

A Modular Numerical Model for  
Stirling Engines and Single-Phase Thermodynamic Machines

by

Steven Mark William Middleton

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical Engineering

University of Alberta

© Steven Mark William Middleton, 2021

# ABSTRACT

---

A numerical model and software interface for the design and modelling of Stirling engines was presented. This model was developed to suit low-temperature Stirling engines, those that run at source temperature of less than 150 °C and run at speeds where temporarily developed losses become significant. The work had three objectives. The first was to create a combined mechanical and thermodynamic model to solve dynamic problems. The second objective was to provide graphical feedback during creation of the geometry and reviewing of a solution. The third objective was to test the model against experimental data taken from a low-temperature gamma-type engine and compare the model against another numerical code.

The resulting model, called the modular single-phase model or MSPM, incorporated a uniform pressure assumption which was used to solve the instantaneous flow rates in a one-dimensional network of pipes. The flow network is generated automatically from arbitrary arrangements of cylindrical or annular extrusions created by the user, within which the solid heat conduction is solved in 2-dimensions. Angular position dependent deformations are driven by the mechanical system, which responded to the forces generated by the gas system. This scheme transferred impulses from the gas network after short increments, which then defined the dynamics next increment. To capture flow losses pressure drops are approximated from gas velocities and the modified pressures are used to calculate the mechanism response.

The software itself presents the user with graphical feedback like that found in CAD software. This makes it possible to generate informative animations of the moving boundaries of an engine. These animations carry forward into the output of the code, presenting temperature, pressure, turbulence, heat flow, flow direction and pressure drop in spatially relevant positions on the virtual engine cross-section. The user can also place sensors, reuse previous simulation data, and run batch tests and optimize engine geometry using the software.

When the uncalibrated model was compared against experimental results featuring an in-lab engine running at 0.56 to 2.26 Hz, this numerical code developed a maximum discrepancy of 43.1% with an average deviation from the experimental results of 30.6%. An exploratory calibration of the effects of compression was conducted drawing on conclusions from the initial

tests, resulting in an overall improvement of the accuracy to an average of 21.9%. The final discrepancy is largely systematic, possibly correctable with reasonable adjustments to the automatically generated convection and friction terms. A sensitivity study of the properties related to heat transfer and friction was presented at two different speeds, the results indicated that the most substantial and predictable effector of power was the convection coefficient. Flow friction became a larger contributor at higher speeds. The code was then compared against SAGE, the numerical code of choice, with 5 tests at 16.7 Hz and 50 bar and with source temperatures ranging from 150 °C to 750 °C. Over these tests MSPM produced a maximum error of 59.1% and an average deviation of 33.5%. When compared against a second patch of in-lab produced SAGE results at slow speeds the two models diverged, it was concluded that the two models featured very different flow loss characteristics at low speeds among a variety of other differences. In a final experiment the optimal design of a beta type Stirling engine was obtained using the geometrical optimization tool within MSPM the results and design process of the beta type engine was presented.

## PREFACE

---

This thesis is an original work by Steven Middleton. Aspects of this research have been published in the following conference publications:

Middleton, S. and Nobes, D.S. (2018) “Dynamic Modelling of Low Temperature Stirling Engines”, *18th ISEC International Stirling Engine Conference*, Tainan, Taiwan, Sept 19-21, 2018

Middleton, S. and Nobes, D.S. (2019) “Modular one-dimensional simulation tool for oscillating flow and thermal networks in Stirling engines”, *4<sup>th</sup> Thermal and Fluids Engineering Conference*, Las Vegas, United States of America, April 14-17, 2019

Middleton, S. and Nobes, D.S. (2021) “Approximations for use in cycling thermodynamic systems: Applications for Stirling engines”, *Proceedings of the CSME International Congress*, Charlottetown, Canada, June 27-30, 2021

The author also contributed to the following publications that did not contribute directly to this Thesis:

Stumpf, C.J.A, Middleton, S. and Nobes, D.S. (2017) "Heat Transfer in Oscillating Fluid Flow Through Parallel Flat Plate Channel Heat Exchangers", *Okanagan Fluid Dynamics Meeting*, Kelowna, British Columbia, Canada, Aug 22-23, 2017

Nicol-Seto, M., Michaud, J. P., Middleton, S. and Nobes, D.S., “Non-Traditional Drive Mechanism Designs for the Improvement of Heat Transfer in Low Temperature Differential Stirling Engines,” *18th ISEC International Stirling Engine Conference*, Tainan, Taiwan, Sept 19-21, 2018

# ACKNOWLEDGMENTS

---

The author would like to acknowledge the assistance of his supervisor,

Dr. David S. Nobes

Lab mates (Team Stirling),

Connor Speer

Jackson Kutzner

Calynn Stumpf

Linda Hasanovich

Jason Michaud

Alex Hunt

David Miller

Gabriel Salata

Michael Nicol-Seto

Matthias Lottmann

as well as co-op students and others that helped inspire this project through to success.

The author would like to thank his wife, Kaybrie, who endured the many hours of programming and inevitable despair this project entailed.

The author would like to acknowledge the financial support for this project from:

Natural Sciences and Engineering Research Council (NSERC) of Canada,

Alberta Innovates Energy and Environmental Solutions

Terrapin Geothermics, and

Future Energy Systems (FES).

# TABLE OF CONTENTS

---

Abstract .....	i
Preface .....	iii
Acknowledgments .....	iv
Table of Contents .....	v
List of Tables.....	xiii
List of Figures .....	xv
List of Symbols .....	xx
CHAPTER 1. Introduction .....	1
1.1 Project Background .....	1
1.1.1 General Background.....	1
1.1.2 Research Activities in DTECL.....	2
1.1.3 Project Goals .....	3
1.2 Stirling Engines .....	4
1.2.1 History.....	4
1.2.2 Stirling Engine Principles.....	4
1.2.3 The Real Stirling Cycle.....	9
1.2.4 What is important for low-temperature engine? .....	14
1.2.5 Summary of Stirling Engines .....	21
1.3 Modeling Techniques .....	21
1.3.1 1 <sup>st</sup> Order Models.....	22
1.3.2 2 <sup>nd</sup> Order Models.....	27
1.3.3 3 <sup>rd</sup> Order Models .....	28
1.3.4 Higher-Order Models .....	30

1.4	Chapter Conclusions.....	31
1.5	Thesis Goals .....	31
1.6	Thesis Outline.....	32
CHAPTER 2. System Development and Architecture.....		33
2.1	Development.....	33
2.1.1	The Gas Medium.....	33
2.1.2	Uniform Pressure.....	33
2.1.3	The Solid Medium.....	34
2.1.4	The Mechanical System .....	34
2.1.5	Gas and Mechanism Relationship.....	35
2.1.6	Axial Symmetry .....	36
2.1.7	First Elements.....	36
2.1.8	Further Abstraction .....	37
2.1.9	The Name .....	37
2.1.10	Final Structure .....	38
2.2	Finite Elements.....	39
2.2.1	Nodes.....	39
2.2.2	Faces.....	41
2.2.3	Node Contacts .....	45
2.2.4	Pressure Contacts .....	46
2.2.5	Shear Contacts.....	46
2.3	Interactable Elements .....	46
2.3.1	Groups .....	47
2.3.2	Bodies.....	47

2.3.3	Matrixes.....	49
2.3.4	Connections.....	55
2.3.5	Bridges .....	56
2.3.6	Leaks .....	57
2.3.7	Non-Connection .....	57
2.3.8	Custom Minor Losses.....	57
2.3.9	Frames .....	58
2.3.10	Mechanism.....	58
2.4	Conclusion .....	61
CHAPTER 3. Core Mathematical Processes.....		62
3.1	Terminology .....	62
3.2	General Heat Transfer .....	62
3.2.1	Thermal Conduction Within Solids.....	62
3.2.2	Thermal Conduction Within Gases.....	63
3.2.3	Thermal Conduction Between Solids and Gases .....	64
3.2.4	Shearing Conduction Enhancement .....	65
3.3	Determining Flow Rates.....	66
3.3.1	Assumptions.....	66
3.3.2	Deriving the Systems of Equations .....	67
3.3.3	Verifying as a Polytropic Process .....	74
3.3.4	Considering Loops .....	75
3.3.5	Considering Flow Losses .....	78
3.3.6	Smooth Property Changes.....	79
3.4	Turbulence .....	81



3.4.1	Open Channels Flows.....	81
3.4.2	Matrix Flows .....	82
3.4.3	Variable Volume Spaces .....	83
3.5	Chapter Conclusions.....	84
CHAPTER 4. Simulation .....		85
4.1	Discretization and Conditioning.....	85
4.1.1	Discretize all Components and Collect Discrete Elements.....	85
4.1.2	Decimate Nodes Based on Size.....	86
4.1.3	Assign Minor Loss Coefficients.....	87
4.1.4	Decimate Triads .....	87
4.1.5	Assign Indexes to all Elements .....	88
4.1.6	Vectorize Node and Face Properties .....	88
4.1.7	Determine Maximum Solid-Conduction Timestep.....	88
4.1.8	Determine the Conduction and Transportation Vectors.....	88
4.1.9	Establish Gas Regions.....	90
4.1.10	Find Loops within each Region.....	91
4.1.11	Define Pressure Loss Matrix .....	93
4.1.12	Vectorize Node Faces .....	95
4.2	Simulation Setup.....	95
4.2.1	Apply Snapshot .....	95
4.2.2	Get Simulation Parameters from the User.....	96
4.2.3	Pre-allocate Memory for Results .....	96
4.2.4	Run Warm-Up Phase.....	97
4.3	Gas Solver Loop .....	97

4.3.1	Calculate Dynamic Properties .....	97
4.3.2	Calculate Flow Independent Flux's.....	98
4.3.3	Calculate Explicit Mass Flux's .....	98
4.3.4	Constrain Time Step Pre-Mass Flux .....	98
4.3.5	Calculate Implicit Mass Flux's .....	98
4.3.6	Constrain Time Step Post-Mass Flux.....	102
4.3.7	Update Properties .....	102
4.3.8	Calculate Turbulence Flux's .....	102
4.3.9	Record Statistics.....	102
4.4	Mechanical Solver Loop.....	103
4.4.1	Calculate Piston Forces .....	103
4.4.2	Calculate Driveshaft Forces .....	103
4.4.3	Calculate Acceleration .....	104
4.4.4	Calculate Next Velocity Target.....	104
4.5	Conclusion.....	104
CHAPTER 5. Advanced Features .....		106
5.1	Solid Temperature Distribution Acceleration.....	106
5.2	Progressive Refinement.....	109
5.3	Geometrical Optimization .....	111
5.4	Conclusions .....	111
CHAPTER 6. Model Usage .....		113
6.1	Constructing a Model .....	113
6.1.1	Display Window.....	114
6.1.2	Left Toolbar – Create, Destroy and Select.....	115

6.1.3	Bottom Toolbar – View options.....	124
6.1.4	Top Toolbar – Save / Load options.....	125
6.1.5	Top Toolbar – Geometrical Optimizer, Relation Toggle & Dropdown mode..	125
6.1.6	Right Toolbar – Property dropdown and Simulation options .....	126
6.1.7	Start the Simulation.....	128
6.2	Discretization.....	129
6.2.1	Spatial Discretization .....	130
6.2.2	Temporal Discretization.....	132
6.3	Simulation Tools.....	133
6.3.1	Snapshot .....	133
6.3.2	Test Set Running.....	133
6.3.3	Geometrical Optimization.....	133
6.4	Model Outputs .....	134
6.4.1	Engine Assessment.....	134
6.5	Chapter Conclusions.....	141
CHAPTER 7. Validation.....		143
7.1	Theoretical Validations.....	143
7.1.1	Steady-State Solid Heat Conduction.....	143
7.1.2	Transient Solid Heat Conduction .....	144
7.1.3	Adiabatic Compression/Expansion of Gas.....	147
7.1.4	Isothermal Compression/Expansion of Gas .....	148
7.2	Comparison with Experiments .....	149
7.2.1	Constant Speed Steady-State Experiments .....	151
7.2.2	In Cycle Speed Variations.....	158

7.2.3	Sensitivity Studies .....	161
7.3	Comparison with SAGE .....	162
7.3.1	In High-Temperature, High Speed Context .....	162
7.3.2	In Low-Temperature, Low-Speed Context .....	169
7.4	Optimization Studies .....	171
CHAPTER 8.	Conclusions .....	175
8.1	Conclusions .....	175
8.1.1	Create a Combined Mechanical and Thermodynamic Model for Low-Temperature Stirling Engines.....	175
8.1.2	Ensure the model is User-Friendly and Intuitive .....	175
8.1.3	Validate the Model against Experimental and a well Established Numerical Model	176
8.2	Sources of Error.....	177
8.2.1	Decoupling of Flow Friction and Volumetric Flow Rate .....	177
8.2.2	Constant Properties .....	177
8.2.3	Ideal Gas Representation of the Fill Gas.....	177
8.2.4	Radiation Heat Transfer is Ignored .....	177
8.2.5	No Contact Resistance .....	178
8.2.6	Nusselt Number is Node Based, not Surface Based.....	178
8.2.7	Constant Friction Coefficients in Mechanism.....	178
8.2.8	One Dimensional Flow Assumption .....	178
8.2.9	Minor Loss Coefficients are Naively Applied .....	179
8.2.10	Fluid Inertia and Acoustics, are Ignored .....	179
8.2.11	Steady-State Convergence.....	180
8.2.12	Calculation Errors.....	180

8.3	Future Opportunities.....	180
8.3.1	Real Gases.....	180
8.3.2	Interface for Simulating Control System .....	181
8.3.3	Multi-Phase simulations.....	181
8.3.4	Source/Sink Simulation.....	181
8.3.5	Material Distortion .....	181
8.3.6	Improved modelling of Entrance Turbulence and Swirl in Open Volumes.....	182
8.3.7	String or Text File Based Test Set Run Files .....	182
8.3.8	Modelling of explicit faces.....	182
8.3.9	Improvements to Geometry Optimizer .....	182
8.3.10	Parallelization .....	183
8.3.11	Other Programming Languages.....	183

# LIST OF TABLES

---

Table 2.1: Discretization specific properties for common regenerator types. ....	54
Table 4.1: Simulation Parameters and Description.....	96
Table 7.1: Steady-state Heat Conduction Validation: Material Properties .....	144
Table 7.2: Experimental properties of Transient heat conduction test.....	145
Table 7.3: EP-1 test sets.....	152
Table 7.4: Velocity variation experiment. “B” is a reference to the “Box” / square wave. “0” refers to the standard / sinusoidal trial (Elliptical factor of 0). .....	159
Table 7.5: Sensitivity Studies (results are colored based on absolute value, -50% uses backwards difference, +50% uses forward difference, $\pm 2\%$ uses central difference for slope calculation).....	161
Table 7.6: Alpha engine geometrical properties for SAGE comparison [44].....	163
Table 7.7: Alpha engine test and specific geometrical properties for SAGE comparison [44].....	163
Table 7.8: Output power comparison between MSP and SAGE simulations at 16.7 Hz .....	164
Table 7.9: Unique properties for low-speed/low-temperature alpha engine SAGE comparisons.....	169
Table 7.10: Test results for low-speed/low-temperature alpha engine SAGE comparisons..	169
Table 7.11 Test properties for Beta-Engine Optimization .....	172
Table 7.12 Geometrical properties for Beta-Engine Optimization, Optimized for Maximum Power vs Gas Volume (1 Hz, filled with air).....	173
Table 7.13 Test properties for Beta-Engine Optimization (listed component volumes are for available gas volume only, heat exchangers have a surface area to volume ratio of about 1.52 m <sup>2</sup> /Litre, 1 Hz, 80% efficient mechanism, wire diameter of 0.1 mm in regenerator, HX volume is for heater and cooler only, with air if not otherwise indicated).....	174

Table 8.1: User inputs and correlations for various properties based on regenerator type. Correlations from (Gedeon, SAGE users manual [35]).	208
Table 8.2: User inputs and correlations for various properties based on heat exchanger type: Fin Enhanced Surface, Fin Connected Channels. Correlations from [35] unless otherwise indicated.	211
Table 8.3: User inputs and correlations for various properties based on heat exchanger type: Tube and Plate Heat Exchangers.	212
Table 8.4: User inputs and correlations for various properties based on heat exchanger type: Individually Finned Tube Heat Exchangers.	213
Table 8.5: User inputs and correlations for various properties based on heat exchanger type: Bare Tube Banks (internal). Correlations from [35] unless otherwise indicated.	214

# LIST OF FIGURES

---

Figure 1.1: Fundamental components of Stirling engines: (top) alpha type engine, (bottom) gamma/beta type engine .....	5
Figure 1.2: The 4 phases of the ideal Stirling engine cycle: arrows represent a motion that is occurring during the phase. Components are defined in Figure 1.1. ....	6
Figure 1.3: Ideal Stirling engine cycle in a pressure-volume diagram form.....	6
Figure 1.4: Illustration of the isothermal idealization.....	8
Figure 1.5: Illustration of the adiabatic idealization .....	8
Figure 1.6: Pressure-volume diagram of an engine with a sinusoidal volume variation compared against one with idealized motions .....	10
Figure 1.7: Effect of thermal losses on the pressure-volume (indicator) diagram.....	11
Figure 1.8: Thermal energy pathways in a Stirling engine .....	11
Figure 1.9: Thermal energy pathway through a heat exchanger.....	12
Figure 1.10: Thermodynamic consequences for of low temperature (ratio) engines.....	14
Figure 2.1: Gas/solid system and mechanical system interaction loop.....	35
Figure 2.2: A body and member connections. In cylindrical elements, the inside vertical connection may be reduced to a single line at the axis. ....	36
Figure 2.3: A flow chart of the system architecture.....	38
Figure 2.4: Geometrical cases and resulting friction length .....	42
Figure 2.5: Shear and velocity factor input variables.....	45
Figure 2.6: Discretization modes for body.....	48
Figure 2.7: How elements are discretized in the fin enhanced surface type heat exchanger ...	54
Figure 2.8: How elements are discretized in the fin connected channels and finned tube type heat exchanger .....	55
Figure 2.9: How elements are discretized in the tubes bank internal type heat exchanger.....	55



Figure 2.10: Permutations of the bridge component definition (1) two horizontal (disk) faces are stacked along a central axis (2) two vertical (annular shell) faces are aligned at some offset from the origin (3) two horizontal faces are stacked with axis offset by a specified amount (4) a horizontal face is perpendicularly mated up against a vertical face with the axis at a prescribed offset from the origin of the vertically aligned face. .... 56

Figure 3.1: Illustration of shear driven mixing ..... 65

Figure 3.2: Examples of common loops found in Stirling engines..... 75

Figure 3.3: Angular locations of  $V_0$  through  $V_3$  ..... 80

Figure 4.1: Process structure of the simulation loop, elements inside of the box are repeated until the simulation has timed out or converged. .... 85

Figure 4.2: Flow of information during discretization ..... 86

Figure 4.3: Example of a triad elimination action: faces are selected based on their relative size, the remaining 2 faces are then modified to compensate. .... 87

Figure 4.4: Loop finding algorithm illustration. (dashed) face that has an area of zero at any point of the cycle. (red) eliminated face. (blue node) starting point for algorithm. (green) discovered loop. .... 91

Figure 4.5: Loop data structure and graphical representation ..... 92

Figure 4.6: Determining the independent faces (dashed) face that has an area of zero at any point of the cycle. (red) eliminated face. (blue node) visited node. (green) set of independent equations obtained at iteration step ..... 94

Figure 5.1: Comparison of accelerated vs natural convergence of Stirling engine performance of the EP-1 model (defined in Appendix C) ..... 109

Figure 5.2: Computational Time vs Number of Gas Nodes, ..... 110

Figure 6.1: The main MSPM graphical user interface. .... 113

Figure 6.2: The main model display window ..... 114

Figure 6.3: Create, Destroy and Select Toolbar ..... 115

Figure 6.4: Examples of Bodies Being Used to Construct Geometry ..... 116

Figure 6.5: Example of an engine containing two groups, one for the main engine assembly and a second for a power piston offset from the main axis. ....	117
Figure 6.6: Example of a usage of the bridge component.....	118
Figure 6.7: Example indicator diagram from PV Output sensor .....	121
Figure 6.8: View options.....	124
Figure 6.9: Save / Load options .....	125
Figure 6.10: Geometrical Optimizer, Relation Toggle & Dropdown mode .....	125
Figure 6.11: Property dropdown and Simulation options .....	126
Figure 6.12: Properties of Bodies including location of Change Matrix where Matrix components are initialized .....	127
Figure 6.13: Run Interface .....	128
Figure 6.14: A plot and definition of the pressure-volume (PV) diagram .....	134
Figure 6.15: Generic Heat Engine Model .....	136
Figure 6.16: Sensor Usage Examples as shown in the GUI.....	137
Figure 6.17: Output of sensor (a) point sensor (b) line sensor, locations shown on Figure 6.16 .....	138
Figure 6.18: An example temperature heatmap snapped during an animation of the modelled Stirling engine.....	139
Figure 6.19: An example conduction heatmap snapped during an animation of the modelled Stirling engine.....	140
Figure 6.20: 4 frames of an instantaneous flow velocity plot snapped during an animation of the modelled Stirling engine (running at 1 Hz) .....	141
Figure 7.1: Steady-state temperature profile obtained via heat conduction through a layered annular conductor compared against analytical predictions .....	144
Figure 7.2: Discretization scheme for transient heat conduction test .....	145

Figure 7.3: Analytically obtained results vs simulated temperature with time measured at the center of test block.....	147
Figure 7.4: Analytically obtained results compared against simulation results in the case of adiabatic compression/expansion.....	148
Figure 7.5: Analytically obtained results compared against simulation results in the case of isothermal compression/expansion.....	149
Figure 7.6: EPM engine body geometry as shown in graphical user interface of software...	151
Figure 7.7: Indicator diagram comparison between EPM-1 experiments and MSPM. (a) DP & PP: Standard at 1.1055 Hz (b) DP: Square Wave Elliptical, PP: Standard at 0.8818 Hz (c) DP & PP: Square Wave Elliptical at 1.1992 Hz .....	154
Figure 7.8: Non-dimensional power for each of the 12 matching experiments.....	155
Figure 7.9: MSPM vs experimental power piston indicated work.....	156
Figure 7.10: MSPM displacer piston indicated work, experimental version not collected ...	157
Figure 7.11: Indicator diagram comparison between EPM-1 experiments and simulations with reduction in stroke by 15.7% and increase in dead volume equivalent to 7.8% of stroke. (test: Standard-Standard at 1.1055 Hz shown) .....	158
Figure 7.12: Instantaneous angular velocity for one cycle for both the EPM-1 physical tests and MSPM simulations.....	160
Figure 7.13: Velocity ratio results for experiment and simulation for 4 different velocities.	160
Figure 14: Annotated alpha engine for Phase 135°, Source 150 °C test as shown in MSPM	162
Figure 7.15: Pressure – volume diagram comparison between MSPM and SAGE, data extracted from: [44].....	168
Figure 7.16: Pressure – volume diagram comparison between MSPM and SAGE for low-speed, low-temperature, low-pressure case.....	171
Figure 7.17: Depiction of beta-layout engine to be optimized. Large volume beneath engine represents the crankcase, the additional bodies jutting out of the engine are added to prevent the	

optimizer from making certain features too small leading to instabilities in the solution. (these bodies will overlap before the gas body becomes too small) ..... 172

Figure 8.1. Slider-crank Mechanism: dimensions, masses, and gravity ..... 191

Figure 8.2. Free Body Diagram of Crank Arm ..... 192

Figure 8.3. Free Body Diagram of Connecting Rod ..... 193

Figure 8.4. Free Body Diagram of Piston Head ..... 194

Figure 8.5: Rhombic Drive Mechanism: dimensions, masses, and gravity ..... 197

Figure 8.6. Free Body Diagram of Crank Arm ..... 199

# LIST OF SYMBOLS

Roman Alphabet Variables		
Symbol	Description	Unit
$A$	Area	$m^2$
$C$	Conduction coefficient	$W / K$
$c$	Constant used in calculation	Various
$C_D$	Constant of turbulent dissipation. In tubes = 0.08.	-
$c_p$	Specific heat under constant pressure.	$J / kg / K$
$c_T$	Heat capacity of solid	$J / kg / K$
$c_v$	Specific heat under constant volume.	$J / kg / K$
$d$	Diameter	$m$
$d_h$	Hydraulic Diameter in the volume element = $4V/S_{solid}$ , as a face element = $4A_{fc}/Perimeter_{fc}$	$M$
$\dot{E}_{ext}$	Power, entering a space, derived from a source that is assumed to be not strongly connected to the upcoming change in velocity, such as conduction or compression.	$W$
$\dot{E}_{flow}$	Power, entering a region from a different region. Based on energy required to push gas into a space of a given pressure.	$W$
$\dot{E}_{kin}, \dot{E}_{pot}, \dot{E}_{int}$	Kinetic, Potential, Internal changes in energy.	$W$
$\dot{E}_{shaft}$	Power as measured at the output drive shaft of the engine.	$W$
$f$	Frequency	$Hz$
$f(...)$	Function of ...	Various
$F_{shear}, F_{velocity}$	Coefficient on angular speed producing instantaneous linear velocity. For shearing rate and linear velocity of face respectively.	$m / rad$
$g$	Gravitational constant = $9.81 m/s^2$	$m / s$
$h$	Convection coefficient.	$W / m^2 / K$
$I$	Moment of Inertia	$kg m^2$
$k$	Thermal conductivity.	$W / m / K$
$K$	Minor loss coefficient	-
$L, l$	Length, linear or characteristic.	$m$
$m$	Mass.	$kg$
$\dot{m}$	Mass change or mass exchange.	$kg / s$
$N_f$	Darcy Friction factor	-
$N_{Fo}$	Fourier Number = $\alpha_t \cdot \delta / \Delta x^2$	-
$N_k$	Streamwise thermal conductivity factor.	-
$N_{Nu}$	Nusselt number = $D_h \cdot h / k$	-
$N_{Pe}$	Pechlet Number = $N_{Re} \cdot N_{Pr}$	-

$N_{Re}$	Reynolds number $= U \cdot D_h \cdot \rho / \mu$	-
$N_{Re_{laminar}}$	Upper limit Reynold's number for a flow to be considered laminar. = 1,700	-
$N_{Va}$	Valensi number $= \rho \omega d_h^2 / \mu$	-
$N_{West}$	West Number = $(W_s / P \cdot V_{swept} f) (T_H - T_L / T_H - T_L)$	-
$P$	Pressure	Pa
$\dot{Q}$	Heat Transfer by conduction and convection into control volume.	W
$R$	Thermal resistance	K / W
$r$	Radius	m
$r_c$	Compression ratio	-
$R_{spec}$	Specific Gas Constant is equal to the $= R_{univ} / M$	J / kg / K
$S$	Perimeter of a face or area.	m
$\dot{S}$	Entropy generation rate.	-
$T$	Temperature, Torque applied by engine load	K, N m
$t$	Time, measured from start.	s
$u, U$	Specific internal energy, total internal energy	J / kg
$U$	Velocity, considered the average over the face.	m / s
$V$	Volume	m <sup>3</sup>
$\dot{V}$	Change in volume or flow rate.	m <sup>3</sup> / s
$\hat{v}$	Specific volume	m <sup>3</sup> / kg
$W$	Work	J
$w$	Weight for interpolation with values between 0 and 1	-
$\dot{W}_b$	Instantaneous energy extracted from moving the boundary of the control volume.	W
$x, y$	Scalar position relative to an origin.	m
$y$	Sign of face, when used with respect to node $i$ , and face $i \rightarrow j$ , the sign is negative because the natural flow is out of node $i$ .	-
$z$	Vertical position	m

---

Greek Alphabet Variables

---

Symbol	Description	Unit
$\alpha_t$	Thermal diffusivity = $k \cdot \rho / c_p$	m / s
B	Ratio of Areas	-
$\beta$	Porosity	-
$\delta$	Small increment in time, often associated with the length of a timestep.	s
$\theta$	Angular Position	rad
$\theta_{inc}$	Angular Increment, over which cycle is divided. Also represents mechanism acceleration lag.	rad
K	Turbulent kinetic energy	J
$\kappa$	Specific Turbulent kinetic energy	J / kg
$\mu$	Dynamic Viscosity	Pa s
$\rho$	Density	kg / m <sup>3</sup>
$\tau$	Turbulence weighting factor, value of zero denotes laminar flow, value of 1 denotes fully turbulent flow.	-
$\omega$	Angular Speed	rad / s

## Subscripts

Symbol	Description
$0$	Measured at start of local period of consideration.
$avg$	Arithmetically averaged over the entire cycle.
$c, k, r, h, e$	Referring to compression, cooler, regenerator, heat and expansion spaces respectively.
$cond$	Associated with conduction or convection
$cycle$	Accumulated over the entire cycle.
$dead$	Dead volume, Eulerian volume that doesn't change its temperature though matter may flow through it.
$DP, PP$	Displacer Piston, Power Piston
$eff$	Not a true measurement but the composite value after the initial is modified by an external, not fully modelled effect.
$fc$	Associated with a face, an interface between two spaces.
$gen$	Internally produced.
$H$	High, often referring to source property.
$i, j, nd$	Indices referring to separate spaces or nodes.
$i \rightarrow fc, fc \rightarrow i, i \rightarrow j, \dots$	Referring to a delta up to an interface from the center point of a space.
$ij$	Arithmetic mean of space $i$ and space $j$ properties.
$in, out$	Into system/space, out of system/space.
$L$	Low, often referring to sink property.
$load$	Refers to load on drive shaft
$loop$	A property characteristic of a fully connected loop of cells
$n$	Iteration subscript, indicating current and next value.
$o, i, max, min$	Outside and Inside, or alternatively farther or closer to the datum, often used for bounds of a shape.
$perp, para, th, g, f, tube, s, c, w, wth$	Perpendicular, parallel, fin thickness, gap between features, fin or fin space, spacing or separation, channel, feature width, wall thickness.
$region$	Bulk property of a region, which is an interconnected zone of gas such as a engine volume. Either the sum or a uniform constant.
$shaft$	Measured at the output shaft of the engine.
$swept$	Quantity forced to enter heat exchange components
$t, t + \delta$	Current time, next time after timestep $\delta$
$up$	Quantity is taken from the space upwind of the current feature.



# CHAPTER 1. INTRODUCTION

---

The following chapter outlines background information and the motivation for investigating Stirling engines in the low-temperature regime. Relevant literature on the topic of numerical modeling of Stirling engines will also be discussed. This chapter concludes with a summary of objectives and thesis structure.

## 1.1 Project Background

---

### 1.1.1 General Background

The research conducted at the University of Alberta's Dynamic Thermal Energy Conversion Laboratory (DTECL), is focused, as of the writing of this thesis, on investigating the operation of Stirling engines with reservoirs at temperatures between 100 °C and ambient (5°C). Such engines are informally classified as low-temperature difference Stirling engines (LTDSE).

Heat sources that produce such low-grade heat streams in high amounts include: industrial waste heat and geothermal heat, with geothermal being a focus of DTECL. Waste heat, in a study by Stricker et al [1], accounts for 70% of the energy used in Canadian industries. With regards to geothermal energy: Banks and Harris [2] estimated that a total of 6,100 MW of thermal energy potential in their studied region (an area of Alberta South of Grand Prairie, West of Edmonton and North West of Calgary). In their study, the highest potential sources (those with source temperature above 120 °C) provide a total up to 712 MW of electrical power which could be reliably gathered over 30 years with existing technology [2]. In addition to these findings, the prolific oil and gas activity throughout Alberta has generated around 170,000 orphaned wells, which represent 37% of all wells in Alberta [3]. Given these numbers, there is a likelihood that many of these may have favorable downhole temperatures. With the high upfront cost of geothermal development, this offers an important jumpstart to both geothermal investigation and capacity installment. Despite this, no active examples of geothermal power generation exist in Canada, with the first example to come online in Saskatchewan at the end of 2021 [4].

There are relatively few competing technologies designed for such temperature conditions. Among them is the organic Rankine cycle (ORC), Kalina cycle and thermoelectric technologies [5]. The ORC, is a relative to the steam Rankine used for high-temperature applications. It has been very successful globally with 250 MW in electrical generating capacity as of 2015, with roughly 40% of that capacity generated from geothermal sources [6]. The Kalina cycle has the potential for higher efficiency [7] than the ORC and improved cycle control. However, it is a proprietary technology with increased complexity, scale, and corrosion than an ORC [5]. Thermoelectric technologies offer simplicity given their solid-state nature, but often have poor efficiencies with state of the art thermoelectric technologies achieving only 12% [5] of the Carnot limit and the vast majority achieving less than 10% of Carnot [8], in the low-temperature regime.

Stirling engines are closed cycle, externally heated, reciprocating engines that utilizes a chemically inert working gas [9] such as hydrogen, helium or air and are made of common structural materials such as steel, polymers and aluminium. Stirling engines offer the additional benefit of being able to increase in power density through pressurization while containing no valves, pumps or turbines. This attribute makes them relatively simpler on an individual basis than competing fluid-based technologies. At larger scales, engines would be coupled together, and the resulting modularity may improve the ability of a generator to provide more consistent power.

### **1.1.2 Research Activities in DTECL**

Stirling engines are a relatively insignificant player in the global low-temperature energy market. It is the objective of the DTECL group to investigate whether this is a valid oversight. Research began with Speer [10] who modified an existing mathematical model, the SIMPLE model by Urieli and Berchowitz [11] to include a host of additional losses. This was used to predict experimental results from a modified 90-degree gamma engine converted from a high temperature solar application to work with low temperatures. The conclusions determined that the error of the SIMPLE model increased substantially at lower temperatures (on the order of 150%) and speeds and established that low temperature engines required different geometry and additional investment in loss prevention. Stumpf [12] optimized the operating parameters of a large diameter low-temperature gamma type engine for maximum shaft power. In doing so established a means of predicting the optimal compression ratio based on the temperature ratio. Stumpf [12] also found

that the West number for well designed low temperature engines rested around 0.21, lower than the expected value for high temperature engines equal to 0.25, indicating that performance expectations are different for low-temperature engines. Miller [13] investigated the effect of flywheel size on transient and steady-state performance. Miller [13] also investigated improvement avenues for the modified model from Speer [10] with regards to low temperature machines, the studies conducted showed that the model was less accurate for low-temperature regimes. The suggestions were to include leakage into the model parameters and include a more sophisticated mechanical power loss calculation and gas spring losses. Michaud [14] performed an experimental optimization study on a low temperature alpha type engine. Though the engine failed to run due to poor heat conduction and higher mechanical losses optimal crank angle was determined by driving the engine with a motor.

The reason's why Michaud's engine [14] failed to run had not been predicted by the SIMPLE model even after modification by Speer [10], where it still predicted 5 Watts of power. This was a consistent finding through all the projects that existing models at the time performed poorly when estimating low-temperature scenarios. For Speer [10] the modelling error increased dramatically at low-temperature due to ignored losses, losses which Miller [13] identified as gas spring hysteresis leakage and mechanical losses. Additionally, feedback from colleges indicates that user error in encoding the engine behaviour into a numerical model is one of the most challenging aspects of engine modelling using such models.

### **1.1.3 Project Goals**

Given the work that has been completed at the University of Alberta, a new numerical model was required. The model, would satisfy the following constraints:

1. It will be designed in the low temperature context, including features that are important to low temperature engines, these will be discussed in further sub-sections.
2. The new model will be validated against experimental data as well as an existing commercially available numerical code known as SAGE.
3. It will assist in the design of new engines, by allowing the incorporation and assessment of geometrical features in a physical realizable arrangement. This will be realized through a

solid modelling interface. Which, given its visual nature, will be simple and intuitive to learn, a benefit, given the user-base is regularly changing.

## **1.2 Stirling Engines**

---

### **1.2.1 History**

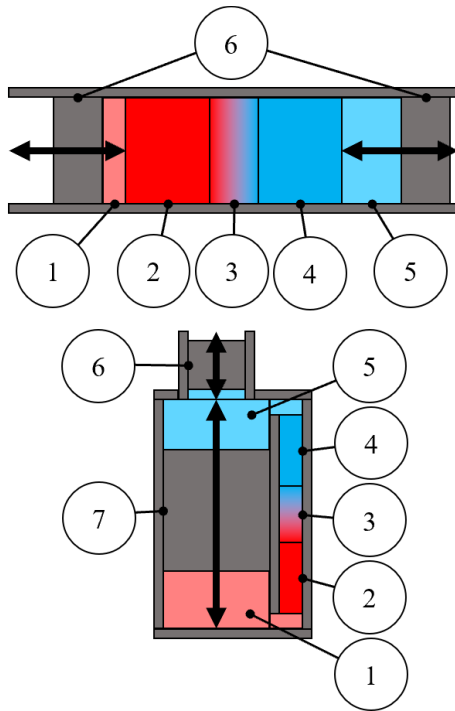
The Stirling engine, often called a hot air engine, as stated in the 1816 patent [15] was invented by Rev. Robert Stirling as an industrial prime mover in the early days of the steam engine. Though its introduction was preceded by other air engines by more than a century, the addition of the economizer, made the Stirling engine a much more efficient invention for the time. This economizer, now termed a regenerator [16], is an internal thermal mass built to store thermal energy for a later point in the cycle. In addition to its efficiency, Stirling engines were noted as safer and quieter but ultimately had poorer economic efficiency [17]. Eventually Steam engines became more efficient and safer and Stirling engines became low-power domestic engines before being overshadowed again by electric motors [17]. The Stirling engine reemerged as a quiet alternative to internal combustion for military radio sets by Philips [9], their research forms the basis of what is known today, but ultimately Philips only made financial success with reverse Stirling cycle cryo-coolers. Today, Stirling engines quietly power submarines [18], domestic combined heat and power systems [19], and high efficiency concentrated solar power systems [20]. Further detail into the history of Stirling Engines is discussed by Lloyd [21].

### **1.2.2 Stirling Engine Principles**

Stirling engines fall under the category of heat engines, a group of processes that generate usable energy from the heat transfer from high to low temperatures. The principles of design and operation of Stirling engines are described in detail by West [22], as well as through preceding thesis: [12], [13], [10] and [14]. The principles are also summarized in brief below.

#### **1.2.2.1 Fundamental Components**

There are several fundamental components of a Stirling engine, which for the purpose of clarity are displayed in Figure 1.1 for both an alpha (top) as well as a gamma type (bottom) engine.

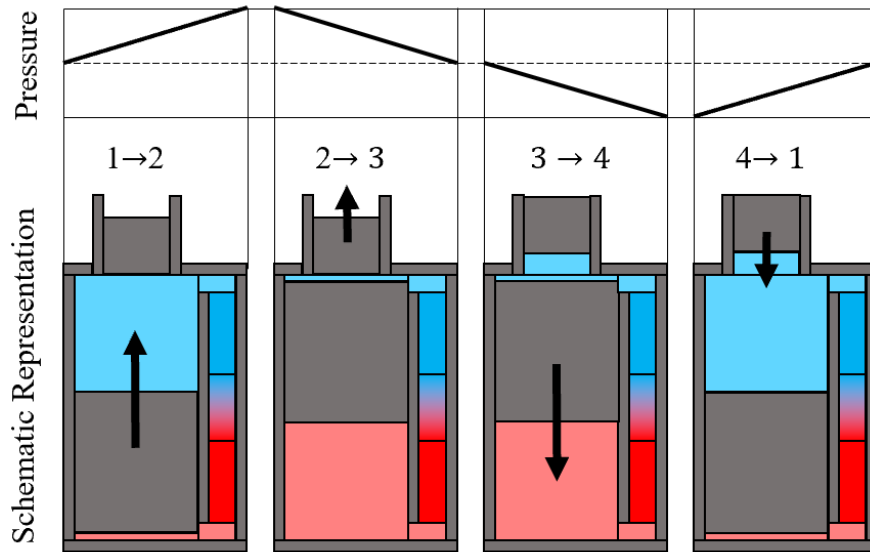


- (1) **Expansion Space** – The variable volume zone that when expanding facilitates the bulk heating of the gas.
- (2) **Heater** – The group of surfaces that facilitate the heating of the gas. These surfaces may be within the expansion space or have their own separate zone.
- (3) **Regenerator** – A set of surfaces and material volume that store a thermal gradient between the heater and cooler, which preheats and precools the gas before entering the opposite exchanger.
- (4) **Cooler** – The group of surfaces that facilitate the cooling of the gas. These surfaces may be within the compression space or have their own separate zone.
- (5) **Compression Space** – The variable volume zone that when expanding facilitates the bulk cooling of the gas.
- (6) **Power Piston** – A piston which separates the engine and a volume outside of the engine. This piston changes the volume of the engine and extracts work.
- (7) **Displacer Piston** – A piston which divides the expansion and compression space. The displacer's function is to facilitate the movement of gas between the expansion and compression space.

**Figure 1.1: Fundamental components of Stirling engines: (top) alpha type engine, (bottom) gamma/beta type engine**

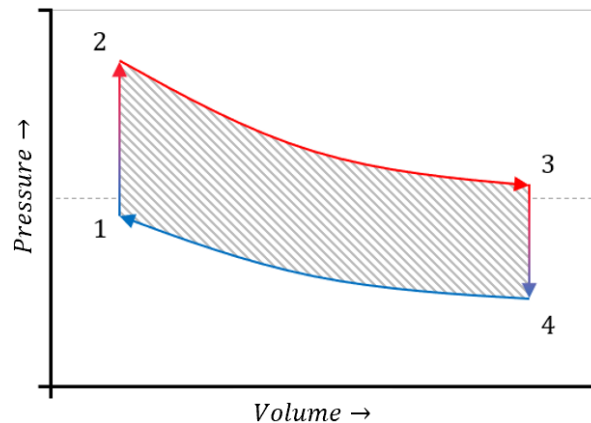
### 1.2.2.2 Ideal Stirling Cycle

All the possible configurations of a Stirling engine are an attempt to mimic the ideal Stirling cycle, while satisfying specific constraints and minimizing losses within their intended application. These configurations are reviewed in greater detail by West [22] as well as Martini [23]. A summary, in the form of a gamma type engine, is also provided here.



**Figure 1.2: The 4 phases of the ideal Stirling engine cycle: arrows represent a motion that is occurring during the phase. Components are defined in Figure 1.1.**

Figure 1.2 schematically illustrates the 4 motions of the ideal Stirling cycle, which appear also in Figure 1.3 as well as in following subsections. Figure 1.3 represents a different style of representation, known in the literature as an indicator diagram. A typical indicator diagram plots engine pressure against internal volume. The line color represents average engine internal temperature, the horizontal dashed line represents the average pressure of the buffer space – the space around the engine which imparts pressure on the backside of the power piston – and the shaded region represents the total sum of work produced by the engine during a single cycle.



**Figure 1.3: Ideal Stirling engine cycle in a pressure-volume diagram form**

The 4 steps of the ideal Stirling cycle outlined in Figure 1.3 are (1-2) isochoric heat addition, (2-3) isothermal expansion (3-4) isochoric heat removal and (4-1) isothermal compression. These steps are described in detail in the following subsections.

#### **1.2.2.2.1. (1-2) Isochoric Heat Addition**

---

The gas is pushed, by the displacer piston, from a mostly cold state to a mostly hot state. During this stage, heat is added through the heat exchange surfaces into the gas. This produces a net increase in the pressure of the engine and occurs while the volume is smallest.

#### **1.2.2.2.2. (2-3) Isothermal Expansion**

---

Energy, in the form of boundary work, is extracted by the expansion of the engine volume via the power piston. Simultaneously, the thermal sources add thermal energy to compensate for expansion cooling, thus maintaining an isothermal environment.

#### **1.2.2.2.3. (3-4) Isochoric Heat Removal**

---

The gas is pushed, by the displacer piston, from a mostly hot state to a mostly cool state. During this stage, heat is discharged through the heat exchange surfaces from the gas. This produces a net decrease in the pressure of the engine and occurs while the volume is largest.

#### **1.2.2.2.4. (4-1) Isothermal Compression**

---

Energy, in the form of boundary work, is added through the compression of the engine volume via the power piston. Simultaneously, the thermal sources remove energy to compensate for compression heating, thus maintaining an isothermal environment.

### **1.2.2.3 Methods of Idealization**

Every idealized heat engine follows Carnot's principle [24], which postulates that a heat engine is most efficient when all of its processes are reversible. A reversible process indicates that it happens in a way that requires the same amount of energy to accomplish a task as it does to undo it. Alternatively, the reversible process does not produce entropy, which is a representation of a loss of potential. There are two idealized thermal models, isothermal [25] and adiabatic [11], which enable the ideal Stirling cycle to achieve its maximal efficiency. These two models establish that heat transfer occurs always over a temperature difference of exactly zero. This negligible temperature difference is important for heat transfer to be reversible, as entropy generation is

proportional to the difference in the inverses of temperatures by the following formula, from Clausius [26]:

$$\dot{S} = \dot{Q} \left( \frac{1}{T_1} - \frac{1}{T_2} \right) \quad (1)$$

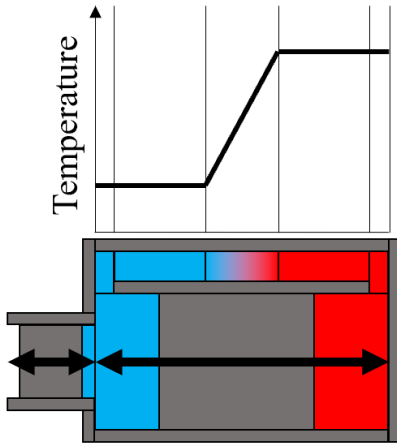
where:  $\dot{S}$ : Entropy generation.

$\dot{Q}$ : Thermal energy flux between temperature sources.

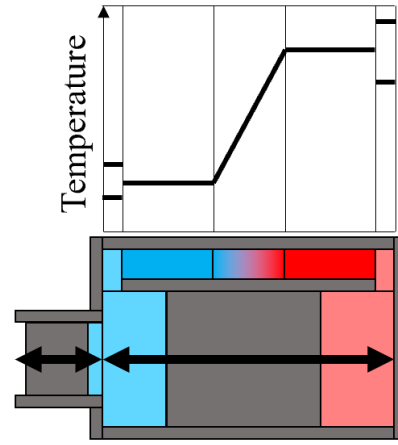
$T_1, T_2$ : Temperatures, measured at two positions.

Notably, at higher temperatures a larger temperature difference can be utilized for the same entropy generation rate due to the asymptotic nature of the inversed temperatures.

The analysis and derivation of these models are described in detail along side the SIMPLE model [11]. The two models are quite similar; however, the adiabatic model allows the temperatures to swing in the compression and expansion spaces according to compression heating and cooling. Conversely, the isothermal model maintains a constant temperature in those spaces. This is illustrated in Figure 1.4 and Figure 1.5 below.



**Figure 1.4: Illustration of the isothermal idealization**



**Figure 1.5: Illustration of the adiabatic idealization**

The isothermal idealization shown in Figure 1.4 is discussed by Urieli [11] and introduced by Gustav Schmidt [25] in 1871. This analytical simplification describes an engine in which the temperature in all spaces remain static. Thus, all temperature change occurs during the flow



through the regenerator and all other heat exchange occurs at a temperature difference of zero. Notably, the compression and expansion spaces are kept consistently at the temperature of their respective heat exchangers. Such an engine would exist only in a slowly cycling engine with heated cylinder spaces and a perfectly effective regenerator.

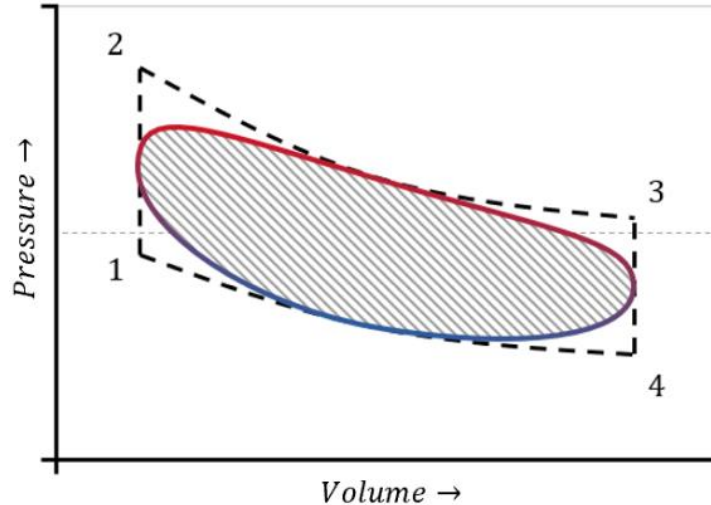
The adiabatic idealization as shown in Figure 1.5 was introduced by Urieli [11] and serves as an alternative to the ideal isothermal model. This model implies that heat transfer only occurs within the boundaries of the heater, cooler and regenerator. The only variation between this and the isothermal model is that the expansion and cooling spaces can oscillate according to compression cooling and heating without heat transfer to the walls. The regenerator and heat exchangers are perfectly effective, and thus all heat transfer pathways are kept at a temperature difference of zero despite the discontinuities. Urieli argued that this modification was more realistic for quickly rotating engines.

### **1.2.3      The Real Stirling Cycle**

The following subsections discuss the idea of a realistic Stirling engine, an engine that diverges from the impractical ideal cycle.

#### **1.2.3.1   Continuous Volume Variations**

Real systems use real mechanisms to move the boundaries of the engine. Kinematic engines utilize slider cranks, yokes, rockers or wobble plates [9], which form motions derived from the sum of several sinusoids, while the resonant free-piston engines use springs, forming near sinusoidal motions. This is a divergence from the discontinuous motion prescribed by the ideal cycle. The effect of these smoothed motions can be seen in Figure 1.6 below, the discontinuous ideal indicator diagram is shown as the dashed outline.

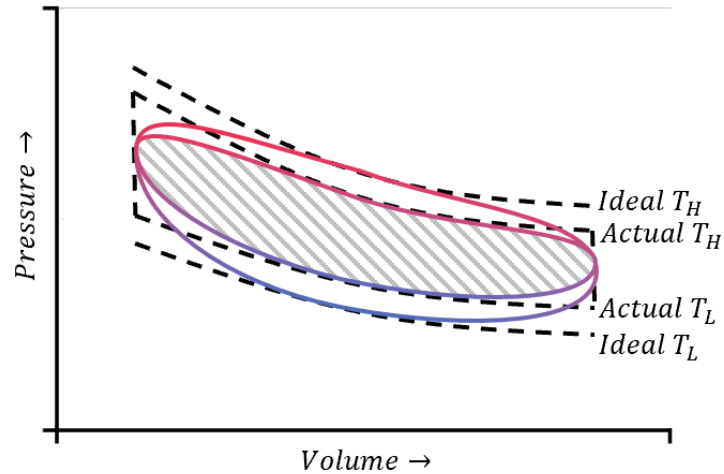


**Figure 1.6: Pressure-volume diagram of an engine with a sinusoidal volume variation compared against one with idealized motions**

Note that the area of the curve is smaller than the ideal case for the same volume and temperature bounds, indicating that the simple act of changing the motion of pistons relative to pressure extremes results in lower engine power. Commonly, mechanisms are selected to maximize the lifetime of the engine, with more specialized mechanisms being avoided for reasons of balancing, design risk or manufacturing complexity [9]. Thus, with Stirling engines there is a drive to produce a mechanism that is both mechanically efficient, produces good motion to maximize the indicated power and offers good control over the phasing of the engine pistons. Senft [27] discusses the mechanical efficiency and design of Stirling engines to great length and produced a series of efficient low-temperature engines. Hargreaves [9], documents the long history of Phillip's Stirling engine technology which includes a variety of tested and successful mechanical configurations.

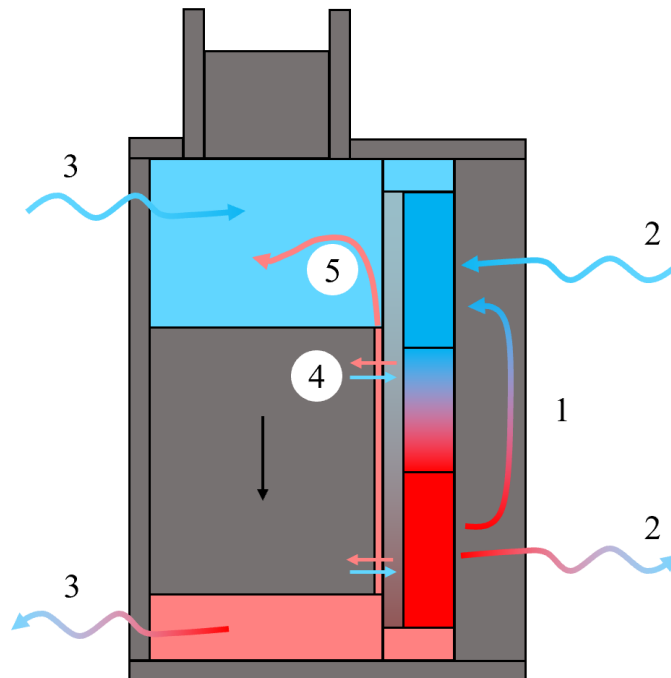
### 1.2.3.2 Imperfect Thermal Control

The ideal Stirling engine cycle is highly temperature-controlled, energy is assumed to transmit only between the gas and the surfaces designed for heat transfer and it does so perfectly without the solid surface changing temperature. In real systems, deviations from this ideal thermal assumption always result in a loss of output energy. The following sub-sections outline how these deviations come about. Losses in this area affect the indicator diagram by reducing the temperature bounds, which has the effect of moving the isothermal/adiabatic lines closer together. This is shown in Figure 1.7.



**Figure 1.7: Effect of thermal losses on the pressure-volume (indicator) diagram.**

### 1.2.3.2.1. Conduction Loss



**Figure 1.8: Thermal energy pathways in a Stirling engine**

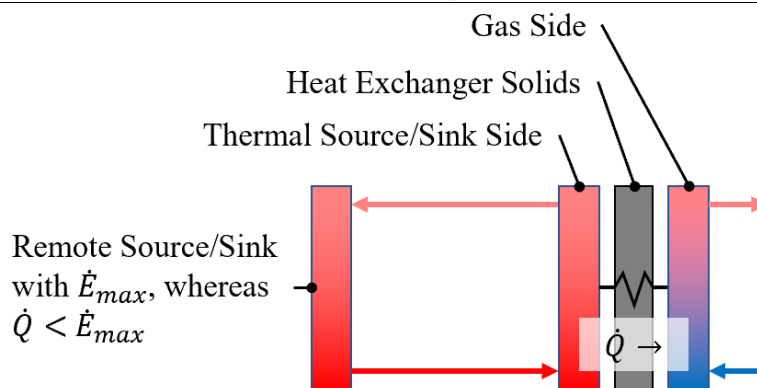
Conduction losses emerge from the conduction of energy through the solid structures of the engine, which has several primary forms as shown in Figure 1.8. All of these are facilitated by the combination of conduction, convection, and radiation. As all conduction between two bodies

produces a loss in exergy it is important that conduction is minimized as much as possible, thus heat transfers beyond the heat exchangers is a drain on the resulting power or efficiency.

- 1) Direct conduction between the hot and cold sources which simply results in that energy bypassing the heat engine entirely. [9]
- 2) Conduction from the hot or cold exchanger through the walls to the surroundings, this energy never engages in the cycle but is a loss component of engine systems.
- 3) Conduction from the interior of the engine to the exterior at ambient temperatures, which represents energy entering the cycle, but exiting partway through the cycle.
- 4) Conduction supported by motion, often referred to as the shuttle loss [22], is where a moving element, primarily a piston, carries with it a temperature gradient that is closely exposed to a wall of a different or offset temperature gradient. Usually, this is reduced by introducing a very long piston and/or modifying the gap between it and the wall. [22]
- 5) Heat transfer from the mixing of two gas streams of different temperatures. This happens when gas from two separate spaces by-passes the heat exchangers, through a seal, or the annular gap. This loss reduces the amount of heat transfer, due to a diminishing of total flow-through, at the same time it reduces the maximum temperatures of both spaces by mixing of hot with cold streams and vis-versa.

All these losses will result in a reduction of the temperature extremes and therefore pressure extremes of the engine.

### 1.2.3.2.2. Non-Ideal Heat Exchanger



**Figure 1.9: Thermal energy pathway through a heat exchanger**

Heat exchange through a heat exchanger relies on 4 regimes, seen in Figure 1.9. For the purposes of this discussion, let us refer to the exchanger as a source, as the same principles apply

to thermal sinks. First of which, is the ability of the source to provide the exchanger with heat. This value  $\dot{E}_{max}$  could be limited by external factors such as rate of fuel injection, fluid flow rate and heat capacity, solar irradiance, or any number of other factors. Exceeding or even approaching this limiting rate may result in a drop in source temperatures. Secondly, there is a loss associated with convection between the source fluid and the heat exchanger geometry. Thirdly, is the conduction of the heat through the solid structure of the heat exchanger. Finally, the convection of the heat from the solid surfaces into the gas. Increasing the resistance of any of these or lowering the potential of the source result in lower gas temperature and thus lower power.

### **1.2.3.3 Gas Friction**

Friction is experienced by the gas as it flows through the geometry of a Stirling engine. The energy required to overcome this friction must be taken from the flywheel. These losses are generated via two main sources. The primary source is flow along the main gas path, where flow encounters the heat exchangers and regenerator and any fittings along the way. The second source called pumping loss [16] is generated from the compression and expansion of gases which causes the fluid to get pumped in and out of every crevasse of the engine, this is most noticeable in tight components such as the annulus of the displacer. The effect of both of these flow losses is detected by the mechanism as a shift in the pressure experienced at the piston faces. The net effect of this disturbance always reduces the overall area of the pressure volume diagram by an amount equal to the loss in energy.

### **1.2.3.4 Mechanical Friction**

Friction, which exists in the sliding or rolling surfaces and seals of a mechanical devices, can absorb a significant portion of the work produced by the engine. This topic is discussed in depth by Senft [27] and often this loss can be as large as 5 to 15% of the output of engine [23]. These losses are dependent on several factors, including engine speed, mechanism weight, balancing, piston loading in both axial and perpendicular directions. Mechanical losses generally have a strong relationship with machine lifetime as friction or imbalances in the engine cause surface wear and fatigue, respectively. Mechanical losses are involved in the transport of energy to and from the engine, the quantity of energy that re-enters the engine to assist in completing the compression or expansion of the cycle can be directly calculated from the indicator diagram and

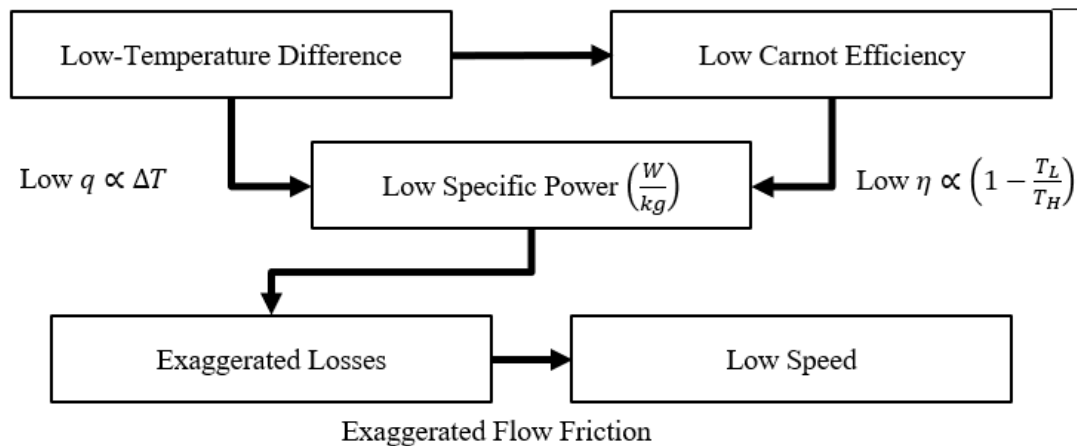
occurs anytime a piston goes against its pressure regime. The study of this phenomenon, called forced work, was pioneered by Senft [27] and is discussed in detail in the preceding theses [10], [12], [13], and [14].

### 1.2.3.5 Compressibility

At higher speeds, gases compress up against accelerating surfaces, slightly changing the experienced pressures. This results in higher pressures when compressing and a higher vacuum when expanding. The study of this particular effect was pioneered by Petrescu et al [28]. Additionally, the density variations caused by pressure losses can cause variations in the local properties of heat transfer, and flow friction.

### 1.2.4 What is important for low-temperature engine?

There is no strict definition of what constitutes a low-temperature engine. In the context of this lab it is taken as having a hot side temperature of less than 150 °C. The cold side is ambient, as opposed to engines which use cryogenic sources. The essence of a low temperature, or low temperature ratio engine can be effectively by the following flowchart.



**Figure 1.10: Thermodynamic consequences for of low temperature (ratio) engines**

The following are consistent with DTECL’s low-temperature laboratory engines.

1. Low-temperature engines run slower than high-temperature engines.
2. Low-temperature engines have lower power density than high-temperature engines.

A non-dimensional number that is often used to represent the power of an engine is the West number ( $N_{West}$ ). In theory, this number allows for the direct comparison between engines of any design with each other. This non-dimensional number, which is the ratio of actual output power to a representation of ideal power, was introduced by West [22] and makes use of the fact that power is proportional to charge pressure, the volume of gas that changes temperature and the engine frequency. The West number is defined as:

$$N_{West} = \frac{\dot{E}_{shaft}}{P_{avg} \cdot V_{swept} \cdot f_{engine}} \frac{T_H + T_L}{T_H - T_L} \quad (2)$$

where:  $\dot{E}_{shaft}$ : Power as measured at the output shaft of the engine

$P_{avg}$ : Mean, internal pressure in the engine over the cycle.

$V_{swept}$ : Swept volume of the expansion space.

$f_{engine}$ : Running frequency of the engine, at which  $\dot{E}_{shaft}$  was measured.

$T_H$ : Source temperature

$T_L$ : Sink temperature

#### 1.2.4.1 Pressure Leakage

The energy loss between a sinusoidally varying pressure region and a constant pressure region over one cycle is equal to the integration of the instantaneous energy loss over the cycle period described by:

$$E_{leak,cycle} = \oint \dot{E}_{leak} \cdot dt = \oint F(P_1 - P_2) \cdot dt \quad (3)$$

$$E_{leak,cycle} = C \frac{1}{f_{engine}}$$

where:  $E_{leak,cycle}$ : Sum of energy lost to a leak over one complete cycle

$\dot{E}_{leak}$ : Instantaneous energy lost to a leak.

$F(\dots)$ : The leak function, which prescribes the amount of energy lost to the leak (proportional to the mass which crosses it) as a function of the pressure difference. Importantly, this function is ideally independent of time.

$C$ : A constant with respect to time. This occurs because  $F(\dots)$  is independent of time, therefore the integral is equal to the integral of  $F(\dots)$  over the range of  $P$  values which are governed by angular position. Therefore, time will simply serve as a scaling factor on the magnitude on  $E_{leak,cycle}$ , as angular speed simply defines the slope of angle vs time.

This results in an energy loss which is inversely proportional to cycle time. With regards to engine speed, a slow engine will accrue more power loss per cycle than a fast cycle. With regards to engine size, given that a leak may be in some way related to engine size, for the same power output a low-temperature engine will have a larger power loss due to leakage.

### 1.2.4.2 Heat Exchange

As sink temperatures are normally ambient temperatures given its accessibility, low-temperature engines are limited by the Carnot efficiency as:  $\eta = 1 - T_L/T_H$ . Low-temperature engines are further disadvantaged by the proportional relationship between heat transfer and temperature difference. Meaning that low-temperature engines require larger heat exchanger surface areas to maintain an adequate temperature ratio, which ultimately balances friction losses against power gains from better temperatures. Low-temperature engines are often limited by heat exchange. Therefore, this requires a more detailed look into heat exchangers for such engines.

The construction of the West number [22] provides the greatest illustration of the effect of temperature on Stirling engines. Specifically, the form of the derivative of shaft power with response to rising source temperatures:

$$\frac{\partial \dot{E}_{shaft}}{\partial T_H} = 2N_{West} \cdot P_{avg} \cdot V_{swept} \cdot f_{engine} \frac{T_C}{(T_H + T_C)^2} \quad (4)$$

The increase in power is approximately dependent on the inverse of source temperature squared, such that at low-temperatures improvements in source temperature will lead to substantial increases in power, while at higher temperatures, there is little incentive to improve gas



temperatures. This will incentivize low-temperature engines to invest more into heat exchange than high temperature engines, even at the increase in flow losses or dead volume.

### 1.2.4.3 Conduction Losses

Thermal conduction is a temporally developing loss found in Stirling engines, this loss has two attributes associated with it: gas spring hysteresis (GSH) and wall conduction, which are innately tied together. GSH has been well studied by the Stirling engine community and in general, all equations for its loss rate follow some variation of the following form [29]:

$$\dot{E}_{GSH} \propto \gamma^n \cdot (\gamma - 1)^{2-n} \sqrt{\pi \cdot k \cdot f_{engine} \cdot P_{avg} \cdot T_{avg}} \left( \frac{\Delta V}{V_{avg}} \right)^2 A \quad (5)$$

where:  $\gamma$ : The ratio of specific heats =  $c_p/c_v$

$n$ : Exponent, varies depending on approximation [29].

$k$ : Thermal conductivity of gas

$T_{avg}$ : Average temperature of engine interior space

$\Delta V$ : Difference between maximum engine volume to minimum engine volume.

$A$ : Surface area over which loss takes place.

If the properties of angular speed, surface area and temperature are isolated the following equation is found.

$$\dot{E}_{GSH} \propto A \sqrt{\omega \cdot T_{avg}} \quad (6)$$

As a note, area is approximately proportional to volume to the 2/3<sup>rd</sup> power. According to the scaling rules of Organ [30] the equivalent speed of a larger engine scales with volume to the negative 1/3<sup>rd</sup> power. Combining these into a single equation gives:

$$\dot{E}_{GSH} \propto V^{\frac{2}{3}} \sqrt{\frac{1}{V^{\frac{1}{3}}} \cdot T_{avg}} \propto \sqrt{V \cdot T_{avg}} \quad (7)$$

According to the West number, power is directly proportional to size, speed and exponentially related to temperature. Thus, the GSH loss may be less of a burden at high speed and temperature ratios. And larger engines will experience a lower GSH loss due to having a diminishing surface area to volume ratio. A large low-temperature engine may have lower GSH than a small high-temperature engine, but the resulting loss is likely to be a larger proportion of power in the low-temperature Stirling engine due to the strong dependence of power on temperature difference.

The second attribute of thermal conduction losses is conduction through the solid containment of the Stirling engine volume. This energy exchange pathway, governed by Fourier's law, is constant in time but linearly proportional to the difference in temperature extremes.

$$\dot{E}_{Conduction} \propto \frac{A}{L} \cdot \Delta T \cdot dt$$

Where:

$$A \propto V^{\frac{2}{3}}, \quad L \propto V^{\frac{1}{3}}, \quad dt \propto V^{-\frac{1}{3}} [30] \quad (8)$$

Thus:

$$\frac{V^{\frac{2}{3}} \Delta T}{V^{\frac{1}{3}} V^{\frac{1}{3}}} \propto \Delta T$$

With regards to engine size this loss will be constant at equivalent speeds, as defined by Organ [30]. The larger engine runs slower but will generate power proportional to  $V^{\frac{2}{3}}$  with the same conduction loss as the smaller engine. With regards to engine speed, a slower engine has longer time to develop this loss, thus allowing it to consume a greater proportion of power. Because power output increases faster with respect to temperature difference than this loss, a high proportion of power will be taken up by conduction losses in low-temperature engines.

### 1.2.4.4 Viscous Friction

Viscous friction loss, or pressure loss, is a function of flow geometry, gas viscosity and gas velocities. With regards to speed, the viscous friction loss is, in turbulent cases, dependent on velocity squared according to the Darcy-Weisbeck equation [31] for pipe-flow. The viscosity of the gas, which generally increases with temperature, will increase the pressure losses. This will be seen to a much less extent than velocity as its effect is only seen in the friction factor ( $f$ ) calculation via (in the pipe flow case) the Colebrook-White equation [32]. The dependence of friction on viscosity varies from linear in the laminar regime to a negligible effect in the fully turbulent regime, where turbulent dissipation overshadows molecular dissipation.

With regards to geometry, an engine that is scaled to be 2 times larger in all dimensions will have flow paths that are 2 times longer with cross-sections that are 4 times higher, hydraulic diameters are 2 times larger and volumetric flow rates that are 8 times larger. This results in 2 times larger flow velocities. Larger scales also effect the surface finish, in theory an equivalent surface finish could be obtained, which would allow a large engine to approach that of a smooth pipe. Inserting these modified dimensions into the Darcy-Weisbeck equation [31]:

$$\Delta P = \frac{f \cdot \rho \cdot L}{2D_h} \left( \frac{\dot{V}}{A} \right)^2 = \frac{f \cdot \rho}{2} \frac{(2L)}{(2D_h)} \left( \frac{8\dot{V}}{4A} \right)^2 = f \cdot 4 \left( \frac{\Delta P}{f} \right)_{1x} \quad (9)$$

where:  $\Delta P$ : Pressure drop over flow path

$f$ : Darcy friction factor: defined as the Colebrook-White equation [32]:

$$\frac{1}{\sqrt{f}} = -2 \log \left( \frac{\epsilon}{3.7D_h} + \frac{2.51}{N_{Re}\sqrt{f}} \right)$$

$$N_{Re} = \frac{\rho \cdot D_h \cdot U}{\mu} = \frac{\rho}{\mu} (2D_h)(2U) = 4N_{Re,1x}$$

$$\frac{\epsilon}{D_h} = \frac{\epsilon}{2D_h}, \text{ thus roughness is halved.}$$

$\rho$ : Fluid density

$L$ : Streamwise length of flow path

$D_h$ : Hydraulic diameter

$\dot{V}$ : Volumetric flow rate

This results in a pressure drop that is roughly the same in the laminar case (as the 4's cancels out and roughness is neglected), and in the fully turbulent case is between 2 to 4 times larger for the large engine, since roughness is – at best – linearly related to the friction factor. Based on the West number, energy produced per cycle increases linearly with volume, so a large engine will have 8 times as much power, but between 8 to 32 times as much flow losses when running at the same speeds. Therefore, low-temperature engines must run at low-speeds to minimize these increased flow losses, caused by their larger sizes. This is further enforced by the more constrictive heat exchangers found in such devices. This is also confirmed by Organ [30], who's scaling laws declare that the speed of the engine will scale with volume to the 1/3<sup>rd</sup>, or in our case here should be halved, corresponding exactly to a 1:1 increase in cycle energy with flow losses.

An alternative to running at lower speeds is to increase the cross-sectional area of the heat exchangers vs its length. This, in theory offers nearly the same temperature change (if the flow is laminar, and therefore having a constant Nusselt number) but can run into problems with preferential flow the lower the pressure drop gets, additionally it may be geometrically challenging to design a high cross-section heat exchanger that does not introduce significant amounts of dead volume from distribution geometry.

#### 1.2.4.5 Mechanical Losses

Kinematic mechanisms, driven by a rotating drive, which are discussed in detail in CHAPTER 2 result in forces on the drive shaft that follow the following structure:

$$F = A(\theta) \cdot \alpha + B(\theta) \cdot \omega^2 + G(\theta) + E(\theta) \cdot F_p \quad (10)$$

where:  $A(\theta)$ : Coefficient on angular acceleration ( $\alpha$ ), represents the system inertia as a function of angular position ( $\theta$ )

$B(\theta)$ : Coefficient on angular velocity squared ( $\omega^2$ ), represents internal inertia as a function of angular position ( $\theta$ )

$G(\theta)$ : Gravitation force, as a function of angular position ( $\theta$ )

$E(\theta)$ : Coefficient on piston force ( $F_p$ ), which represents how the piston force, when translated through the mechanism produces a load on the drive shaft.

With most engines the value of angular acceleration ( $\alpha$ ) is small, the value of angular velocity squared ( $\omega^2$ ) increases internal and external loads as a function of engine speed squared. With a larger and slower engine, loads from the piston ( $E(\theta).F_p$ ) and static loads from gravity will dominate. Friction is often taken as linearly proportional to the normal force [33] (as in Coulomb friction), or rolling resistance such as in dry ball bearings, lubrication depends on both normal force and speeds, and often the friction coefficient decreases with speed as lubrication films even out. In addition, at high speeds lower viscosity lubricants are required [34] and many high-speed Stirling engines utilize bearings that at high enough speeds utilize none contact gas bearings [21] with very little friction or wear.

A high-speed engine will have larger internal loads due to the internal inertia term, which may result in larger friction losses, however such an engine may reduce these loads with lower friction components. Such an engine will require a lighter flywheel and suffer from lower static loadings which would reduce friction further through leaner component design. Low-speed engines require larger flywheels but suffer less from internal inertia forces, however they have to be designed for larger static loads. Low-temperature engines could be the clear loser, with heavier components, thicker lubricants and a power density that is exponentially less than their high-temperature cousins.

### **1.2.5 Summary of Stirling Engines**

In summary, while the ideal Stirling cycle provides the highest power per cycle, it is also impossible to accomplish. Therefore, designers must be able to manage the losses that appear as part of the practical modifications to the ideal cycle. The accurate modelling of these phenomena is particularly important for low-temperature engines, which as shown in section 1.2.4 suffer to a greater proportion in every loss category identified with the exception of inertia losses. Often the first step in loss identification is numerical modelling of the engine, which is the topic of the following section.

## **1.3 Modeling Techniques**

---

There are many modeling tools developed specifically for Stirling engines or related machines. The complexity, and variety of tasks that Stirling engines have been applied to has forced the

creation of many specific models. Many have significant simplifications for the sake of computational efficiency which limit the scope of applicability of those models. Several advanced models are commercially available [35]–[37], which are outlined below and many models are described in the literature. Few are publicly available to the author’s knowledge, as a result, this list is not all-encompassing, and many models and their implementation are only partially described in the literature.

Stirling engine models are classified by their order, a classification scheme introduced by Martini [23] in their “Stirling Engine Design Manual” of 1978. It is important to note that most models have been designed for a high-temperature context and that although many models may work under low-temperature conditions, the selection of correlations and factors common in lower-order models becomes a challenge [10].

### **1.3.1 1<sup>st</sup> Order Models**

The most basic requirement for a model to be 1<sup>st</sup> order is that it produces closed-form solutions. What is meant by this is that the power can be determined explicitly through an equation, with often little more than the source and sink temperatures. Generally, they are recommended only for those who wish to begin investigating the possibility of a Stirling engine [23].

#### **1.3.1.1 Efficiency Prediction**

The most basic of 1<sup>st</sup> Order models described by Martini [23] is described by multiplying the Carnot efficiency by a series of efficiencies that represent different major losses:

$$\eta_{eff} = \left(1 - \frac{T_L}{T_H}\right) \cdot \eta_C \cdot \eta_H \cdot \eta_M \cdot \eta_A \quad (11)$$

where:  $T_H, T_L$ : Hot (source) and Cold (sink) gas temperatures.

$\eta_C$ : Carnot Efficiency, ratio of indicated efficiency to Carnot Efficiency, includes all gas and thermal internal losses. Often = 0.65 → 0.75, but can be as high as 0.80.

$\eta_H$ : Heater Efficiency, a measure of how efficiency the heating element delivers heat to the engine. Often related to a burner, but can be related to a heat delivery system. Often = 0.85  $\rightarrow$  0.95

$\eta_M$ : Mechanical Efficiency, related to the reduction of energy from the indicated power to the shaft power. Often = 0.85  $\rightarrow$  0.95

$\eta_A$ : Represents the power loss due to the driving of auxiliary systems, such as pumps, valves and instrumentation. Often = 0.95

This is multiplied by the input thermal power to form the expected power. Though useful, if estimating the power output in a well understood design space, this sort of model offers little to the designer for parametrization.

### 1.3.1.2 West Number

Introduced in the previous section, the West number [22] ( $N_{West}$ ) is a non-dimensional number that represents the performance of a Stirling engine as is defined as:

$$\dot{E}_{shaft} = N_{West} \cdot P_{avg} \cdot V_{swept} \cdot f_{engine} \frac{T_H - T_L}{T_H + T_L} \quad (12)$$

where:  $\dot{E}_{shaft}$ : Power as measured at output shaft of engine, after all engine specific losses.

$P_{avg}$ : Average internal pressure of engine, averaged over both space and time.

$V_{swept}$ : Volume of gas pushed through (in one direction) the exchangers.

$f_{engine}$ : Engine (cycle per second) frequency at which  $\dot{E}_{shaft}$  is measured.

$T_H$ : Temperature of source, or average temperature in expansion space.

$T_L$ : Temperature of sink, or average temperature of compression space.

The normal range of West numbers is around 0.25 [22]; a number that was obtained from a variety of high temperature high-performance engines. The power can be predicted by rearrangement. The West number establishes a baseline given that an engine is well designed it

should provide output power proportional to the macroscopic properties. However, it offers no key information on what exactly an engine, well designed for its conditions, should look like.

### 1.3.1.3 Schmidt Model

The Schmidt model is a semi-ideal analytical model, created by Gustav Schmidt in 1871 [25], that takes advantage of several assumptions about Stirling Engines. The Schmidt model offers greater insight into parametrization, of importance is the influence of piston motions on the indicated power. Due to its relative information density, it is the basis on which many 2<sup>nd</sup> and 3<sup>rd</sup> order models are constructed. As such it will be outlined in detail in the following pages, as outlined by Walker [38]. The isothermal simplification was classically used in this model that includes the following assumptions:

1. Perfect Regenerator:
  - a. While within the regenerator the gas and regenerator material are the same temperature.
  - b. The Regenerator temperature is constant in time.
  - c. The Regenerator temperature follows a linear trend from the hot to the cold side.
2. The pressure is the same throughout the engine.
3. The working gas is ideal and therefore follows the equation of state:  $P.V = R_{spec} \cdot T$
4. There is no leakage internally or with the surroundings.
5. The piston motion and therefore the volume variations are perfectly sinusoidal in time.
6. Heat exchangers are uniform temperatures both spatially and temporally.
7. The cylinder wall and piston temperatures are constant.
8. There is perfect mixing in the expansion and compression volumes.
9. The temperature of the dead volume is constant.
10. The speed of the machine is constant.
11. Flow conditions are steady state.
12. The expansion and compression spaces are assumed to be isothermal.

Starting from assumption 4, the constant total mass is equal to the mass of all the gas spaces within the engine for every point in the cycle. The total mass ( $M$ ) is the sum of masses ( $m$ ) in individual components as:



$$M = m_{compression} + m_{cooler} + m_{regenerator} + m_{heater} + m_{expansion} \quad (13)$$

$$M = m_c + m_k + m_r + m_h + m_e$$

where:  $M$ : Total mass

$m$ : Mass of the contents of individual components

By substituting mass with its equivalents within the ideal gas law and applying assumptions 2 and 3:

$$M \cdot R_{spec} = P \cdot \left( \frac{V_c + V_k}{T_L} + \frac{V_r}{T_r} + \frac{V_h + V_e}{T_H} \right) \quad (14)$$

where:  $R_{spec}$ : Specific gas constant of the gas inside the engine.

$P$ : Pressure inside of engine (considered uniform, cycle position dependent)

$V$ : Instantaneous volume of individual component

$T$ : Average temperature of whole gas contents of individual component

By assumption 1, a linear temperature profile must exist in the regenerator:

$$T(x) = \frac{(T_H - T_L) \cdot x}{L_r} + T_L \quad (15)$$

where:  $L_r$ : Length, in flow direction, of regenerator.

$x$ : Distance, in flow direction, from cold heat exchanger.

To calculate the mass of the regenerator, the temperature profile can be substituted in to give:

$$m_r = \int_0^{V_r} \rho(x) \cdot dV_r = \frac{V_r \cdot P}{R_{spec}} \int_0^{L_r} \frac{1}{\frac{(T_H - T_L) \cdot x}{L_r} + T_L} dx \quad (16)$$

where:  $\rho$ : Local gas density, is a function of  $P$ ,  $R_{spec}$  and  $T(x)$

This reduces to:

$$m_r = \frac{V_r \cdot P \cdot \ln\left(\frac{T_H}{T_L}\right)}{R_{spec} \cdot (T_H - T_L)} \quad (17)$$

which gives the temperature for the regenerator as:

$$T_r = \frac{(T_H - T_L)}{\ln\left(\frac{T_H}{T_L}\right)} \quad (18)$$

This can also give the pressure of the engine volume as:

$$P = \frac{M \cdot R_{spec}}{\frac{V_c + V_k}{T_L} + \frac{V_r \cdot \ln\left(\frac{T_H}{T_L}\right)}{T_H - T_L} + \frac{V_h + V_e}{T_H}} \quad (19)$$

The work done by a single cycle ( $W_{cycle}$ ) is simply:

$$W_{cycle} = \oint P \cdot \left( \frac{dV_c}{d\theta} + \frac{dV_e}{d\theta} \right) \cdot d\theta \quad (20)$$

where:  $\theta$ : Cycle angular position, from which  $V_c$  and  $V_e$  are both derived.

The volume variations as a function of  $\theta$  are different for each of the 3 classical engine configurations but given a sinusoidal pattern the integral above may be solved simply by hand. The final work may then be multiplied by the cycles per second to get the work at any running speed. Using numerical integration, the above equations can be solved for non-sinusoidal motions as well which allows the model to generally applicable. However, based on the discussions in section 1.2.3, the isothermal idealization present in the Schmitt model make it a poor predictor of actual engine performance.

### **1.3.2 2<sup>nd</sup> Order Models**

Second-Order models are simulations that are based upon or are closed-form solutions. According to Martini [23], 2<sup>nd</sup> Order models start with a 1<sup>st</sup> Order model, and then degrade the resulting power output by losses that are decoupled from the engine cycle. 2<sup>nd</sup> Order models often apply empirical correlations for these losses which can be subjected to calibration. However, they generally do not considered loss dependencies and often require expert knowledge to apply the correct correlations [23]. This makes accuracy susceptible to a case-by-case basis. Like 1<sup>st</sup> order models, speed is an input to these models, making them incapable of determining speed as an output or modelling dynamic speed scenarios. Due to their uncoupled nature, 2<sup>nd</sup> Order models are only capable of converging towards a steady state solution and the user must consult the produced engine curves when determining the engine speed for a given loaded condition. The following details the main 2<sup>nd</sup> order models discussed in the literature.

#### **1.3.2.1 Urieli & Berchowitz**

The SIMPLE model – so called as it is a simplification of the actual non-steady flow heat exchange – was derived by Urieli and Berchowitz in their 1984 publication [16]. The scheme was originally written in Fortran, but updated to MATLAB [39] and hosted online by Urieli [11]. This software used the ideal adiabatic model assumption and integrates into the solution the non-ideal performance of the heat exchanger sections. The model assumes quasi-steady assumption of friction and heat exchange through the heat exchangers. This model is decoupled from losses not associated with non-ideal heat exchanger or regenerator performance. The base model by default subtracts regenerator wall leakage and pumping losses from the power output. The model was expanded by Speer [10] to include a host of recognized Stirling engine losses as decoupled subtractors.

#### **1.3.2.2 Babaelahi & Sayyaadi**

The SIMPLE model [16] was expanded in 2014 by Babaelahi and Sayyaadi [40]. The authors included heat absorbed and rejected by the displacer piston and mass leakage between the engine and buffer spaces in addition to using different equation forms for representing pressure. Additional decoupled losses included finite speed losses, mechanical friction, and the longitudinal conduction along the regenerator wall. The study predicted the power and efficiency of the high-

temperature 3kW GPU-3 Stirling engine by General Motors with at least 5 times as much accuracy compared to the original SIMPLE model and better than the state of the art of equivalent models at the time.

Babaelahi and Sayyaadi [41] developed a new model in 2015, called the polytropic analysis of Stirling engine with various losses or PSVL. The model takes into consideration that Stirling engines are a continuously varying polytropic process. The author's devised a method for determining polytropic indexes for each working space as a function of crank angle. The method also introduces 3 categories of loss considerations. The first of which was direct partial differential equation representation for polytropic heat transfer, gas leakage and shuttle effect. The second, including non-ideal heat transfer, pressure drops which applied their corrections to the temperatures between each iteration. The third category involves losses that don't affect the temperature distribution and were therefore subtracted from the power afterward. The polytropic index is derived from properties and used in heat conduction for each increment of the cycle, the model is iterated until the index forms a continuous loop. The model proved to be more accurate than the author's previous models.

### **1.3.2.3 Commercial Codes**

One of two commercially available 2<sup>nd</sup> order models: SNAP pro by Altman [37] is a model implemented within the MS Excel environment for straight forward user modification. By default, an engine parameter set can be optimized via a genetic algorithm. SNAP Pro is based on the work of various researchers including Martini [23], Berchowitz's linear analysis [42] for free-piston analysis and includes losses due to forced work from Senft [27].

The second commercially available software, PROSA 2 by Thomas [36] is implemented in a self-contained program with a detailed parametrization scheme with simplex optimization. The author also developed a 3<sup>rd</sup> order model that allows for speed variations and non-sinusoidal piston variations, but further documentation for either model could not be found.

### **1.3.3 3<sup>rd</sup> Order Models**

According to Martini [23], third-order models divide the working volume into distinct volumes and the basic equations are solved using numerical methods. This, as opposed to the 2<sup>nd</sup> order

assumption, allows interdependent processes to affect one another. The field was pioneered by Finkelstein [43], who developed the first nodal analysis, whose work is updated by the models discussed below. These models are regarded as more accurate than 2<sup>nd</sup> order models in general, in particular a 3<sup>rd</sup> order model will have greater accuracy when exploring a new design space, but may be comparable to a 2<sup>nd</sup> order model in a well-defined space [23]. Due to their sensitivity to such features and higher general accuracy, they are often used in the later stages of engine design and as part of optimization studies.

In addition to greater generality, 3<sup>rd</sup> Order models offer outputs that cannot be measured experimentally or by lower order models. Temperature is notoriously difficult to measure reliably due to response times, whereas the 3<sup>rd</sup> Order model defines the instantaneous values. Depending on the solution scheme, dynamic solutions may also be available which can detect phenomena such as stalling after a change in load and even allow initial design of engine control systems before a physical prototype is even made.

### **1.3.3.1 SAGE**

Possibly the most well known commercially available 3<sup>rd</sup> Order model for Stirling engines is SAGE, developed by Gedeon [35]. This modular model, which is well documented, creates discretized networks of nodes, that are solved using the harmonic assumption. The harmonic assumption, which represents cyclic values as a Fourier series, is largely accurate for a wide variety of Stirling engine designs and very computationally efficient because parameters are defined as a series of phases and magnitudes as opposed to individual timestamped values. SAGE has been used for both low-temperature [44] and high-temperature Stirling engine designs [45]. A detailed discussion of this model can be found in Gedeon [35], as well as Hoegel [45]. SAGE features high performance, but through communication with users within the University of Alberta, it is noted to have a high learning curve and has difficulty converging in unique circumstances. In addition, SAGE is unable to support dynamic scenarios [35] due to the nature of the harmonic solver. Regardless, this model was studied extensively during the development of the current model presented later in this thesis.

### **1.3.3.2 Nlog Thermodynamic Analysis Code**

The Nlog thermodynamic analysis code is a 1D control volume code. The model was created by researchers investigating neural-network-based performance prediction [46] and was even modified to support dynamic scenarios [47]. Given a set of detailed parameters, Nlog solves the equations of mass, momentum, and energy for each volume in the engine. The model ignores solid-conduction effects and instead has two wall types: An isothermal wall type, i.e., temperature remains constant and a regenerative wall type, i.e., net heat transfer to its control volume is zero over the cycle. It is not well documented to the authors knowledge and may be difficult to configure to engine types.

### **1.3.3.3 Anderson**

Anderson et al [48], from the Technical University of Denmark, developed a modular model for research usage using the MusSim framework, which the author also developed. The author studied the effect of the regenerator's thermal response within the cycle and found that handling it dynamically introduced a relatively substantial change in calculated engine power. Anderson's model is constructed out of modular groups much like SAGE's implementation but in the incremental form, making it capable of deriving dynamic scenarios given that the mechanism dynamics are provided. However, this model is not publicly available to the authors knowledge and little information could be found on its implementation.

## **1.3.4 Higher-Order Models**

Models that consider flow to be a one-dimensional phenomenon often have increased error when the engine is not symmetric about its axis. Discretizing gas space in a 2<sup>nd</sup> dimension is generally what changes a 3<sup>rd</sup> Order model into a higher-order model. Substantially more computationally expensive, this form of engine modeling is generally very accurate, but the approach is currently too slow for any but final engine design and optimization. Two groups have made advances in this level of modeling, researchers at the University of Northumbria [49] and researchers at NASA [50].

## 1.4 Chapter Conclusions

---

There is a great opportunity for alternative energy generation through recovery of low-temperature energy source. These include: waste heat which accounts for up to 70% of all energy used in Canadian industries [1] and geothermal energy. The energy recoverable from these processes, which are limited by thermodynamics to between 2-4% in most cases still amounts to significant energies. Geothermal energy has potential [2], but is completely untapped by existing technologies in Canada. It is the goal of DTECL to investigate Stirling engines as potential contributor to utilizing these energy sources.

Work is being conducted in this area, but a major weakness of the investigation is the models that are being used, which are inaccurate [10], [13], [14], particularly at low-temperatures and are difficult or inextensible in usage. A significant weakness in all the models presented is either being too specific or opaque, as in 1<sup>st</sup> and 2<sup>nd</sup> Order models. Or being inaccessible to new users such as with many 3<sup>rd</sup> Order models.

## 1.5 Thesis Goals

---

The goals of this thesis are as follows.

To develop a numerical model to

1. Simulate low-temperature engines. This entails a model that focuses on heat transfer, flow friction, leakage and allows detailed definition of the mechanism with its internal friction. To improve computational efficiency, loss modes such as radiation, fluid inertia and acoustics will be ignored.
2. Produce results that are validated against an experimental laboratory engine, analytical experiments and results from SAGE, an accessible and well validated numerical code.

In addition, the software will

1. Allow a user to construct a full working model of arbitrary geometry using a graphical user interface.
2. Present the model geometry in an intuitive and visual manner.
3. Provide the user with tools to record data of interest and interpret results.

The goal of the remainder of this thesis is to devise an implementation of the above goals in MATLAB [39] and test its ability to accurately model Stirling engines within the low-temperature context.

## **1.6 Thesis Outline**

---

CHAPTER 2 documents the development process for the structure of the software and the problems that need to be solved using the model.

CHAPTER 3 documents the core mathematical and numerical processes.

CHAPTER 4 documents the implementation of the discretization and numerical solving.

CHAPTER 5 documents the algorithms used to enhance the convergence of the model.

CHAPTER 6 documents important information on how to use the software module, as well as the importance of model outputs.

CHAPTER 7 documents the tests used to verify the performance of the model.

CHAPTER 8 concludes the thesis with an assessment of the project goals.

Further information for each of these chapters as well as the entire project code can be found in the Appendices.



# CHAPTER 2. SYSTEM DEVELOPMENT AND ARCHITECTURE

## 2.1 Development

---

The following sections outline the type of problem that this model targets. This begins with defining the processes that go on within an engine and then abstracting those phenomenon and features into components that can be added by the user. Based on the discussion in section 1.2, a low-temperature engine's power is limited to a great extent by its losses. These losses stem from poor heat conduction within heat exchangers, problems with heat control in other volumes, pressure leakage, flow friction and mechanical friction. These losses are all interconnected, poorer heat transfer into the engine, also reduces the loss of that energy to the wall. Internal to external leaks or even leaks between different areas of the engine effect heat transfer and flow friction through changing gas flow rates. Flow friction effects the force on the piston and thus the mechanism losses.

### 2.1.1 The Gas Medium

The best way to model these losses is to model the physical system that produces these losses. The extreme of this is to build physical prototypes. Outside of that the most conservative approach is to use CFD or 4<sup>th</sup> Order models which attempt to model all the physics with as little assumptions as possible. A further compromise then exists by assuming that the flow is 1-dimensional, which reduces the number of calculations but increases the complexity of each of them. This complexity thankfully is very well studied as it reduces the Stirling engine into a dynamic pipe network problem. Having modelled the gas inside the engine as a pipe network, albeit a cyclically changing one, the viscous friction, leaks and the gas side of heat conduction can be relatively easily solved using empirical correlations to match the encountered flow geometry and conditions.

### 2.1.2 Uniform Pressure

Stirling engines rely on the compressibility of their contents to function; therefore, the gas must be modelled as compressible. This leads to a problem though, as the low-temperature Stirling engines generally don't operate at speeds that would require properly modelled acoustics, a

phenomenon which requires that the timestep is restricted by the speed of sound and not the speed of the gas. Practitioners have come up with ways of modelling the acoustic component implicitly [51], which avoids this restriction, but the assumption is made here is that acoustics has a negligible impact on the operation of the low-temperature Stirling engine. One way of modelling this is to assume, as in the Schmidt model, that the pressure is uniform throughout the engine. Friction losses can still be approximated as required by using the flow rates, but without compressibility, the gas speed – not the speed of sound – contributes to the timestep. This is thoroughly discussed in CHAPTER 3.

### **2.1.3 The Solid Medium**

The next problem exists when solving the solid medium, solid conduction problems are simple compared to their gas counterparts, the equations are linear, and the conduction does not appreciably change with temperature within reasonable bounds. Therefore, it is easy to model thermal conduction in the solid body of the engine as a 2D or 3D network, provided that an engine is reducible to such a form. This will allow the system to naturally arrive at conduction losses without resorting to empirical formulas or surface idealizations.

### **2.1.4 The Mechanical System**

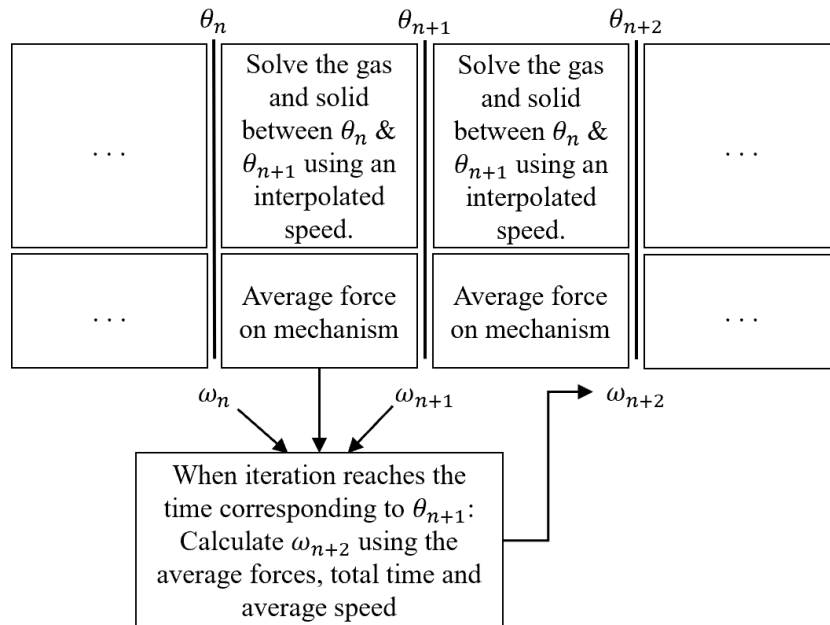
The last component of development is the mechanical system. At this stage, for simplicity, only kinematic mechanisms will be studied. Thus, the volume of the piston cylinder spaces is solely dependent on the angular position. The gas spaces, in return, provide a force on the piston itself; composed of many individual pressures and shears. This force is translated through the mechanism, which uses the equation form introduced in section 1.2.4.5 to output a set of normal forces and torques. The normal forces are used to generate losses on the drive shaft. The torque, minus losses, feeds into the flywheel, generating an acceleration of the system.

The inclusion of a mechanism was important, as it was desired that the software could model dynamic scenarios. This would differentiate it from previous works such as SAGE [35] most of which are optimized for solving steady-state solutions. This restriction is often justified due to the fact that the majority of operating conditions Stirling engines will be connected to a reliable supply of thermal energy and therefore run at steady-state. However, permitting dynamic speed allows

MSPM to, in the future, support the design of control schemes for Stirling engine power systems. It allows the designer to properly size flywheels, which are required to be large for slowly rotating devices due to the inclusion of the  $\omega^2$  term in rotational kinetic energy. Additionally, it can prediction of conditions required for successful start-up of Stirling engines and predict stalling behavior from changes in load, which are hypothesized to occur due to the thermal distribution changes for different operating speeds.

### 2.1.5 Gas and Mechanism Relationship

The gas and mechanical systems are intimately linked, the question is then how to solve these two systems together. In normal Stirling engine operation, the mechanical system does not change speed very much over a cycle. The solution then is to break up the cycle into periods of time where the speeds are precomputed, this would allow the mechanical system to change speed and is due to the fact that the actual acceleration lags the force generated by the gas. A depiction of this can be seen in Figure 2.1. It is important to note that the lag can be reduced to be as small as desired by increasing the number of increments per cycle.



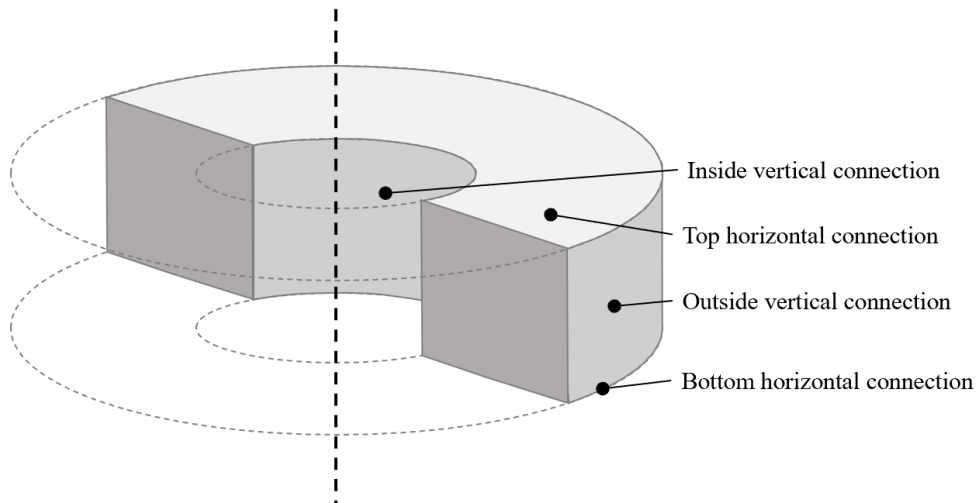
**Figure 2.1: Gas/solid system and mechanical system interaction loop**

### 2.1.6 Axial Symmetry

The design of Stirling engines revolves around the shape of its pistons, which make up the bulk of its internal space. Extra space in a Stirling engine not swept by the pistons, called dead volume, reduces the pressure swing and therefore the power of the engine. As a result, areas that are not pistons are compact and conform to the engine. Since pistons are round, based on manufacturing techniques used in the manufacture of both the bore and the piston head, Stirling engines are very commonly symmetric about a single axis or at the very least symmetric about multiple axis connected by round pipes. Round pipes due to the minimization of perimeter per cross-section as well as pressure resilience are also optimal. In addition, asymmetry in a Stirling engine leads to preferential flow, which makes them less efficient. All these factors enforce that a well-designed Stirling engine ought to be axially symmetric. Additionally, this restricts the solid modelling to only 2 dimensions which improves the readability of the model construction and reduces the computational complexity.

### 2.1.7 First Elements

The preceding discussion leads to a virtual engine that is composed of groups of cylindrical or annular elements. These elements are called bodies. A graphical depiction of this can be found in Figure 2.2.



**Figure 2.2: A body and member connections. In cylindrical elements, the inside vertical connection may be reduced to a single line at the axis.**

These bodies have rectangular cross-sections when cut by a plane coincident with the center axis. This restriction, while it limits the variation in geometry that can be modelled, drastically reduces the complexity and handled cases from a programming perspective. A rectangle is characterized by having 4 sides at right angles to each other, thus sides can only be cylindrical shells or planes perpendicular to the center axis. As blocks of material in a Stirling engine are packed together, it makes sense that these surfaces – called connections – rather than the blocks themselves contain the dimension information, represented either a radius or distance along the axis. This trait prevents blocks from overlapping, except for the case where the interface moves enough to give an affected body a negative volume.

### **2.1.8 Further Abstraction**

Often Stirling engines are not wholly axially symmetric. Gamma engines are classic examples as characterized by power pistons that are not inline with their displacer pistons. This then requires that to model these aspects, at least in a visually interpretable manner, that the ability to have multiple sets of these elements; lying upon potentially different axis is required. These sets will be called groups and allow the user to visually arrange the engine. With elements now separated in this manner, the only way to connect them would be to artificially produce a connection between a body in one group, to a body in another group, this component is called a bridge. Several additional components such as the leak, custom minor loss coefficient and non-connection also connect, modify, or disconnect two remote bodies in their own way. All these components will be discussed in greater detail further into this chapter.

Some structures such as those found in regenerators or heat exchangers are too small or complex to be modelled at the body level. Instead, it leads to an additional component called a matrix. This matrix component will be added to a body, modifying it by introducing representations of the fine geometry. These modifications will be such that interactions during simulation approximate the macroscopic behaviour of the structure.

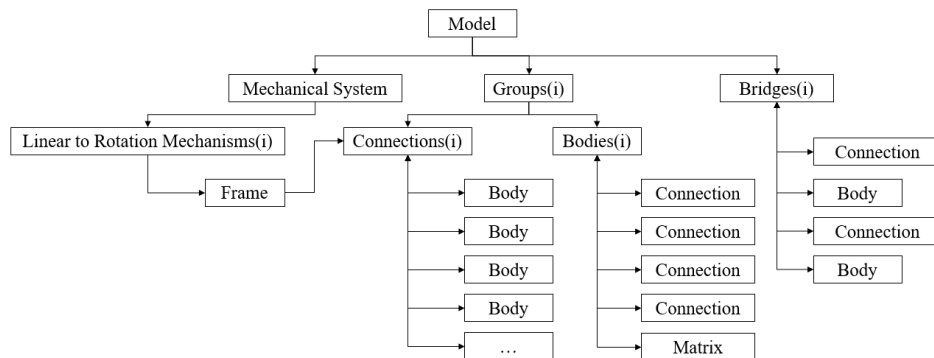
### **2.1.9 The Name**

All models need a name. This software and the solving system that is contained within is intended to solve the thermodynamics and losses of Stirling engines. As will be seen later in the

following sub-sections and CHAPTER 3 the model is a modular structure that is intended to solve single phase problems that have cyclically varying motions. Thus, from now on, the model is called MSPM or Modular Single-Phase Model, as it does not have to be restricted to just Stirling engines. The term modular refers to the fact that the network that is solved is formed from blocks representing components such as bodies of solid material, flow channels, heat exchangers or open volumes which change shape. These modules can be arranged into arbitrary arrangements, which the software converts into a network, and solves for pressure, temperature and mass.

### 2.1.10 Final Structure

Figure 2.3 represents the final hierarchical map of the different components of a definition file. These components – bodies, connections, groups, and bridges – contain the bulk geometrical information as well as the information and functionality needed to discretize themselves. These linking lines do not represent the programming concept of inheritance, but rather which objects contain references to other objects.



**Figure 2.3: A flow chart of the system architecture**

The core element, a class called model – so called as it is a representation of the physical model of an engine – includes a single mechanical system which links to multiple linear to rotational mechanisms (such as slider crank mechanisms). Each linear to rotational mechanism provides a motion profile, here called a frame. The model can contain multiple groups, which are collections of bodies which lie around the same rotational axis. Each of these bodies contains a reference to 4 connections. Those connections that are oriented perpendicular to the normal axis can also reference a frame and in turn are able to move in accordance with that frame using their current

position as the datum. In gas bodies a representation of an internal structure, here called a matrix, can be added.

The next sub-sections, for the purposes of terminology, will discuss the numerical elements followed by a section on the macroscopic features.

## 2.2 Finite Elements

---

Finite elements represent small sections of the engine, within which the properties are assumed to be constant. The smaller these elements are, the more the modelled system reflects the continuum of the real world. Following subsections will define the finite elements that play important roles in the creation of the mesh.

### 2.2.1 Nodes

A node represents a small element of matter. The shape of gas nodes may evolve in response to the motion of engine components, solid nodes - though incompressible - may translate through space, for example as a part of piston assemblies. There are 4 variants of nodes as outlined in the following sub-sections:

#### 2.2.1.1 Common Properties

**Volume ( $V$ )** – The total volume of the node:  $V = \pi(r_o^2 - r_i^2)/(y_o - y_i)$

**Internal Energy ( $u$ )** – The specific internal energy of the node, this property is initially determined as a function of the initial temperature, but later defines the temperature.

**Temperature ( $T$ )** – The temperature of the node

#### 2.2.1.2 Gas

**Mass ( $m$ )** – The total amount of mass in the node, initial mass is derived from the set pressure, temperature, and volume:  $m = VP/R_{spec}T$

**Pressure ( $P$ )** – The pressure of the gas node

**Turbulence ( $\tau$ )** – If the node is of constant volume, this value is the weighting factor between laminar and fully turbulent. For variable volume nodes, it is a representation of the specific turbulent kinetic energy, which is the amount of oscillatory kinetic energy for each unit mass.

**Hydraulic Diameter ( $d_h$ )** – Geometry and orientation dependent

Radial Flow	Annular Flow	Cylindrical Flow
$2(y_{max} - y_{min})$	$2(r_{max} - r_{min})$	$2r_{max}$

**Nusselt Number Function ( $F_{N_{Nu}}(N_{Re})$  or  $F_{N_{Nu}}(N_{Re}, N_{Pr})$ )** – Geometry and orientation dependent as:

Radial Flow	Annular Flow	Cylindrical Flow
Laminar		
3.66	Inner: $4.4438 \cdot \left(\frac{r_o}{r_i}\right)^{-0.43}$ Outer: $4.6961 \cdot \left(\frac{r_o}{r_i}\right)^{0.0548}$	3.66
Turbulent		
$0.035 \cdot (N_{Re})^{0.75} (N_{Pr})^{0.33}$		

The properties of the specific gas constant ( $R_{spec}$ ), inverse heat capacity ( $F_{u2T}(T)$ ) correlation, thermal conduction  $F_k(T)$  correlation and viscosity  $F_\mu(T)$  correlation are taken from the gas the node inherits from its parent.

### 2.2.1.3 Solid

**Mass ( $m$ )** – The total amount of mass in the node, the mass is constant and initial determined as  $m = V\rho$ , whereas density is extracted from the material of its parent.

Heat capacity ( $C_T$ ) is taken from the material the node inherits from its parent.



### 2.2.1.4 Environment

**Static convection coefficient ( $h$ )** – Corresponds to the ambient atmospheric conditions, a stagnant environment would be associated with a low convection coefficient. A well-ventilated space or windy space may correspond to a higher value.

The environment node represents the atmospheric surrounding of the engine, it serves as a source of gas, a place of exhaust and a constant temperature source. It contains all the properties of a gas node, but never change from initial calculation. Additionally, the properties of mass ( $m$ ) and volume ( $V$ ) are equal to infinity for calculation purposes.

## 2.2.2 Faces

A face is a physical interface between nodes. A face can take many forms, whether gas-gas, gas-solid or solid-solid or any nodes with the environment. In general, a face contains an area and transmit energy between nodes based on transport, conduction and convection. Faces can have evolving properties or vanish when their two interacting nodes are no longer overlapping during periods of the cycle. There are 4 variants of faces as outlined in the following subsections

### 2.2.2.1 Gas-Solid

**Surface area ( $A$ )** – The wetted area of the solid node concerning the gas node:

- Normal to the radial direction:  $A = 2r_{fc}\pi(y_{max} - y_{min})$
- Normal to the axial direction:  $A = \pi(r_{max}^2 - r_{min}^2)$

**Resistance ( $R$ )** – Calculated from the surface to the center of the solid node:

- Conduction in the radial direction:

$$R = \frac{r_{fc} \ln(r_{ratio})}{A \cdot k} \quad (21)$$

where:  $r_{int}$  = radius of the nodal interface.

$r_{ratio}$  = The minimum of  $\frac{r_{fc}}{r_i}$  and  $\frac{r_i}{r_{fc}}$  (see Note)

Note:  $r_i = \sqrt{r_{min} \cdot r_{max}}$  as the position in the element where the conduction coefficient towards the inside is equal to the conduction coefficient towards the outside. In cases where  $x_{min}$  is zero,  $r_i = \frac{2}{3}x_o$  representing the average radius of the body.

- Conduction in the axial direction:

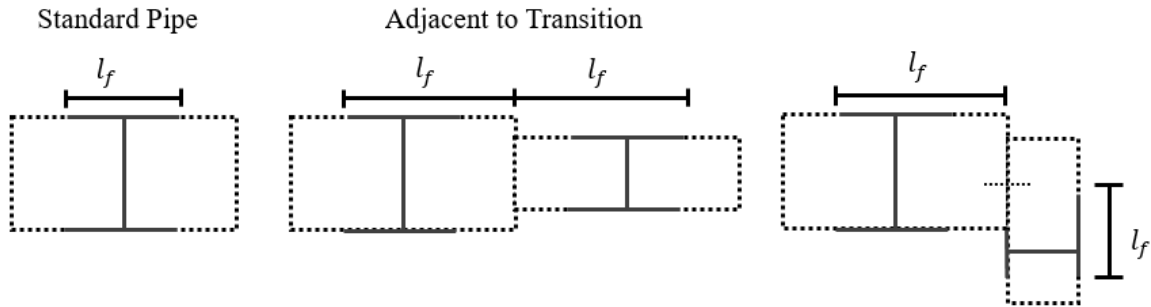
$$R_{fc \rightarrow i} = \frac{L_{fc \rightarrow i}}{A_{fc} \cdot k_i} \quad (22)$$

where:  $L_{fc \rightarrow i}$  = Distance from the interface to the center of the solid node in the direction parallel to the group axis.

### 2.2.2.2 Gas-Gas: & Gas-Environment

**Area (A)** – Calculated the same as with Gas-Solid faces

**Friction distance ( $l_f$ )** – Represents the length that the face calculates its friction over. A graphical representation is found in Figure 2.4.



**Figure 2.4: Geometrical cases and resulting friction length**

In cases where the node leads into a transition where a minor loss coefficient is used, then this friction length extends to cover up until the transition. This only occurs in scenarios where the node does not branch, for example, where each node involved only has 2 gas faces.

**Minor loss coefficient ( $K_{12}, K_{21}$ )** – replaces the Darcy friction factor pressure loss, minor loss coefficients are calculated by first determining three areas: the initial area ( $A_1$ ), the orifice area ( $A_2$ ) and the final area ( $A_3$ ). The minor loss coefficient is then calculated by the following equation.

$$K = \begin{cases} (1 - B^2)^2 & B > 0.76 \\ 0.42(1 - B^2) & \textit{otherwise} \end{cases} \quad (23)$$

where:  $B$  = ratio of cross-sectional areas, equal to the smaller area divided by the larger area.

The vast quantity of potential permutations of this quantity as well as its evolution under laminar regimes are ignored under this implementation, this coverage is merely included as an estimate and is not the primary contributor to flow losses when compared to matrix passages found in Stirling engines.

**Hydraulic diameter ( $d_h$ )** – calculated the same as it is for gas nodes.

**Conduction distance ( $l_c$ )** – represents the length between neighbor node centers for thermal conduction, this is the average dimension of the interacting nodes normal to the orientation of the face.

**Stability distance ( $l_s$ )** – is the length used when solving for the local Courant number. This is the minimum dimension of the interacting nodes normal to the orientation of the face. In most situations it is equal to the minimum of the two aligned-node sizes. The Courant number is typically defined as:

$$N_{Co} = \frac{\delta \cdot U}{l_s} \quad (24)$$

where:  $\delta$ : The length of a time-step during iteration.

$U$ : Gas velocity normal to the direction of  $l_s$ .

**Reynolds Number ( $N_{Re}$ )**

The Reynold's number is calculated at the start of each cycle and as required after that for friction updates. The Reynold's number is defined here as:

$$N_{Re} = \frac{\rho \cdot U \cdot d_h}{\mu} \quad (25)$$

where:  $\rho$ : Density of fluid.

$U$ : Velocity of fluid.

$\mu$ : Dynamic viscosity of fluid.

$d_h$ : Hydraulic diameter of channel in direction of  $U$ .

### Friction function ( $F_{N_k}(N_{Re})$ )

Calculates the Darcy friction factor ( $N_f$ )

Radial Flow	Annular Flow	Cylindrical Flow
Laminar		
$96/N_{Re}$	$96/N_{Re}$	$64/N_{Re}$
Turbulent		
$0.11 \left( \frac{l_r}{d_h} + \frac{68}{N_{Re}} \right)^{0.25}$ [35]		

### Mixing function ( $F_{N_k}(N_{RE})$ )

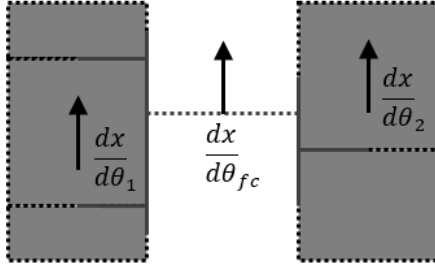
Calculates the mixing enhancement factor ( $N_k$ ), for all geometries.

Laminar:  $N_k = 1$

Turbulent:  $N_k = 0.022(N_{Re})^{0.75} \cdot N_{Pr}$

### Shear Factor ( $C_{shear}$ ) & Velocity Factor ( $C_{velocity}$ )

Shear Factor is defined as the shearing rate of the gas node, in the axial direction, declared with units of m/radians. Often combined with velocity factor. Velocity Factor is defined as the relative motion of this gas face relative to the walls around it. Declared with units of m/radians.



$$F_S = \left| \frac{dx}{d\theta_1} - \frac{dx}{d\theta_2} \right| \quad (26)$$

$$F_V = \left( \frac{dx}{d\theta_{fc}} - \frac{1}{2} \left( \frac{dx}{d\theta_1} + \frac{dx}{d\theta_2} \right) \right) \quad (27)$$

**Figure 2.5: Shear and velocity factor input variables**

### 2.2.2.3 Solid-Solid

**Conductance (C)** – Calculated from the center of one node to the other, these are the inverses of the resistances calculated under the gas-solid face (equations (21) and (22)), combined in series to allow for material discontinuities.

$$C = \left( \frac{1}{C_{i \rightarrow fc}} + \frac{1}{C_{fc \rightarrow j}} \right)^{-1} \quad (28)$$

### 2.2.2.4 Solid-Environment

**Conductance (C)** – Calculated from the center of the solid node to the environment using the constant convection coefficient combined in series based on the following formula.

$$C = \left( \frac{1}{C_{i \rightarrow fc}} + \frac{1}{h_j A_{fc}} \right)^{-1} \quad (29)$$

In practice the conductance of the engine with the environment will be a constant value, and therefore it possesses the same symbol as the conductance used for solid-solid interactions.

## 2.2.3 Node Contacts

A node contact is a temporary element that is used to define the contact of a node against the 1D surface of a connection. This allows all bodies to be discretized and these temporary data structures to be sent to the respective connection to be later combined into faces. For connections in the form of a cylindrical shell, a node contact represents a thin ring shape composed of a lower

and upper bound in the direction of the local axis normal. For other connections of the horizontal type, a node contact will be composed of an annular disk shape, with an inner and outer radius, lying on the connection plane and centered concerning the axis. These boundaries also move with the node, which facilitates the creation of faces with angularly dependent areas, existing only during the crossing point of two nodes. The interaction with bridges necessitates the inclusion of a local porosity, which represents how much of the total surface area is available, i.e. the hole drilled to access the top of the engine body for the power piston would reduce the surface area of the top plate.

#### **2.2.4 Pressure Contacts**

A pressure contact is a special surface used during simulation to connect a mechanism to surfaces on which pressure acts. Pressure contacts are exclusive to faces that are both moving and horizontally oriented. The properties of pressure contact include its area, direction and node index from which the pressure can be recovered from the output array. This pressure is converted into a force and sent to the mechanism during the mechanism loop.

#### **2.2.5 Shear Contacts**

Like a pressure contact, the shear contact approximates the proportion of shear drag from any gas channels parallel to the moving surface. The properties of a shear contact include the acting area, force direction and the face index from which shear is calculated. The force is calculated as one half of the pressure drop across the face, which is attributed to the shear drag against the wall.

### **2.3 Interactable Elements**

---

Interactable objects encompass the group of objects that the user modifies, moves about, and interacts with. These objects can be thought of as a high-level perspective of the final numerical model, which is produced when the model file is discretized. These elements all work within the property inspection interface, can be created via the GUI, and contain the functionality to translate their settings into numerical elements containing their own complex functionality.

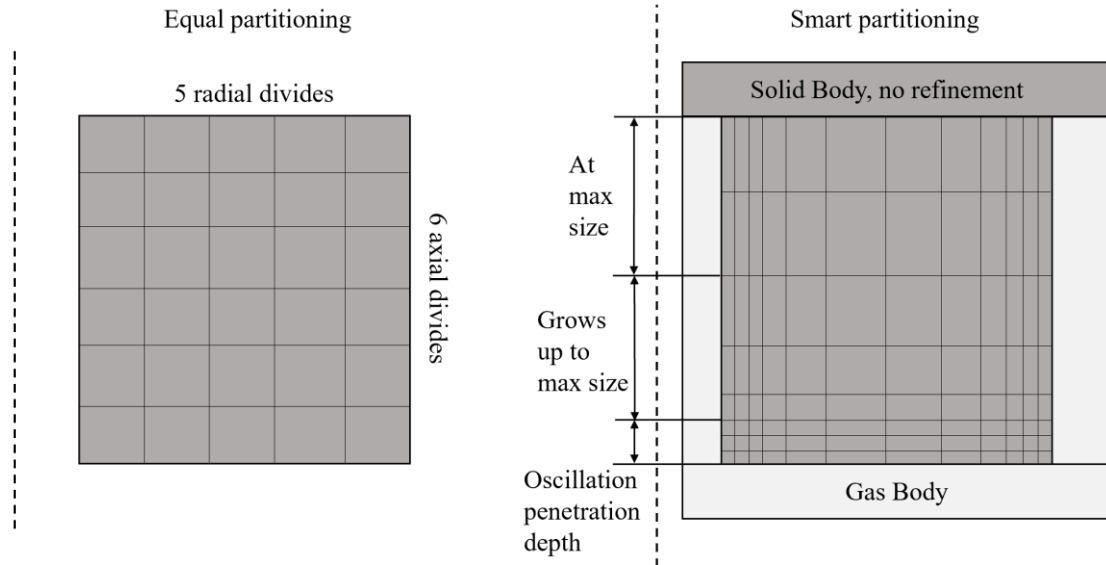
### **2.3.1 Groups**

Groups are collections of bodies that all lie around a common axis of symmetry. Groups contain a set of physical properties, including orientation data for their axis and the bounding box that surrounds its contents. The group component also includes the functionality to form connections between the bodies it encapsulates and the environment. In brief, this function collects all the segments of all the connection at which a body intercepts, it then removes all the segments that are covered twice by any body (i.e. a body on either side). The remaining segments after this algorithm are either part of an illegal open space internal to the engine or are exposed to the surroundings. At the point of discretization this information is passed down to the respective connections a temporary construct known as a node contact featuring the gas node that represents the environment. After processing, this results in connections that facilitate energy and mass exchange with the environment when the model is simulated.

### **2.3.2 Bodies**

Bodies represent a cylindrical or annulus shaped element that is aligned with a group axis. A body can be any material, solid or gaseous. Bodies contain references to 4 connections that serve to define its dimensions in the axial and radial directions. The primary function of a body is to be discretized into nodes and faces. Those nodes that lie on a boundary are passed as node contacts to connections. A body containing gas can also contain an internal structure represented by a Matrix component, which it provides its created nodes to be further modified. Bodies also contain functionality for validation, depending on the material. For example: solid bodies cannot change its dimensions but may translate. Meanwhile, a gaseous volume can both translate and stretch in their axial direction. Neither material allows for bodies to overlap or invert, which prevents the model from simulating an engine that couldn't exist or function.

Bodies can be discretized by two modes, which are shown in Figure 2.6, these modes can be optionally applied to one of or both axial and radial directions:



**Figure 2.6: Discretization modes for body**

**Equal partitioning** – The body is sliced into several equal layers in both the radial and axial directions.

**Smart partitioning** – In solid bodies this discretization is dependent on whether a side is in contact with a gas node or within a specific distance of a gas body. Only a small part of a side must be in contact with a gas node to activate this function. But at those edges a specified number of nodes will be placed within the oscillation penetration zone, a zone defined by the thermal diffusivity and expected frequency of the engine test (discussed in section 6.2.1). Beyond that zone the node size grows by a specified growth factor to a maximum node size. If neither side is in contact with a gas node the body is discretized coarsely with equal partitioning such that the node size is not greater than the maximum node size.

In Gas nodes this discretization works along the discretized direction and creates a series of thin entrance nodes over the first and last 15% of the total length. Beyond these regions the nodes grow using the prescribed growth factor. This style of discretization is inspired by the work of Anderson [48] who applied this technique to regenerator and heat exchanger elements. These elements are discussed in the next subsection as matrixes.



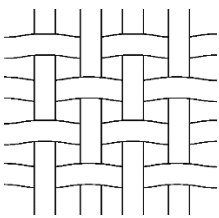
### 2.3.3 Matrixes

Matrixes are found nestled within gaseous bodies; these elements represent a variety of fine channeled geometries such as regenerators and heat exchangers. These elements are closely in contact with the gas network and each has a porosity and geometry that merits an override of the default defined hydraulic diameter ( $D_h$ ), volume ( $V$ ), area ( $A$ ), Darcy friction factor ( $N_f$ ) correlation, Nusselt number ( $N_{NU}$ ) correlation, and Axial Mixing enhancement coefficient ( $N_k$ ). When the matrix takes its parents nodes, it both modifies the provided nodes for the listed properties and adds solid sources, surface nodes and faces which model the solid components of the heat exchanger or regenerator as a nodal network. The following sub-sections discuss the different types and their different properties.

#### 2.3.3.1 Regenerators

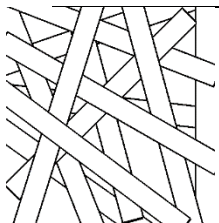
The types of common Stirling engine regenerator structures and their discretization are outlined here. For regenerators which are notably almost always laminar due to the domination of viscosity, only the laminar correlation is provided. Property correlations, which for regenerators are taken from Gedeon [35], are summarized in Appendix B of this thesis. The Darcy-Weisbach equation is used here instead of Darcy's law for flow through porose media both because it is used successfully in SAGE [35] and as a consequence of maintaining consistency with the rest of the model.

##### 2.3.3.1.1. Woven Screen



A woven screen regenerator is composed of a tight weave of filaments, often arranged in layers perpendicular to the flow direction to minimize parallel conduction losses. This is like the perforated screen type regenerator that is not implemented here. The woven screen has 2 inputs: porosity ( $\beta$ ) and wire diameter ( $d_o$ ).

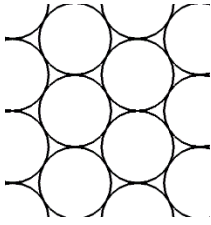
##### 2.3.3.1.2. Random Fiber



A random fiber regenerator is usually composed of felt or batting of randomly oriented fibers. The random fibre matrix has 2 inputs: porosity ( $\beta$ ) and wire diameter ( $d_o$ ).

### 2.3.3.1.3. Packed Sphere

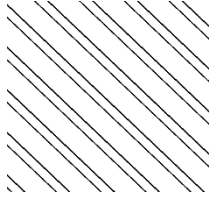
---



A packed sphere regenerator is composed of many small packed spheres, sometimes sintered together. The packed sphere matrix has 2 inputs: porosity ( $\beta$ ) and sphere diameter ( $d_o$ ).

### 2.3.3.1.4. Stacked Foil

---



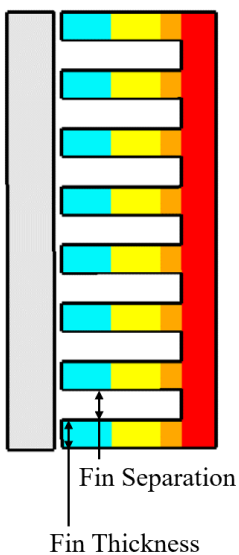
A stacked foil regenerator is composed of many thin parallel channels between thin foil elements. As this model is the most like an open channel, it provides a laminar and turbulent friction and Nusselt number definition. The stacked foil matrix has 3 inputs: the gap width ( $l_g$ ), thickness ( $l_t$ ) and surface roughness ( $l_r$ ).

## 2.3.3.2 Heat Exchangers

Heat exchangers come in a wide array of different types; a selection of implemented types is described here. In each of the diagram's heat flow is identified by coloration with red at the heat source and blue as it gets farther from the heat source. Air either travels into the page through the open spaces or as indicated by arrows. All the correlations and property calculations for heat exchangers can be found in Appendix B.

### 2.3.3.2.1. Fin Enhanced Surfaces

---



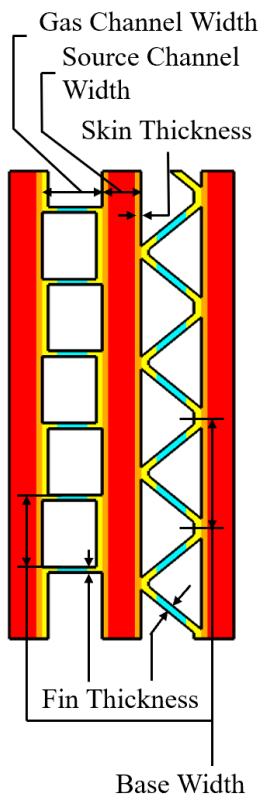
This type of heat exchanger is the case where one of the walls is the heat exchange surface, this surface is covered with long parallel fins that span across the gas space. This type of heat exchanger is common in a scenario where the engine body itself conducts heat to and from a source/sink of heat. Cases, where there are no fins and only a bare wall are handled natively by the conduction with the wall, without need for a matrix component.

Functionally, this matrix component allows the user to select a connection from which to grow the fins, the surface of this connection is

then integrated into the internal solid conduction network, which layers nodes normal to the selected connection.

Fin enhanced surface exchangers have 4 inputs and assumes rectangular fins: target connection, distance between fins ( $l_g$ ), fin thickness ( $l_{th}$ ) and surface roughness ( $l_r$ ). Fin length is determined automatically by assuming that the fins go right up to, but don't touch the opposite side of the parent body.

### 2.3.3.2.2. Fin Connected Channels



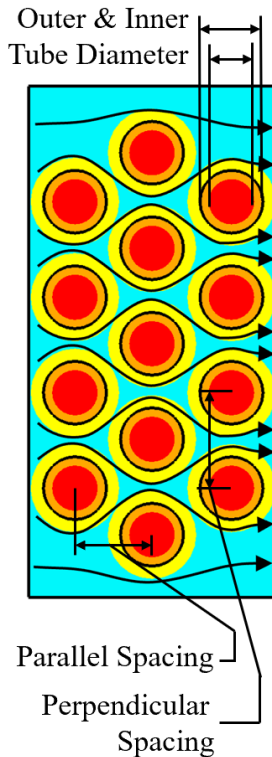
This type of heat exchanger encompasses the subclass of heat exchangers known as plate and frame heat exchangers. The main feature being that the cross-section is an alternating pattern of the two fluids, which persists through the depth.

The following two sub-types refers to the structure of the interstitial fins:

- Rectangular Gaps: The fins cross at 90° across the gas side forming many rectangular paths for the gas to flow through.
- Triangular Gaps: The fins zig-zag across the gas side forming many triangular paths for the gas to flow through.

This heat exchanger has several inputs including: fin type (rectangular / triangular), gas space between source channels ( $l_{c,g}$ ), source channel width ( $l_{c,w}$ ), skin thickness ( $l_{c,wth}$ ), surface roughness ( $l_r$ ), base width / fin separation ( $l_{f,g}$ ) and fin thickness ( $l_{th}$ ).

### 2.3.3.2.3. Fin Connected Tubes



This type of heat exchanger encompasses the subclass of heat exchangers known as compact heat exchangers or finned tube heat exchangers. The main feature of these heat exchangers is that one fluid (assumed to be the source/sink fluid) traverses through a series of tubes that are covered in surface enhancing fins or plates. The tubes are most often arranged perpendicular to the flow direction.

#### Fin Type

**Continuous Plate:** The fins of this subtype bridge across from tube to tube forming continuous plates. These are automatically produced when the user does not submit a fin length.

**Individually Finned:** The fins of this subtype are associated with just one tube.

#### Tube Pattern

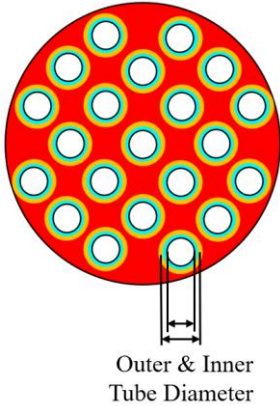
**Staggered:** Staggered tube means that each consecutive layer of tubes is offset relative to the previous one by exactly half the perpendicular tube spacing. This is seen in the figure to the left.

**Aligned:** Aligned tubes are aligned, such that each tube lies in the wake of the previous one. Aligned is generally not used unless a very low-pressure drop is the essential requirement [52]. Which is why it is not implemented here.

This type of heat exchanger accepts several inputs from the user including: spacing perpendicular to flow ( $l_{perp}$ ), spacing parallel to flow ( $l_{para}$ ), fin thickness ( $l_{th}$ ), fin separation ( $l_g$ ), tube outer diameter ( $d_o$ ) and tube inner diameter ( $d_i$ ). In which fin thickness and fin separation are identical to that identified for fin enhanced surfaces.

#### 2.3.3.2.4. Tube Bank Internal

---



This type of heat exchanger is relatively common in high temperature engines, it involves the forcing of air through many parallel tubes each submersed in a thermally charged environment. The design attempts to minimize flow losses while providing maximum surface area and pressure containment ability.

This type of heat exchanger accepts 3 inputs: number of tubes ( $N$ ), outer tube outer diameter ( $d_o$ ) and tube inner diameter ( $d_i$ ).

### 2.3.3.3 Discretization

#### 2.3.3.3.1. Regenerators

---

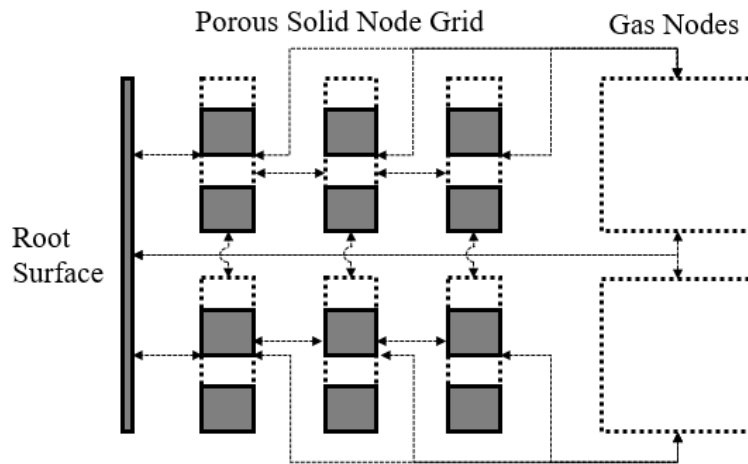
Regenerators typically contain extremely fine geometry, a well-designed regenerator, one that behaves most like an ideal reversible device will maintain its temperature as close as possible with the gas's temperature. Thus, for a well-designed regenerator, the lumped mass assumption should be able to be applied with little error. As a result, a typical regenerator only requires a single layer of nodes. Discretization is straightforward then, simply construct a solid node for each gas node in the body and form a mixed face between the two elements. The properties relevant to discretization are as follows for each regenerator:

**Table 2.1: Discretization specific properties for common regenerator types.**

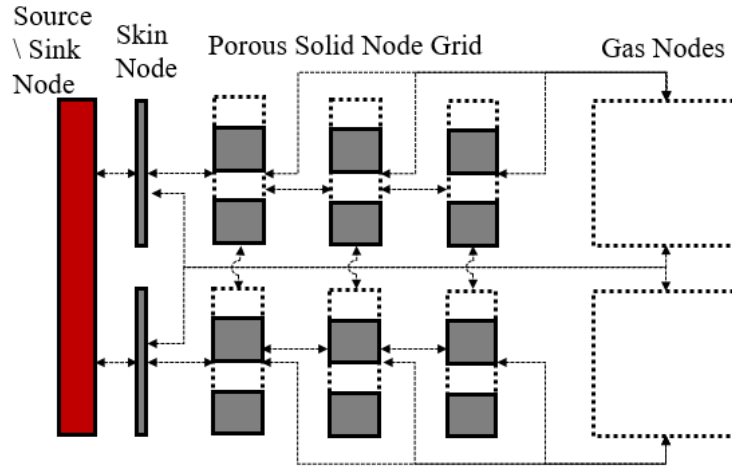
	Woven Screen & Random Fiber	Packed Sphere	Stacked Foil
Surface Area / Total Volume (A/V)	$4 \frac{1 - \beta}{d_o}$	$6 \frac{1 - \beta}{d_o}$	$\frac{2}{l_t + l_g}$
Resistance to average radius times Area (R.A)	$\frac{\ln\left(\frac{3}{2}\right) d_o}{2k}$	$\frac{d_o}{6k}$	$\frac{l_t}{4k}$

### 2.3.3.3.2. Heat Exchangers

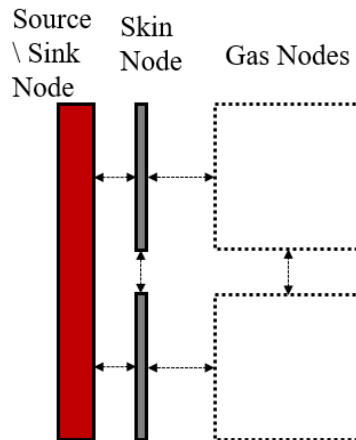
Heat exchangers are different compared to regenerators, as their purpose is to transport heat from a physically separate space - the thermal reservoir - to the gas through their conducting surfaces. A well-designed heat exchanger should have as little as possible resistance as this ensures that the surfaces of the heat exchanger are as close as possible to the thermal source/sink, but often they include extended geometry which may help the heat transfer but not be exactly at the desired temperature. Therefore, heat exchangers are discretized along the path from the source to the gas. The discretization scheme for the implemented types of heat exchangers is depicted in Figure 2.7, through Figure 2.9 below.



**Figure 2.7: How elements are discretized in the fin enhanced surface type heat exchanger**



**Figure 2.8: How elements are discretized in the fin connected channels and finned tube type heat exchanger**



**Figure 2.9: How elements are discretized in the tubes bank internal type heat exchanger**

### 2.3.4 Connections

A connection represents a surface either as a cylindrical shell or as an axially normal plane. Connections have a position, an orientation and a reference to a frame. A connection can use the frame's motion profile to shift its position depending on the angular position, which can move the associated bodies. This only works when the connection is aligned as an axially normal plane.

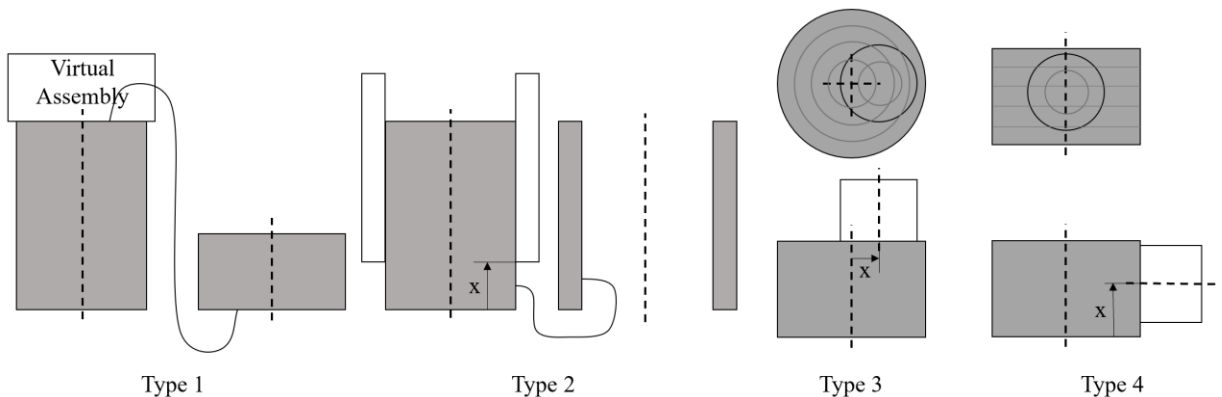
To be discretized, connections find overlaps of node contacts, which over the course of discretization, have been received from the environment, bodies and bridges. In many cases the

properties of the faces that are generated from these overlaps will vary with respect to angular position, thus these properties are stored as arrays, one value for each angular increment.

### 2.3.5 Bridges

A bridge is a geometric construct that facilitates the interface between two bodies, which may or may not be part of the same group. These interfaces can occur between connections of perpendicular and aligned orientations. The referenced bodies then connect, in disregard for any existing external connections at that location.

There are four permutations of bridge definition. The first two are types which could have been constructed as a combined structure from the start but convenience or perhaps requirements by the program have made it more convenient to have separate. These versions are shown on Figure 2.10 below along with the third and fourth type. The third type of bridge is where the two selected connections are both horizontal, much like type 1, but their axis are misaligned such that the node interactions are only partial interactions. The fourth type is where there is a mix, where a horizontal connection contacts the side of a vertical connection.



**Figure 2.10: Permutations of the bridge component definition (1) two horizontal (disk) faces are stacked along a central axis (2) two vertical (annular shell) faces are aligned at some offset from the origin (3) two horizontal faces are stacked with axis offset by a specified amount (4) a horizontal face is perpendicularly mated up against a vertical face with the axis at a prescribed offset from the origin of the vertically aligned face.**

Type 1 and 2 add their own node contacts to the register of the foundation connection, and trim node contacts on that register that both overlap the new contacts and are not from either body. Types 3 and 4, as their construction is much more involved, create a series of faces of size equal



to that which was produced by the overlap of 2 annular rings – in the case of type 3 – or an annular ring and rectangle – in the case of type 4. The software then modifies the existing node contacts by changing their porosity value, which will modify the area of faces created in subsequent operations within the connection. It is important to note that because the software assumes that the model is axially symmetric, any resulting faces are also axially symmetric, thus faces end up stretched around the entire body rather than towards one side of the body.

### **2.3.6 Leaks**

A leak component is a special feature that connects two separate bodies as if connected by a small channel. When the leak is discretized, it forms a special face that stores the rate law parameters for use by the solver. The rate law employed by default here is of the general form, which has been applied as far back as 1881 [53].

$$\dot{V} = C \cdot (P_1 - P_2)^{N1} \quad (30)$$

where:  $C$ : Leakage number

$N1$ : Leakage exponent

### **2.3.7 Non-Connection**

A non-connection was added so that the designer could create idealized representations. If two bodies were not supposed to interact via conduction, convection, or transport, then the non-connection can turn off these interactions by filtering the produced faces.

### **2.3.8 Custom Minor Losses**

The custom minor loss feature was added to override the default defined minor loss coefficient created between any two bodies. This feature also allows the user to add facsimile of check valves by having a small loss in one direction, but a large loss in the other. Given the discussion defined in CHAPTER 3, this will only work as a check valve when the face is a part of a loop, due to the uniform pressure assumption, thus an alternative path must be provided, or the fluid will flow regardless.

### 2.3.9 Frames

A frame is a container for holding a position vector and mechanism reference. Connections that contain a frame reference and are subject to pressure or shear force produce a Pressure or Shear Contact object, which during simulation will pass these forces onto the mechanism itself.

### 2.3.10 Mechanism

Within the code, the mechanism is a container for a series of connected linear to rotational mechanism. These sub mechanisms all share a common angular motion state. The mechanism itself takes an accumulated driveshaft load as an input to its own internal friction, inertia, and load calculations to return an acceleration. Generally, the flywheel and power outputs are defined within this object. Currently the code does not support multiple mechanisms with multiple angular positions, which would be one way to simulate free-piston engines or engines that have a free-floating displacer piston.

The child linear to rotational mechanisms turn the rotation of the drive shaft into a translation, that can be applied to a boundary. During each calculation step, occurring once per angular increment, an average pressure force is calculated for all faces aligned normal to their motion. These pressure forces form a combined piston load for each installed linear to rotational mechanism – the forms of which can be found in the following sections. Each of these linear to rotational mechanisms is derived to solve its internal friction and inertia and provide to the driveshaft a pair of normal forces as well as a torque force.

Each mechanism contains stored coefficient vectors in the form of:

$$F = A(\theta)\alpha + B(\theta)\omega^2 + G(\theta) + E(\theta)F_p \quad \text{same as} \quad (8)$$

where:  $A(\theta)$ : Coefficient on angular acceleration ( $\alpha$ ), represents the system inertia as a function of angular position ( $\theta$ )

$B(\theta)$ : Coefficient on angular velocity squared ( $\omega^2$ ), represents internal inertia as a function of angular position ( $\theta$ )

$G(\theta)$ : Gravitation force, as a function of angular position ( $\theta$ )

$E(\theta)$ : Coefficient on piston force ( $F_p$ ), which represents how the piston force, when translated through the mechanism produces a load on the drive shaft.

In addition to the coefficients, all internal loads required to calculate the friction load ( $F_f$ ) are also stored, which have their own equations of the same form. Friction is a relatively small component of internal forces, thus its effect on itself and normal forces is ignored and simply subtracted from the torque after its power consumption is calculated, via:

$$\dot{E}_{friction} = -|\omega T_{friction}| = -|v F_{friction}| \quad (31)$$

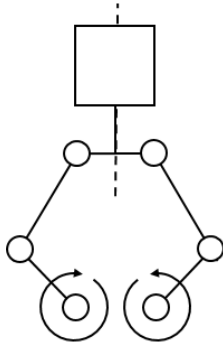
The following sub-sections are common mechanisms used by DTECL; additional mechanisms can be added to the code by following the template laid out by these. If a motion is desired, more than the mechanism behind it, then the custom profile mechanism attempts to predict some of the physics for such an unknown mechanism.

### 2.3.10.1 Slider-crank Mechanism



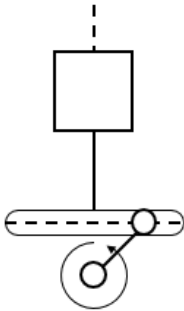
The slider-crank is by far the most recognizable linear to rotational mechanism. The slider-crank is used almost universally by internal combustion engines, this commonality leads it to be a convenient mechanism when designing Stirling engines. Constructed out of 3 components: the crank arm, the connecting rod, and the piston. There are three loss mechanisms associated with slider-cranks that are considered: friction between the crank and connecting rod, friction between the connecting rod and the piston and friction of the piston seal, all of which are subtracted from the final torque. The derivation of the parameters can be found in Appendix A.1.

### 2.3.10.2 Rhombic Drive



The rhombic drive is a very common mechanism in beta-type Stirling engines, utilized primarily for its compact design and reduced side loads. Also, the rhombic drive can be constructed such that it drives both the displacer and power piston on the same mechanism through the addition of a secondary set of connecting rods and a nested shaft. In its essence, a rhombic drive is simply a double slider-crank mechanism and thus its parameter derivation is built upon the solution of the slider-crank with some specific modification, this derivation can be found in Appendix A.2.

### 2.3.10.3 Scotch Yoke



The Scotch yoke mechanism is a relative of the slider-crank mechanism that, instead of a connecting arm, depends on a slot linkage to drive the piston forward. The Scotch yoke is often used because its side loads are more easily managed or relatively low due to the roller system. Another notable property of the scotch yoke is that it produces a truly sinusoidal motion profile. The derivation of the parameters can be found in Appendix A.3.

### 2.3.10.4 Ideal Sinusoidal Mechanism

The ideal sinusoidal motion model is intended for use at the early stage of engine development. The sinusoidal model is a frictionless scotch yoke and is only distinct as its definition interface is based solely on the properties of the sine wave itself without introducing linkage dimensions. The derivation of these parameters can be found in Appendix A.4.

### 2.3.10.5 Custom Motion Mechanism

The user can test different motion profiles for their effect on the thermodynamic cycle using a custom motion mechanism. This mechanism results in a model that approximates the inertia of the piston and driving mechanism through analysing the provided motion profile. The formulation is discussed in Appendix A.5.

## 2.4 Conclusion

---

Through the discussions presented in this chapter, it was established that the gas would be modelled as a one-dimensional pipe network of uniform pressure. All geometry would be considered axially symmetric, under the reasoning that well-designed Stirling engines are built in that fashion. Under that, the solid would be modelled as a two-dimensional network. The mechanism, due to low accelerations, would lag the gas network by short angular increments, in which the velocity was deterministic.

Those discussions lead to the establishing of a model definition composed of annular blocks of material, that connect to each other via mobile surfaces, grouped around a common axis. Bridges and a variety of surface modifiers were added to support specialized geometry. Fine structures are represented by network generating matrix components. These constructs decompose into a network of smaller allotments of material called nodes and the faces that link them together. The following chapter will discuss the mathematics required to solve this network.

# CHAPTER 3. CORE MATHEMATICAL PROCESSES

---

The following sections outline most of the mathematics used to solve the model during a simulation. The mathematical complexities of specific components outlined in the subsections within CHAPTER 2. This chapter in combination with CHAPTER 5 brings it all together into a network solvable by a computer.

## 3.1 Terminology

---

The following terminology is defined here, other terms such as nodes, faces and the environment are defined in CHAPTER 2.

**Region:** A region is a set of gas nodes, which are always connected during the entire cycle through one path or another.

**Loop:** A loop occurs whenever there is more than one path between one part of a region to another part of a region. A loop that often occurs in gamma type engines is the path past the displacer and the path through the heat exchangers, which represent two different ways to get to form the compression to expansion space.

## 3.2 General Heat Transfer

---

### 3.2.1 Thermal Conduction Within Solids

The solid conduction model is based upon Fourier's Law, which calculate the thermal energy transfer rate ( $\dot{E}_{cond}$ ).

$$\dot{E}_{cond} = k \cdot A \cdot \frac{dT}{dx} \quad (32)$$

where:  $k$ : Conduction coefficient of conducting material.

$A$ : Cross-sectional area over which conduction occurs.

$x$ : Position along the direction of heat flow.

This can be rearranged to suit the case of conduction between nodes.

$$\dot{E}_{cond,fc} = \frac{A_{fc}}{\frac{L_{i \rightarrow fc}}{k_i} + \frac{L_{fc \rightarrow j}}{k_j}} (T_i - T_j) \quad (33)$$

where:  $A_{fc}, \dot{E}_{cond,fc}$ : Cross-sectional area and thermal energy transfer rate of face, from node  $i$  to  $j$ .

$L_{i \rightarrow fc}, L_{fc \rightarrow j}$ : Absolute distance, normal to the face, from the center of node  $i$  or  $j$  to the face's surface.

$k_i, k_j, T_i, T_j$ : Conduction coefficients and temperatures associated with the nodes  $i$  or  $j$ .

In its final form, as seen in the model:

$$\dot{E}_{cond,fc} = C_{cond} \cdot (T_i - T_j) \quad (34)$$

where:  $C_{cond}$ : Combined coefficient that converts a temperature difference into an energy flux.

Within the model faces can vary with angular position, thus this conductance property is condensed into a single interpolated property. For static faces only the result of the above coefficient is stored. This property is assumed to be constant concerning temperature.

### **3.2.2 Thermal Conduction Within Gases**

Conduction between gas nodes is much like conduction between two solid nodes. Here only molecular conduction is solved. Radiation would also be solved here but is ignored in this model due to the low temperatures assumption. Thermal energy that is carried with mass flows is solved within the volumetric flow rate solving step, section 3.3.

$$\dot{E}_{cond} = k \cdot A \cdot \frac{\partial T}{\partial x} \quad (35)$$

This is arranged to suit the case of internodal conduction/convection. The factor  $N_k$  is a dimensionless conduction enhancement factor, which arises when turbulence or geometrical pathing enhances the streamwise mixing action.

$$\dot{E}_{cond,fc} = \frac{N_{k_{fc}} \cdot k_{ij} \cdot A_{fc}}{L_{i \rightarrow j}} (T_i - T_j) = C_{cond} \cdot (T_i - T_j) \quad (36)$$

where:  $N_{k_{fc}}$ : Conduction enhancement factor.

$k_{ij}$ : Thermal conduction coefficient measured at the face, equal to the average value of node  $i$  and  $j$ 's conduction coefficient.

$L_{i \rightarrow j}$ : Cumulative distance between the center of node  $i$  to the center of the face to the center of node  $j$ .

### 3.2.3 Thermal Conduction Between Solids and Gases

The conduction model between gases and solids is based upon the combination of conduction and convection; radiation is ignored.

$$\dot{E}_{cond} = \frac{A_{fc}}{\frac{L_{i \rightarrow fc}}{k_i} + \frac{1}{h_j}} (T_i - T_j) \quad (37)$$

where: Node  $i$  is the solid node and node  $j$  is the gas node.

$h_j$ : Is the convection coefficient produced by gas node  $j$ 's internal geometry and flow conductions.

This is arranged to suit the case of internodal conduction/convection.

Due to the existence of variable area contacts, resistance is stored as the product of thermal resistance and area. The following equation is modified to move area to the numerator, and use  $RA$  in the denominator. As resistance is equivalent to a constant over area, multiplying by area gives a constant. In this way, only area must be interpolated as resistance is proportional to the inverse of area.



$$\dot{E}_{cond,fc} = \frac{A_{fc}}{R_{i \rightarrow fc} \cdot A_{fc} + \left( \frac{d_h}{N_{NU} \cdot k} \right)_j} (T_i - T_j) \quad (38)$$

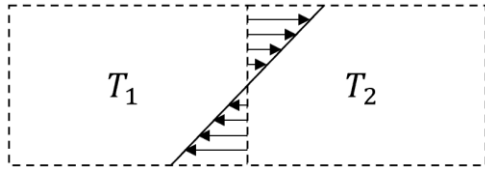
where:  $N_{NU}$ : The Nusselt number equal to  $\frac{h \cdot d_h}{k}$

$d_h$ : Hydraulic diameter with respect to the flow in node  $j$

In a special case of constant convection coefficient, the system can be simplified to that of the solid conduction. This is what is applied for connections to the environment.

### 3.2.4 Shearing Conduction Enhancement

In Stirling engines with a displacer piston, there is always a region of gas, called the annular gap, that undergoes shearing. The act of shearing will affect the effective axial conduction coefficient. This effect is approximated by assuming two things, which are also displayed graphically in Figure 3.1:



**Figure 3.1: Illustration of shear driven mixing**

1. Shearing conduction enhancement occurs independently of transport via bulk convection.
2. A gas node is a uniform temperature with a discontinuity at the face between.
3. The shear velocity profile is fully developed and linear.
4. After crossing the boundary, flows mix completely.

This equation introduces a non-dimensional number called the shear velocity factor, shear velocity is equal to the ratio of shear speed over rotational speed. The mass exchanged is equal to the following:

$$\dot{m}_{fc} = \underbrace{\left( \frac{A_{fc}}{2} \right)}_{\text{Cross-section over which it occurs}} \underbrace{\left( \frac{F_{shear} \cdot \omega}{2} \right)}_{\text{Average Velocity}} \underbrace{\rho_{fc}}_{\text{Density at Face}} = \frac{1}{4} A_{fc} \cdot F_{shear} \cdot \omega \cdot \rho_{fc} \quad (39)$$

where:  $F_{shear}$  = The face shearing rate expressed as velocity over angular speed.

$\rho_{fc}$ : Density calculated at face. In the current implementation this value is taken from the node directly upstream from the face.

$\omega$ : Angular velocity of engine.

The rate of thermal energy conducted downstream by shear ( $\dot{E}_{shear}$ ) is equal to the following:

$$\dot{E}_{shear} = c_v \cdot \dot{m}_{fc} \cdot (T_i - T_j) = \frac{1}{4} c_v \cdot F_{shear} \cdot \omega \cdot \rho_{fc} \cdot A_{fc} \cdot (T_i - T_j) \quad (40)$$

where:  $c_v$ : Thermal heat capacity with respect to constant volume.

This converts into a conduction coefficient for shear ( $C_{shear}$ ) which is added to the existing conduction coefficient:

$$C_{shear} = \frac{1}{4} c_v \cdot F_{shear} \cdot \omega \cdot \rho_{fc} \cdot A_{fc} \quad (41)$$

### 3.3 Determining Flow Rates

---

The following sections outline the determination, implicitly, of the flow rates between nodes of the engine. This is derived starting from the foundation of equal pressures.

#### 3.3.1 Assumptions

The following assertions lead to inertia independent scheme for solving for the approximate internal thermodynamics of a Stirling Engine.

1. Pressure throughout connected regions is uniform in space but not time, in other words, pressure change due to temperature and volume changes is much greater than pressure changes from flow losses or acoustics.
2. The air within the engine behaves as an ideal gas, following the law:  $P = \frac{m \cdot R_{spec} \cdot T}{V}$

3. The engine rotational velocity changes are negligible within small angular increments of the cycle. Within these increments velocity is assumed to be known, but velocity may still evolve in a rate that lags by an increment.
4. Boundary work on a single node within a region is distributed among nodes of the region as if all the nodes were grouped into one.
5. Potential energy and kinetic energy concerning the gas is ignored given the low densities of gas molecules and small vertical displacements.
6. Energy transport via radiation is ignored it is dependent on temperature to the fourth power which for low temperatures results in a much lower effect at low temperature compared to other thermal energy transport modes.

### 3.3.2 Deriving the Systems of Equations

The values of volume flow rate, internal energy, nodal mass and temperature of the engine are solved based on the equations derived in the following section.

In consideration of the inherent compressibility of the Stirling engine system and avoiding methods that encourage the emergence of undesirable acoustic effects, the following assumption was used as the basis on which to derive the set of equations. The engine was first divided into distinct regions, within which a uniform pressure would be enforced:

$$P_i = P_j \quad (42)$$

where:  $i, j$ : refer to two different nodes that fall within the same gas region, whereas a gas region is an area of the engine which is always fully connected throughout the cycle.

This equation states that the pressure of each node constituting a connected volume within the engine has equal pressure. Starting from here the following is true for all nodes within a region:

$$\left(\frac{m \cdot R_{spec} \cdot T}{V}\right)_{i,t} = \left(\frac{m \cdot R_{spec} \cdot T}{V}\right)_{j,t}$$

where:  $t$ : refers to the state at the current timestep.

Thus, after a time step, the same condition applies:

$$\left(\frac{m \cdot R_{spec} \cdot T}{V}\right)_{i,t+\delta} = \left(\frac{m \cdot R_{spec} \cdot T}{V}\right)_{j,t+\delta} \quad (43)$$

where:  $t + \delta$ : refers to the next state, after the timestep is traversed.

The specific gas constant:  $R_{spec}$  is assumed constant over a region, therefore it is removed in further calculations. The first quantity, the volume at the new time ( $V_{i,t+\delta}$ ), is easily determined, as the volume at any angular position is predetermined by assumption 3:

$$V_{i,t+\delta} = f_{V_i}(\theta_{t+\delta}) \quad (44)$$

The second property, the mass of the gas node at the new time ( $m_{i,t+\delta}$ ), is as follows:

$$m_{i,t+\delta} = m_{i,t} + \sum \left( (\dot{V} \cdot y \cdot \delta \cdot \rho)_{fc} \right) \quad (45)$$

where:  $y$ : Sign of face with regards the sign convention of volume flow rate.  
If  $i$  is listed second with regards to face  $fc$  then  $y = 1$ . Otherwise,  $y = -1$ .

The third property, nodal temperature at the new time ( $T_{i,t+\delta}$ ), must be determined via an energy balance. The 1<sup>st</sup> law of thermodynamics is commonly represented as the following:

$$\Delta E_{tot} = \Delta Q_{tot} - \Delta W_{tot} \quad (46)$$

Whereas  $\Delta U_{tot}$  is the change in the total internal energy of the control volume.  $\Delta Q_{tot}$  is the transference of energy to the control volume through thermal energy.  $\Delta W_{tot}$  is the transference of energy away from the control volume by a force acting over a distance? The internal energy can be expanded into kinetic, potential, and internal energy.

$$\Delta E_{tot} = \Delta E_{kin} + \Delta E_{pot} + \Delta U \quad (47)$$

The term  $\Delta Q_{tot}$  can be expanded to include energy that is conducted to the control volume and energy that is transported to the control volume in the form of internal thermal energy.

$$\Delta Q_{tot} = \Delta t \sum \dot{V} \cdot y \cdot \rho \cdot u + \Delta Q \quad (48)$$

The term  $\Delta W_{tot}$  can be expanded to include the flow work, kinetic energy and potential energy of a flow as well as the boundary work.

$$\Delta W_{tot} = -\Delta t \sum \dot{V} \cdot y \cdot \rho \left( P \cdot \hat{v} + \frac{U^2}{2} + g \cdot z \right) + P \Delta V \quad (49)$$

Combining the expanded terms and converting into a derivative form the following equation is obtained:

$$\begin{aligned} & \underbrace{\dot{E}_{kin}}_{\substack{\text{Rate of change} \\ \text{of Kinetic} \\ \text{Energy}}} + \underbrace{\dot{E}_{pot}}_{\substack{\text{Rate of change} \\ \text{of Potential} \\ \text{Energy}}} + \underbrace{\dot{U}}_{\substack{\text{Rate of change} \\ \text{of Internal} \\ \text{Energy}}} \\ &= \sum \underbrace{\dot{V} \cdot y \cdot \rho}_{\substack{\text{Mass} \\ \text{Transport} \\ \text{Rate Across} \\ \text{Boundary}}} \left( \underbrace{u}_{\substack{\text{Specific} \\ \text{Internal} \\ \text{Energy}}} + \underbrace{P \cdot \hat{v}}_{\substack{\text{Specific} \\ \text{Pressure-} \\ \text{Volume} \\ \text{Energy}}} + \underbrace{\frac{U^2}{2}}_{\substack{\text{Specific} \\ \text{Kinetic} \\ \text{Energy}}} + \underbrace{g \cdot z}_{\substack{\text{Specific} \\ \text{Gravitation} \\ \text{Potential} \\ \text{Energy}}} \right) \\ &+ \underbrace{\dot{Q}}_{\substack{\text{Heat} \\ \text{Power} \\ \text{into the} \\ \text{System}}} - \underbrace{(P\dot{V})}_{\substack{\text{Rate of Work} \\ \text{Applied on} \\ \text{the surroundings}}} \end{aligned} \quad (50)$$

Several simplifications are applied to this equation:

1. Gravitational potential energy is ignored due to the short vertical distances found in Stirling engines in combination with the low density of the working fluid.
2.  $\dot{Q}$  and  $\dot{W}_b$  are combined into a single term  $\dot{E}_{ext}$  which encompasses the thermal energy conducted into the node and the boundary work or flow work acting externally on the region, based on assumption 4.

3. Kinetic energy between nodes is ignored due to the low speeds and low densities of the target engine environments.
4. Internal energy and flow work is combined into a single enthalpy term.

These simplifications result in:

$$\underbrace{\dot{E}_{kin}}_3 + \underbrace{\dot{E}_{pot}}_1 + \dot{U} = \sum \dot{V} \cdot y \cdot \rho \cdot \left( u + \frac{P}{\rho} + \frac{U^2}{\underbrace{2}_3} + \frac{g \cdot z}{\underbrace{1}_1} \right) + \dot{Q} - P \cdot \dot{V}$$

Which results in the following:

$$\dot{E}_{int} = \sum \left( \dot{V} \cdot y \cdot \rho \cdot \left( u + \frac{P}{\rho} \right) \right)_{fc} - P \cdot \dot{V} + \dot{E}_{ext} \quad (51)$$

where:  $u_{fc}$ : Specific enthalpy, measured at the face, in the current implementation this value is calculated using the Van-Alibaba flux limiter [54] between a 1<sup>st</sup> order unwinding and 4<sup>th</sup> order polynomial to prevent numerical artifacts.

$P_{fc}$ : Pressure as measured at the face, is the same as the pressure throughout the region.

$\rho_{fc}$ : Density, measured at a face. Calculated using average of upstream and downstream values.

The composite term:  $(\dot{V} \cdot y \cdot \rho)_{fc}$  is collapsed into  $\dot{m}_{fc}$  in the next section for clarity.

For ideal gases it is commonly known that the change in temperature is dependent on the internal energy.

$$\frac{dT}{dt} = \frac{1}{c_v} \frac{du}{dt} \quad (52)$$

To calculate the next temperature:

$$T_{i,t+\delta} = T_{i,t} + \frac{1}{c_{v,i,t}} \left( \frac{m_{i,t} \cdot u_{i,t} + \delta \sum \dot{m}_{f_c,i,t} \cdot \left( u_{f_c} + \frac{P_{f_c,t}}{\rho_{f_c,t}} \right) - \delta \cdot P_i \cdot \dot{V}_i + \delta \cdot \dot{E}_{ext_i}}{m_{i,t+\delta}} - u_{i,t} \right) \quad (53)$$

The equation for  $T_{i,t+\delta}$  can be inserted into equation (43).

$$\left( T_{i,t} + \frac{1}{c_{v,i,t}} \left( \frac{m_{i,t} \cdot u_{i,t} + \delta \sum \dot{m}_{f_c,i,t} \cdot \left( u_{f_c} + \frac{P_{f_c,t}}{\rho_{f_c,t}} \right) - \delta \cdot P_i \cdot \dot{V}_i + \delta \cdot \dot{E}_{ext_i}}{m_{i,t+\delta}} - u_{i,t} \right) \right) \frac{m_{i,t+\delta}}{V_{i,t+\delta}} = \frac{P_j}{R_{spec}} \quad (54)$$

Multiplying  $m_{i,t+\delta}$  into the bracketed expression yields the following equation:

$$\left( T_{i,t} \cdot m_{i,t+\delta} + \frac{1}{c_{v,i,t}} \left( m_{i,t} \cdot u_{i,t} + \delta \sum \dot{m}_{f_c,i,t} \cdot \left( u_{f_c} + \frac{P_{f_c,t}}{\rho_{f_c,t}} \right) - \delta \cdot P_i \cdot \dot{V}_i + \delta \cdot \dot{E}_{ext_i} - u_{i,t} \cdot m_{i,t+\delta} \right) \right) \frac{1}{V_{i,t+\delta}} = \frac{P_j}{R_{spec}} \quad (55)$$

Expanding the value of  $m_{i,t+\delta}$  here:

$$\left( T_{i,t} \cdot m_{i,t} + T_{i,t} \cdot \delta \sum (\dot{m}_{f_c} \cdot u_{up})_{i,t} + \frac{1}{c_{v,i,t}} \left( m_{i,t} \cdot u_{i,t} + \delta \sum \dot{m}_{f_c,i,t} \cdot \left( u_{f_c} + \frac{P_{f_c,t}}{\rho_{f_c,t}} \right) - \delta \cdot P_i \cdot \dot{V}_i + \delta \cdot \dot{E}_{ext_i} - u_{i,t} \cdot m_{i,t} - u_{i,t} \cdot \delta \sum \dot{m}_{f_c,i,t} \right) \right) \frac{1}{V_{i,t+\delta}} = \frac{P_j}{R_{spec}} \quad (56)$$

Collapsing the sums into a single sum, this is possible because they all reference the same series of faces that access the node  $i$ .

$$\frac{1}{V_{i,t+\delta}} \left( T_{i,t} \cdot m_{i,t} + \delta \sum \left( \dot{m}_{fc,i,t} \left( T_{i,t} + \frac{1}{c_{v,i,t}} \left( u_{fc,t} - u_{i,t} + \frac{P_{fc,t}}{\rho_{fc,t}} \right) \right) \right) \right) + \frac{m_{i,t} \cdot u_{i,t} - \delta \cdot P_i \cdot \dot{V}_i + \delta \cdot \dot{E}_{ext i} - u_{i,t} \cdot m_{i,t}}{c_{v,i,t}} = \frac{P_j}{R_{spec}} \quad (57)$$

Canceling out duplicate terms and reordering yields the following equation, which includes both  $T_{i,t+\delta}$  and  $m_{i,t+\delta}$ . When a set over the entire region, implicitly solves for the value of  $\dot{V}_{fc}$  when subjected to matrix inversion. Here,  $\dot{m}$  is re-expanded to expose  $\dot{V}$ .

$$\frac{1}{V_{i,t+\delta}} \left( T_{i,t} \cdot m_{i,t} + \frac{\delta \cdot (\dot{E}_{ext i} - P_i \cdot \dot{V}_i)}{c_{v,i,t}} \right) + \frac{1}{V_{i,t+\delta}} \sum (\dot{V} \cdot y \cdot \rho \cdot \delta)_{fc,i,t} \left( T_{i,t} + \frac{1}{c_{v,i,t}} \left( u_{fc,t} - u_{i,t} + \frac{P_{fc,t}}{\rho_{fc,t}} \right) \right) = \frac{P_j}{R_{spec}} \quad (58)$$

The exact same procedure can be conducted on the right-hand side to produce the following equation.

$$\begin{aligned} & \frac{1}{V_{i,t+\delta}} \left( T_{i,t} \cdot m_{i,t} + \frac{\delta \cdot (\dot{E}_{ext i} - P_i \cdot \dot{V}_i)}{c_{v,i,t}} \right) \\ & + \frac{1}{V_{i,t+\delta}} \sum (\dot{V} \cdot y \cdot \rho \cdot \delta)_{fc,i,t} \left( T_{i,t} + \frac{1}{c_{v,i,t}} \left( u_{fc,t} - u_{i,t} + \frac{P_{fc,t}}{\rho_{fc,t}} \right) \right) \\ & = \frac{1}{V_{j,t+\delta}} \left( T_{j,t} \cdot m_{j,t} + \frac{\delta \cdot (\dot{E}_{ext j} - P_j \cdot \dot{V}_j)}{c_{v,j,t}} \right) \\ & + \frac{1}{V_{j,t+\delta}} \sum (\dot{V} \cdot y \cdot \rho \cdot \delta)_{fc,j,t} \left( T_{j,t} + \frac{1}{c_{v,j,t}} \left( u_{fc,t} - u_{j,t} + \frac{P_{fc,t}}{\rho_{fc,t}} \right) \right) \end{aligned} \quad (59)$$

The terms of the full equation in for form  $A\dot{\mathbf{V}} = \mathbf{b}$  are as follows:

Values of each element of column vector  $\mathbf{b}$ . Here, each entry is for a face. Each face has two nodes a node which a positive flow exits and a node which a positive flow enters, these are presented as  $i_k$  and  $j_k$  respectively – where  $k$  is the face index.



$$b_k = F(i, j) = \frac{1}{\delta V_{i,t+\delta}} \left( T_{i,t} \cdot m_{i,t} + \frac{\delta \cdot (\dot{E}_{ext_i} - P_i \cdot \dot{V}_i)}{c_{v_{i,t}}} \right) - \frac{1}{\delta V_{j,t+\delta}} \left( T_{j,t} \cdot m_{j,t} + \frac{\delta \cdot (\dot{E}_{ext_j} - P_j \cdot \dot{V}_j)}{c_{v_{j,t}}} \right)$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N_{fcs}} \end{bmatrix} = \begin{bmatrix} F(i_1, j_1) \\ F(i_2, j_2) \\ \vdots \\ F(i_{N_{fcs}}, j_{N_{fcs}}) \end{bmatrix} \quad (60)$$

where:  $N_{fcs}$ : The number of faces being solved.

Values for each row of matrix  $A$ , whereas the column is defined by the face associated with the value of  $\dot{V}$ . Each row contains all the inflows and outflows for a pair of nodes  $i$  and  $j$ .

$$g(i, fc) = \begin{cases} \frac{\delta}{V_{i,t+\delta}} (y \cdot \rho)_{fc,i,t} \left( T_i + \frac{1}{c_{v_i}} \left( u_{fc} - u_i + \frac{P_{fc}}{\rho_{fc}} \right) \right)_t & \text{if node } i \text{ and face } fc \text{ touch} \\ 0 & \text{otherwise} \end{cases}$$

$$A = \begin{bmatrix} g(j_1, 1) - g(i_1, 1) & g(j_1, 2) - g(i_1, 2) & \cdots & g(j_1, N_{fcs}) - g(i_1, N_{fcs}) \\ g(j_2, 1) - g(i_2, 1) & g(j_2, 2) - g(i_2, 2) & \cdots & g(j_2, N_{fcs}) - g(i_2, N_{fcs}) \\ \vdots & \vdots & \ddots & \vdots \\ g(j_{N_{fcs}}, 1) - g(i_{N_{fcs}}, 1) & \cdots & \cdots & g(j_{N_{fcs}}, N_{fcs}) - g(i_{N_{fcs}}, N_{fcs}) \end{bmatrix} \quad (61)$$

There is a simplification to this when the region includes the environment, which is a node with constant pressure.

$$\left( T_{i,t} \cdot m_{i,t} + \frac{\delta \cdot (\dot{E}_{ext_i} - P_i \cdot \dot{V}_i)}{c_{v_{i,t}}} \right) + \delta \sum_{\text{faces in "i"}} \dot{V}_{fc,t} \left( y_{fc,i} \cdot \rho_{fc} \left( T_i + \frac{1}{c_{v_i}} \left( u_{fc} - u_i + \frac{P_{fc}}{\rho_{fc}} \right) \right) \right)_t$$

$$= \frac{V_{i,t+\delta} \cdot P_{env}}{R_{spec}} \quad (62)$$

$$b_k = F(i, j) = \frac{1}{\delta} \left( \frac{V_{i,t+\delta} \cdot P_{env}}{R_{spec}} - T_{i,t} \cdot m_{i,t} - \frac{\delta \cdot (\dot{E}_{ext_i} - P_i \cdot \dot{V}_i)}{c_{v_{i,t}}} \right) \quad (63)$$

$$g'(i, fc) = \begin{cases} \dot{V}_{fc,t} \left( y_{fc,i} \cdot \rho_{fc} \left( T_i + \frac{1}{c_{vi}} (u_{fc} - u_i) \right) \right) & \text{if node } i \text{ and face } fc \text{ touch} \\ 0 & \text{otherwise} \end{cases} \quad (64)$$

$$A = \begin{bmatrix} g'(i_1, 1) & g'(i_1, 2) & \cdots & g'(i_1, N_{fcs}) \\ g'(i_2, 1) & g'(i_2, 2) & \cdots & g'(i_2, N_{fcs}) \\ \vdots & \vdots & \ddots & \vdots \\ g'(i_{N_{fcs}}, 1) & g'(i_{N_{fcs}}, 2) & \cdots & g'(i_{N_{fcs}}, N_{fcs}) \end{bmatrix}$$

The term  $\dot{E}_{ext}$  represents the components of energy exchange to the node that can be approximated using a previous value of the Reynolds number as between iterations the flow rate changes are assumed to be small. This is defined:

$$\dot{E}_{ext_i} = \sum_{Gas-Solid} (y_i \cdot C_{cond} \cdot \Delta T)_{fc} + \sum_{Gas-Gas} (y_i \cdot C_{cond} \cdot \Delta T)_{fc} \quad (65)$$

### 3.3.3 Verifying as a Polytropic Process

Stirling engines are a polytropic process, meaning that they exist anywhere along the spectrum of processes. The polytropic index is a representation of the type of process that is going on.

$$n = (1 - \gamma) \frac{\delta q}{\delta w} + \gamma \quad (66)$$

where:  $n = 0$ : Isobaric Process     $n = \gamma$ : Isentropic Process

$n = 1$ : Isochoric Process     $n = \infty$ : Isochoric Process

The work from a polytropic process is equal to the following:

$$w = \frac{R(T_2 - T_1)}{1 - n} \quad (67)$$

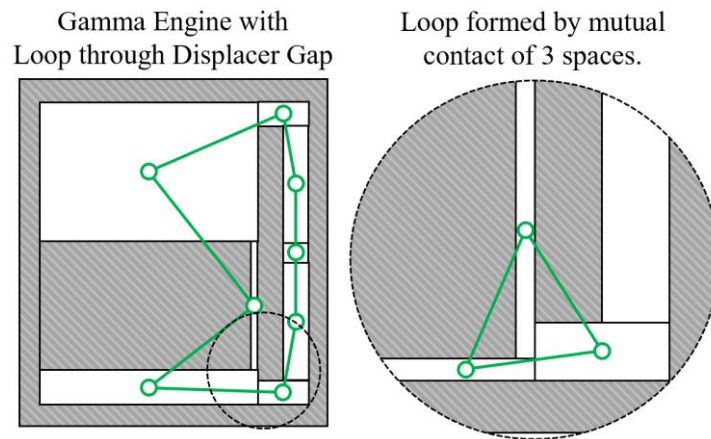
$$w \left( 1 - (1 - \gamma) \frac{\delta q}{\delta w} - \gamma \right) = (1 - \gamma)(w - q)$$

$$\frac{R}{1-\gamma}(T_2 - T_1) = \frac{c_p - c_v}{1 - \frac{c_p}{c_v}}(T_2 - T_1) = -c_v(T_2 - T_1)$$

$$q - w = c_v(T_2 - T_1)$$

This now aligns with the mathematics developed in the previous section; thus, the math should have no problem simulating the polytropic processes of a Stirling engine.

### 3.3.4 Considering Loops



**Figure 3.2: Examples of common loops found in Stirling engines.**

In systems that contain loops, as illustrated in Figure 3.2, it is observed that for each independent loop added wherein the number of these loops is the same as in mesh analysis for solving circuits [55], a new equation is required. A loop exists anywhere a gas particle can take more than one path between any node and any other node. A common example, found in gamma type engines, is the thin boundary around the displacer, which allows some gas to not pass through the heat exchangers when traveling between compression and expansion spaces. This space is often minimized by close running seals such that it could be modelled by a narrow gas path or even a leakage component.

Each loop is represented by a characteristic volumetric flow rate, called a loop flow rate. There is one characteristic flow rate for each additional equation. The extra rows in the matrix will be used to assign a value to the loop flow rates, producing, out the infinite set of solutions from the indeterminate matrix, the solution that has those flow rates. This is done by having the row be all zeros except for a 1 in the corresponding face's column. The corresponding entry in  $\mathbf{b}$  will be the

value of this loop flow rate. The following discussion will outline how these loop flow rates are determined.

Considering pressure to be akin to electric potential and a pressure drop to be akin to voltage in an analogous circuit then over the entire loop the pressure drop should be zero:

$$\oint \Delta P = 0 \quad (68)$$

Calculating pressure drop using the Darcy-Weisbach equation [31].

$$\Delta P_{fc} = \left( f(N_{Re}) \frac{L}{d_h} \frac{\rho \cdot |U| \cdot U}{2} \right)_{fc} \quad (69)$$

We can modify this term by considering the face's movement relative to the geometry around it. Hypothetically a translating face may have a gas flow rate that is zero across it but may still develop a pressure drop because it is moving relative to the walls. The faces velocity is precalculated in such a way that the product of the angle-dependent face velocity factor ( $F_V$ ) and the instantaneous rotational speed equals the instantaneous face velocity.

$$U' = U + U_{fc} = U + F_V \omega$$

$$\Delta P_{fc} = \left( f(N_{Re}) \frac{L}{d_h} \frac{\rho \cdot |U'| \cdot U'}{2} \right)_{fc} \quad (70)$$

This results in the following equation for each loop.

$$\sum \left( y \frac{f(N_{Re})}{2} \frac{L}{d_h} \rho \cdot |U'| \cdot U' \right)_{fc} = 0 \quad (71)$$

The equation introduces a non-linearity in the form of a squared velocity term as well as the friction factor's dependence on Reynold's number, which is also dependent on velocity. This equation is therefore solved by a root-finding scheme which sets the value of the set of characteristic loop velocities until pressure drop across all loops is equal to zero.

The derivative of this function concerning the change in the volume flow of a characteristic volumetric flow rate is equal to the following:

$$\sum_{\substack{\text{For each} \\ \text{face in} \\ \text{loop}}} \left( \frac{\partial \dot{V}_{fc}}{\partial \dot{V}_{loop}} \frac{\partial}{\partial \dot{V}_{fc}} \left( \frac{y \cdot L \cdot \rho}{2d_h \cdot A^2} \right)_{fc} f(N_{Re,fc}) \cdot |\dot{V}_{fc}| \cdot \dot{V}_{fc} \right) = \frac{\partial}{\partial \dot{V}_{loop}} \oint \Delta P \quad (72)$$

where:  $\dot{V}_{loop}$ : Characteristic flow rate for loop, selected to be independent of all other loops.

Manipulation of the expression gives:

$$\frac{1}{2} \sum_{\substack{\text{For each} \\ \text{face } k \text{ in} \\ \text{loop } i}} \frac{\partial \dot{V}_k}{\partial \dot{V}_{loop}} \left( \frac{y \cdot \rho \cdot L \cdot |U|}{d_h \cdot A} \right)_k \left( \frac{U}{|U|} N_{Re} \frac{\partial f(N_{Re})}{\partial N_{Re}} + 2f(N_{Re}) \right)_k = \frac{\partial}{\partial \dot{V}_{loop}} \oint \Delta P \quad (73)$$

where:  $\frac{\partial f(N_{Re})}{\partial N_{Re}}$ : The change in the Darcy friction factor with respect to a change in Reynold's number.

The intent in finding the derivative of the loop pressure drop is to use it as part of the Newton-Raphson gradient descent algorithm. This algorithm makes incremental updates to a set of parameters – in this case a set of velocities called generally here as  $X$  – seeking the set of velocities at which the pressure crosses or equals zero. Each iteration step is as follows:

$$X_{n+1} = X_n - J_n^{-1} \cdot F_n \quad (74)$$

where:  $X_n$  = The vector of independent variables at iteration  $n$ .

$F_n$  = The vector of dependent variables at iteration  $n$ .

$J_n$  = The Jacobian of  $F$  concerning inputs  $X$  at iteration  $n$ .

Here the independent variable is the loop volume flow rate, which is a single flow rate that lies in and only in the loop of interest, to minimize correlation during solving. The dependent variable is the loop pressure drop. The Jacobian is constructed from the derivatives of the loop pressure drop; the Jacobian formula is given below:

$$J = \begin{bmatrix} \frac{\partial F_1(X_n)}{\partial x_1} & \dots & \frac{\partial F_1(X_n)}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_M(X_n)}{\partial x_1} & \dots & \frac{\partial F_M(X_n)}{\partial x_N} \end{bmatrix} \quad (75)$$

where: 
$$\frac{\partial F_i}{\partial x_j} = \frac{1}{2} \sum_{\substack{\text{For each} \\ \text{face } k \text{ in} \\ \text{loop } i}} \frac{\partial \dot{V}_k}{\partial \dot{V}_{loop}} \left( \frac{y \cdot \rho \cdot L \cdot |U|}{D_h \cdot A} \right)_k \left( \frac{U}{|U|} N_{Re} \frac{\partial f(N_{Re})}{\partial N_{Re}} + 2f(N_{Re}) \right)_k$$

While the selected velocities are defined by this algorithm, all other velocities are still dependent on the matrix defined in section 3.3.2. Additionally, the derivative with respect the loop flow rate for all other flow rates can be obtained through querying the corresponding column in the inverted matrix by (which is precalculated as only  $\mathbf{b}$  is modified as part of the loop solving scheme).

### 3.3.5 Considering Flow Losses

Due to the uniform pressure assumption all nodes have the same pressure. In the real system, however, the flow loss would enact its effect by a rise or fall of the pressure acting on the piston faces. This must be considered to some degree as the work of some pistons, namely the displacer, is solely a function of this loss. This loss is solved by calculating the pressure in all nodes such that the total pressure is equal to the  $P_{region}$  defined in equation (77), and that the pressure drop between adjacent nodes matches equation (70). This results in a modified set of pressures that is an estimate of the pressure drop, this will be particularly effective when the difference in pressure is all that is required, such as the pressure difference across a displacer.

For each face, using the Darcy-Weisbach equation [31]:

$$P_1 - P_2 = \frac{1}{2} \left( K + \frac{N_f \cdot L}{d_h} \right) \rho \cdot |U_{fc}| \cdot U_{fc} \quad (76)$$

For all nodes:

$$\sum P_{nd} \cdot V_{nd} = P_{region} \cdot V_{region} \quad (77)$$

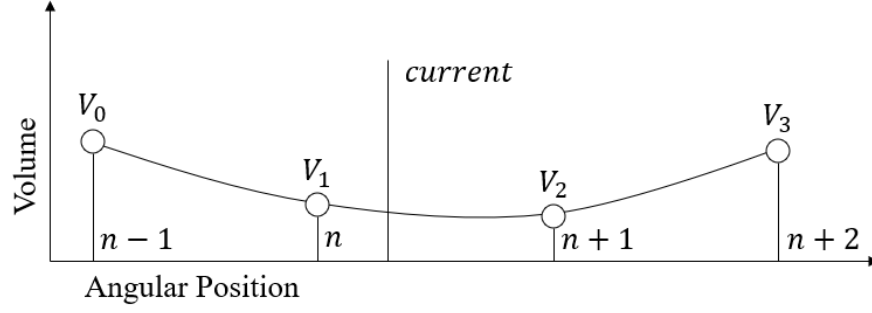
The matrix in the style of  $Ax = b$ :

$$\begin{bmatrix} 0 & \dots & \pm 1 & \dots & \mp 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ V_1 & V_2 & \dots & \dots & \dots & \dots & V_N \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_N \end{bmatrix} = \begin{bmatrix} \left( \frac{1}{2} \left( K + \frac{N_f \cdot L}{d_h} \right) \rho \cdot |U| \cdot U \right)_1 \\ (\dots)_2 \\ \vdots \\ (\dots)_M \\ P_{region} \cdot V_{region} \end{bmatrix} \quad (78)$$

### 3.3.6 Smooth Property Changes

For many properties in the model, their value changes relative to the crank angle. These properties called “ $x$ ” may be linearly interpolated between crank angle or, for better results, they can be interpolated using a cubic spline. The cubic spline provides several benefits, it ensures changes, particularly in the derivative are smooth which improves the convergence of sensitive components such as the loop solving in section 3.3.3. The following discussion outlines the interpolation procedure, which considers the gradual change in speed of the engine between angular positions.

Taking the volume as an example, the instantaneous change in volume is interpolated via a cubic spline of the 4 points around the current simulation region. This cubic spline is derived such that the 2<sup>nd</sup> order central difference derivative at the start and endpoint of the section is the same, this avoids discontinuities in rate when transitioning between angular increments. The locations of the interpolation points are provided in Figure 3.3.



**Figure 3.3: Angular locations of  $V_0$  through  $V_3$**

$$A \cdot t'^3 + B \cdot t'^2 + C \cdot t' + D = V(t')$$

$$3A \cdot t'^2 + 2B \cdot t' + C = \frac{dV}{dt}(t')$$

where:

$$t' = \frac{t - t_n}{t_{n+1} - t_n} \quad \Delta t = t_{n+1} - t_n$$

$$\left(\frac{dV}{dt}\right)_1 = \frac{(V_2 - V_0)}{2 \frac{\theta_{inc}}{\omega_1}} \quad \left(\frac{dV}{dt}\right)_2 = \frac{(V_3 - V_1)}{2 \frac{\theta_{inc}}{\omega_2}} \quad (79)$$

$$A = \Delta t \left(\frac{dV}{dt}\right)_1 + \Delta t \left(\frac{dV}{dt}\right)_2 - 2(V_2 - V_1)$$

$$B = -2\Delta t \left(\frac{dV}{dt}\right)_1 - \Delta t \left(\frac{dV}{dt}\right)_2 + 3(V_2 - V_1)$$

$$C = \Delta t \left(\frac{dV}{dt}\right)_1$$

$$D = V_1$$

A similar procedure is applied for all dynamic properties. In addition to the cubic spline interpolation, each interpolated value is clamped to be above zero depending on the physical realism of a negative value occurring. Some properties, such as volume, which are featured on the



denominator of a calculation are clamped to be above some small tolerance value to avoid the small time-steps or errors that occur with smaller or zero values.

## 3.4 Turbulence

---

### 3.4.1 Open Channels Flows

Turbulence here is adapted from the implementation found in SAGE by Gedeon [35]. This implementation represents turbulence in open channel flows as a conserved property bounded between 0 and 1. Gedeon, through development of SAGE, observed that for oscillating flows while the flow was turbulent it exhibited properties like that of steady-state turbulent flows and while it was observed to behave in a laminar manner, its properties mimicked steady-state laminar flows. Thus, this factor,  $\tau$ , attempts to predict the altered transitions between laminar and turbulent. In practice, this factor serves as a weight between the laminar and turbulent values of the Darcy friction factor ( $N_f$ ) and the Nusselt Number ( $N_{Nu}$ ). The associated conservation equation is as follows [35]:

$$\frac{\partial(\tau \cdot A)}{\partial t} + \frac{\partial(\tau \cdot U \cdot A)}{\partial x} = \tau_{gen} \quad (80)$$

The weighting factor is carried with the flow and its generation follows a few rules.

Flows in the wake of large geometrical features such as those representing bends or area changes always receive a fully turbulent flow corresponding to a value of  $\tau = 1$ .

Generation of turbulence within nodes occurs exclusively when the flow condition is above  $Re_{crit}$  represented below [35]:

$$N_{Re_{crit}} = 200 \max\left(\frac{\sqrt{N_{Va}}}{0.075 + 0.112\omega \cdot (t - t_0)}, 11.5\right) \quad (81)$$

The Valensi number ( $N_{Va}$ ) represents the ratio of oscillatory fluid inertia to viscosity and is defined as:

$$N_{Va} = \frac{\rho \cdot \omega \cdot D_h^2}{\mu} \quad (82)$$

The value  $t_0$  represents the time at which the flow was at a velocity of zero. The function in the denominator represents the growth of the momentum layer under sinusoidal oscillations, which is close enough for this model.

The generation or decay term for turbulence weighting factor is represented as [35]:

$$\frac{\partial \tau}{\partial t} = \frac{\omega \delta}{N_{Va}} \begin{cases} 0.008 N_{Re} \cdot (1 - \tau) & N_{Re} > N_{Re\,crit} \\ -0.25 N_{Re\,laminar} \cdot \tau^{3/2} & N_{Re} \leq N_{Re\,crit} \end{cases} \quad (83)$$

Outflows and inflows induce changes in turbulence weighting equal to the following:

$$\frac{\partial \tau_i}{\partial t} = \frac{\dot{m}_{fc}}{m_i} \begin{cases} (\tau_{fc} - \tau_i) & \dot{V}_{fc} \text{ is entering node } i \text{ through a pipe} \\ (-\tau_{fc} + \tau_i) & \dot{V}_{fc} \text{ is leaving node } i \\ (1 - \tau_i) & \dot{V}_{fc} \text{ is entering node } i \text{ through a minor loss} \end{cases} \quad (84)$$

### 3.4.2 Matrix Flows

Flow within the fine geometry of features such as regenerators or heat exchangers is considered a Matrix flow. Such flows develop rapidly and under oscillatory conditions are very close to steady-state due to the domination of the viscosity term. Therefore, the weighting factor between a turbulent and laminar value is as follows [35]:

$$F(N_{Re}^*) = \begin{cases} F_{lam} & N_{Re}^* \leq 0 \\ w \cdot F_{lam} + (1 - w) \cdot F_{turb} & 0 < N_{Re}^* < 1 \\ F_{turb} & 1 \leq N_{Re}^* \end{cases}$$

$$\text{where: } N_{Re}^* = \frac{N_{Re} - 2,300}{4,000 - 2,300} = \frac{N_{Re} - 2,300}{1,700} \quad (85)$$

and

$$w = N_{Re}^{*2} \cdot (3 - 2N_{Re}^*)$$

The incorporation of the smooth step function is an improvement over the discontinuous curve applied in SAGE. This function applies to both the Darcy friction factor and the Nusselt number.

### 3.4.3 Variable Volume Spaces

Turbulence within variable volume spaces is generated by inflows and reduced by outflows and remaining turbulence is decayed as a function of time. An important distinction is that turbulence in variable volume spaces is not measured by the weighting factor  $\tau$  but is rather a specific turbulent kinetic energy  $\kappa$ .

The turbulence produced by an inflow is a function of the entrance velocity [35]:

$$\frac{\partial \kappa_i}{\partial t} = \frac{\dot{m}_{fc}}{m_i} \begin{cases} \left( \frac{1}{2} \left( \frac{\dot{V}_{fc}}{A_{fc}} \right)^2 - \kappa_i \right) & \dot{V}_{fc} \text{ is entering node "i"} \\ 0 & \dot{V}_{fc} \text{ is leaving node "i"} \end{cases} \quad (86)$$

where:  $\kappa_i$ : The specific turbulent kinetic energy for node  $i$

Flows that leave a variable volume space do not change the specific turbulent kinetic energy. The entrance velocity turns its kinetic energy into the swirling motion observed in open chambers. This swirling energy is decayed according to the following rule, derived by Gedeon [35], [56] and presented by Cantelmi [57].

$$\frac{\partial m\kappa}{\partial t}_{decay} = -\frac{C_D}{0.021D_h} m \cdot \kappa^{\frac{3}{2}} \quad (87)$$

where:  $C_D$ : the turbulent energy dissipation constant for tube flow = 0.08

$0.021D_h$ : the spatial averaged length scale for tube flow and

$\kappa$ : is the specific turbulent kinetic energy

Through manipulations:

$$\frac{\partial \kappa_i}{\partial t} = -\frac{5.8 \kappa_i^{\frac{3}{2}}}{D_h} - \frac{\kappa_i}{m_i} \frac{\partial m_i}{\partial t} \quad (88)$$

Turbulence in variable volume spaces affects the values of the effective conduction and viscosity factors, also extracted from Gedeon [56]. These overwrite the default values of these properties and result in higher wall conduction and higher friction losses in variable volume spaces.

$$k_{eff} = k + 0.021 \rho \cdot d_h \cdot c_p \cdot \sqrt{\kappa} \quad (89)$$

$$\mu_{eff} = \mu + 0.021 \rho \cdot d_h \cdot \sqrt{\kappa}$$

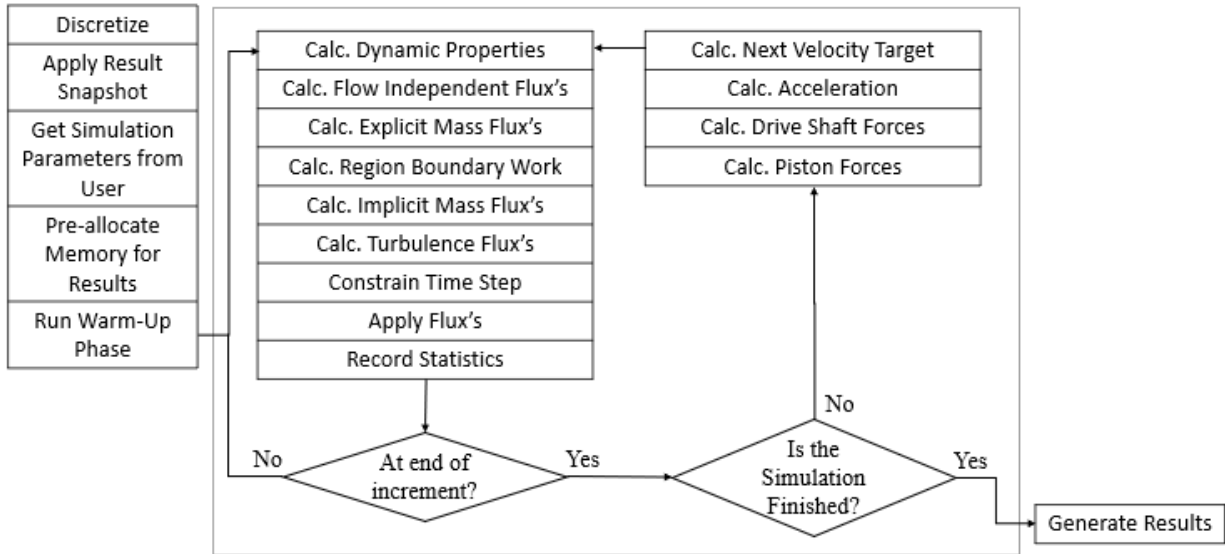
### 3.5 Chapter Conclusions

---

Through this chapter the equations needed to solve for the energy exchange due to conduction was presented for all possible node combinations. Following this, the mathematics needed to solve for the volumetric flow rates of ideal gases with volume and energy changes was outlined. Then, the expansions applied when the network contains loops were discussed, allowing the code to solve arbitrary networks. The pressure correction due to flow friction was provided, allowing the uniform pressure assumption to still support the effect of flow losses across pistons. The cubic spline used for smooth property transitions and handling of angular accelerations was presented. The chapter concluded with the presentation of the turbulence handling mathematics derived from SAGE's [35] implementation. These theoretical formulations, lead into the next chapter where they are implemented into algorithms which decompose the interactable blocks into the network, and finally into a solution.

# CHAPTER 4. SIMULATION

During the solving phase MSPM follows the flow structure in Figure 4.1 below, each of the actions are discussed in the following sections. All the project code can be found in Appendix G.



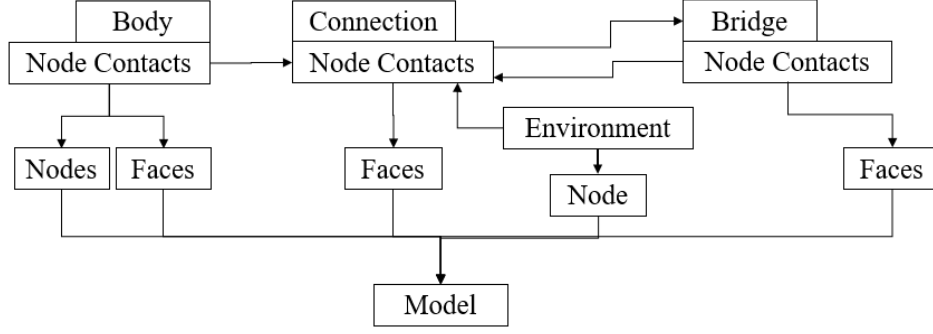
**Figure 4.1: Process structure of the simulation loop, elements inside of the box are repeated until the simulation has timed out or converged.**

## 4.1 Discretization and Conditioning

Many steps are involved in creating and conditioning the network for solving, these are outlined in the following sections.

### 4.1.1 Discretize all Components and Collect Discrete Elements

The model discretizes components in a particular order, which is shown in Figure 4.2. The discretization of each element is discussed in 0.



**Figure 4.2: Flow of information during discretization**

Bodies are discretized first, which generates the node contacts that are used by connections. Then the environment is discretized, and its singular node is distributed among the boundary connections. Following this, bridges are discretized as they will provide their own node contacts as well as modify the existing ones. Lastly the connections are discretized, which pair up the node contacts to form faces.

#### **4.1.2 Decimate Nodes Based on Size**

After the mesh is produced, a post processing step eliminates particularly thin nodes from the calculation. This step prevents the existence of nodes that would otherwise produce an unreasonably small timestep, dragging the entire simulation down for negligible gains in accuracy. Small nodes are merged into larger nodes if the timestep, as defined by the Fourier number, falls below a certain threshold. This timestep is constructed as follows:

$$\delta = N_{Fo} \frac{\left(\frac{dU}{dT}\right)_{nd}}{\left(\frac{d\dot{Q}}{d(\Delta T)}\right)_{fc}} = N_{Fo} \frac{\text{Heat Capacity}}{\text{Conductance}} \quad (90)$$

where:  $N_{Fo}$ : The maximum allowable Fourier number.

$\left(\frac{dU}{dT}\right)_{nd}$ : The derivative of total internal energy vs temperature of the node. Equivalent to the heat capacity of the material found in the node.

$\left(\frac{d\dot{Q}}{d(\Delta T)}\right)_{fc}$ : The derivative of heat flow rate across a face of the node verses the temperature difference across it. Equivalent to the conductance of the face.

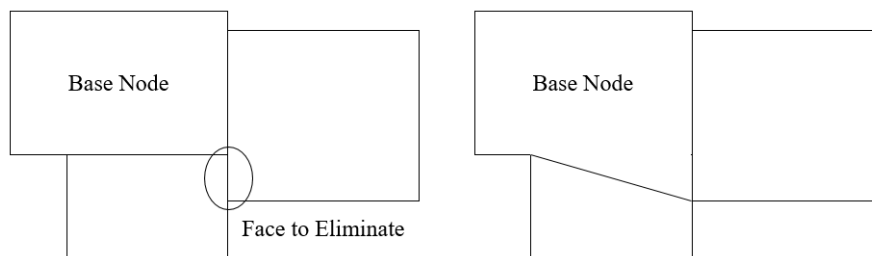
The larger node then takes on the unmodified faces and appends the properties of the smaller node. This has a significant impact on the performance of the model at the loss of some small elements.

### 4.1.3 Assign Minor Loss Coefficients

A minor flow loss coefficient ( $K$ ) applies to a face when the flow area changes between the first and second nodes. Faces in which this applies have two such coefficients a forward ( $K_{12}$ ) and a backward ( $K_{21}$ ) minor loss coefficient, corresponding to the two possible flow directions. These can be applied using the default equation presented in section 2.2.2 or can be overridden by the custom minor loss coefficient component.

### 4.1.4 Decimate Triads

The number of loops is a major factor in the run speed of the simulation. Oftentimes junctions occur between 3 nodes, which form micro-loops, increasing the load on the solver. The decimate triad's function finds such triads of nodes and assesses them, looking at the relative size of the faces. The node with the largest contact to the other two nodes becomes the base node and replaces the connection that lies between the two other nodes. Figure 4.3 below illustrates this process. The final area and loss coefficients of the non-eliminated faces are equal to the area weighted average of the values.



**Figure 4.3: Example of a triad elimination action: faces are selected based on their relative size, the remaining 2 faces are then modified to compensate.**

#### **4.1.5 Assign Indexes to all Elements**

Though it is convenient to work with discrete elements as objects (as in object-oriented programming), it is much faster to access vectorized elements in MATLAB. Therefore, a unique index is assigned to each node and face in preparation for vectorization. The indexes are assigned such that when sorted in ascending order the gas nodes appear before the environment nodes, which are followed by solid nodes. Similarly, gas faces lead mixed faces, which lead solid faces.

#### **4.1.6 Vectorize Node and Face Properties**

Using the previously defined index, nodes and faces insert their physical properties – which are defined in 0 – into arrays that are more computationally efficient to access than their current object-oriented format. Here also the dynamic properties are referenced and stored in a master array. Due to previous sorting the resulting property arrays are as small as possible as the most memory demanding elements are listed first.

Specialized elements such as leaks, shear and pressure contacts remain in their object-oriented format as they are typically few in number.

#### **4.1.7 Determine Maximum Solid-Conduction Timestep**

A loop through all the Solid Nodes calculates the absolute minimum allowable time step per angular increment based on the maximum Fourier number ( $N_{Fo}$ ) defined in equation (90). If the conductance changes due to a sliding boundary the most conservative value (smallest timestep) is retained, this is derived from the definition of the Fourier number.

#### **4.1.8 Determine the Conduction and Transportation Vectors**

With irregular grids, one of the challenges is efficiently calculating the change in the nodal properties. This is a challenge because a node can have many faces and finding them for each node is a tedious task. even if a list of face indexes were obtained this would constitute many short and therefore inefficient vector operations. As MATLAB is very fast at large vector calculations, the goal is to define as many large vector calculations as possible, because the conduction values can change every frame it would be a challenging pursuit to do this efficiently by constructing a matrix.



The prework pseudo-code below produces sets of faces that can be added to the nodal values without overlapping each other. The result of this operation is a vector of vectors which contains parallel sets of 3 vectors, the first contains the sign of the operation, the second contains the vector of nodes on which the operation is performed and the third indicates the face. In the case of a grid mesh, this would produce at most 8 sets, no matter how large the grid was. (one for each face of a node, one for each sign).

---

*Inputs: List of all Nodes and Faces*

**Step 1: Initialize Vectors**

*CondNds = vector containing the indexes of all Nodes*

*CondFcs = vector containing the indexes of all Faces*

*Nds1 = vector containing only the first node listed by each Face*

*Nds2 = vector containing only the second node listed by each Face*

**Step 2: Make backups of Nds1 and Nds2**

*CondNds1 = Nds1 --- Duplicate of Nds1*

*CondNds2 = Nds2 --- Duplicate of Nds2*

**Step 3: Exclude nodes that are constant values (surrounding, source, etc.)**

*Nds1(indexes where the Node has static properties) = 0*

*Nds2(indexes where the Node has static properties) = 0*

**Step 4: Calculate how many sets of vector to produce, based on the maximum number of times any one node is referenced.**

*if any(Nds1 are not equal to 0)*

*x = mode of Nds1 excluding 0's*

*N1 = number of times x occurs in Nds1*

*else N1 = 0*

*if any(Nds2 are not equal to 0)*

*x = mode of Nds2 excluding 0's*

*N2 = number of times x occurs in Nds2*

*else N2 = 0*

**Step 5: Create vectorized references**

*i = 1*

*CondVectors = vector of vectors of length = 3\*(N1 + N2)*

*if N1 is not 0*

*for k = 1 to N1 by stepping 1*

*N = count of unique values in Nds1 excluding 0's*

*CondVectors[i] = -1*

*CondVectors[i+1] = vector of 0's of length = N*

*CondVectors[i+2] = vector of 0's of length = N*

*Counter = 1*

*for x = 1 to length(Nds1) by stepping 1*

*if Nds1[x] > 0*

*if (Nds1[x] is not already in CondVectors[i+1])*

*CondVectors[i+1][Counter] = Nds1[x]*

*CondVectors[i+2][Counter] = x*

*Nds1[x] = 0*

*Counter = Counter + 1*

*i = i + 3*

---

**Step 6: Repeat Step 5 for N2 and Nds2**

```
return CondVectors, CondNds1, CondNds2, CondFcs, CondNds
```

**-- Usage (as found in simulation.m)**

Using CondFcs, update all conductances and calculate energy flows using CondNds1 and CondNds2

```
for i = 1 to length(CondVectors) - 2 by stepping 3
    Qnode[CondVectors[i+1]] = Qnode[CondVectors[i+1]] +
        CondVectors[i]*Qface[CondVectors[i+2]]
```

---

The equivalent data structure for transportation is obtained by the same method above but for solely the gas-gas faces and nodes. This is used to determine the nodal Reynold's number, which is simply the area weighted average of the Reynold's number of its participating faces.

### **4.1.9 Establish Gas Regions**

Given that the model can be defined arbitrarily, the interconnected gas spaces within the engine, henceforth called regions, must be determined automatically. A region is distinct from another gas space if at any point in the cycle it is completely cut off from that other space; thus, allowing a build up in pressure. Such spaces are found via a recursive space filling algorithm starting at a root node. The output is a vector with region indexes for each gas node including the environment node. The pseudo-code below outlines the algorithm:

---

*Inputs: List of all Nodes*

**Outer Loop: Loop through all the nodes to find ones that are not grabbed by the recursive function. Any new ones indicate an unreached region.**

```
n = 0
region = vector of 0's of length = number of Gas & Environment Nodes
for each Nd in Nodes
    if the Nd is a Gas Node and it does not have an assigned region
        n = n + 1
        region = PropegateRegion(Nd, region, n)
    if all nodes have a region then exit for each Nd loop
return region
```

**Inner Loop: Add the current node as a region member, look for unreached neighbors, call the function using them as a starting point**

*Sub Function PropegateRegion(Nd, region, n)*

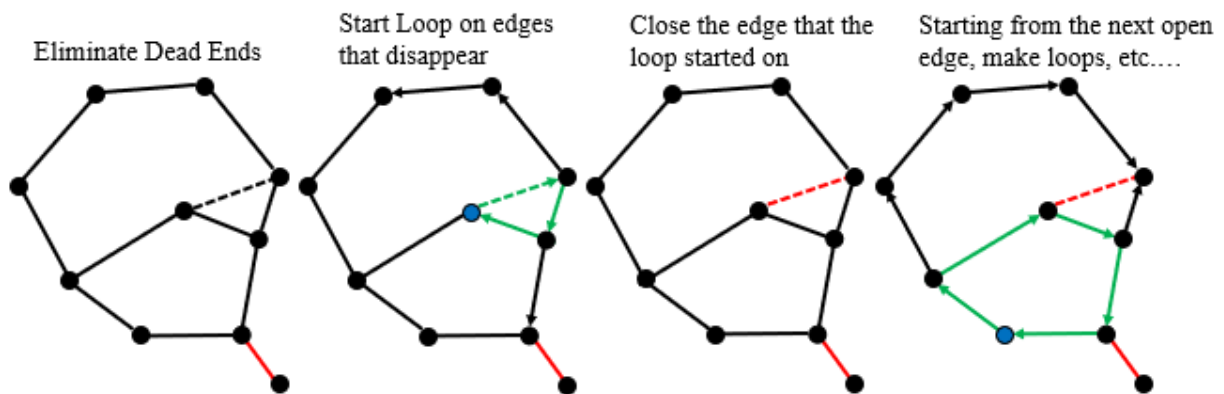
```
if region[Nd.index] is 0 ... Nd does not have a region
    region[Nd.index] = n
    for each face associated with Nd
        if the face is a gas face and never completely closes
            node = other node of face relative to Nd
            region = PropegateRegion(node, region, n)
return region
```

---

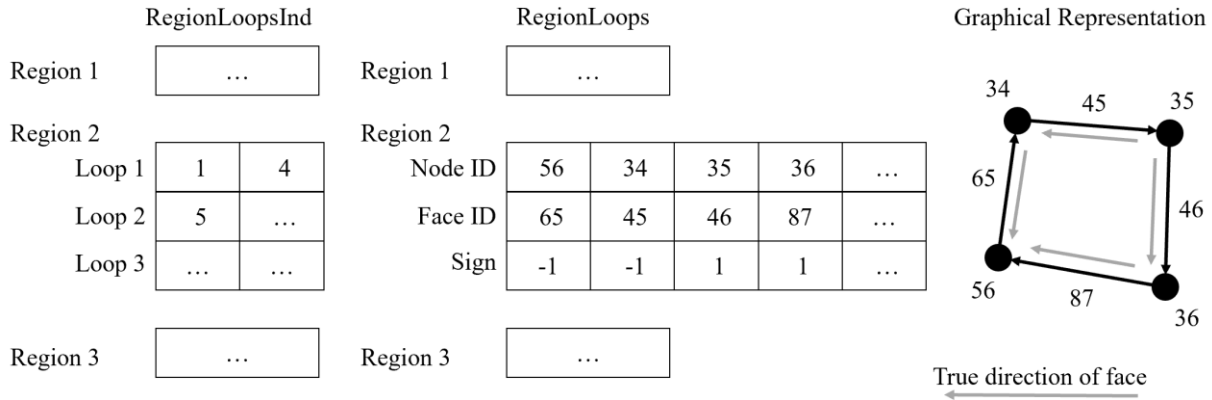
Post-processing of the output vector provides a list of nodes for each region. Faces are assigned to a region if both member nodes are within the same region. Also, the region that contains the environment node is treated differently during simulation, so it is indicated with a Boolean value if the current region is an environment region. It is treated differently because its pressure does not change, resulting in a simplified solution.

#### 4.1.10 Find Loops within each Region

Loop finding is another area where a specialized algorithm is required that is composed of several steps. The first step eliminates dead-ends. Then faces that are within the region but are closed for parts of the cycle are found. Followed by the actual loop finding algorithm. The final output of this series of algorithms is two data structures. The first, **RegionLoops**, is a vector of arrays with an array for each region that has 3 rows, the first for a node index, the second for a face index and the third for a sign. The second, **RegionLoopsInd**, is a vector (corresponding to each region) of arrays that contains pairs of numbers, in columns, indicating the start and end column in **RegionLoops** for each distinct loop in the region. This data structure is shown in Figure 4.5. This algorithm is described in Figure 4.4 and also by the following code.



**Figure 4.4: Loop finding algorithm illustration. (dashed) face that has an area of zero at any point of the cycle. (red) eliminated face. (blue node) starting point for algorithm. (green) discovered loop.**



**Figure 4.5: Loop data structure and graphical representation**

---

**-- Elimination of dead-ends**

A node is a dead end if they have only one gas face, or they have two gas face but one or both leads to a dead-end node. Mark these and all their faces as closed in the closed\_edge array.

```

open = empty
for i is 1 to the number of regions
  -- Region Enclosed Closing Edge Discovery
  Collect faces whose area goes to zero at some point but both nodes lie
  within the same region. Add these to the "holes" array and mark them
  closed.

  -- Loop Finding
  visited_edge = closed_edge
  lequ = 1
  lcount = 0
  N = number of faces in region i - number of nodes in region i + 1
  for k = 1 to N by stepping 1
    if k is less than or equal to the number of holes
      -- Find a loop that covers this hole, one must exist
      Start_Face = holes[k]
    else
      -- Find open edges and find a loop that covers it
      closed = []
      for each face in Faces
        if face is a gas face, close_edge[face.index] is false
          ... and it is with region i
          Start_Face = face
          exit for each face loop
      Closed_Edge[face.index] = true
      Vis_Edge[face.index] = true

      closed = LoopNode([], Start_Face, first node of Start_Face)
      target = first node of Start_Face
      open = LoopNode(closed, Start_Face, second node of Start_Face)
      edge_closed = Start_Face

```

---

---

```

-- Use open as a starting point to the path to "target"
done = false
while open is not empty and not_done
  len = length of open
  for x = len to 1 by stepping -1
    -- Expand open[x]
    LpNd = open[x]
    Add LpNd to closed
    For each face associated with LpNd.Nd
      If the face is a gas face, is in region i
        ...and Vis_edge[face.index]
        visited_edge[face.index] = true
        newNd = other node of face relative to LpNd.Nd
        if newNd is the target
          done = false
          add LoopNode(LpNd,Fc,newNd) to closed
        else
          add LoopNode(LpNd,Fc,newNd) to open
  cut elements 1 to len from open

if done
  -- Backtrace the loop
  current = last element of closed
  lcount = lcount + 1

  RegionLoopsInd[i][1,lcount] = lequ

  while not current is empty
    RegionLoops[i][1,lequ] = current.Nd.index
    RegionLoops[i][2,lequ] = current.parentFc.index
    if the first node of current.parentFc is current.Nd
      RegionLoops[i][3,lequ] = 1
    else
      RegionLoops[i][3,lequ] = -1
    lequ = lequ + 1
    current = current.parent

  RegionLoopsInd[i][2,lcount] = lequ - 1

  if the length of holes >= k
    RegionLoopsInd[i][3,lcount] = holes[k].index
  else
    RegionLoopsInd[i][3,lcount] = 0

  -- Close dead-ends from the nodes of edge_closed
  Not counting closed edges as faces, remove dead ends.

LoopNode(parent, parentFc, node)

```

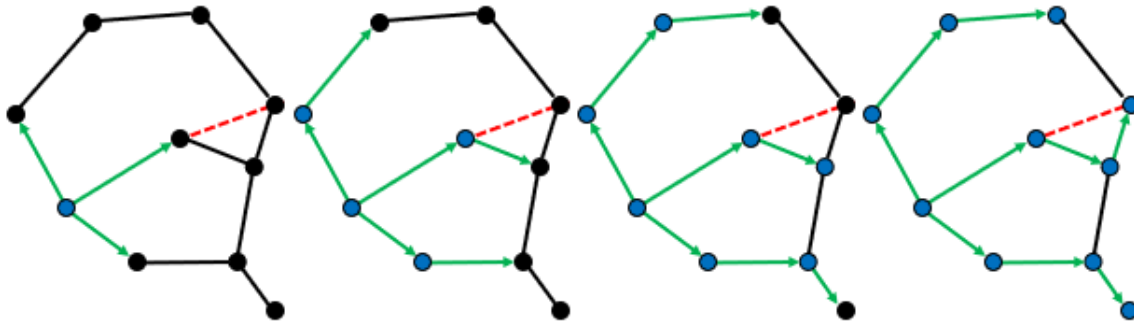
---

#### 4.1.11 Define Pressure Loss Matrix

Given the assumption that pressure does not change throughout a region, it does not allow pressure drop to be naturally obtained by the solution. An approximation of the actual pressure is

obtained in solution by solving for the nodal pressure. Thus, there is a need to determine the faces that represent all the independent equations. The algorithm that does this is graphically represented in Figure 4.6 and in the following pseudo-code. During this phase a matrix was created which includes the sign of all the faces, the algorithm that creates this is not shown here. This algorithm is akin to finding the minimum spanning tree of the network with the modification of excluding the closing faces.

Starting at a random node, grow outwards, marking nodes as visited, only go to non-visited nodes, holes are closed



**Figure 4.6: Determining the independent faces (dashed) face that has an area of zero at any point of the cycle. (red) eliminated face. (blue node) visited node. (green) set of independent equations obtained at iteration step**

---

```

isvisited = vector of false of length = number of gas nodes
ActiveRegionFaces = vector, length = number of regions, of vectors
for i = 1 to number of regions by stepping 1
    k = 0
    ActiveRegionFaces[i]
    node = node in region i
    [ActiveRegionFaces[i], isvisited] =
        PropegateActiveFaces(node, isvisited, ActiveRegionFaces[i])

Function [fcs, visited_node] = PropegateActiveFaces(node, isvisited, fcs)
    isvisited [node.index] = true
    for each face associated with node
        if the face is a gas face in the region and whose area is always > 0
            if any of the nodes of this face are not visited
                add face to fcs
                [fcs, isvisited] = PropegateActiveFaces(
                    unvisited node of face, isvisited, fcs)
    return fcs, isvisited

```

---

### **4.1.12 Vectorize Node Faces**

A different form than the transportation matrix, each node also requires, for the construction of the volumetric flow rates solving matrix, a list representing its faces. This takes the form of a vector of matrices. Each submatrix has 3 columns corresponding to the face index, the sign relative to the node and a 0 or 1 which indicates whether the face is implicit or explicit respectively.

Faces that do not fall within a single region are defined explicitly using the pressure difference across them. Explicit definition does bring instability but since it is between two regions, rather than 2 nodes, the effective node size is much larger. Explicit calculation of those faces is done because the two regions may be closed to each other during parts of the cycle. Over these parts they can generate very different pressures. Within Stirling engines however, this sort of phenomenon is relatively uncommon except for leaks and introduction of an explicit face will likely drastically slow down or destabilize the simulation.

## **4.2 Simulation Setup**

---

### **4.2.1 Apply Snapshot**

If the user wishes they may apply a snapshot of a previous test, snapshots are always taken at the same angular position (angular position of 0) which is the same angular position all simulations start on. All bodies, when created, are assigned an integer, which serves as a unique identifier. The same identifier is used to recast the data onto the body for the next simulation. A snapshot of nodal temperatures and pressures is saved for each body based on nodal positions relative to the bodily extremes (a value of 0 to 1). When recasting a snapshot, these non-dimensional positions are used in the 2D interpolation of nodal properties. Otherwise, if a body is new relative to the snapshot or has changed the phase of its material then the body maintains its default values. This flexible setup allows a much greater useful lifetime for results, as a modified engine would have similar property distributions to its original form.

### 4.2.2 Get Simulation Parameters from the User

The user is provided with a series of options that control the type of simulation that they are running. The controls and their summary are outlined in the following table:

**Table 4.1: Simulation Parameters and Description**

---

<b>Maximum Simulation Time</b>	Determines the absolute maximum amount of time the simulation will run. The actual simulation time will depend on other properties such as steady-state convergence or stalling.
<b>End Condition</b>	The user can type “SS” for a steady-state simulation termination condition or nothing to allow the simulation to run till time-out. The SS condition looks for convergence in the cycle power output to be under some tolerance.
<b>Motion Condition</b>	The user can type “C” for constant velocity simulation or “V” for variable velocity simulation. For a constant velocity, the simulation will run on the initial velocity for all increments. For the variable velocity simulation, the mechanism will be used to calculate changes in velocity throughout the cycle.
<b>Initial Velocity</b>	This is the initial velocity that the engine will be running at the start of the simulation.
<b>Maximum Time Step</b>	Generally, this not recommended as time-step is already automatically constrained, but the user has the option to prescribe an absolute maximum timestep when desired.

---

### 4.2.3 Pre-allocate Memory for Results

For simulations that are run using the constant velocity assumption the total number of data points to record is known, therefore these output arrays can be allocated at the beginning. For



variable velocity simulations, the output arrays must periodically expand in chunks to allow for further records. While MATLAB will automatically expand arrays when the provided index is beyond the scope of the array, it is more efficient to expand the array in large chunks to minimize the number of times the array must be copied.

#### **4.2.4 Run Warm-Up Phase**

The warmup-phase is inspired by laboratory experiments where engines with substantial thermal inertia are warmed up for some time to allow them to start running, as a warmed engine will generally run fast before slowing down to steady-state. With this simulation the same scenario occurs, allowing the engine to approach steady-state from a faster speed (in a variable speed test). This also will speed up convergence as components such as the piston, regenerator and walls will already have a temperature gradient, which may take a great deal of time to develop.

The warm-up phase converts all nodes to solid, using the cycle average geometry and a Nusselt number of 1 corresponding to pure conduction, region pressures are calculated to conserve mass, but warm-up phase calculations do not simulate gas flow, a neglect that significantly speeds up the calculation. When using thermal convergence acceleration (section 5.1) this step is not required as the temperature profile can show up rapidly by using the quickly acclimated gas temperatures, but this feature is retained as it can mimic an experimental process.

### **4.3 Gas Solver Loop**

---

This is the actual simulation; the following sub-sections occur for each timestep.

#### **4.3.1 Calculate Dynamic Properties**

Properties that change according to angular position such as the hydraulic diameter, volume, area, friction length, relative velocity factor or stability length (length that is used in Courant number calculations) are interpolated according to the cubic spline formula. Also, within this section, properties that rely on correlations are updated, these include conduction coefficient, heat capacity, dynamic viscosity, nodal Reynold's number and the Nusselt number are all recalculated. If a function call is involved, then elements that use the same function are grouped together and

all functions are created such that they can handle vector inputs; this improves computational speed significantly as function calls to dynamic or anonymous functions are remarkably slow.

### **4.3.2 Calculate Flow Independent Flux's**

Calculate any conduction fluxes. Some of these may be dependent on the flow conditions such as the Reynold's number, or Nusselt number but these properties are assumed to be slowly evolving relative to the size of the timestep. This section utilizes the conduction network produced in section 4.1.8 to apply calculated flux to their respective nodes.

### **4.3.3 Calculate Explicit Mass Flux's**

Explicit mass fluxes are determined via the pressure drop across a boundary. Generally, these passages are small, given that during some part of the cycle their area is zero. However, they cannot be part of a region because they represent a perfect seal during the closed-off period, allowing the two regions to exhibit different pressures. The internal energy and mass transport are automatically handled in the implicit mass flux stage. Open channels are iterated to converge on a volume flow rate appropriate for the change in pressure, leaks determine flow rate via the pressure difference. Explicit faces are excluded from the timestep calculation as they can introduce extremely high velocities, the timestep will still dramatically fall as the gas is distrusted through the network.

### **4.3.4 Constrain Time Step Pre-Mass Flux**

At this point the time step is restricted based on the Courant number for velocity and the maximum Fourier number for mixed heat conduction. Solid conduction is already analyzed using the precalculated angular timestep limitation, serving as the initial maximum timestep size.

### **4.3.5 Calculate Implicit Mass Flux's**

Produce matrix  $A$  and  $b$  such that the solution of  $Ax = b$  will give uniform pressure over all region nodes. The first section of pseudo code occurs before the differentiation between regular and environment region. The region with loops section occurs conditionally after both of those.

---

*faces = ActiveRegionFcs of region i*

---

---

```

nodes = nodes of region i
F2C = array that takes a face index and returns a column index associated
... with A
-- Structure of data:
... 1 column for each face
... The first row = face index
... The second row = sign of face relative to the node
... The third row = 1 if the face is explicit, 0 otherwise
A = matrix of square size = count all of region i's faces
b = column vector of size matching matrix A

```

---

## 1. Environment Region

Since the pressure of the environment is static, each node of a region that contains the environment maintains this static pressure in a simplified set of equations which are outlined in CHAPTER 3. The implementation of pseudo-code is as follows:

---

```

for row = 1 to the length of faces
  nd = nodes[row]
  b[row] = Vnew[nd]*Penv - T[nd]*m[nd] - dT_du[nd]*h*Qnode[ni]
  data = faces associated with nd
  for p = 1 to count of rows in data
    fc = data[p,1]
    X = h*data[p,2]*Fcrho[fc]*(T[nd] + dT_du[nd]*(Fcu[fc] - u[nd]))
    if data[p,3] is 1
      b[row] = b[row] - Fc_V[fc]*X
    else
      A[row,F2C[fc]) = A[row,F2C[fc]] + X
newV = A \ b
Fc_V[faces] = newV[F2C[faces]]
Update Fc_U, Fc_RE, Friction Factors, etc.

```

---

## 2. Standard Region

Each node is compared with one of its neighbors through equations outlined in section CHAPTER 3. Through the restricting action of all the paired equations, all nodes advance to a future point having a uniform pressure. The implementation of pseudo-code is as follows:

---

```

for row = 1 to the length of faces
  nd1 = first node for face[row]
  nd2 = second node for face[row]
  b[row] = (T[nd1]*m[nd1] + dT_du[nd1]*h*Qnode[nd1])/Vnew[nd1] -
           (T[nd2]*m[nd2] + dT_du[nd2]*h*Qnode[nd2])/Vnew[nd2]
  data = faces associated with nd1
  for p = 1 to count of rows in data
    fc = data[p,1]
    X = h*data[p,2]*Fcrho[fc]*(

```

---

---

```

        ... T[nd1] + dT_du[nd1]*(Fcu[fc] - u[nd1])/Vnew[nd1]
    if data[p,3] is greater than 0 // It is a explicit face
        b[row] = b[row] + Fc_V[fc]*X
    else
        A[row,F2C[fc]) = A[row,F2C[fc]] - X
    data = faces associated with nd2
    for p = 1 to count of rows in data
        fc = data[p,1]
        X = h*data[p,2]*Fcrho[fc]*(
            ... T[nd2] + dT_du[nd2]*(Fcu[fc] - u[nd2])/Vnew[nd2]
        if data[p,3] is greater than 0 // It is a explicit face
            b[row] = b[row] - Fc_V[fc]*X
        else
            A[row,F2C[fc]) = A[row,F2C[fc]] + X
    newV = A \ b
    Fc_V[faces] = newV[F2C[faces]]
    Update Fc_U, Fc_RE, Friction Factors, etc.

```

---

### 3. Region with Loops

In regions that have excess faces, additional independent equations are provided in the form of loops, which all such regions with excess faces are guaranteed to contain. The equations are outlined in section 3.3.3. The method begins by estimating the root using the previous 2 answers. A record of the solution values is kept so that values can be extrapolated through time to improve the initial guess.

---

```

Function Inputs(this,region,F2C,startrow,A,b,Fcrho,Fcmu,time)
-- this = the simulation object, contains all nodal and face properties
-- region = index of current region, used to get recorded points and loops
-- F2C = F2C[face index] returns the column index that face is mapped to
-- startrow = row number in A where loop entries start
-- A = matrix as constructed by volume low rate solver prior to this
-- b = array as ...
-- Fcrho = array of face densities
-- Fcmu = array of face viscosities
-- time = current time, for extrapolation

-- Loop Definitions
Loop = RegionLoops array for region i
Ind = RegionLoopInd array for region i
Nloops = number of loops in region

-- UnCollapsed References (some faces are removed from the calculation as they
... do not affect the loop pressure drop.
Rows = vector of numbers from startrow to startrow + Nloops by stepping 1

-- Extrapolate values at this time-step
if 3 points are recorded
    for each loop velocity
        0 indicates the "t-3" recorded point

```

---

---

```

1 indicates the "t-2" recorded point
2 indicates the "t-1" recorded point
prediction = y0*((time-t1)*(time-t2))/((t0-t1)*(t0-t2)) +
            y1*((time-t0)*(time-t2))/((t1-t0)*(t1-t2)) +
            y2*((time-t0)*(time-t1))/((t2-t0)*(t2-t1))

-- Define SkipLoop as a boolean indicator of whether or not the loop is considered
during this solution phase.
SkipLoop = vector of false booleans of length = number of loops
For p = 1 to Nloops by stepping 1
    If the loop is closed off during this increment
        A(rows(p), F2C(Ind(3, p))) = 1
        SkipLoop(p) = true
    Else
        A(rows(p), F2C(loop(2, Ind(2,p)))) = 1
        If prediction was made
            b(rows(p)) = prediction(p)
        Else
            b(rows(p)) = volume flow rate of face index= loop(2, Ind(2, p))

-- Calculate Inverse of "A" Matrix
Ainv = inv(A) // By LU Decomposition

-- Eliminate the rows of the solution that are not important
Ind1 = vector containing increments from 1 to the number of loops stepping 1
Remove skipped entries from both rows and Ind1

-- Initialize Solving Loop
Iteration = 1
Max_Iterations = 300
Fn = vector of ones equal of length equal to number of active loops
Tol = 1e-8

If any loops need to be solved

    -- Newton-Raphson Method
    J = square array of zeros of length = the number of active loops
    While Iteration < Max_Iterations
        -- Define Jacobian
        For i = 1 to the length of Ind1
            Elements = Ind[1, Ind1[i]] to Ind[2, Ind1[i]] stepping 1
            S = Loop[3, elements]
            Fcs = Loop[2, elements]
            For j = 1 to length of Ind1
                DeltaV = column "rows[j]" of Ainv
                If i == j
                    [dfi_dxj, fni] = getCost(this, x[F2C[Fcs]], S, Fcs, ...
                        Fcmu[Fcs], Fcrho[Fcs], DeltaV[F2C[Fcs]])
                    J[i,j] = dfi_dxj
                    Fn[i] = fni
                Else
                    [dfi_dxj] = getCost(this, x[F2C[Fcs]], S, Fcs, ...
                        Fcmu[Fcs], Fcrho[Fcs], DeltaV[F2C[Fcs]])
                    J[i,j] = dfi_dxj

-- Test Convergence

```

---

---

```

    If the sum of Fn is less than Tol then exit While loop

    -- Calculate the shift in x
    dx = Ainv[:,rows]*(J\ -Fn)
    x = x + dx

    Iteration = Iteration + 1

Record Calculated values and their associated timestamp

this.Fc_V[faces in region] = x[F2C[faces in region]]

-- getCost Function
Inputs(this, x, S, Fcs, Fcmu, Fcrho, DeltaV)
-- Calculates the loop pressure loss given loop volume flow rates "x", also
calculates the derivative of the loop pressure loss.

```

---

### 4.3.6 Constrain Time Step Post-Mass Flux

After flow velocities are initially calculated the maximum timestep is updated. If the maximum timestep is smaller than the timestep that was used, then the flow rate calculation is repeated until this condition is satisfied.

### 4.3.7 Update Properties

Using the flow rates, the nodal internal energy, mass, and temperature (derived from internal energy) are calculated, in preparation for the next iteration.

### 4.3.8 Calculate Turbulence Flux's

Turbulent fluxes are calculated according to each element type. Turbulence values are incremented at steps that are limited by a maximum change in turbulence weight. The turbulence loop is repeated until the timestep is completely traversed by the limited steps.

### 4.3.9 Record Statistics

Having stored the indexes and sign of all faces that conduct to a source, to a sink or the environment in a pre-processing step, the program adds the total of these over the increment to a collection variable, that can be accessed by outside processing.

## 4.4 Mechanical Solver Loop

---

### 4.4.1 Calculate Piston Forces

The first step to calculating piston forces is to calculate the pressure drop. This value is defined exclusively by the volumetric flow rate. The Reynolds number is calculated as:

$$N_{Re} = \left| \frac{\left( \frac{\dot{V}}{A} + F_V \cdot \omega \right) \rho \cdot d_h}{\mu} \right|_{fc} \quad (91)$$

The following matrix contains 1 row for each independent face's pressure drop and a single additional row which ensures that the partial pressures of each node add up to the pressure of the region, multiplied by its volume. This region pressure is the pressure established through the volume flux solving section. The first  $n - 1$  rows of this matrix are precalculated.

$$\begin{bmatrix} \Delta P_1 \\ \Delta P_2 \\ \vdots \\ \Delta P_n \\ P_{region} \cdot V_T \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 & \dots & 0 \\ 0 & 1 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 & -1 \\ V_1 & V_2 & V_3 & \dots & V_n \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_{n-1} \\ P_n \end{bmatrix} \quad (92)$$

The final pressures are then used in conjunction with the pressure and shear contacts to calculate forces which pass onto the linear to rotational conversion functions via the following equations:

$$F_P = \sum y \cdot A \cdot P \quad (93)$$

$$F_S = \sum \frac{y \cdot A}{2} (P_i - P_j) \quad (94)$$

where:  $y$ : The sign of the face relative to the orientation of the mechanism.

### 4.4.2 Calculate Driveshaft Forces

Each mechanism contributes a processing function that takes an input (including position, angular velocity, angular acceleration, and index of sub mechanism) and outputs the driveshaft forces including the two normal forces and torque. These are simply added up, as driveshafts can

take a wide variety of forms and the total force including the weight of the flywheel is calculated is multiplied by the driveshaft friction coefficient.

#### **4.4.3 Calculate Acceleration**

Acceleration is determined by simply calculating the acceleration induced on the flywheel via the sum of driveshaft torques.

$$\alpha = \frac{T}{I} \quad (95)$$

where:  $T$ : The total torque generated by the mechanism acting on the flywheel. This is produced via the translation of piston forces through the linear-to-rotational mechanisms onto the drive shaft.

$I$ : The moment of inertia of the flywheel.

$\alpha$ : The angular acceleration of the flywheel.

#### **4.4.4 Calculate Next Velocity Target**

The angular acceleration produced in the previous section is assumed to have happened over the angular increment immediately preceding this calculation. Therefore, the acceleration is multiplied by that timestep to produce the change in velocity. The boundary motions experienced by the gas, over the next angular increment, will ramp up to that next velocity.

In closing this step, the solver has two potential strategies. It can re-enter the gas loop to solve the next angular increment. Or it can exit having satisfied one of the following conditions: it has reached steady state, it has run out of time or it has detected that the engine has stalled. Steady-State is detected via the difference between the current and previously calculated power falling below a tolerance. An engine stall is detected via the calculation of a negative angular velocity.

## **4.5 Conclusion**

---

This chapter outlined the entire simulation process. The process started with discretization, then filtered the nodes for unnecessary loops and poorly sized nodes. Then the nodes were converted



into arrays of properties, along side several other structures designed to facilitate the solving of the solid and gas networks. Then the user was queried for details on the setup, the conditions to start from and the conditions of termination.

The main solving loop was characterized by an initialization phase, where temperature or flow dependent properties were recalculated given new conditions. Then flow independents were calculated such as conduction and explicit faces. Then the gas network was solved, considering changes in volume and energy introduced to each node. Then flow rate dependent properties such as mass and energy were distributed. Following this turbulence was transported, generated, and decayed.

At the end of each increment the mechanism took the cumulative pressure on its pistons as an impulse. This impulse was transferred through the linear to rotational mechanism, onto the driveshaft until finally causing the flywheel to either accelerate or decelerate for the next angular increment. The cycle can continue like this for many steps. The next chapter presents a series of advanced methods available in MSPM to reduce the number of steps, the time required for each step or use MSPM to automatically improve the engine.

## CHAPTER 5. ADVANCED FEATURES

---

The following sub-sections outlines some of the successful tools used to enhance the convergence of the algorithm when a steady state is desired.

### 5.1 Solid Temperature Distribution Acceleration

---

One of the primary restrictions to reaching steady state is the rate at which heat diffuses into the body of the engine. A technique used by some researchers in the massively parallel computational fluids areas [58] is to decouple the gas and solid networks and on occasion calculate the steady-state temperature regime of the solid component based upon the mean thermal fluxes from the gas component. Often this is validated by the concept that temperatures in the solid evolve on a much slower rate than the temperatures in the gas. The algorithm, in this case is as follows.

The algorithm builds off the following equation, which applies for every solid node for a given instant in time:

$$Q_{in,i}(t) = \left( \sum_{\substack{\text{other solid} \\ \text{nodes } (j)}} C_{ij}(T_j - T_i) + \sum_{\substack{\text{adjacent} \\ \text{gas nodes } (k)}} C_{ik}(T_k - T_i) \right)_t \quad (96)$$

The solution that this algorithm is seeking will solve for all  $T_i$  such that  $Q_{in,i}$ , the flux into each solid node, is zero when integrated over the entire cycle. Since the conductance between solid nodes is only dependent on angular position, and velocity change during the cycle is assumed to be negligible in a well, then the cycle averaged conductance between solid nodes  $C_{ij,eff}$  is the following:

$$C_{ij,eff} = \begin{cases} C_{ij} & \text{For static faces} \\ \frac{1}{2\pi} \sum C_{ij}(\theta) d\theta & \text{For dynamic faces} \end{cases} \quad (97)$$

where:  $C_{ij}$  = Static value assigned to conductance between node  $i$  and  $j$

$C_{ij}(\theta)$  = Angular dependent value of conductance between node  $i$  and  $j$

$d\theta$  = Angular increment that separates unique values in the dynamic values lookup table.

The conductance between the gas and a solid node depends on velocities and temperatures, and therefore the effective conductance  $C_{ik,eff}$  for these nodes is equal to the following:

$$C_{ik,eff} = \frac{1}{\sum \delta_t} \sum \left( \frac{\delta}{R_{ik} + \frac{1}{A_{ik}h_k}} \right)_t = \frac{1}{\sum \delta} \sum \delta C_{ik_t}$$

where:  $\delta_t$  = Instantaneous timestep between which each conductance samples are gathered.

$C_{ik_t}$  = Instantaneous conductance between the gas node and solid node. (98)

$A_{ik_t}$  = Instantaneous value of the surface area of the mixed face.

$R_{ik}$  = Thermal resistance of the solid component of conduction.

$h_k$  = Convection coefficient, as derived from the Nusselt number, associated with the gas node.

The effective value of  $T_{ik,eff}$  with respect to the faces that it interacts with is dependent on the instantaneous value of conductance for each face. Thus:

$$T_{ik,eff} = \frac{\sum(\delta C_{ik} T_k)_t}{\sum(\delta C_{ik})_t} \quad (99)$$

The cycle averaged version of equation (96) is as follows:

$$\oint Q_{in,i} = \sum_{\substack{\text{other solid} \\ \text{nodes}}} C_{ij,eff} (T_j - T_i) + \sum_{\substack{\text{adjacent} \\ \text{gas nodes}}} C_{ik,eff} (T_{ik,eff} - T_i) \quad (100)$$

Since every solid flux should integrate to zero over the cycle, this forms a set of equations, which can be arranged in matrix form ( $AT = \mathbf{b}$ ):

$$\begin{bmatrix} \sum_j C_{1j,eff} + \sum_k C_{1k,eff} & -C_{12,eff} & \cdots & -C_{1n,eff} \\ -C_{21,eff} & \sum_j C_{2j,eff} + \sum_k C_{2k,eff} & \cdots & -C_{2n,eff} \\ \vdots & \vdots & \ddots & \vdots \\ -C_{n1,eff} & -C_{n2,eff} & \cdots & \sum_j C_{nj,eff} + \sum_k C_{nk,eff} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad (101)$$

where:

$$b_i = \sum_k C_{ik,eff} T_{ik,eff} = \sum_k \sum_t (\delta C_{ik} T_k)_t$$

The resulting set of temperatures are adjusted to account for the current offset from the mean cycle temperature, this equation appears as:

$$T_{i,new} = A^{-1}b + T_{i,0} - \left( T_{i,avg} + \frac{1}{2}(T_{i,0} - T_{i,0 prev}) \right)$$

where:  $T_{i,0}$  = Temperature of node at the end of this cycle (last measured temperature) (102)

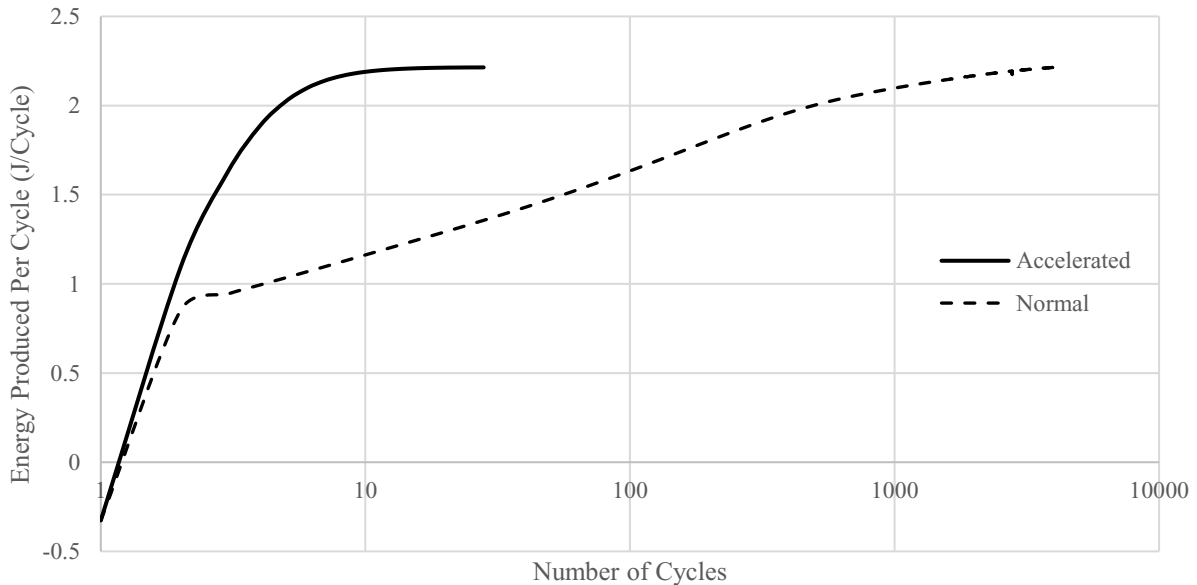
$T_{i,0 prev}$  = Temperature of node at the end of previous cycle

$T_{i,avg}$  = Average temperature of node as measured over the previous cycle.

The concluding formula attempts to recapture the oscillations that each node experiences over the cycle. As used in the solver, the engine is cycled until reaching convergence with this method. The last cycle is then conducted without this acceleration and with a tighter timestep. This final step ensures that the results are relatively free of numerical artifacts.

The performance of this algorithm was compared against natural convergence on the EP-1 test engine, an experimental engine compared against in CHAPTER 7, from a cold state. The log plot in Figure 5.1, below, illustrates the convergence behaviour. The initial sharp climb in both models is due to the rapid temperature changes of the gas volume experienced over the first few cycles. The remaining period involves the gas exchanging heat with the solid elements of the body until the amount of energy exchanged over the cycle is zero. The resulting algorithm converged to 2% of the final value in 9 cycles, equivalent to approximately 9 seconds in simulated time, the

unaccelerated model would converge to this level of tolerance after 2057 cycles, approximately 34 minutes of simulated time. The two simulations converged on the same value within 0.01% of each other. This indicates that acceleration did not significantly alter the result of the simulation, but rather improved the rate at which this result is obtained.



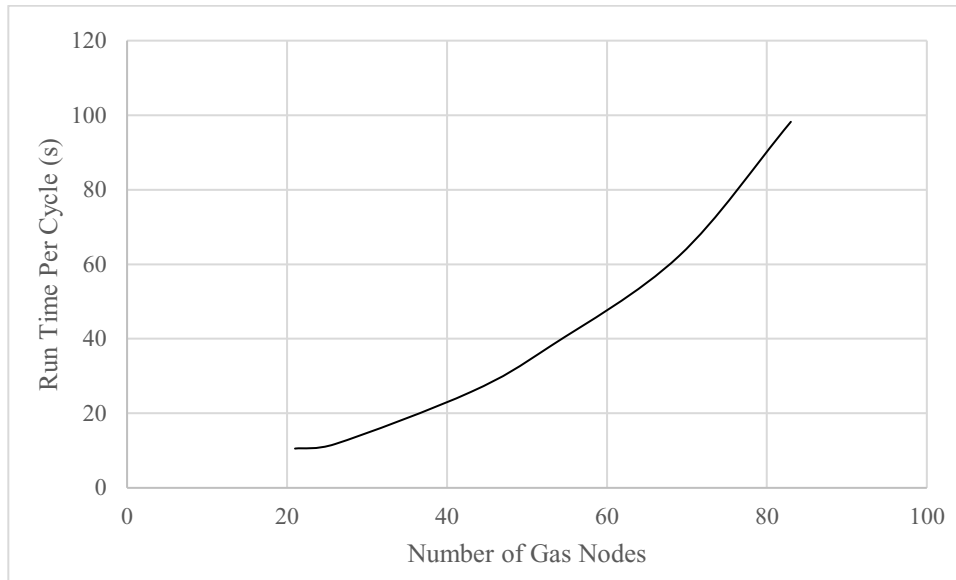
**Figure 5.1: Comparison of accelerated vs natural convergence of Stirling engine performance of the EP-1 model (defined in Appendix C)**

## 5.2 Progressive Refinement

Taking inspiration from multi-grid methods discussed in [59] which are effective convergence enhancement tools for CFD solvers, the following section outlines progressive refinement as a tool to accelerate an incremental simulation to a steady-state position. The strategy that would be undertaken by this is to run the engine on a coarse model to establish a quick overall picture of the macroscopic behaviour of the engine. Then progressively run the engine on a series of finer grids, before ultimately establishing the final temperature distribution with the desired grid size. The Snapshot feature would assist with interpolating between different granularities.

The speed at which the gas system may run is theoretically proportional to the number of gas nodes divided by the size of each node. Ideally, due to the reduction of the number of nodes by the same amount, the simulation will run at speeds proportional to the inverse of the number of nodes

squared. In the EP-1 simulation model, which is a comparison with a real-world engine found in CHAPTER 7, the trend shown in Figure 5.2 is found.



**Figure 5.2: Computational Time vs Number of Gas Nodes,**

The trend follows closely the power curve:  $0.0011x^{2.5568} + 7.625$ , it varies from the theoretical case of  $Cx^2$  for the following reasons.

- Node densities are not high nor uniform enough such that node growth could be considered a uniform action.
- Loop calculation may be relatively independent of the number of nodes and have a significant impact on the run speed.
- Larger memory requirements can slow down operations through higher computational overhead, this is not included in the theoretical case.

During test sets this progressive refinement does not activate if following a similar experiment such as the snapshot from a previous experiment as this is typically close to the steady state for the following experiment. As a result, time investment into further development of this tool was not considered. It does however allow the user to quickly estimate the result from a more advanced model by reducing the number of nodes through modification of the de-refinement factor entry on the GUI, as discussed in CHAPTER 6.

## 5.3 Geometrical Optimization

---

Traditionally, engines have been designed by skilled experts, who start with defining the basic parameters of a design using low-order models and then through use of higher models, make small adjustments to improve the predicted performance of the engine. These adjustments can be made by computers as well, and learning algorithms have been incorporated into a variety of 3<sup>rd</sup> and 2<sup>nd</sup> order models in the past [35]–[37], [46]. The choice of learning algorithm depends on the runtime of each test, which for this model can take between 10 minutes for a detailed model to several seconds for a simple model. Other involved factors include the presence of local maximums, the existence of saddle points and the non-linearity of results. Thankfully, the power response of a Stirling engine given reasonable geometry will be continuous as discontinuities will only exist when two parts of the engines are separated into two regions for part of the cycle, likely a sub-optimal process. There is no guarantee of a single optimal geometry as Stirling engines can exhibit two possible optimal configurations, isothermal and adiabatic.

To conduct this optimization a combination of standard gradient descent with a line searching algorithm is applied. Wherein, after a step is made in the direction it is tested, if the step resulted in greater power then it is the new point, if the step did not then the step is backtracked. If the first step resulted in a reduction of power then the step length is backtracked then the step length is reduced by half trying again at the intermediate position, this is conducted until an increase is recorded, after which a new gradient is taken. If the first step was successful, then more steps are taken in that direction until a reduction in power occurs. The inclusion of this improves the rate of convergence of the optimization as a gradient calculation is very expensive.

The MSPM software can modify piston strokes, the position of surfaces (connections), the fill pressure and rotational speed to seek the optimal power. These are governed by the optimization scheme builder on the GUI.

## 5.4 Conclusions

---

This chapter contained a series of optional features which were designed to enhance the model. Temperature convergence acceleration resulted in 2 orders of magnitude improvement in convergence rate at a small computational cost while converging to the same state as natural

methods. The second section included a mesh refiner which allows the designer to choose between accuracy and computational speed. The third section outlined a geometrical optimizer which can be applied to optimize a given part of the engine, or the entire engine, allowing the designer to automate this part of the design process.

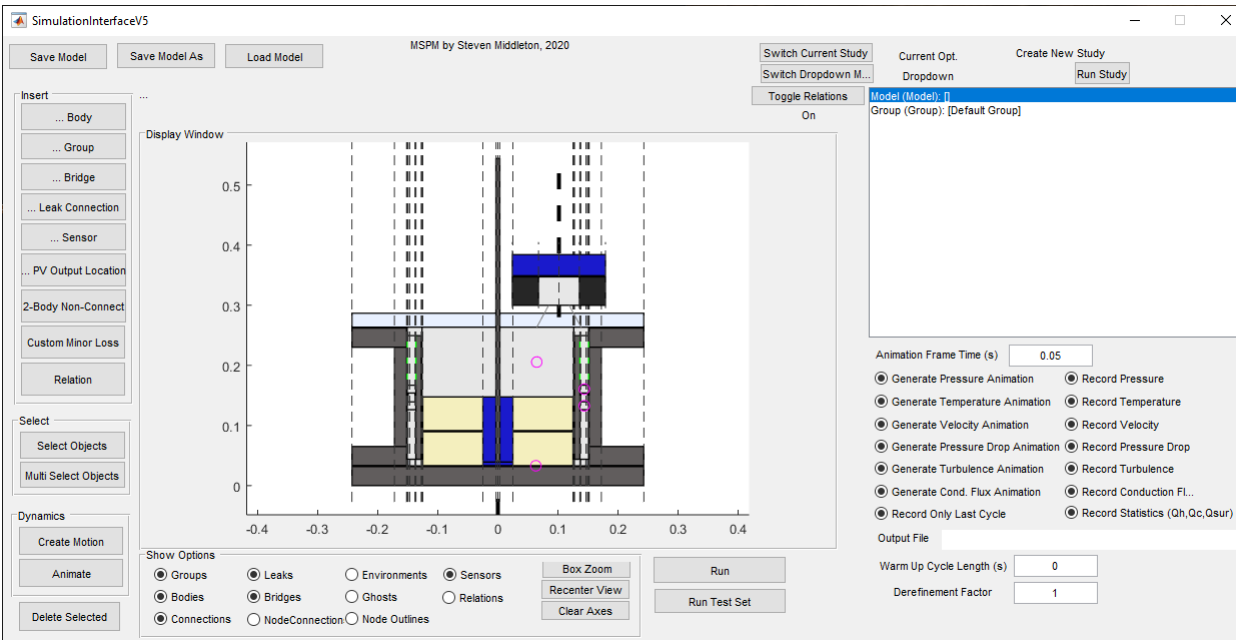


# CHAPTER 6. MODEL USAGE

## 6.1 Constructing a Model

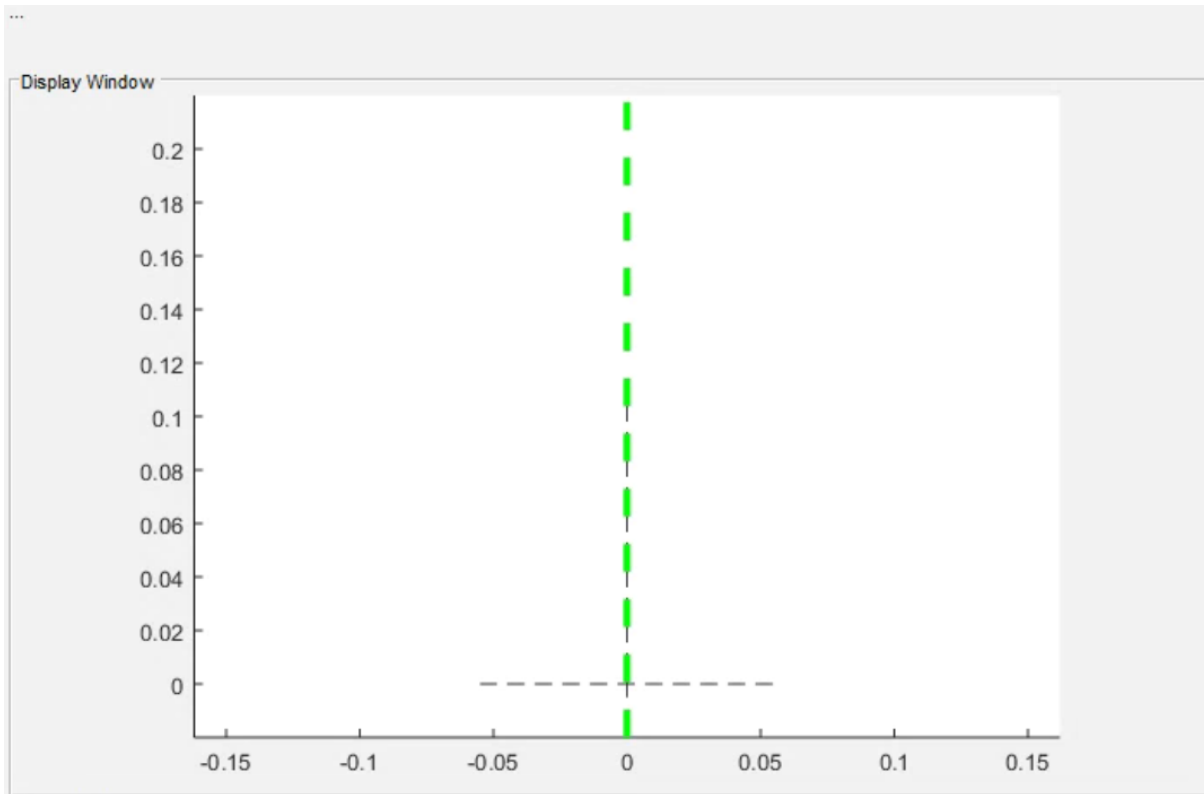
A model of a Stirling engine is initialized by opening the GUI interface. By default, a single group is created along with 2 orthogonal connections placed at the origin. From this point the user can construct their design.

Figure 6.1 shows the entirety of the main interface, from which all of the main functionalities can be accessed. In further sections each area of this GUI is broken down.



**Figure 6.1: The main MSPM graphical user interface.**

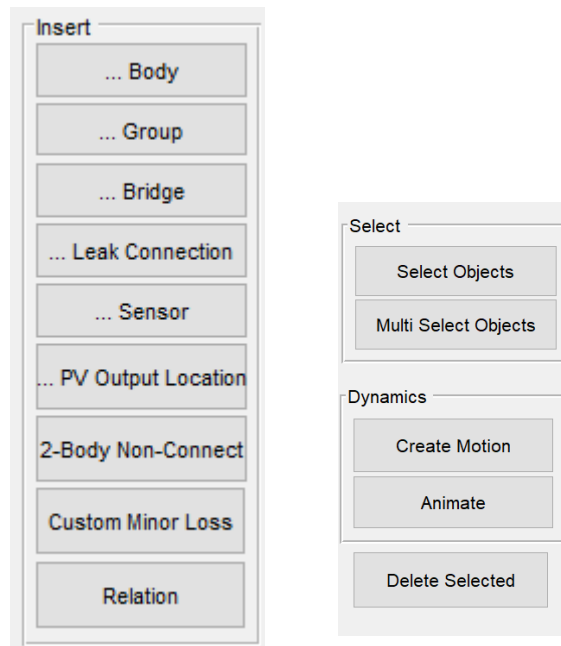
### 6.1.1 Display Window



**Figure 6.2: The main model display window**

The display window displays the model view. The user can modify the view with the box zoom options. The default window contains all the geometry of the model, default views can be reset via the re-center view option. Mode specific instructions will appear above the display window, informing the user of the next step that they need to accomplish in the insertion of elements. When producing both live and output animations, the current display window acts as the animation scope. The model within this window is displayed in cross-section and each group is assumed to be azimuthally symmetrical.

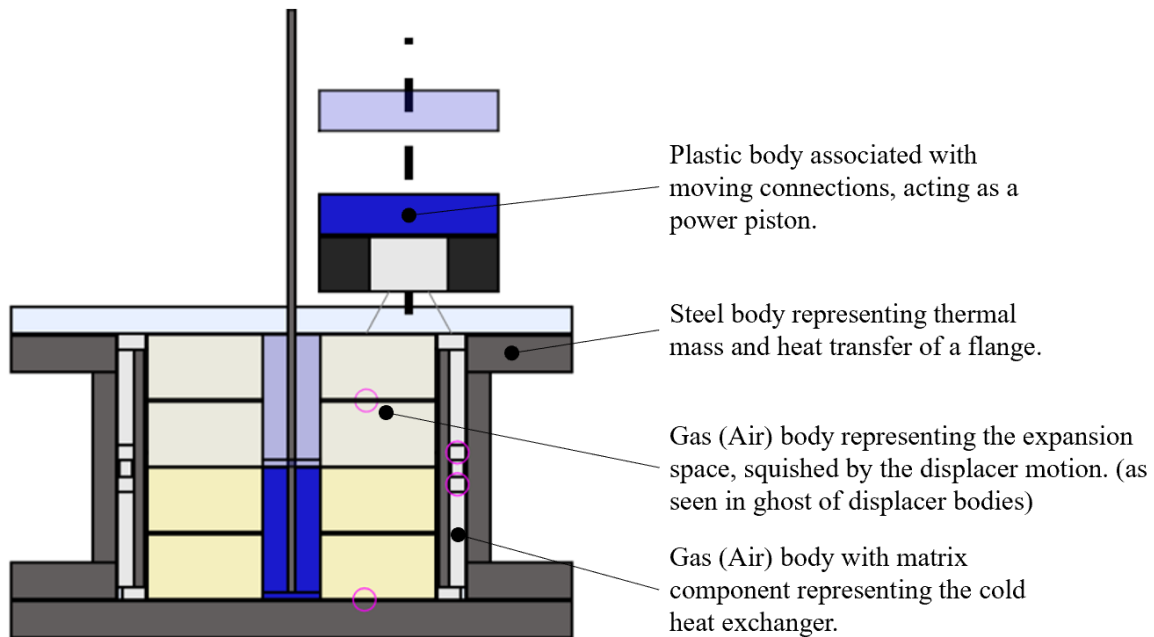
## 6.1.2 Left Toolbar – Create, Destroy and Select



**Figure 6.3: Create, Destroy and Select Toolbar**

Here, using the group of controls seen in Figure 6.3, the user has the control to add to or delete elements from the virtual engine. The elements in order of appearance and their description is as follows:

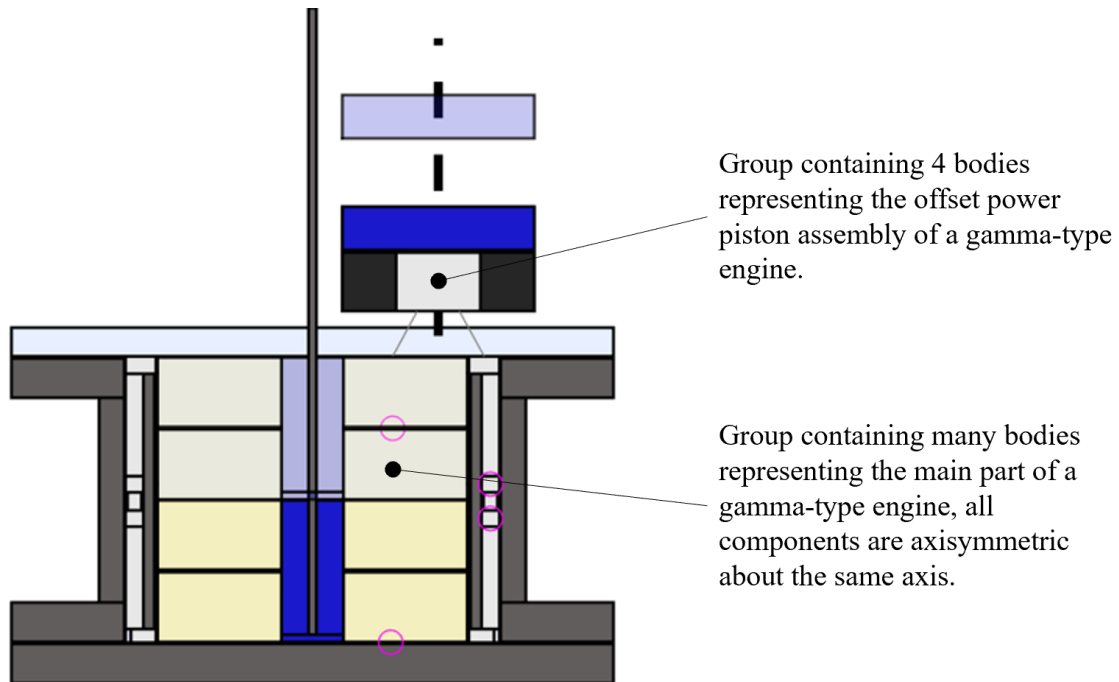
### 6.1.2.1 Insert: Body



**Figure 6.4: Examples of Bodies Being Used to Construct Geometry**

Bodies as shown in Figure 6.4, are foundational elements of an engine model and the only means of representing a physical object. This is defined through the definition of 4 constraining surfaces, referred to as connections. There are multiple ways of creating these. A left mouse click will select the nearest properly oriented connection. On the 1<sup>st</sup> connection this can be any connection, horizontal or vertical. The even numbered connections will be oriented the same as their preceding connection with the 3<sup>rd</sup> connection being the opposite orientation as the first 2. On a right click, the model will ask for either an offset from the previous connection or, in the case of an odd numbered connection, an offset from the origin. If no connections have been created, then this first right creates a connection that is parallel to the axis (the radial direction). Care should be taken to always define mobile elements at their bottom position (bottom being the farthest in the negative direction). If a new group is required, then this can be added using the insert group functionality.

### 6.1.2.2 Insert: Group

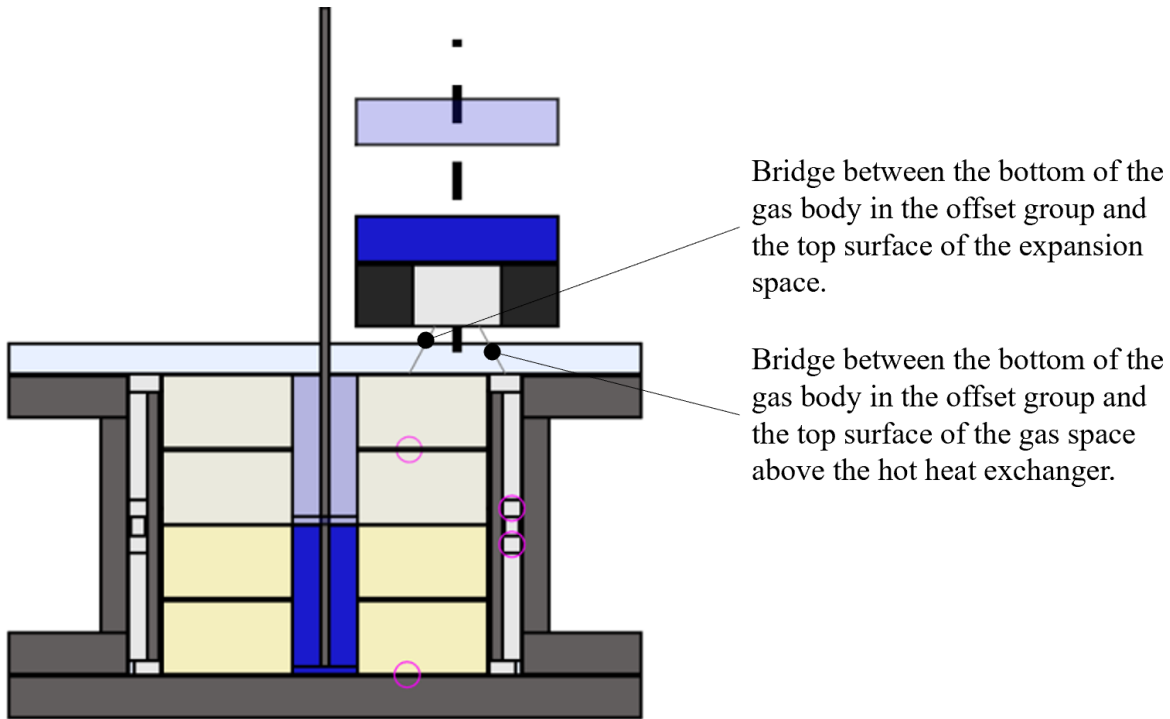


**Figure 6.5: Example of an engine containing two groups, one for the main engine assembly and a second for a power piston offset from the main axis.**

A group, as shown in Figure 6.5, is a container for a set of bodies which lie around a common axis of rotation. For a wholly axially symmetric engine, only one group would be required. A group can be inserted by using the left mouse button in the empty space of the display window. The software will insert a new group there. In post model assessment the created groups can be moved into a more compact or physically representative location by editing the position property of the group using the property inspector.

### 6.1.2.3 Insert: Bridge

If the engine design incorporates multiple groups, there may be a need to include a bridge into the design. Bridges connect one body to another via a connection interface. An example of the usage of a bridge in a practical engine is found on Figure 6.6.



**Figure 6.6: Example of a usage of the bridge component**

Bridges can represent axially aligned connections, offset connections or even T-junction connections. It is important to note that the simulation does not support the non-symmetries that arise from any of the scenarios as there is no discretization in the direction of rotation. However, the software still utilizes the same area and hydraulic diameters of the connection. The available configurations to be used by a bridge are found on Figure 2.10. To produce a bridge the following steps must be followed:

**Select the first connection:** Select a connection which touches one of the bodies, in particular this body is called the foundation onto which the other body is added on. This connection may be in the horizontal or vertical orientation and represents the surface that the two bodies will meet against.

**Select the first body:** Select the body, that is touching the first connection. In the event of an offset, this body is considered static.

**Select the second connection:** This is the face of the second body that will interface with the first body. This may ask several times, as this connection may be in a different group than the first body-connection set.

**Select the second body:** Select the second body.

**Input the offset from the origin:** If the two connections are horizontally aligned, provide an offset from being coaxial. If one of the connections is vertically aligned, provide the offset of the center of the horizontally aligned participant from the origin of the vertically aligned participant. If both connections are vertical then the second body is offset from the first body (in addition to the local coordinates of the second body within its group) by the set amount.

#### **6.1.2.4 Insert: Leak Connection**

The leak connection connects two separate bodies with a leak function that provides a leak flow rate that is dependent on the pressure difference through the general formula:

$$\dot{V} = C \cdot (P_1 - P_2)^{N1}$$

where:  $C$ : Leakage number

$N1$ : Leakage exponent

The exponent and number depend on the properties of the leak. These can be obtained by observing the pressure drop of a physical engine of similar design or used as an aid in sensitivity studies. A leak is created by selecting two bodies to link together and then by providing the coefficients.

#### **6.1.2.5 Insert: Sensor**

Sensors are intended to be used to measure a specific property at a specific location and show the evolution of that property over the course of the experiment or over the course of a single cycle. This has two forms: a single point, which generates a line plot (the parameter vs time) and a line sensor, which generates a surface plot (the parameter at N points down the line vs time). The steps to create a sensor are as follows:

**Selecting the target body:** Select a body, from within which you will record the data.

**Selecting an orientation:** There are several options for this, you can select the center of the body, the center of the bottom, top, inside or outside face or an axial line through the middle either going in the  $y$ -direction ( $y$ -axis) along the group axis, or in the  $x$ -direction ( $x$ -axis) which is the radial direction.

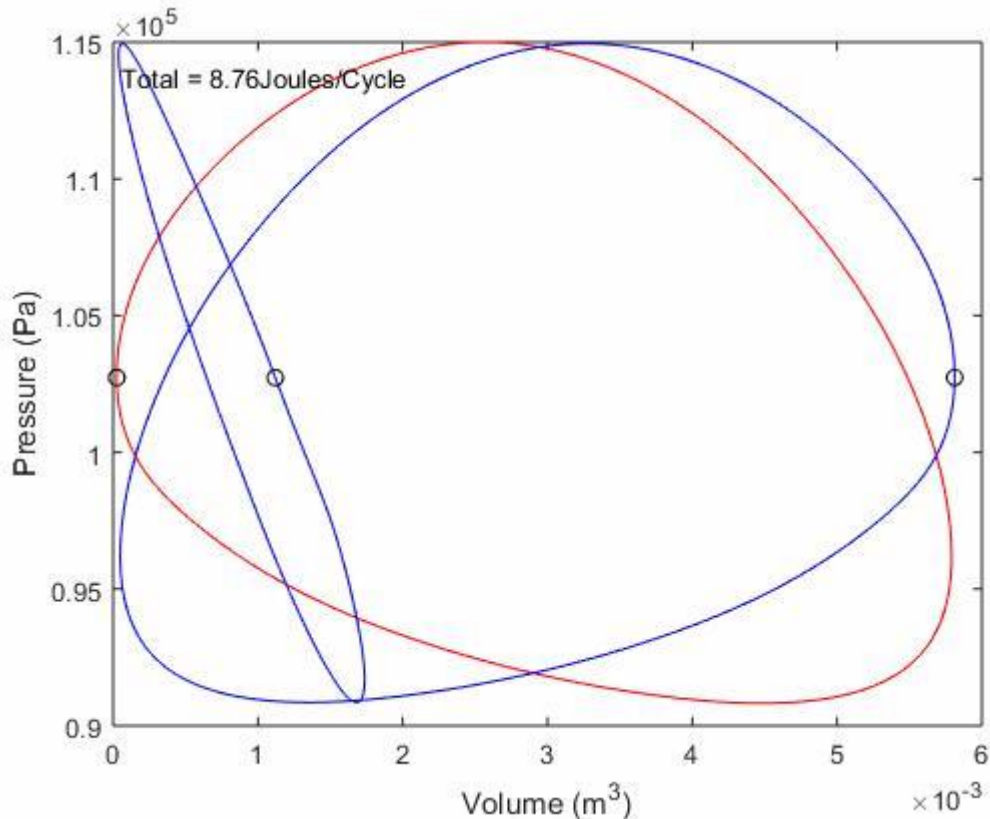
**Selecting the independent variable:** The independent variable is the variable that changes in time but is only dependent on the motion of the engine. Currently, the two independent variables are time and angle. If angle is selected the variable will, in the end, have only recorded the last cycle (as it over-writes the same angular positions). If time is selected it will record each value uniquely with time until the simulation ends.

**Selecting the dependent variable:** Currently, this can be either temperature, pressure, or the turbulence weighting factor.

### **6.1.2.6 Insert: PV Output Location**

PV Output locations are specialized sensors that are designed to output an indicator (pressure vs volume over a cycle) diagram. These features are created by **selecting a gas body**. Internally, the code will then scan the selected region (all gas bodies touching the selected body). Each variable volume space that it finds will be represented by an individual indicator diagram on the final plot. For example, the plot displayed in Figure 6.7 below is the indicator diagram produced by this sensor for a gamma type engine. A gamma engine has 3 variable volume spaces: the expansion space, the compression space, and the power piston space. The sensor also colors plots as blue if they produce positive power and red if they produce negative power and places a marker where the cycle start position was.





**Figure 6.7: Example indicator diagram from PV Output sensor**

### 6.1.2.7 Insert: Two Body Non-Connection

This feature was added to fit a specific problem, which is outlined here for clarity. The EP-1, the test engine that will be compared against, contains a flexible thin, rubber bellow. However, as solid bodies currently cannot stretch, and an immobile rubber sleeve would fail to model the thermal transport across the real system. Thus the Two-Body Non-Connection was added and prevented the gas body under the power piston from interfacing with the environment.

Thus, this feature involves two steps as follows:

**Selecting a Body:** This body will not connect to the second body by any means.

**Selecting a Second Body:** This second body will not connect to the first body through any means.

### 6.1.2.8 Insert: Custom Minor Loss Coefficient

This feature enables the user to create a custom minor loss at any boundary between bodies. The minor loss can be non-symmetric, which allows the user to, to a degree, simulate check valves in cases where a different flow direction is available. In cases where the check valves completely close off two gas spaces they will not work as only the loop solver uses flow losses to determine flow rates. In these cases, the user may have to create their own solution or create a piston that blocks the two spaces when desired.

**Selecting a Body:** Defines body 1.

**Selecting a Second Body:** Defines body 2.

**Providing a name, K12 and K21:** This user-form appears after the user creates the feature, the name is important for identification later, K12 is the minor loss for flow going from Body 1 to Body 2, while K21 is the flow going in the opposite direction.

These features can be deleted by finding them in the Model and following the instructions to delete the desired Custom Minor Losses.

### 6.1.2.9 Insert: Relation

This feature allows the selection of 2 connections to associate with each other. Doing so ensure that when one connection is moved that any connections or mechanism strokes that are associated with it will also move. A relation comes in several forms: (a) *constant distance*: when one connection moves the second moves the same amount, (b) *scaled*: when one connection moves the other moves an amount scaled by its distance from zero, (c) *scaled based on lowest value*: when one connections moves the other connection moves based on distance from the minimum extreme of this group of connections (or relative to the maximum if it is the minimum that moves), (d) *width*: when 4 or 6 connections are grouped together by having this relation type, any modifications are reflected. If the two extreme points move, the inner points shift by half that amount in the same direction. This is of particular use in cases where the heat exchangers sandwich the regenerator, lengthening one heat exchanger will length the other one and keep the regenerator centered between. (e) *Stroke*: selecting 2 connections and a mechanism will allow the difference between those connections to define the stroke. A unit change in that distance results in a unit change of

the stroke. (f) *Piston*: selecting 2 connections and mechanism will allow the difference between those connections to define the length of a piston. This relationship is only of physical suitability in the case of a displacer piston which moves within an enclosure. A unit increase in the distance will result in a unit decrease in the mechanism stroke. Relationships can be toggled on and off via the Toggle relations button along the top of the GUI.

This feature was introduced to reduce the degrees of freedom used by the gradient descent algorithm.

#### **6.1.2.10 Select: Select Objects**

While this is activated, clicking on the graphical window will give you a list of elements that you may have wanted to select. Including, within a selection tolerance, a group, a connection and a body. Selecting an object with this mode will remove any currently selected objects from the list.

#### **6.1.2.11 Select: Select Multiple Objects**

Similar to select objects. However, this option appends the new object to the selected object list.

#### **6.1.2.12 Dynamics: Create Motion**

Using the create motion interface a frame can be connection. After creation, clicking on a body or a connection exposes the drop-down menu, where a reference frame can be added. The animate function can provide some feedback on the resulting motion. As a note, stretching solid bodies are not supported realistically in the model, so ensure that every element in compound shapes such as pistons are given the same movement.

Note that the motions that are define here can be found and edited through the property inspector window, as they can are stored under the default expandable object found there.

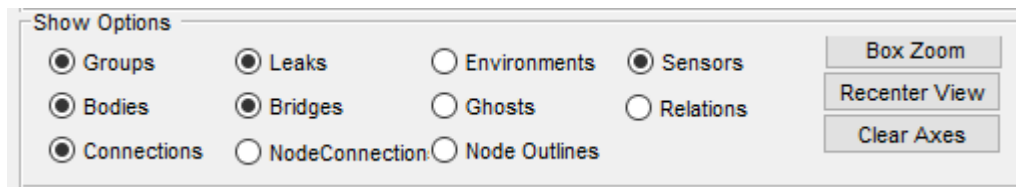
#### **6.1.2.13 Dynamics: Animate**

Pressing this button will begin a 30 second animation of the model's defined motion. Useful for understanding the phasing or interference caused by a prescribed motion.

### 6.1.2.14 Delete Selected

This function deletes (without undo) the selected element. This will also delete any element that relied on the deleted object.

### 6.1.3 Bottom Toolbar – View options



**Figure 6.8: View options**

The view options bar provides the user with the ability to modify what is shown in the display window. Several of these are self-explanatory: *Groups*, *Bodies* and *Connections* are common properties of any model. *Bridges* appear as lines that go from the center the side of one body, to the center of the side of another. *Leaks* is a placeholder for future development but will appear like bridges. *Node-Connections* connect the center of nodes together if those nodes share a connection via a face. *Node Outlines* simply places a marker at the center of all the nodes. *Environments* does not show the environment, but rather shows what the software has identified as the environment exposed surfaces. *Ghosts* show the maximum positions of any solid body that translates (as the minimum position is covered by the current placement). *Sensors* place a magenta marker or line segment along the area that will be measured by the sensor during simulations. Showing *Relations* update the colors on connections based on their existing geometrical relations with each other, those with multiple associates are colored in the default color, however.

The view option *Box Zoom* allows the user to zoom in on a box, the code maintains the aspect ratio, zooming in as much as it can. *Recentre View* resets the display window to its default all encompassing mode. *Clear Axis* is useful for removing graphics are plotting incorrectly or may represent deleted objects.

### 6.1.4 Top Toolbar – Save / Load options

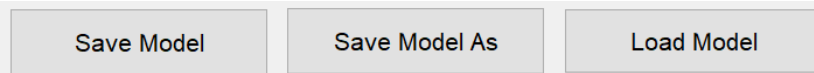


Figure 6.9: Save / Load options

The save options, allow a user to save a model, save the model as a specific name and load models. If there is an existing model that is called by the default name (which is the model's name), then the save model button asks for permission or a new name.

### 6.1.5 Top Toolbar – Geometrical Optimizer, Relation Toggle & Dropdown mode

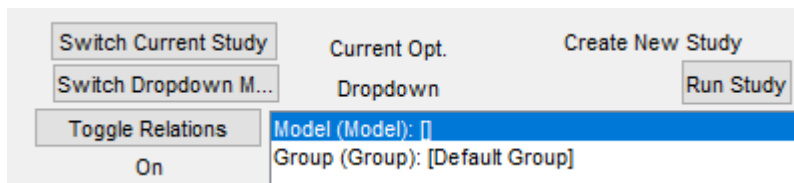


Figure 6.10: Geometrical Optimizer, Relation Toggle & Dropdown mode

#### 6.1.5.1 Switch Current Study

The *Switch Current Study* button switches the optimization study that will run or be appended through interfacing with the dropdown, via the dropdown mode being in the “optimizer” setting. The text label to the right of it will cycle between the named studied or display “Create New Study”, indicating that when this study is appended it will create a new optimization study. The study displayed here will be the one ran by selecting “Run Study”.

#### 6.1.5.2 Switch Dropdown Mode

This button toggles the text label to the right of it from blank to “optimizer”, when the text label is blank the drop down will work normally. Otherwise, anything clicked on in the dropdown menu will be added as a degree of freedom to the optimization study indicated by the *Current Opt.* text label.

#### 6.1.5.3 Run Study

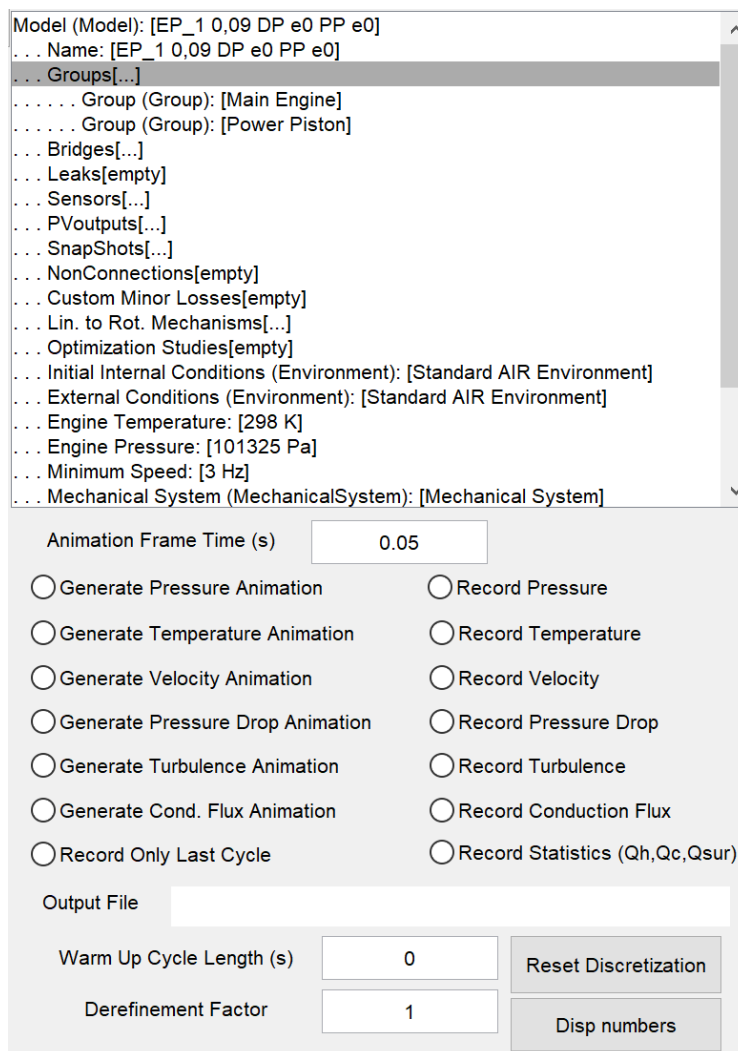
This runs the indicated optimization study. It will also ask for a set of run conditions, which are in the same format as the test set structures. If the run condition structure contains a field called

“PressureBounds” it will assume that pressure is a degree of freedom. If the run condition structure includes “SpeedBounds” it will assume that speed is a degree of freedom.

### 6.1.5.4 Toggle Relations

Toggling this button from “On” to “Off” will make relations not work when a change is made, allowing you to change the position of connections or length of strokes in isolation. Turning it back to the “On” position will reactivate these relations.

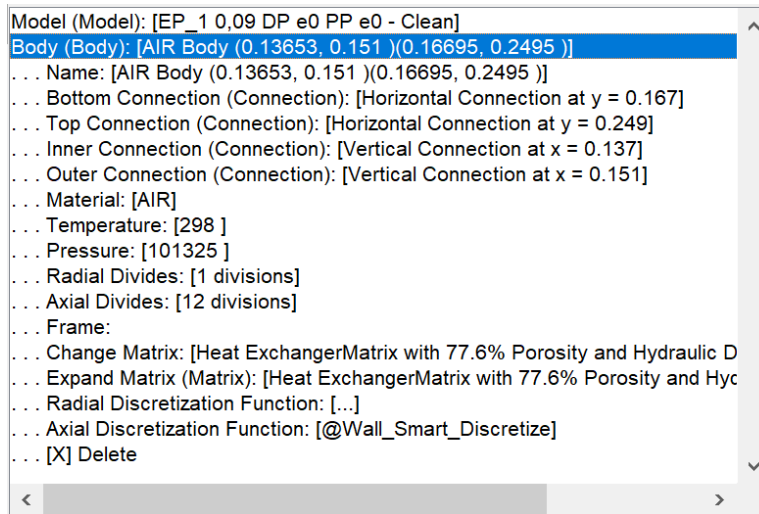
## 6.1.6 Right Toolbar – Property dropdown and Simulation options



**Figure 6.11: Property dropdown and Simulation options**

The property drop-down shows by default the root object, represented by “Model (Model) [the name of this file]”, which can be expanded to see a host of options or lists of other expandable

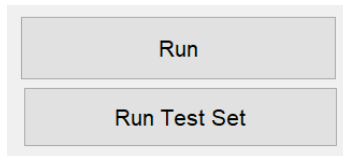
objects. Any selected objects will also appear at the bottom of this dropdown menu. Each object has multiple properties and child objects, that can either be edited, expanded or in some cases deleted from this menu. This is where motions are added to connections (or bodies), where discretization schemes are assigned and where matrixes are added to a body using the “Change Matrix” option as shown in Figure 6.12.



**Figure 6.12: Properties of Bodies including location of Change Matrix where Matrix components are initialized**

The options on the lower half of this side of the GUI refer to the simulation and simulation outputs. Checking these radio buttons will permit the software to record the referenced property and optionally generate an animation for quick review. The animation frame time refers to the amount of time that each frame of the animation covers. The output file is the path of a folder where you want the files to be saved. The warm-up cycle is a simulation option that is added for transient scenarios from start-up where the engine, is warming up for a period before turning over. The de-refinement factor is a global modifier that decimates the mesh by a set, approximately uniform, amount. A factor of greater than 1 will result in more nodes, while a factor of smaller than 1 result in less nodes.

## 6.1.7 Start the Simulation



**Figure 6.13: Run Interface**

### 6.1.7.1 Run

Runs a single test, after querying the user for simulation parameters.

### 6.1.7.2 Run Test Set

Runs a single test by calling a test set function by name. Said function returns a structure that contains simulation parameters for several tests in series, these functions are stored as files in the Test\_Running folder. An exemplar test definition file is included below. This version is the default test definition for a gradient ascent and shows all the properties available. A test set function can output an array of structs to perform several tests in series.

```
function [RunConditions] = GA_Template()
    RunConditions = struct(...
        'Model','EP_1 0,09 DP e0 PP e0 - GA',...
        'title','',...
        'simTime',60,... [s]
        'SS',true,...
        'movement_option','C',...
        'rpm',60,... [rpm]
        'max_dt',0.1,... [s]
        'SourceTemp',90 + 273.15,... [K]
        'SinkTemp',5 + 273.15,... [K]
        'EnginePressure',101325,...
        'NodeFactor',1,...
        'Uniform_Scale',1,...
        'PressureBounds',[101325 3*101325],...
        'SpeedBounds',[60 1000] ,...
        'HX_Convection',1.0,...
        'Regen_Convection',1.0,...
        'Outside_Matrix_Convection',1.0,...
        'Friction',1.0,...
        'Solid_Conduction',1.0,...
        'Axial_Mixing_Coefficient',1.0,...
        'HX_C1',1.0,...
        'HX_C2',1.0,...
```



```

'HX_C3', 1.0, ...
'HX_C4', 1.0, ...
'HX_SA_V', 1.0, ...
'Regen_C1', 1.0, ...
'Regen_C2', 1.0, ...
'Regen_C3', 1.0, ...
'Regen_C4', 1.0, ...
'Regen_SA_V', 1.0, ...
'Regen_Porosity', 1.0);
end

```

*Model* refers to the file to load in, which is the same as the name property of the root object. *Title* will be the name that the results are saved under. *SimTime* is the amount of in simulation time allotted. *SS* is a flag indicated whether or not the simulation will stop at steady state. *Movement Option* can have defined either constant speed (C) or variable speed (V) indicating if the results should contain the velocity variations. *Rpm* indicates the initial or target speed in revolutions per minute. *Max dt* indicates the maximum timestep to use, normally this only applies for exceedingly slow scenarios. *SourceTemp*, *SinkTemp* indicates the temperature that will applied to constant temperature elements flagged as source or sink respectively, which are automatically identified by relative temperature. *EnginePressure* indicates the fill pressure of the engine, the internal volume of the engine is identified by the placement of a PVoutput sensor. *NodeFactor* is the mesh refinement factor applied to the test instance, which scales the number of nodes by this amount. *Uniform\_Scale* scales the geometry by the specified amount in all directions. *PressureBounds* and *SpeedBounds* is used by gradient ascent to bound the search space for pressure and speed respectively. The next 6 fields allow the user to apply factors on each of the described properties, a factor of 1 uses the results of the default equations. The final set of equations modify custom type heat exchangers with correlations  $N_{Nu} = C_1 N_{Re}^{C_2} N_{Pr}^{0.33}$ ,  $N_f = C_3 N_{Re}^{C_4}$ , surface area to gas volume ratios (SA\_V) and porosity.

## 6.2 Discretization

In simulations there are two types of discretization, spatial and temporal. The following sections outline some recommendations and theory on discretization. These are controllable through interfacing with each body via the property drop-down menu, where the number of nodes in each direction can be controlled. Specialized functions that allow for better modelling of surface

gradients can be added by loading the wall smart discretize function as a discretization function. This function uses the global mesher properties. Those properties can be found under Model->Mesher. Other meshing functions can be added if required.

### 6.2.1 Spatial Discretization

The spatial discretization is how much space is broken up; it is simplest to assign this at the start of the simulation. If this is the case, certain assumptions have to be made by the designer to ration nodes to areas that truly need it. With spatial discretization of the solids the main concern is high gradients, as high gradients require greater node density to be properly modelled, particular at their edges. Three phenomena utilize these high gradients, which will be discussed in the next three paragraphs.

The first aspect is gas spring hysteresis which occurs between the gas and the very surface of the solids surrounding it. If this is the case then these surface nodes will experience strong gradients and curvature, which would vary depending on the expected frequency, and the thermal diffusion of the material. These oscillations are experienced throughout the material, but at a specific depth they become negligible. This depth, which is called the oscillation penetration depth was studied by Wang [60], who presented the following. The formula here is modified such that at this distance only 5% of the oscillation is present.

$$x_{0.05} = 3 \sqrt{\frac{2\alpha_t}{\omega}} \quad (103)$$

where:  $x_{0.05}$ : Represents the distance at which the temperature fluctuations are at 5% of their original value.

$\omega$ : Angular velocity

$\alpha_t$ : Thermal diffusivity  $\left(\alpha_t = \frac{k}{\rho c_T}\right)$

Beyond this point, temperatures experience a slowly evolving or static temperature profile, which is a phenomenon assumed by several Stirling engine modelers [35], [46]. The ratio of conduction, density and heat capacity is equal to the thermal diffusivity, a representation of the thermal inertia

of the material. The angular velocity ( $\omega$ ) is established at the beginning of the simulation as an estimate of the final angular velocity.

The second aspect is static conduction within the solids of the body. With first order discretization schemes the error associated with discretization increases proportional to the local element size, a length representing how far elements are from each other. This error also increases based on the distribution of values among the nodes, if a high gradient or curvature is to be represented then a fine grid is required. These areas namely exist in features that divide two areas of very different temperatures, or at interfaces of materials of very different thermal properties.

The third aspect is dynamic conduction. As Stirling engines contain moving components, there is an opportunity for momentarily high gradients to be generated when two components closely cross paths. This leads to gradients which exist both into the material depth as well as along the interface length.

For gases the same thing is true as for solid elements, however, the areas in which they occur are different. As the one-dimensional assumption prevents the modelling of the temperature gradients in large chambers, particularly off of walls, these areas are considered well mixed and don't require many nodes. To partially account for this and other factors gas nodes should include turbulence, a representation of how disturbed the flow is. Therefore, there may exist areas of high gradients in turbulence, such as areas directly following geometrical non-uniformities. These gradients persist for about one diameter from the entrance according to Gedeon [35]. Temperature gradients exist at the start and end of heat exchangers and regenerators, Anderson [48] studied this extensively, the exact breadth of the inlet gradient will depend on the effectiveness of the heat exchangers. In addition, Anderson identified that gradients can persist in the areas around heat exchangers, in particular after reversal events, highly diffusive schemes require many nodes to properly represent these moving gradients. Anderson utilized an advanced flux limiting scheme to preserve these gradients.

## 6.2.2 Temporal Discretization

Temporal discretization is how much time progresses after each flux calculation. For solids this is calculated based on the Fourier number, which sets a limit to the timestep based on the numerical volatility of a numerical element. This non-dimensional number is identified here:

$$N_{Fo} = \frac{\alpha_t \cdot \Delta t}{\Delta x^2} = \frac{k}{\rho \cdot c_T} \frac{1}{\Delta x^2} \Delta t = \underbrace{\left( \frac{k \cdot A}{\Delta x} \right)}_{C_{fc}} \underbrace{\left( \frac{1}{\rho \cdot c_T \cdot \Delta x \cdot A} \right)}_{1/C_T} \Delta t = \frac{C_{fc} \cdot \Delta t}{C_T} \rightarrow \Delta t = N_{Fo} \frac{C_T}{C_{fc}} \quad (104)$$

where:  $N_{Fo}$ : Fourier number

$\alpha_t$ : Thermal Diffusivity

$\Delta t$ : Timestep

$C_{fc}$ : Thermal conductance of a face, used as  $\Delta Q = \Delta t C_{fc} (T_2 - T_1)$ , where  $T_2$  and  $T_1$  are temperatures of two nodes connected by a face.

$C_T$ : Specific heat capacity of node, used as  $\Delta Q = C_T (T_2 - T_1)$ , where  $T_2$  and  $T_1$  are temperatures of the same node from different times.

Recommendations from Hensen and Nakhi [61] indicate that the iteration is stable – errors do not grow – for a Fourier numbers of 0.25 or less. Therefore, a Fourier number of 0.25 is selected as the maximum of any node. In determining the timestep, the entire pool of nodes is queried for the timestep limit.

For gases the maximum time step is established based on the Courant Friedrichs Lewy [62] condition:

$$N_C = \frac{U \cdot \Delta t}{\Delta x} = \frac{\dot{V}}{A_{fc} \cdot \Delta x} \Delta t \leq N_{C,max} \quad (105)$$

where:  $N_C$  = Courant number, less than 1 for theoretical stability.

$U$  = Speed at which information travels, the speed of the gas

$\Delta x$  = Spatial difference between two measurement points. Between which the information is travelling.

This condition ensures that a property is not transported any farther than is calculatable by the underlying numerical system. The term  $\Delta t$  represents the time step,  $\Delta x$  represents the spatial distance between adjacent nodes and  $U$  represents the velocity.

Numerical algorithms that cover a longer distance, such as one that considers 2 neighbors on either side may be stable with a maximum courant number of 3 due to the added information. In practice however, these theoretical maximums are limited by lower node quality, the presence of destabilizing features and numerical errors, often by an order of magnitude.

## **6.3 Simulation Tools**

---

### **6.3.1 Snapshot**

A snapshot is an image of the engine at the 0<sup>th</sup> angular position on the last cycle of the engine, this obtains a snapshot of all the bodies of the engine and records to the granularity of the body discretization. These are recorded in arrays of temperatures that are accompanied by  $X$  and  $Y$  values scaled to the body as if it were of unit dimensions, such that interpolating onto a modified body is a trivial manner. This option does not handle new bodies, instead leaving them with the default temperature values.

### **6.3.2 Test Set Running**

A series of tests may be run using test definitions. These run in series, each test starting from the conclusion of the previous, the goal would be that an engine curve could be defined by setting the engine to run at a set of constant speeds in sequence. Each run can be told to look for a snapshot title which will by default be used as the starting point, otherwise it uses the last defined snapshot as a starting point. The use of snapshots allows this approach to construct an engine curve more quickly than starting from scratch each time.

### **6.3.3 Geometrical Optimization**

The MSPM software includes a geometrical optimizer, which when given a series of parameters, will tune the geometry until the engine gives optimal power output. These parameters

include connection positions, mechanism strokes, charge pressure and engine speed. The optimizer makes use of gradient ascent to make small adjustments to the geometry in the direction of positive slope until it reaches either the maximum number of iterations or reaches a point at which the RMS of all gradients is below a set tolerance. Further details on this is found in 6.1.5 for interfacing with it and 5.3 for details into the algorithm.

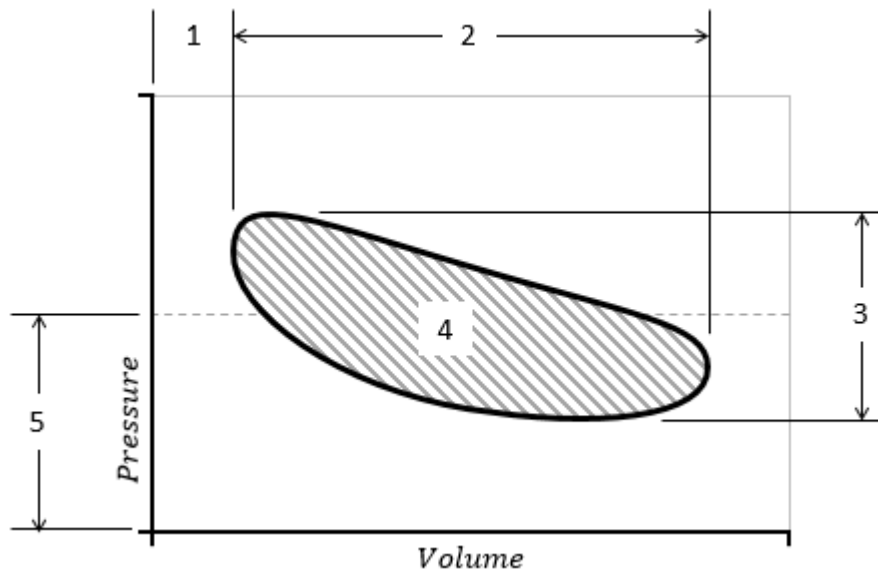
## 6.4 Model Outputs

---

### 6.4.1 Engine Assessment

#### 6.4.1.1 PV Diagram & Thermodynamic Work

Pressure-volume (PV) or indicator diagrams, produced by the PV output sensor, are an excellent descriptor of the engine's thermodynamic cycle. Several quantitative measures can be quickly extracted from the PV diagram. These are labeled on the sample indicator diagram below.



**Figure 6.14: A plot and definition of the pressure-volume (PV) diagram**

1. **Minimum Volume** – This quantity represents the minimal volume of the engine. This can be used for two imported parameters. The dead volume and the compression ratio.

- a. **Dead Volume**       $V_{dead} = V_{min} - V_{DP,swept}$

This parameter represents the volume that does not change its temperature during the cycle, in practice these are all the spaces not swept by the pistons. This dead volume acts as a form of compliant boundary for the engine, which reduces the maximum pressure that the engine reaches. In general, this quantity should be minimized as much as possible.

b. **Compression Ratio**  $r_c = V_{max}/V_{min}$

This non-dimensional parameter represents the proportion that the volume changes throughout the cycle. The optimum point of this value is a function of the temperature ratio but is influenced by a variety of factors including the mechanism design. Review and experimentation on low-temperature engines by Stumpf [12], indicates that an approximate value of this optimal point can be obtained as the value:  $0.624(T_H/T_L) + 0.376$ .

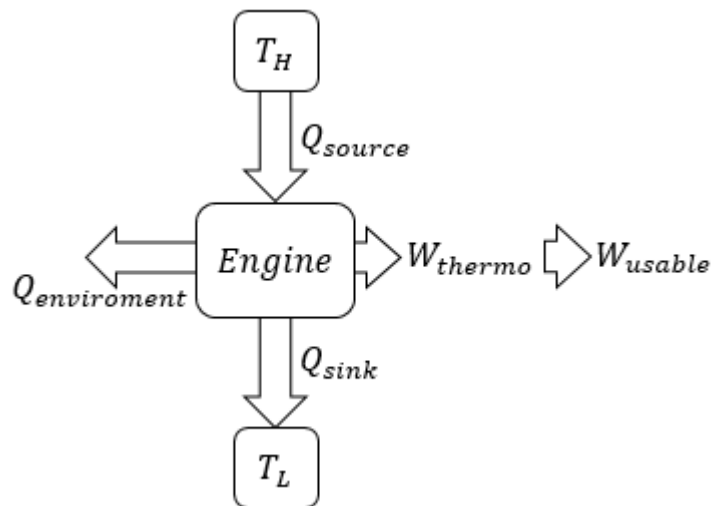
2. **Total Change in Volume** – This quantity represents the  $dV$  in the basic equation for pressure work  $W = PdV$ . However, increasing this quantity does not increase the amount of power produced linearly as various factors influence the power output of the engine.
3. **Engine Pressure Swing** – This dependent property of the cycle represents the  $P$  in the basic equation for pressure work  $W = PdV$ . This property is indicative of the magnitude of the temperature swing in combination with the volume change.
4. **Area Enclosed by the Curve** – This measure represents the amount of work, as defined by  $W = PdV$  that the gas volume being observed sent to the mechanism. An engine that produces negative work would still maintain an area here, but the border would progress through time in the counter clockwise direction. A discrepancy between this value and the measured shaft power indicates the amount of energy lost to the mechanism.
5. **Average Pressure** – This quantity indicates the mean pressure that the engine operates at. This property is directly proportional to the amount of produced power, according to the West Number.

When used properly an indicator diagram captures most interactions between the gas and the mechanism. Proper use of an indicator diagram would capture all spaces that are being compressed or expanded. This is important as minute differences in pressure in these spaces can be

embodiments of flow friction effects. This results in an indicator diagram with multiple loops. The indicator diagram does capture shear from flows pushing past the piston.

### 6.4.1.2 Energy Transfer Statistics

The series of information defined here as Energy Transfer Statistics include values that go to and from major elements of the engine. These are defined in Figure 6.15 through the generic Heat Engine Diagram.



**Figure 6.15: Generic Heat Engine Model**

These terms are automatically generated by analyzing the given model for constant temperature elements and making note of all possible avenues of exit to such nodes, sources and sinks are differentiated by assigned temperature. The actual efficiency and Carnot adjusted efficiency of the engine are calculated as follows from these values.

$$\mu = \frac{W_s}{Q_{in}} \quad (106)$$

$$\mu' = \frac{\mu}{\left(1 - \frac{T_L}{T_H}\right)} \quad (107)$$

The actual efficiency is a measure of how well an engine converts one form of energy into another. However, all technologies that utilize heat to generate energy must be subjected to the limitation of Carnot which artificially undervalues machines that have a lower maximum



efficiency. Therefore, the second efficiency adjusts for this and is considered a better measure of the quality of an engine design, irrespective of the quality of its thermal sources.

### 6.4.1.3 Mechanical Work

Mechanical work is the integral of the equation  $W = Td\theta$  over the tested time frame. The difference between the area of the sum of the PV diagrams and this measured quantity indicates the efficiency of the mechanical system at converting linear into rotational force. In dynamic and constant speed cases the instantaneous mechanical power is equal to the following.

$$Power = \underbrace{I \cdot \omega \cdot \alpha}_{\substack{\text{Excess Power} \\ \text{given to Flywheel}}} + \underbrace{T_{load} \cdot \omega}_{\substack{\text{Power Consumed} \\ \text{by Load}}} \quad (108)$$

### 6.4.1.4 Sensors

Sensors define an explicit output from the model that will be automatically produced when the simulation completes. Single point sensors will produce data suitable for a line graph, line sensor output data that is suitable for a surface plot. The user can select from any of the properties in the model and compare them against either time or angular position. An example of the plots produced by this at the end of the simulation is found in Figure 6.17.

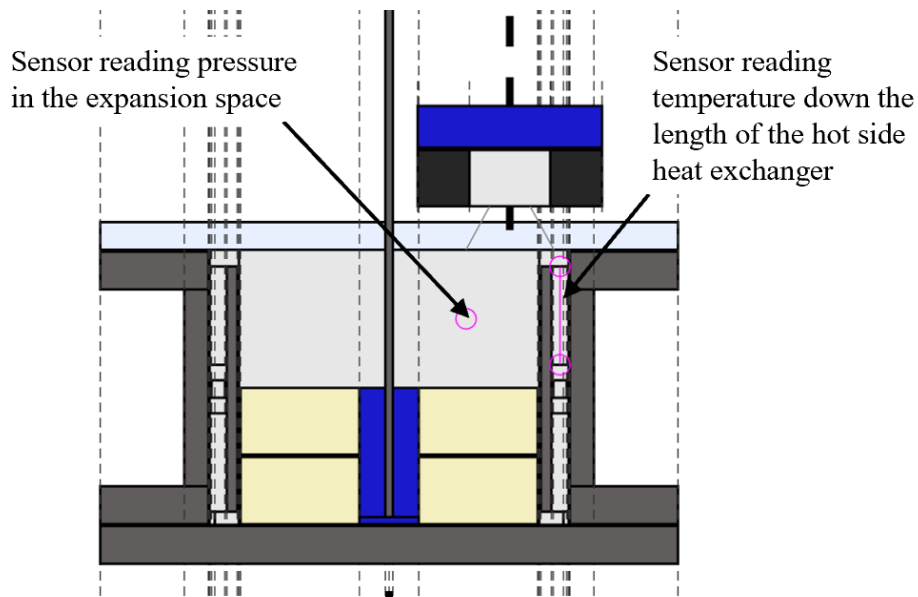
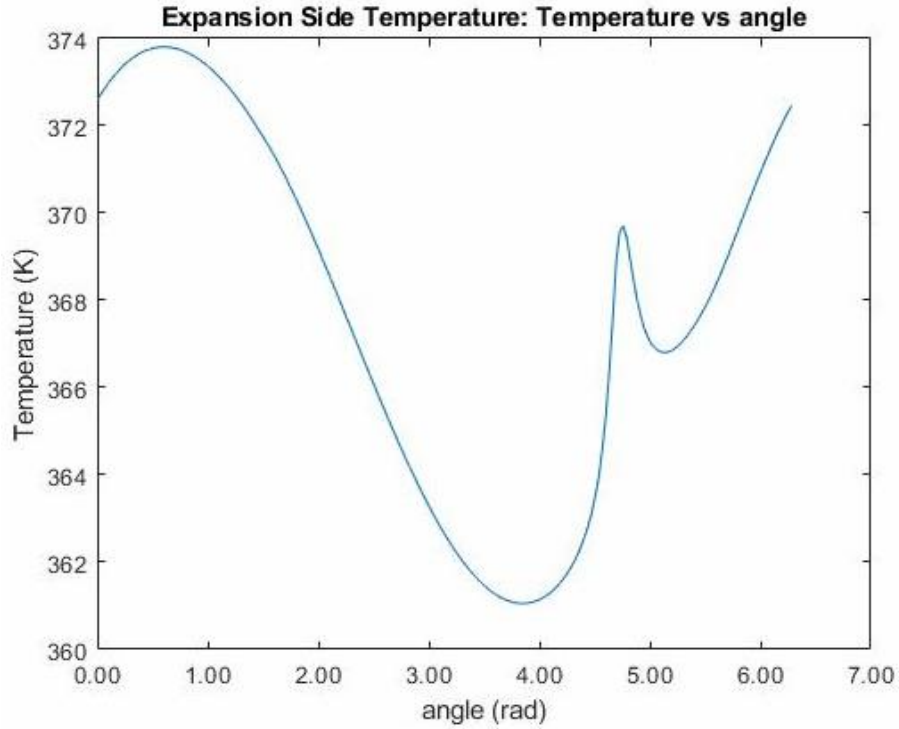
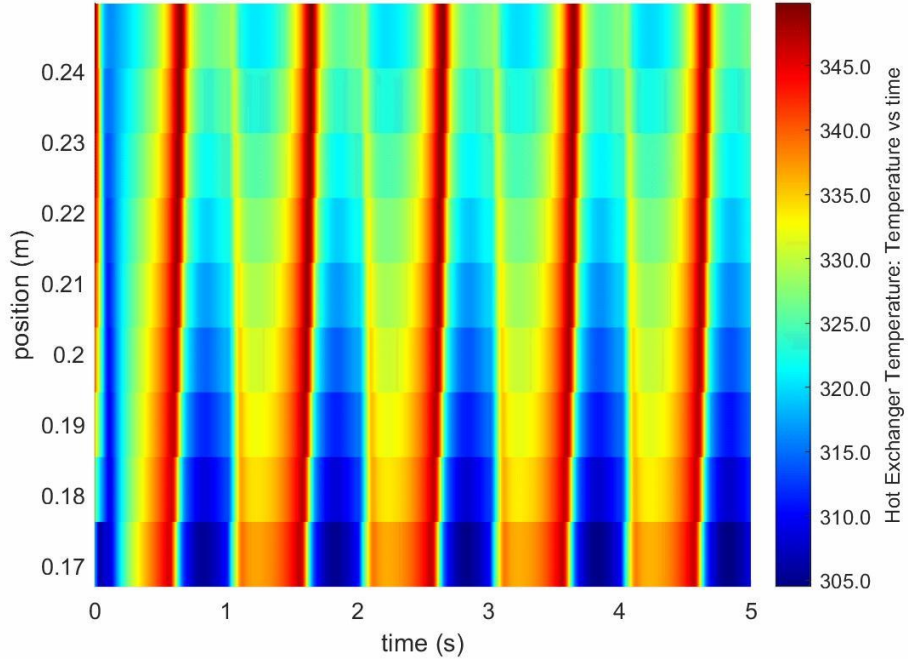


Figure 6.16: Sensor Usage Examples as shown in the GUI



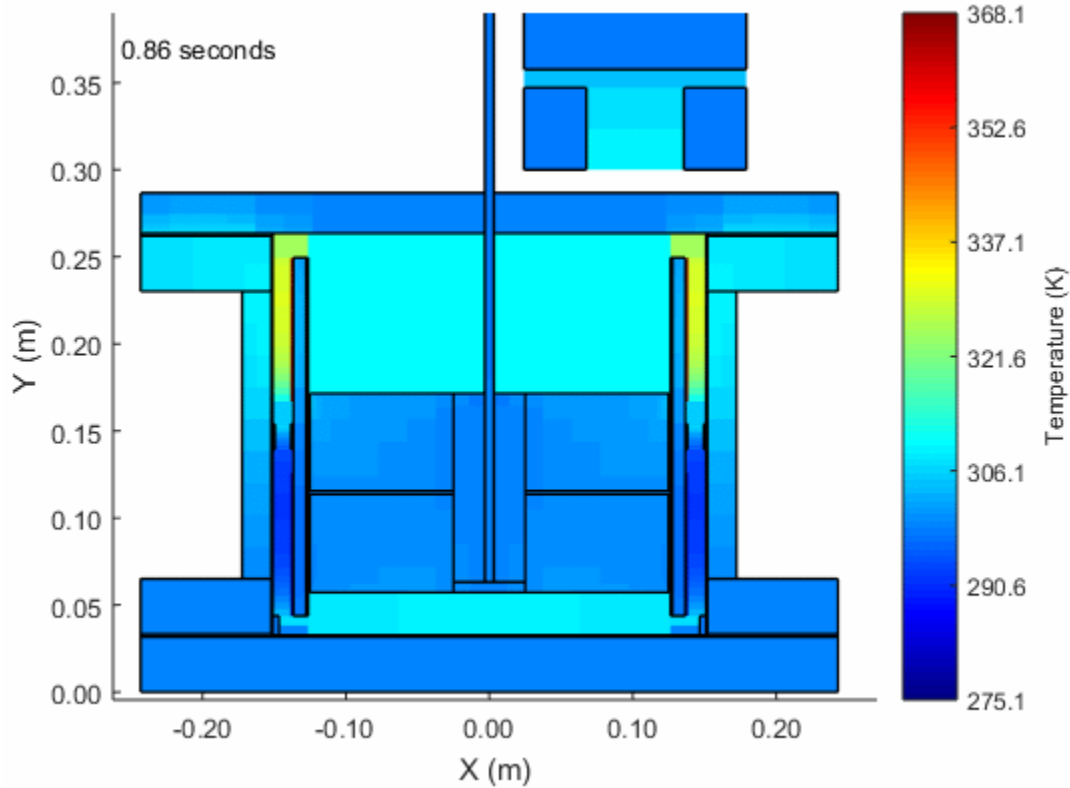
(a) Temperature vs angle plot of expansion space of engine with a source temperature of 368 K, discontinuity between 4 and 5 matches up with when flow would reverse.



(b) Temperature vs time plot down length of a low effectiveness heat exchanger.

**Figure 6.17: Output of sensor (a) point sensor (b) line sensor, locations shown on Figure 6.16**

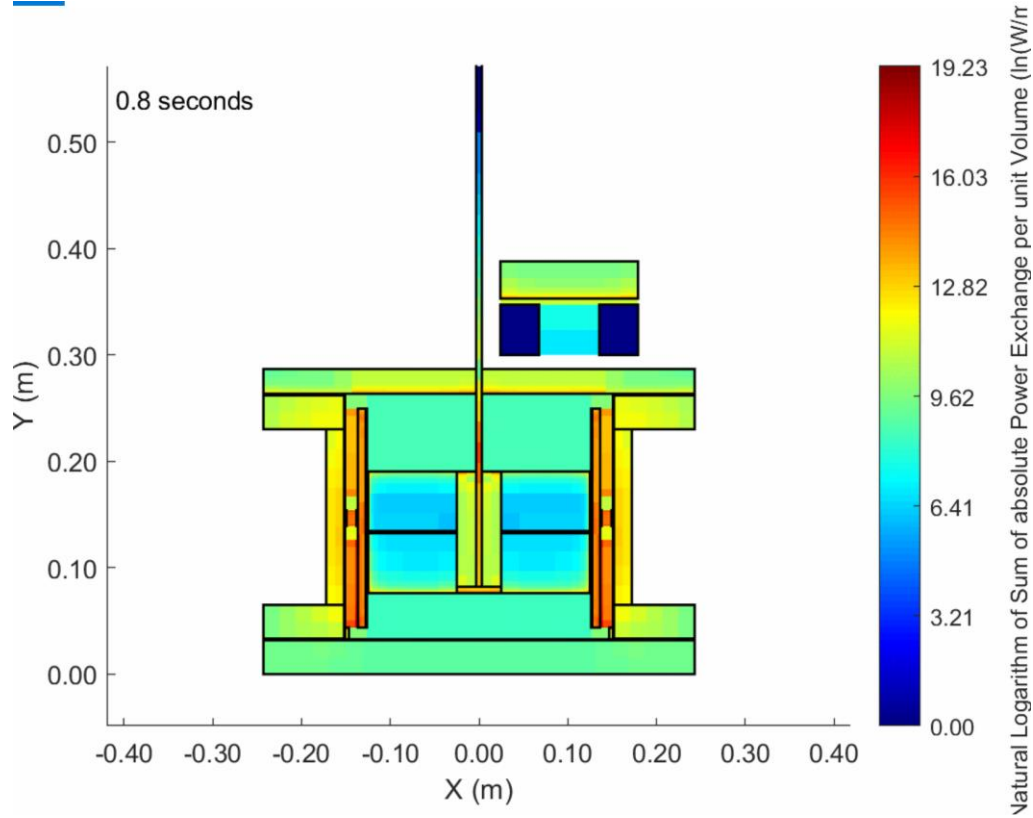
### 6.4.1.5 Heatmap Animations



**Figure 6.18: An example temperature heatmap snapped during an animation of the modelled Stirling engine**

While not an integral part of the model, the software has the built-in capability to generate graphical outputs of properties. Figure 6.18 presents one of these outputs, which displays the temperature of different nodes through the engine. These types of animations may allow the designer to visualize the flow of matter, energy and motion of the engine allowing conclusions to be drawn more quickly. Pressure, turbulence, temperature, and conduction are plotted in this fashion. This is a function that is not normally found by default in Stirling engines codes, even commercial ones such as SAGE [35], but is relatively mainstream among generic CFD codes.

### 6.4.1.6 Conduction



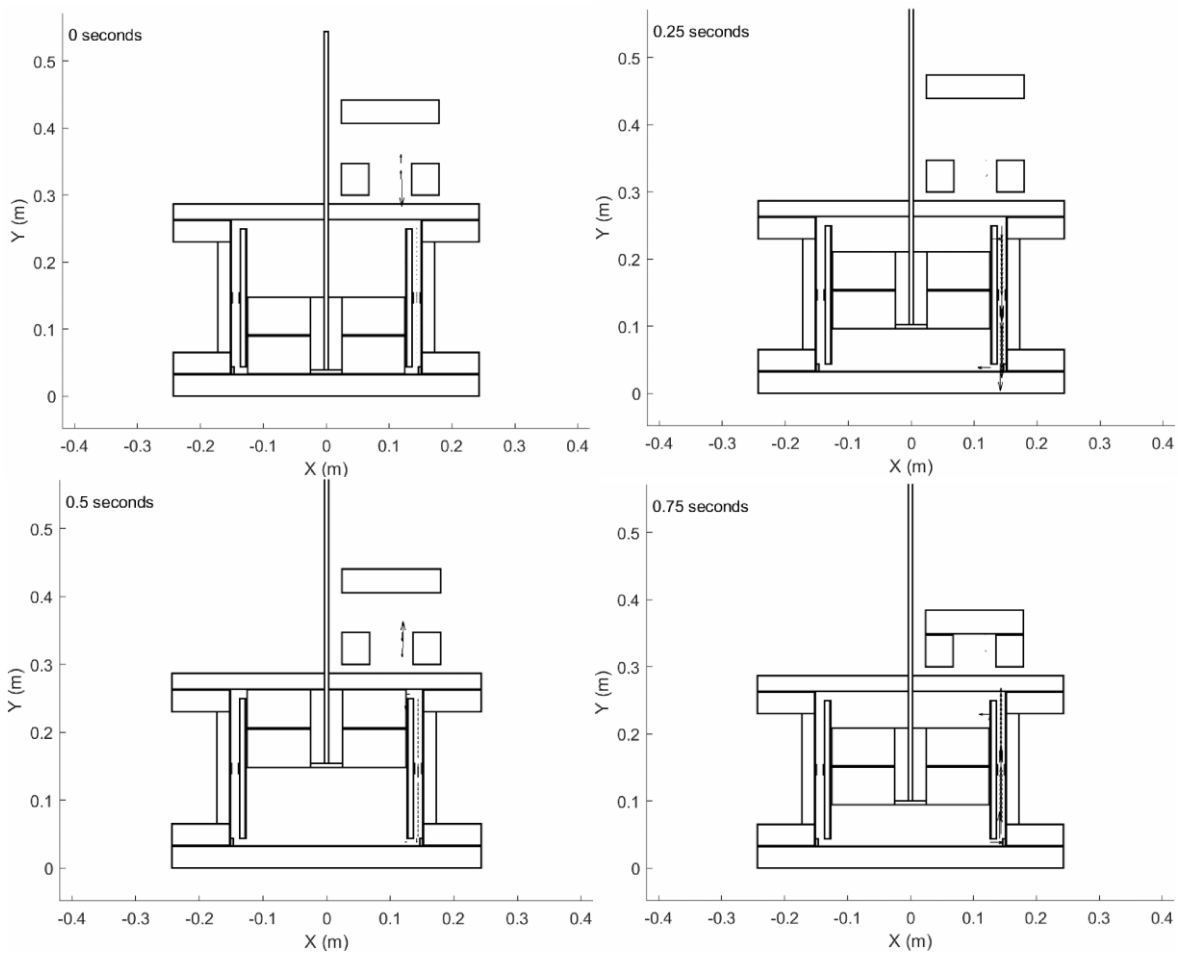
**Figure 6.19: An example conduction heatmap snapped during an animation of the modelled Stirling engine**

Volumetric conduction is a tool introduced in this thesis as a measure of the energy passing through an element as a function of its volume at any given time. This is calculated via the formula:

$$Cond = \ln \left( \frac{\sum_{all\ faces\ in\ nd} |e_{fc}|}{V_{nd}} \right) \quad (109)$$

The application of the natural logarithm is simply for enhancing the contrast when the value is plotted in the heatmap style. The purpose of this plot, though it lacks physical meaning, is to provide designers with the locations in which heat flows, which is generally undesired, outside of the heat exchangers. From the plot in Figure 6.19 it is observed that the central divider, that lies between where the displacer piston moves and the heat exchangers, exchanges quite a lot of heat, comparable to the heat exchangers themselves at that moment. The top and bottom caps of the engines on the other hand lose very little heat to the surroundings.

### 6.4.1.7 Face Based Animations



**Figure 6.20: 4 frames of an instantaneous flow velocity plot snapped during an animation of the modelled Stirling engine (running at 1 Hz)**

Velocity and pressure drop animations give insight into areas of the highest flow and pressure loss, it may serve as an indication to the designer of areas where the flow is bypassing or features that introduce most of the flow loss experienced by the system. An example of a face-based animation is in Figure 6.20. This animation depicts relative gas speeds in the engine.

## 6.5 Chapter Conclusions

This chapter outlined the basic usage of the GUI, an interface that allows the user to construct a complete model from blocks without any contact with the code. The software offers a variety of simulation tools which aid in the running of groups of experiments while the user is away and use

the results from previous experiments to kickstart the progress of the next. Outputs from the model use the GUI to generate indicator diagrams, acquire virtual sensor data and construct animations that provide a visual perspective to the acquired results. These visual features are unheard of among Stirling engine modelling programs beneath 4<sup>th</sup> order simulations.

# CHAPTER 7. VALIDATION

## 7.1 Theoretical Validations

---

Each of the following tests are conducted within the same environment and aims to show the flexibility of this software to provide results at varied levels of detail and geometry. These early tests allow for a certain level of confidence with regards to the performance of the model, before simulating full scale engines.

### 7.1.1 Steady-State Solid Heat Conduction

Steady-state solid heat conduction was validated using a composite annular ring, the inside was heated to an elevated temperature of 100 C and the outside was cooled to a temperature of 0 °C. The ring was composed of multiple materials, which exhibited different material properties. The equations used for the analytical model are as follows:

$$R_i = \frac{\ln\left(\frac{r_{max,i}}{r_{min,i}}\right)}{2\pi \cdot k_i \cdot L} \quad (110)$$

$$T_{start,i} = T_{end,i-1} = T_{start} + \frac{T_{end} - T_{start}}{\sum R_i} \sum_{i=1}^{i-1} R_i \quad (111)$$

$$T_i(r) = T_{start,i} - (T_{start,i} - T_{end,i}) \frac{\ln\left(\frac{r}{r_{min,i}}\right)}{\ln\left(\frac{r_{max,i}}{r_{min,i}}\right)} \quad (112)$$

where:  $R_i$  = The overall resistance of material layer  $i$

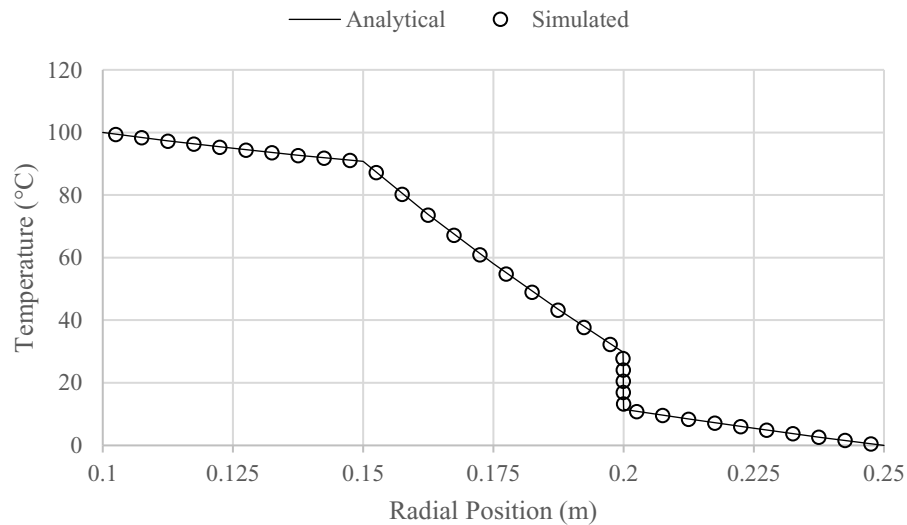
$T_{start,i}$  = The temperature on the inside radius of material layer  $i$

$T_{end,i}$  = The temperature on the outside radius of material layer  $i$

The discretization and material composition are displayed in Table 7.1. The resulting temperature profile appears in Figure 7.1. The steady-state profile which emerges closely matches the analytically derived profile.

**Table 7.1: Steady-state Heat Conduction Validation: Material Properties**

Radius (m)	Material	Thermal Conductivity (W/mK)	Number of Nodes
0.1 – 0.15	Copper	401	10
0.15 – 0.1999	Carbon Steel	43	10
0.1999 – 0.2	ABS Plastic	0.25	5
0.2 – 0.25	6061 Aluminum	176.5	10

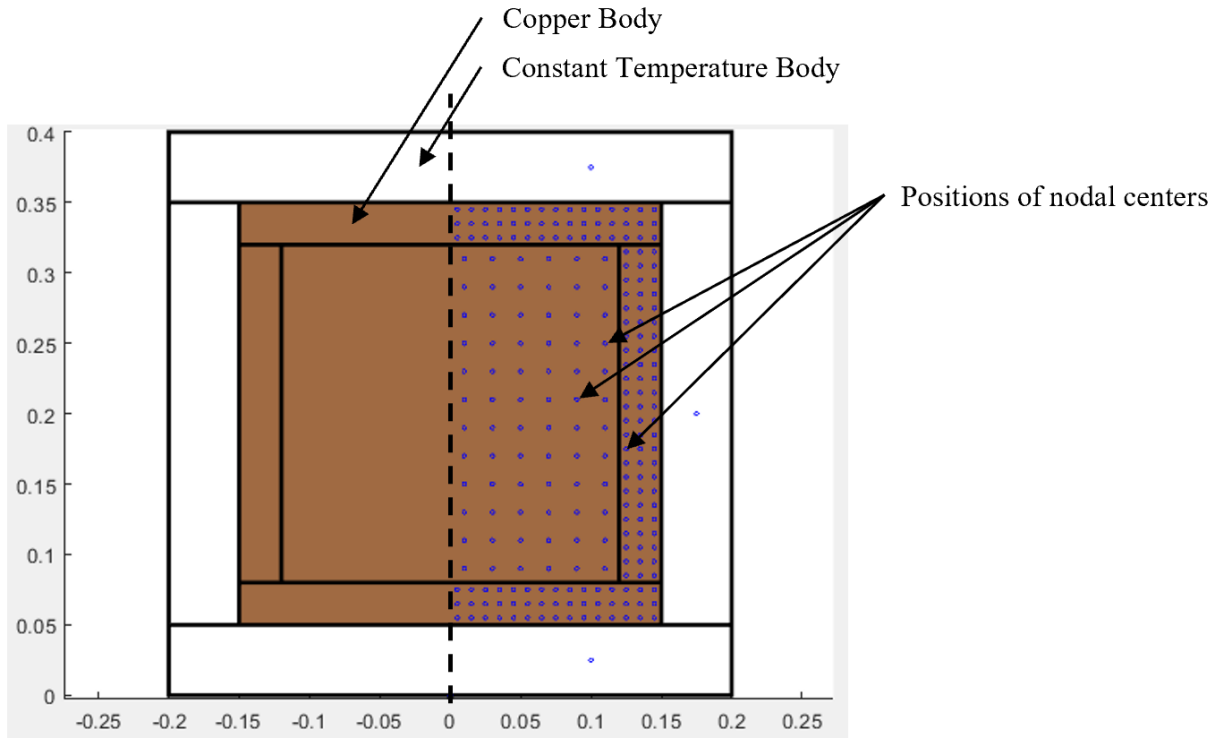


**Figure 7.1: Steady-state temperature profile obtained via heat conduction through a layered annular conductor compared against analytical predictions**

### **7.1.2 Transient Solid Heat Conduction**

Transient solid heat conduction was validated through a comparison of the analytical solution of a cylinder heated by uniform convection. The discretization scheme and material composition are displayed in Figure 7.2 and Table 7.2 respectively. The cylinder of material is 15 cm in radius, 30 cm in length with a 3 cm thick outer layer of nodes of that are 1 cm square and an inner core of nodes of square dimensions of 2 cm square.





**Figure 7.2: Discretization scheme for transient heat conduction test**

**Table 7.2: Experimental properties of Transient heat conduction test**

Property	Value	Unit
Density	8960	kg/m <sup>3</sup>
Thermal Conductivity	401	W/mK
Specific Heat Capacity	0.385	kJ/kg
Surface Temperature	100	°C
Initial Temperature	0	°C
Cylinder Length ( $L$ )	0.3	(m)
Cylinder Radius ( $r_o$ )	0.15	(m)
Small Node Size	0.01	(m)
Large Node Size	0.02	(m)

The analytical profile is produced by the product solution between an infinite plate of thickness  $L$  and an infinitely long cylinder of radius  $r_o$ . For the plate's contribution the solution is derived from 200 terms of the Fourier series [63] using:

$$\theta_{x_{2L} Plate}^* = \sum_1^{\infty} \frac{4}{\pi} e^{-\left(\frac{n\pi}{2L}\right)^2 \alpha t} \sin\left(\frac{n\pi x}{2L}\right) \quad (113)$$

where:  $n = 1, 3, 5, \dots$

The cylinder's contribution is derived from the numerically derived solution of the following equation, which is created from the heat equation in cylindrical coordinates. This is provided to MATLAB's ode45 solver in the subsequent set of equations.

$$\frac{\partial T}{\partial t} = \alpha \left( \frac{1}{r} \frac{\partial T}{\partial r} + \frac{\partial^2 T}{\partial r^2} \right) \quad (114)$$

$$\frac{dT_1}{dt} = \alpha \frac{T_2 - 2T_1 + T_1}{dr^2} + 0$$

$$\frac{dT_i}{dt} = \alpha \left( \frac{T_{i+1} - 2T_i + T_{i-1}}{dr^2} + \frac{1}{r_i} \frac{T_{i+1} - T_{i-1}}{2dr} \right)$$

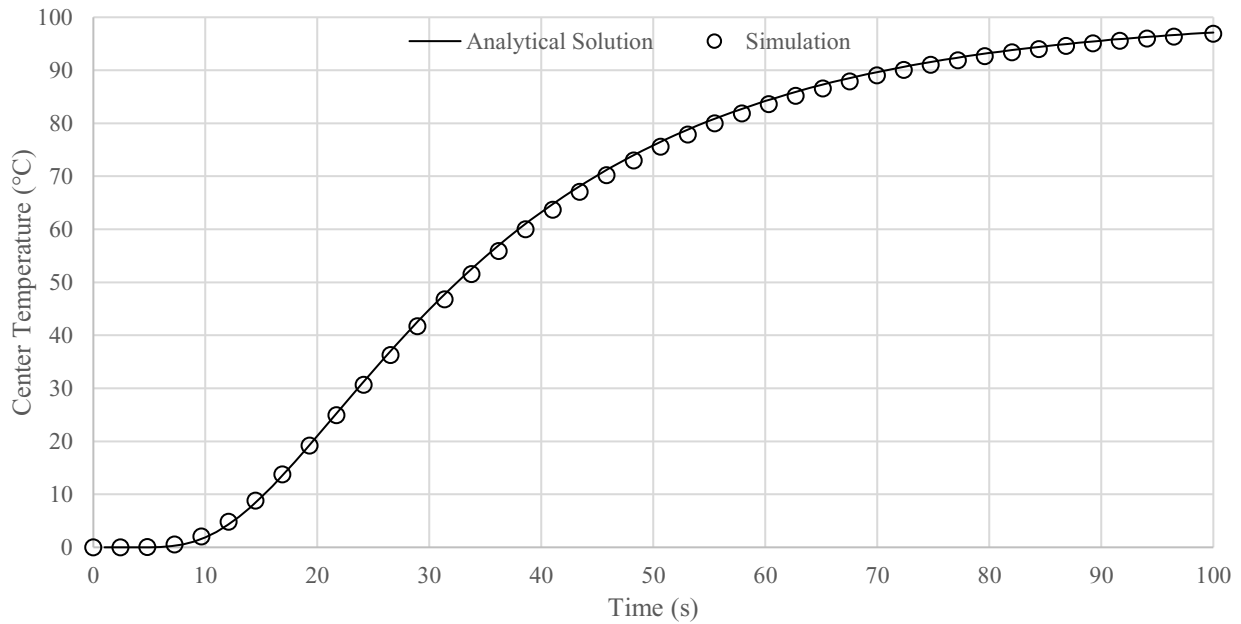
$$\text{Boundary Conditions} \quad (115)$$

$$T_N = 100$$

Initial Conditions:

$$T_1, T_2, T_3, \dots, T_{N-1} = 0$$

Thus, the center temperature of the short cylinder, Figure 7.2, is mathematically determined. A comparison between the analytical and simulated center temperature is shown in Figure 7.3. The simulated transient center temperature matches up closely with the analytically obtained solution.



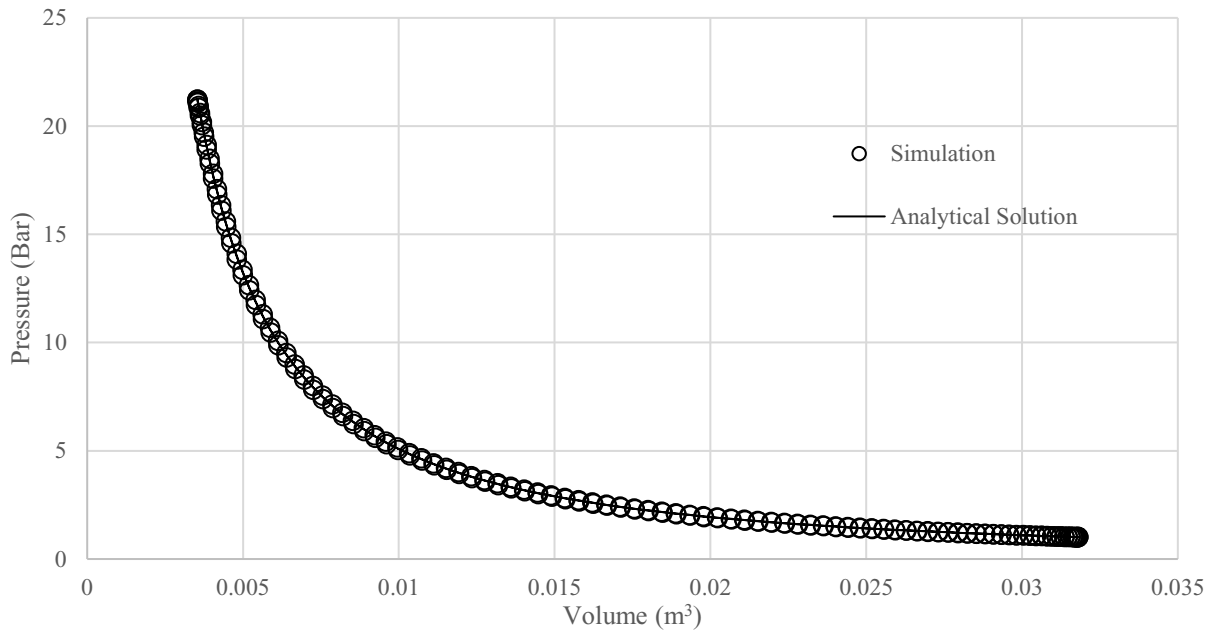
**Figure 7.3: Analytically obtained results vs simulated temperature with time measured at the center of test block.**

### 7.1.3 Adiabatic Compression/Expansion of Gas

The compression of a volume of ideal gas within a perfectly insulated chamber results in a curve in the Pressure vs. Volume space that follows the relationship defined by.

$$\frac{P \cdot V}{T} = \text{Constant} \quad (116)$$

A perfectly insulated chamber was constructed within the program, a moving boundary applied a compression, expansion cycle to the gas. The pressure of the gas was measured and plotted after compressing from atmospheric conditions. The results are plotted in Figure 7.4, along with the simulated curve. The simulated compression-expansion cycle matched up well with the expected adiabatic trend.



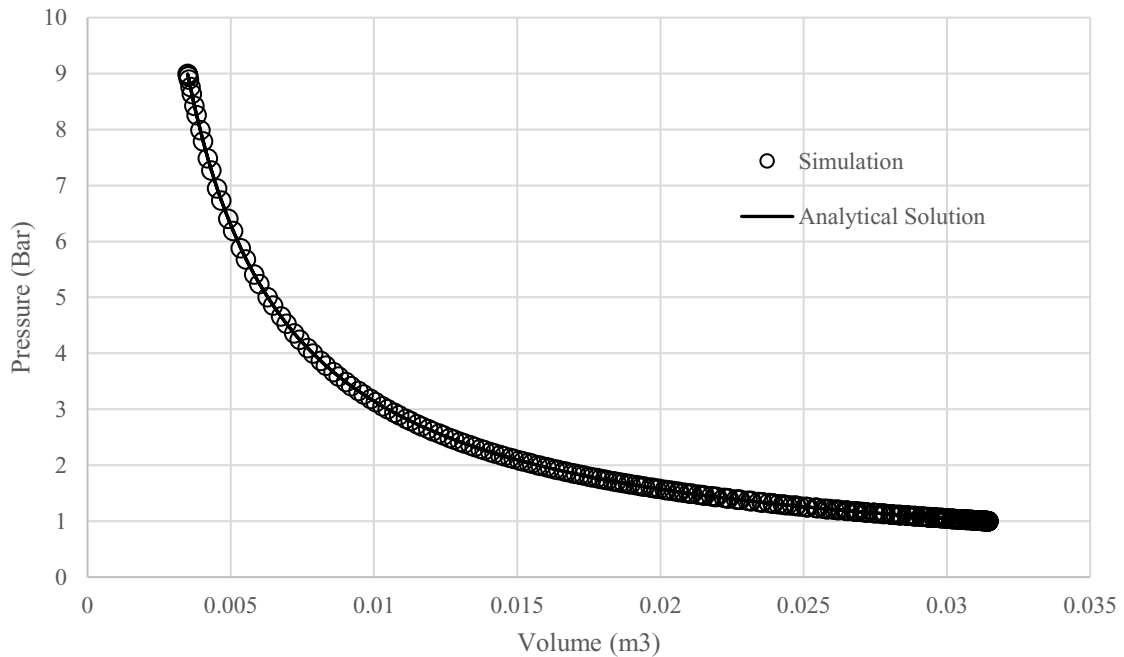
**Figure 7.4: Analytically obtained results compared against simulation results in the case of adiabatic compression/expansion**

### **7.1.4 Isothermal Compression/Expansion of Gas**

The compression of a volume of ideal gas within a perfectly conductive chamber under compression over an infinitely long-time scale results in a curve in the Pressure vs. Volume space that follows the relationship:

$$P.V = Constant \quad (117)$$

A Chamber was constructed out of constant temperature material at a temperature of 298 K, the chamber was cycled at a frequency of 0.01 rpm to allow enough time for the gas to exchange energy with the wall. The initial conditions are used to generate the isothermal curve. The results are plotted in Figure 7.5, along with the simulated curve. The simulated compression-expansion cycle matched up well with the expected isothermal trend.



**Figure 7.5: Analytically obtained results compared against simulation results in the case of isothermal compression/expansion**

## 7.2 Comparison with Experiments

---

It was important to compare with experimental results within the low temperature regime in order to get a correct reflection of the accuracy of MSPM's predictions of the losses and effects which were important to a low temperature engine. Experimental results with suitable detail on the engine geometry could not be found in the literature, thankfully the works of DTECL had produced several low temperature engines with the EP-1 being the latest.

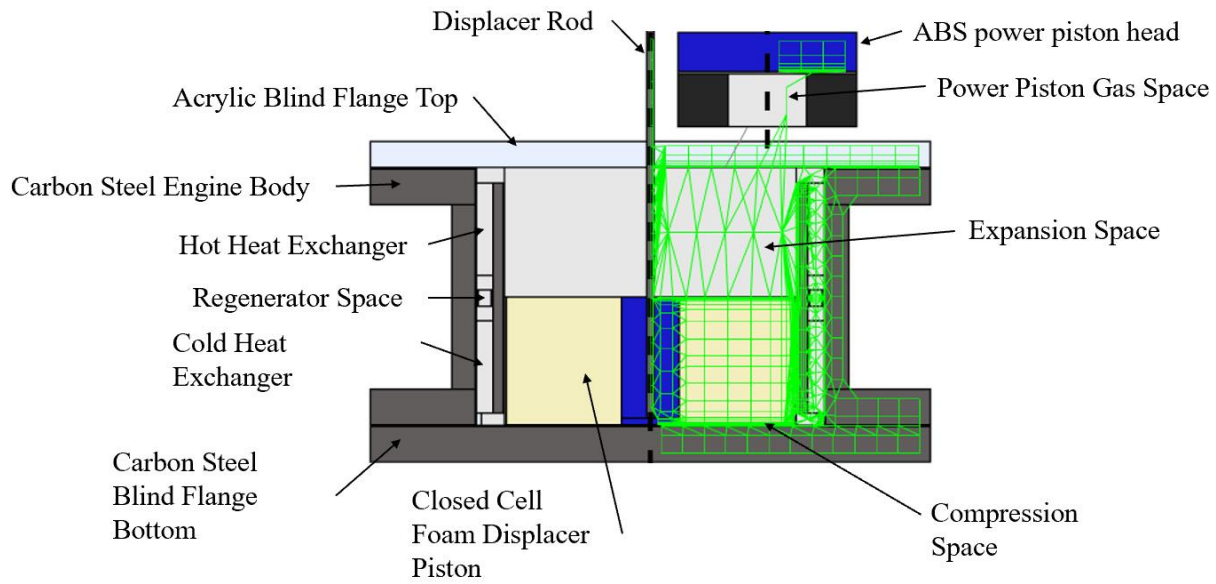
The following validations are a comparison of the model output with values obtained through physical experimentation on the EP-1 engine. These serve to assess the general performance of the model. Unless otherwise noted, error is calculated based on the percent difference in non-dimensional indicated power. This non-dimensional power is derived from non-dimensional pressure and volume, which are defined below:

$$P^* = \frac{P}{P_{avg}} \quad (118)$$

$$V^* = \frac{V - V_{min}}{V_{min} - V_{max}} \quad (119)$$

$$P' = \frac{P}{P_{avg}} \quad (120)$$

The EP-1 as discussed by Stumpf [12] was modified to host a drive train composing of circular or elliptical gears [64]. These studies were intended for the experimental analysis of the effect of non-sinusoidal piston motion on engine performance by Nicol-Seto [64]. Because this model can host mechanical components of arbitrary linear motions, this is an important comparison to see the accuracy of the model under those conditions as this phenomenon is one of the most difficult scenarios to model due being lesser studied oscillatory phenomena. The detailed geometry is found in Appendix C. Within the user interface the engine appears as a 2D cut-away view of the geometry, here the pistons are shown at their lowest point, the default position of the simulator.



**Figure 7.6: EPM engine body geometry as shown in graphical user interface of software**

### **7.2.1 Constant Speed Steady-State Experiments**

The engine's offset power piston is connected using a bridge component such that the geometry may be maintained as closely as possible. The model was run for several combinations of elliptical and circular gears, as outlined in the following table, the motion profiles for each of the following are calculated in Appendix D. All tests are conducted with a charge pressure equal to atmospheric (101,325 Pa) with a hot source temperature equal to 95°C and a cold sink temperature of 5°C. All tests covered a range of speeds from 0.2 – 2.6 Hz. All mechanisms for displacer have a crank to connecting rod length ratio of 6, a value of 2 is applied for the power piston. All tests use a slider crank mechanism, but between the drive shaft either have a set of circular gears (standard) or a set of elliptical gears arranged in either a dwelling cycle (square wave), or to have a minimal velocity cycle (saw wave). Only tests 1-3 were conducted on a physical test engine, 4-6 are expansions upon the existing data set using untested but promising mechanisms. Test 1-3 were performed with the engine driving itself, loaded by a friction brake. The pressure was measured using a pressure diaphragm sensor. The volume of the experimental results is estimated with the bellow volume being calibrated via a shadowgraph technique [64]. Additional tests were also conducted on speeds from 0.2 Hz to 4.0 Hz, a scope beyond the 12 selected experimental tests.

**Table 7.3: EP-1 test sets**

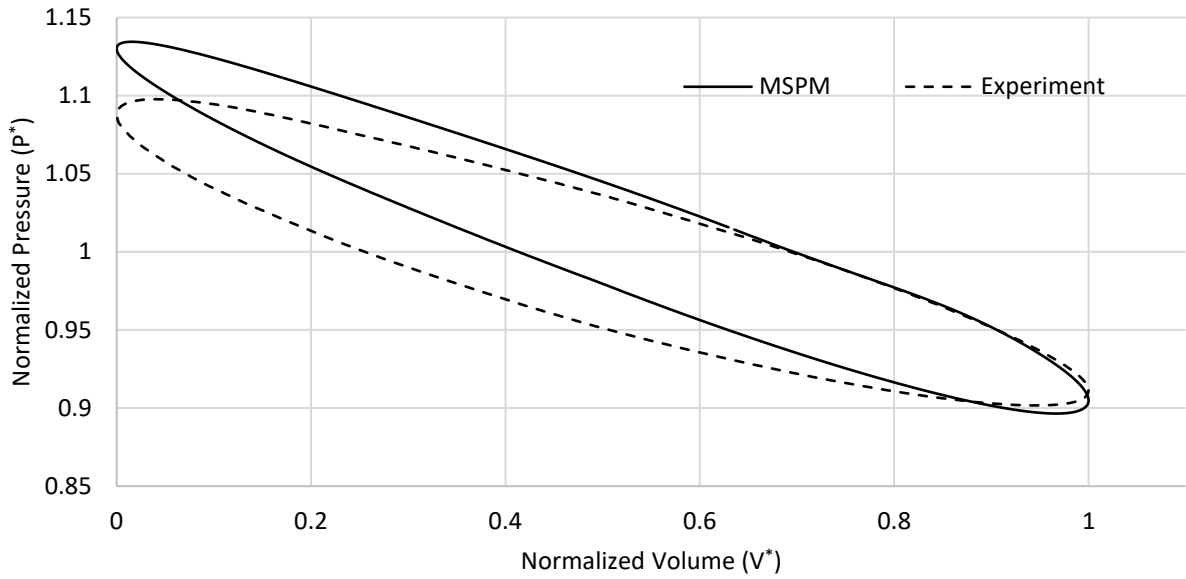
Test Set	Displacer Piston	Power Piston
1	Standard	Standard
2	1/5 Elliptical for square wave, dwelling cycle	Standard
3	1/5 Elliptical for square wave, dwelling cycle	1/5 Elliptical for square wave, dwelling cycle
4	1/5 Elliptical for saw wave, minimum velocity cycle	Standard
5	1/5 Elliptical for saw wave, minimum pressure loss cycle	1/5 Elliptical for square wave, dwelling cycle
6	1/5 Elliptical for saw wave, minimum pressure loss cycle	1/5 Elliptical for saw wave, minimum pressure loss cycle

The mesh density settings applied to both the experimental and SAGE comparisons has an approximate mesh sensitivity related error of 0.8%, these results are shown in Appendix F and is assumed to be representative of all speeds and configurations.

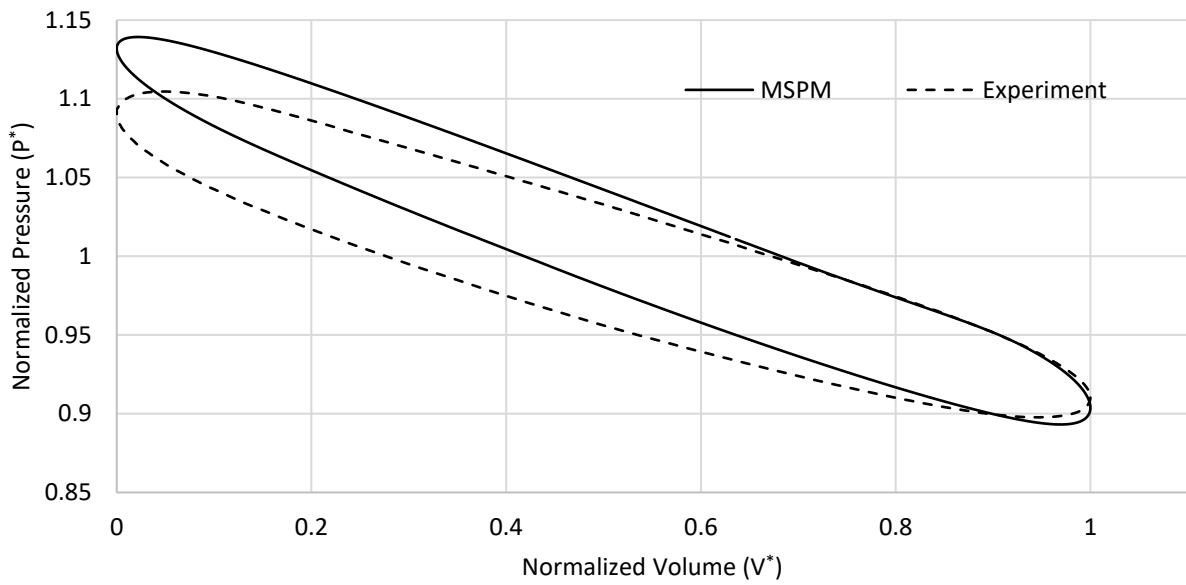
The PV diagrams and thermodynamic powers are compared in terms of non-dimensional pressure and volume as the model was simulated at an average pressure of 101,325 Pascals, while the test engine tended to rest somewhere above or below that value.

Figure 7.7 displays an indicator / pressure-volume diagram from both the experiment and simulation for the same conditions, given the default uncalibrated model output using properties taken from the solid model of the engine that was used for construction. Experimental results are attributed to Nicol-Seto [64].

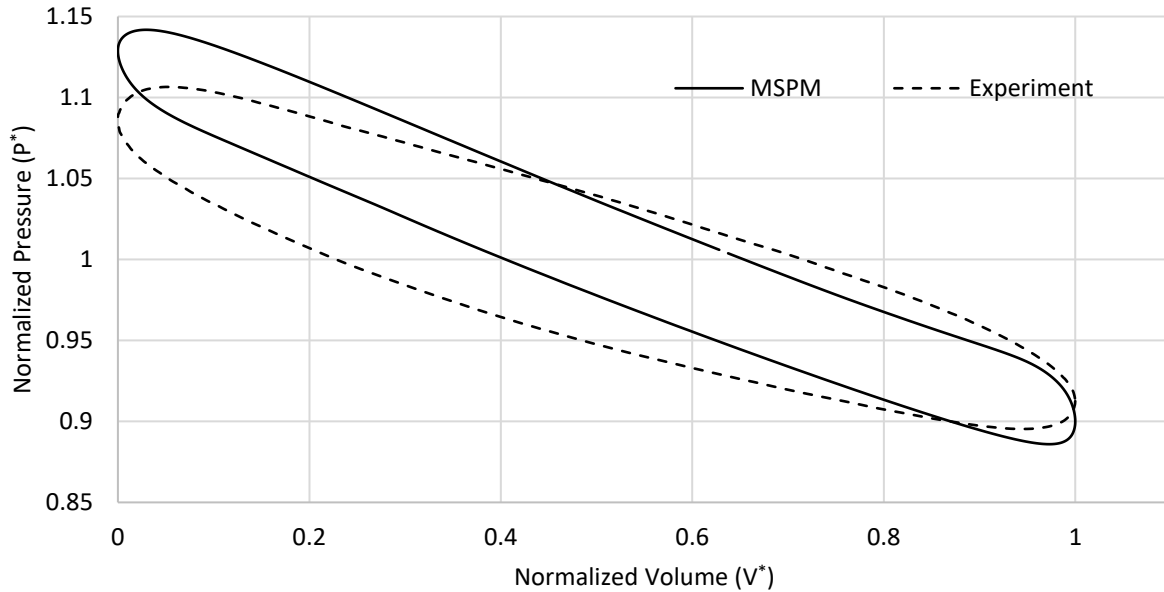




(a) Both in standard arrangement – 1.1055 Hz



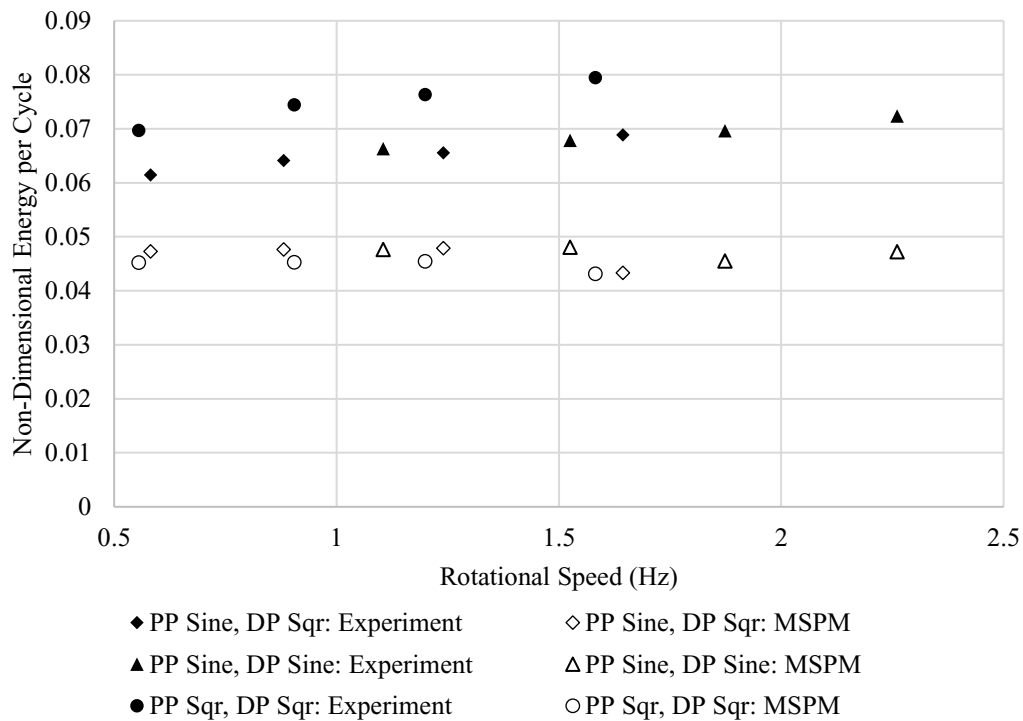
(b) Displacer piston with dwelling cycle, power piston with standard slider crank – 0.8818 Hz



(c) Both with dwelling cycle – 1.1992 Hz

**Figure 7.7: Indicator diagram comparison between EPM-1 experiments and MSPM. (a) DP & PP: Standard at 1.1055 Hz (b) DP: Square Wave Elliptical, PP: Standard at 0.8818 Hz (c) DP & PP: Square Wave Elliptical at 1.1992 Hz**

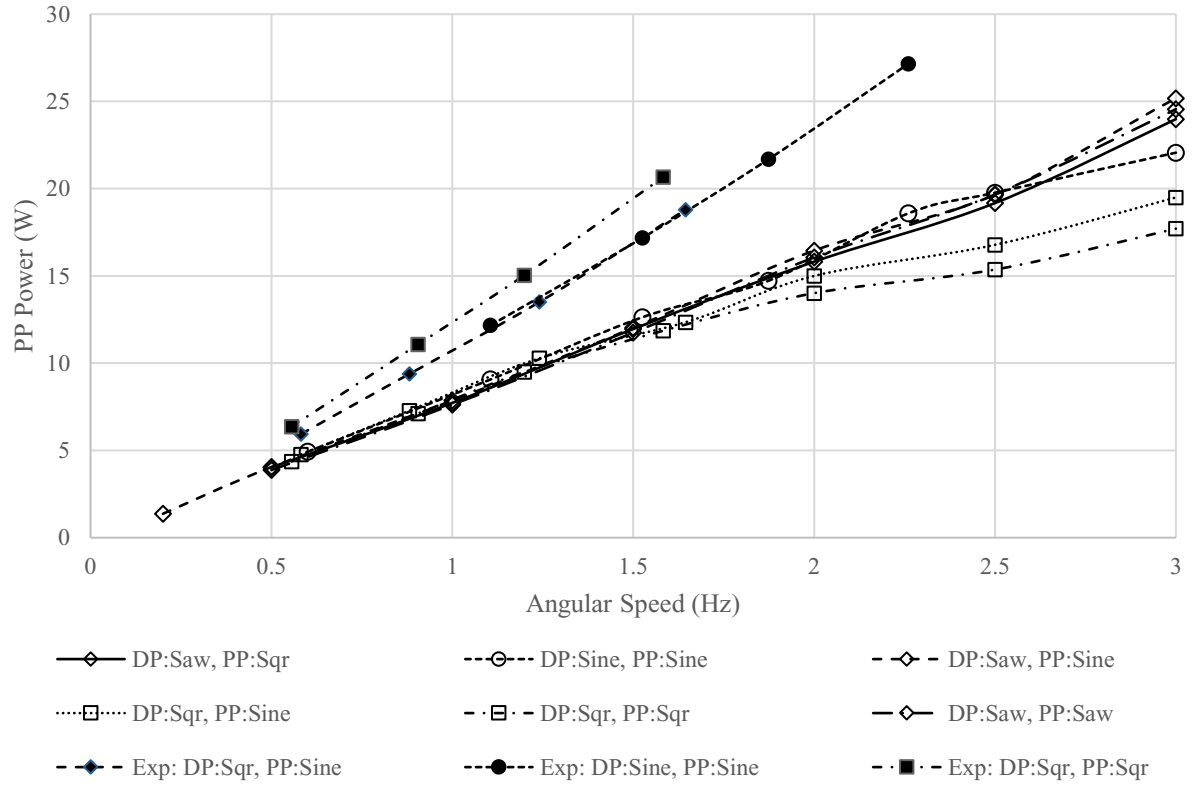
Over the 12 tests considered, MSPM had a maximum error of 43.1% and an average error of 30.6%. The shape of the indicator diagram is close, particularly at the ends where each of the configurations has a distinctive shape. The non-dimensional cycle energy extracted from the 12 matching experiments is displayed in Figure 7.8 below:



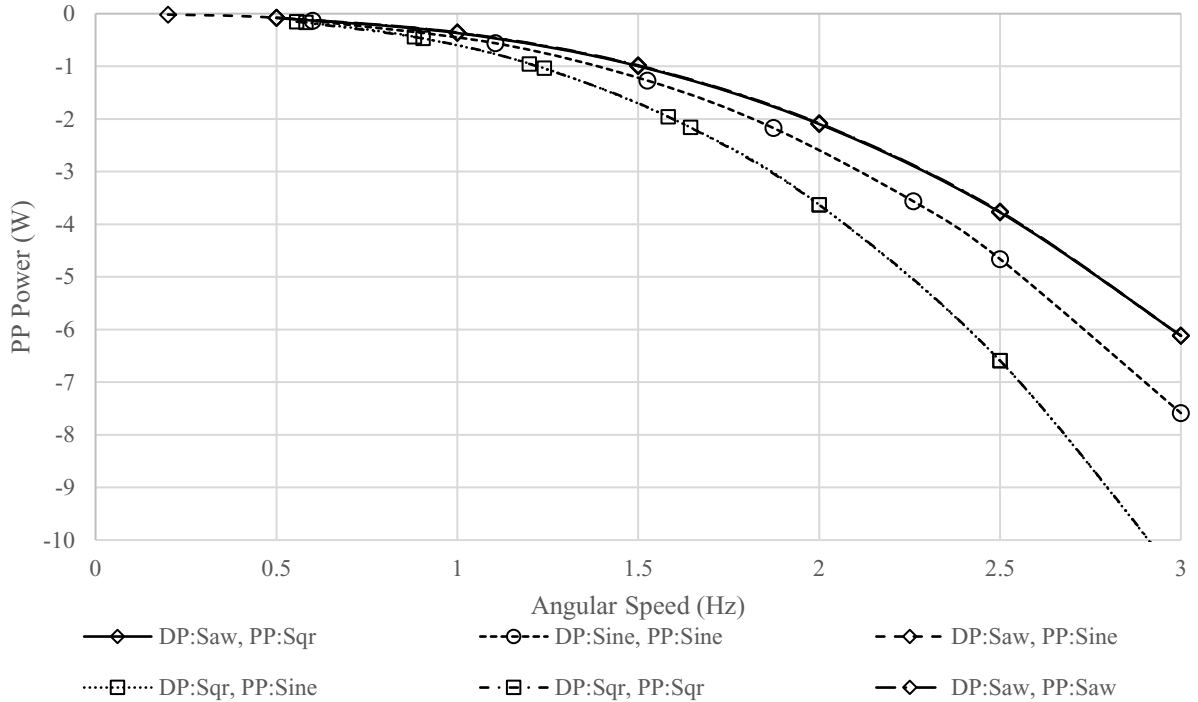
**Figure 7.8: Non-dimensional power for each of the 12 matching experiments**

The main sources of error are likely resulting from 2 areas. Firstly, the EPM-1 features a power piston which is facilitated by a rubber bellows. The bellows serves as a well sealed and low friction alternative to a piston-cylinder design. It is known [12], [14], however, to affect the pressure maximums by expanding and contracting in response to internal pressures. Secondly, the exact heat transfer and friction characteristics of the inline set of finned tubes with directing geometry is unknown. With the directing geometry, the exchangers resemble staggered tube banks, but with the directing geometry being non-conductive the model was compensated by multiplying the total surface area by half. It is likely that the friction characteristics are the most off as MSPM disproportionately disadvantaged motions that included rapid flushing through the heat exchangers with relation to the experiments. Determining the exact characteristics of these heat exchangers is out of the scope of this thesis. This second group of effects is strongly correlated to the broadness of the indicator diagram, which can be seen earlier in Figure 7.7. These combined errors are systematic, amendable by correction to the source temperatures, convection and friction correlations, though it is possible that they mask the errors produced via the modelling assumptions, further study with a more well understood experimental engine is necessary to assess the magnitude of those errors.

Along side the 12 matching tests, a total of 39 tests were conducted including the 3 mechanisms not tested experimentally (DP: Saw PP:Sqr, DP:Saw PP:Sine & DP:Saw PP:Saw), these are shown following on Figure 7.9 and Figure 7.10.



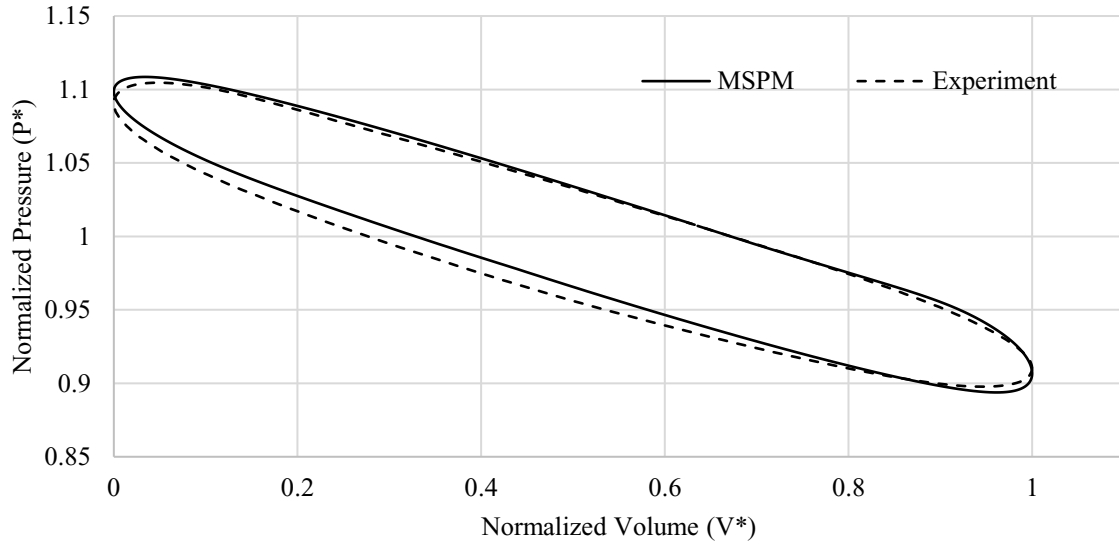
**Figure 7.9: MSPM vs experimental power piston indicated work**



**Figure 7.10: MSPM displacer piston indicated work, experimental version not collected**

The results show that the highest power output will come from configuration 4, with a saw wave on the displacer and the power piston set to a sinusoidal motion, the lowest power is from the tests that used the dwelled motion for the displacer. The fact that the dwelling motion is such a disadvantage is found by inspecting the indicated work of the displacer piston. The configurations become more distinct at higher speeds, seen by the 2<sup>nd</sup> order growth of the flow losses at higher speeds.

The clearest change that could be made to improve the correlation of the simulation with the experiment is to modify the bellows. By observing the indicator diagram, the pressure swing of the simulation is proportional to the compression of the power piston. The effect of compression is tested in a second set of tests in which the stroke is reduced to 84.3% of its original value. This value was determined by assuming that pressure is inversely related to volume and given that the pressure swing was  $18.7 \pm 1.4\%$  among the 12 tests. Dead volume is also increased by 7.8% of original stroke to account for bellow expansion and contraction at stroke extremes. A sample result appears in Figure 7.11 below. Only the 12 comparison experiments were repeated.



**Figure 7.11: Indicator diagram comparison between EPM-1 experiments and simulations with reduction in stroke by 15.7% and increase in dead volume equivalent to 7.8% of stroke. (test: Standard-Standard at 1.1055 Hz shown)**

The resulting error in non-dimensional pressure is decreased to an average discrepancy of 21.9% (from 30.6%) and maximum error of 35.2% (from 43.1%) over the 12 modified tests. The discrepancy in pressure maximums appears to be completely corrected, indicating that the property of indicator diagrams is strongly dependent on volume in the regime of interest. Remarkably the broadness in the Sinusoidal cases is also very close with the 4 sinusoidal tests (test set 1) providing an average error of 19.0% with a maximum of 28.5%. The error in the set with the sinusoidal power piston and dwelling displacer provided an average of 24.0% and the set with both pistons dwelling provided an average error of 31.5%. The error of MSPM increased with dwelling cycles, it is possible that this is due to overestimating the flow losses in the heat exchangers. The overall error appears systematic, indicating that it is likely a problem with an equation used to calculate heat transfer, flow losses or even turbulence; as turbulence was originally designed for mostly sinusoidal motions.

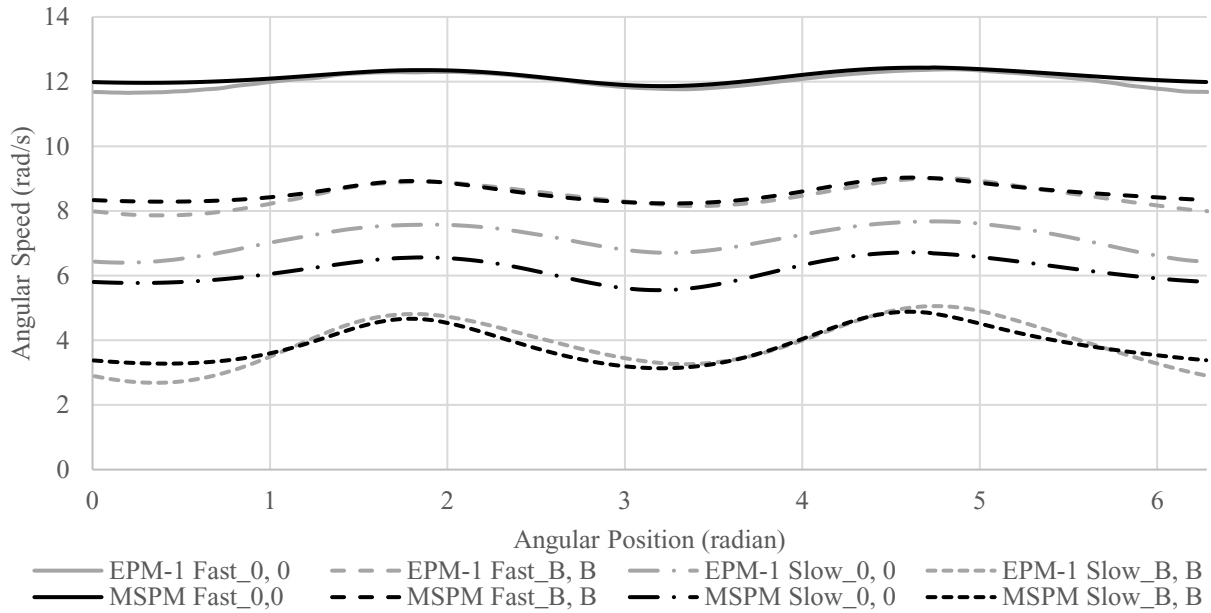
### **7.2.2 In Cycle Speed Variations**

As a complement to the above studies the dynamic response portion of the code was evaluated against actual in-cycle velocity variations for 4 tests. These represent the actual conditions of the tests that were compared against previously, where the simulation was set to a constant speed. These tests are outlined in Table 7.4:

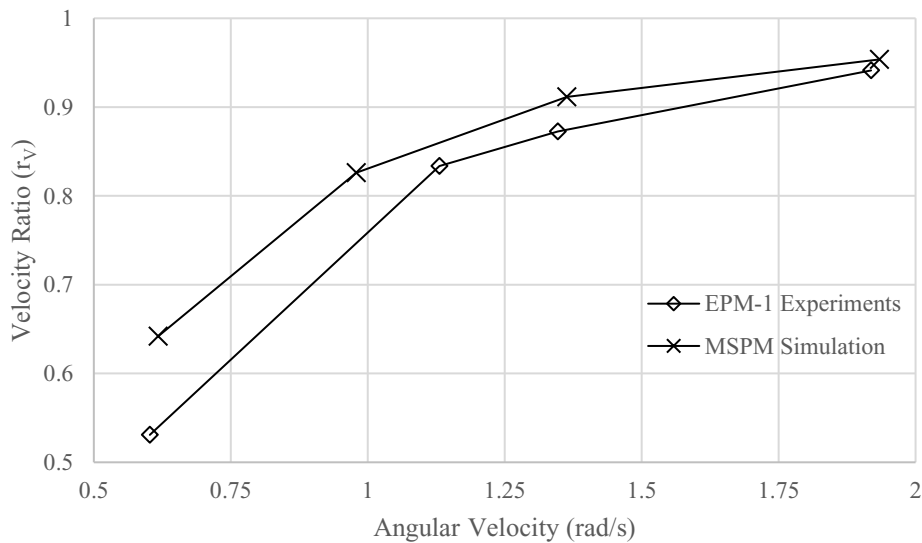
**Table 7.4: Velocity variation experiment. “B” is a reference to the “Box” / square wave. “0” refers to the standard / sinusoidal trial (Elliptical factor of 0).**

<b>Test</b>	<b>Properties:</b>
Fast_0, 0	<b>PP:</b> Standard <b>DP:</b> Standard
	<b>Average Speed:</b> 1.918805 Hz
Fast_B, B	<b>PP:</b> 1/5 Elliptical for square wave
	<b>DP:</b> 1/5 Elliptical for square wave
Slow_0, 0	<b>PP:</b> Standard <b>DP:</b> Standard
	<b>Average Speed:</b> 1.130698 Hz
Slow_0, 0	<b>PP:</b> 1/5 Elliptical for square wave
	<b>DP:</b> 1/5 Elliptical for square wave
	<b>Average Speed:</b> 0.602163 Hz

Here a new non-dimensional number is introduced, the velocity ratio ( $r_v = \frac{\omega_{min}}{\omega_{max}}$ ). This ratio is equal to the minimum instantaneous speed ( $\omega_{min}$ ) divided by the maximum instantaneous speed ( $\omega_{max}$ ). A comparison of the angular velocity curves between the EPM-1 tests and MSPM under the above scenarios is found in Figure 7.12 below, it is also presented in an alternative form in Figure 7.13:



**Figure 7.12: Instantaneous angular velocity for one cycle for both the EPM-1 physical tests and MSPM simulations.**



**Figure 7.13: Velocity ratio results for experiment and simulation for 4 different velocities**

This result indicates that MSPM follows the same trend as the experimental simulations and is within a close margin on the error, on the 3 tests which line up closely in speed, due to its asymptotic nature the error is calculated based on the difference between the speed ratio and the ratio of 1:  $(1 - r_v)$ , the error in this value is largest at the highest speed with a value of 30.6%.



The differences in velocity variations may be caused by various factors, including the differences when and how much power is entering the mechanism at any one time, which is indicated in the differences between the experimental and modelled indicator diagrams. These differences are caused by a variety of effects. Firstly, are flow losses, which would be different in the true heat exchangers. Secondly, leaks and displacer by-passing which are assumed negligible in the model and may be associated with lower power produced by the experimental engine. Thirdly, as the mechanism frictions were not included in these experiments, their effect, specifically, their intermittent and phase dependent effects are not captured. The actual experiment used a set of 3D printed elliptical gears to define the motion of the pistons for both the sinusoidal and dwelled tests which may have inconsistent loss characteristics. While the model maintains its ability to model friction as an angularly varying loss, it was outside the scope of this thesis to determine what these values were in the experimental engine.

### 7.2.3 Sensitivity Studies

A set of sensitivity studies were performed on the EP-1 model (filled at atmospheric pressure, and using sinusoidal motions for both pistons) and the change in the pressure-volume diagram was observed. These studies were performed on a test at 0.5 Hz and a test running at 2 Hz. A summary of the changed properties can be found in Table 7.5 below.

**Table 7.5: Sensitivity Studies (results are colored based on absolute value, -50% uses backwards difference, +50% uses forward difference, ±2% uses central difference for slope calculation)**

Change	0.5 Hz Test, 7.824 J/cycle			2 Hz Test, 1.707 J/cycle		
	Slope (J/cycle/%)			Slope (J/cycle/%)		
	-50%	±2%	+50%	-50%	±2%	+50%
Heat Exchanger Nusselt	8.10E-02	5.35E-02	3.84E-02	8.20E-02	7.10E-02	5.86E-02
Regenerator Nusselt	1.16E-02	8.00E-03	7.40E-03	8.20E-03	6.75E-03	6.40E-03
All Friction Factors	-1.60E-03	-1.50E-03	-1.60E-03	-1.56E-02	-1.53E-02	-1.58E-02
All Solid Conduction	-1.16E-02	-7.50E-03	-5.60E-03	-1.66E-02	-1.18E-02	-1.00E-02
Axial Mixing Coefficient	-1.00E-03	-7.50E-04	-8.00E-04	-6.00E-03	-5.00E-04	-6.00E-03

The property that has the strongest response is the heat exchanger Nusselt number. In both the slow and fast trials reducing the convection coefficient resulted in a drop of roughly 4 Joules per cycle. Increasing it increased the power output by a smaller margin. The increase was more significant on the faster trial, presumably as this trial existed farther down the arctan shaped

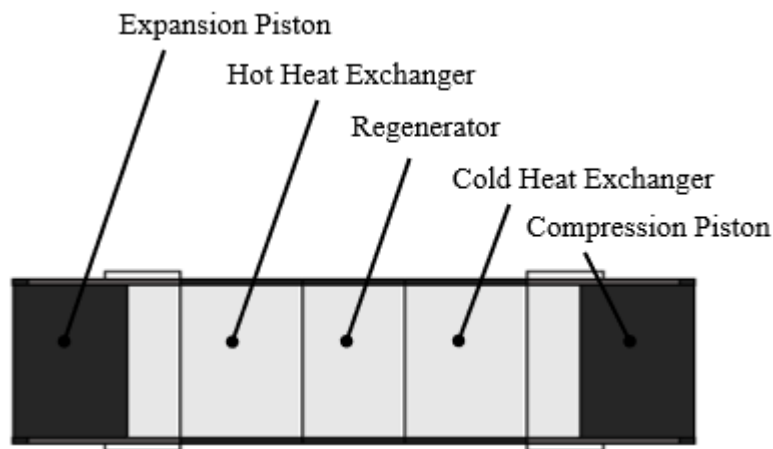
response curve for Nusselt number. The effect was similar when the regenerator including that the up and down values were closer together in magnitude on the fast trial. Increasing/decreasing the solid conductance and axially Nusselt number was more significant on the fast trials, possibly due to the heat exchangers being less capable of compensating for any drops in temperature. The increased effect of viscous friction is inline with the close to order of magnitude increase in friction losses. A sensitivity study like this can help to guide the designer in identifying where improvements may lie with respect to the running conditions of the engine.

## 7.3 Comparison with SAGE

---

### 7.3.1 In High-Temperature, High Speed Context

To assess the model performance against published literature the paper by Hoegel et al [44] was selected. This work was selected because it provided geometry details and results for a set of different engines and included several tests at with lower source temperatures. The premise of the referenced work was to compare optimized designs between low and high-temperature alpha type Stirling engine using the commercial software SAGE, the MSPM model of this engine is shown below:



**Figure 14: Annotated alpha engine for Phase 135°, Source 150 °C test as shown in MSPM**

For the purposes of comparison SAGE is correct, however this is not necessarily true as SAGE is not perfect, and the given results do not provide an experimental validation. The model parameters of the tested opposed piston alpha type engine are shown in Table 7.6.

**Table 7.6: Alpha engine geometrical properties for SAGE comparison [44]**

Component	Value	Unit
Material (heat conductors and regenerator)	Steel	
Bore	0.2	m
Stroke	0.1	m
Phase Angle	Test Dependent	degrees
Angular Speed	16.7	Hz
Heat Source Temperature	Test Dependent	°C
Heat Sink Temperature	40	°C
Mean Pressure	5,000,000	Pa
Working Fluid	Helium	
Heat Exchanger Type	Tube Bundle	
Tube Diameter	0.003	m
Tube Wall Thickness	0.0005	m
Tube Number	Test Dependent	
Heat Exchanger Length	Test Dependent	
Regenerator Cavity Diameter	0.2	m
Regenerator Wall Thickness	0.0075	m
Regenerator Matrix	Random Fibre	
Fibre Diameter	0.00005	m
Porosity	Test Dependent	
Length	Test Dependent	

**Table 7.7: Alpha engine test and specific geometrical properties for SAGE comparison [44]**

Test	HX Tube Number	HX Length (m)	Regen. Porosity	Regen. Length	Power (W)
All at 16.7 Hz					
Phase 135°, Source 150 °C	1054	0.1604	0.9686	0.6709	3,630
Phase 165°, Source 150 °C	1042	0.1441	0.9067	0.01441	4,840
Phase 90°, Source 750 °C	Cold: 277 Hot: 398	Cold: 0.2628 Hot: 0.2592	0.7846	0.03220	96,286
Phase 135°, Source 750 °C	Cold: 233 Hot: 401	Cold: 0.2496 Hot: 0.1895	0.6902	0.009823	104,470
Phase 165°, Source 750 °C	Cold: 330 Hot: 510	Cold: 0.1607 Hot: 0.1108	0.7301	0.009823	45,736

Some information, such as the thickness of the cylinder walls is missing. Therefore, a thickness of 7.5 mm is also selected for those points. All components including pistons and walls are assumed to be steel. The results from the 5 tests conducted are presented in the following table:

**Table 7.8: Output power comparison between MSP and SAGE simulations at 16.7 Hz**

<b>Test</b>	<b>MSPM Output Power (W)</b>	<b>SAGE Output Power (W)</b>	<b>Error (%)</b>
Phase 135°, Source 150 °C	1,484	3,630	59.1%
Phase 165°, Source 150 °C	3,507	4,840	27.5%
Phase 90°, Source 750 °C	73,146	96,286	24.0%
Phase 135°, Source 750 °C	72,478	104,470	30.6%
Phase 165°, Source 750 °C	33,734	45,736	26.2%

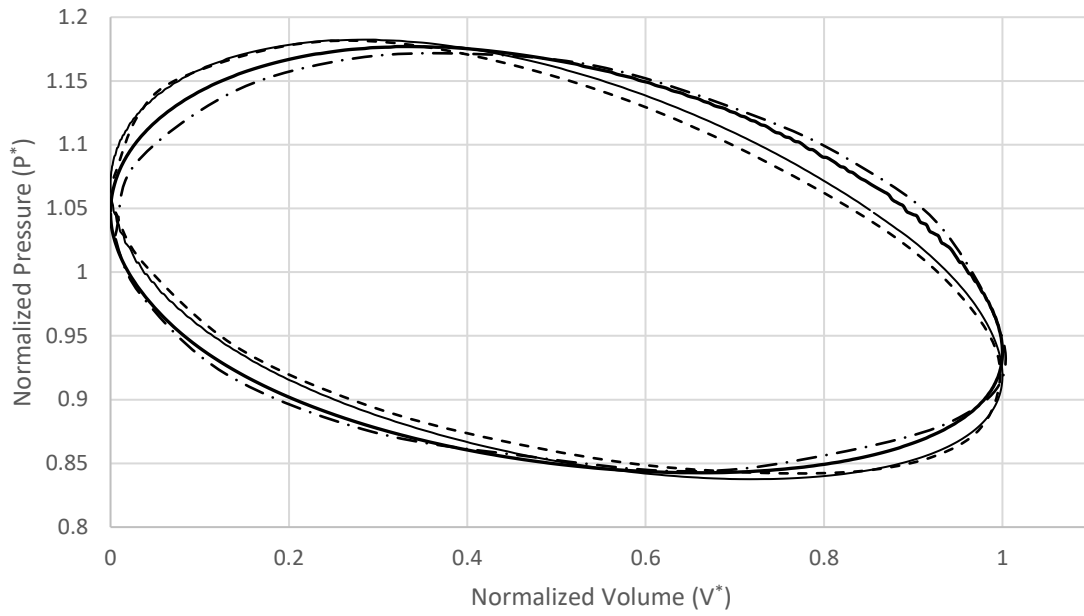
The PV diagrams for each of the tests are compared in Figure 7.15. Note that pressures and volumes are converted to their non-dimensional form.

Normalized Pressure:

Normalized Volume:

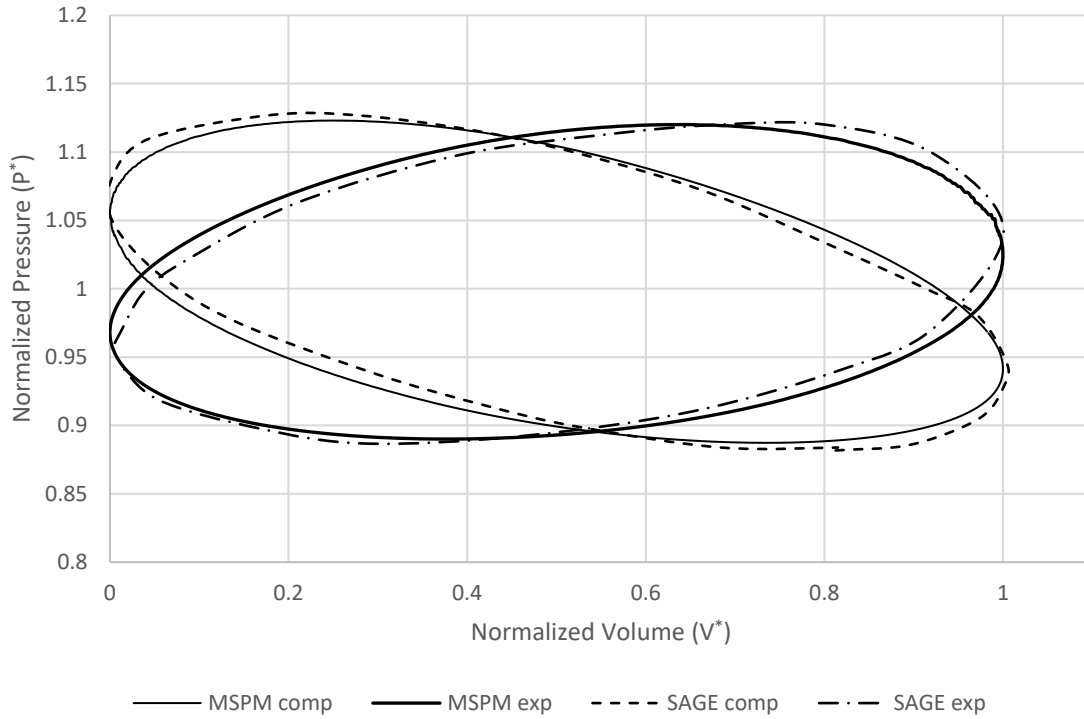
$$P/P_{mean}$$

$$V/V_{max}$$

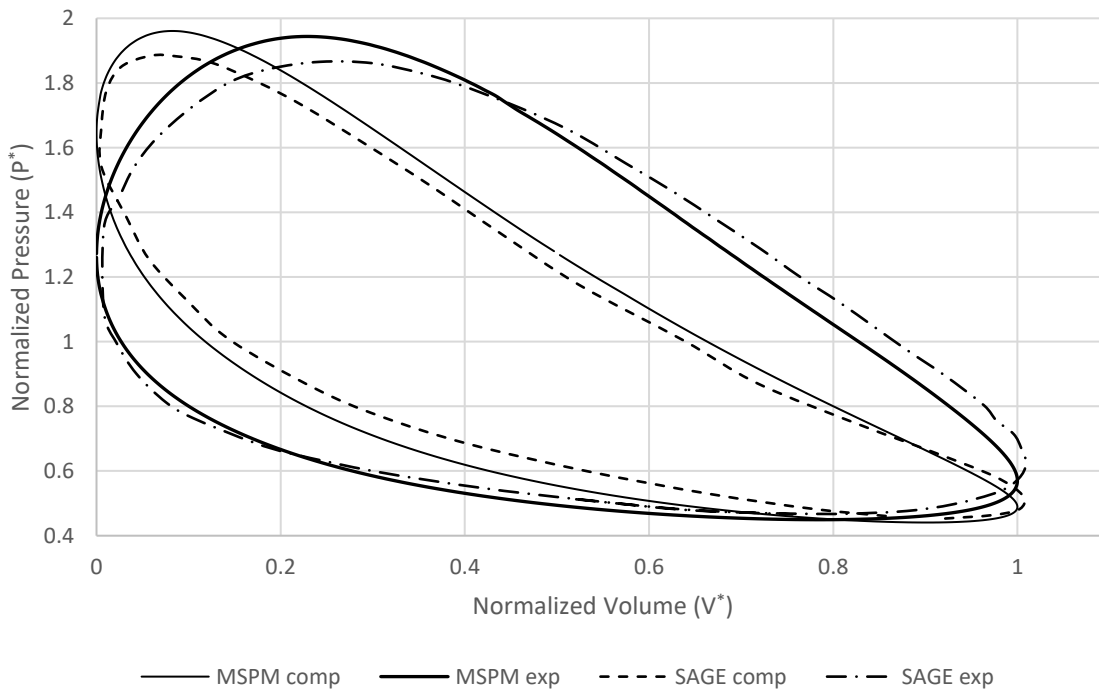


— MSPM comp    — MSPM exp    - - - SAGE comp    - · - SAGE exp

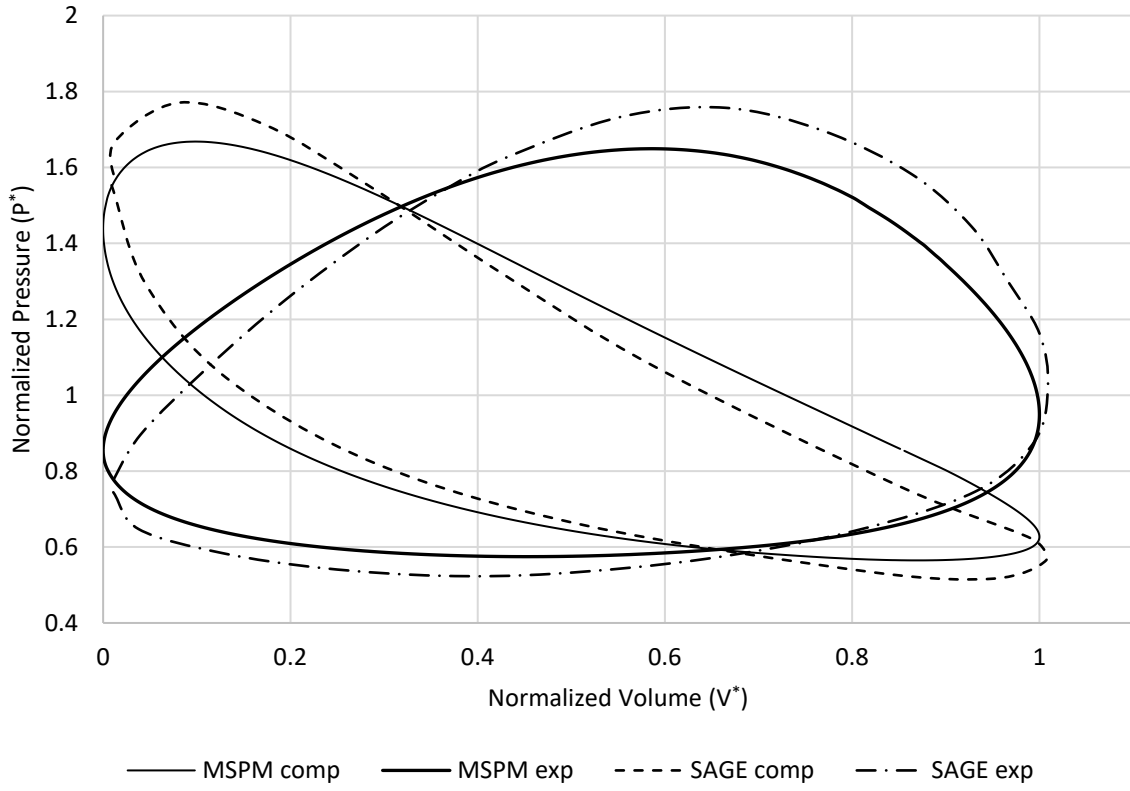
Phase 135°, Source 150 °C



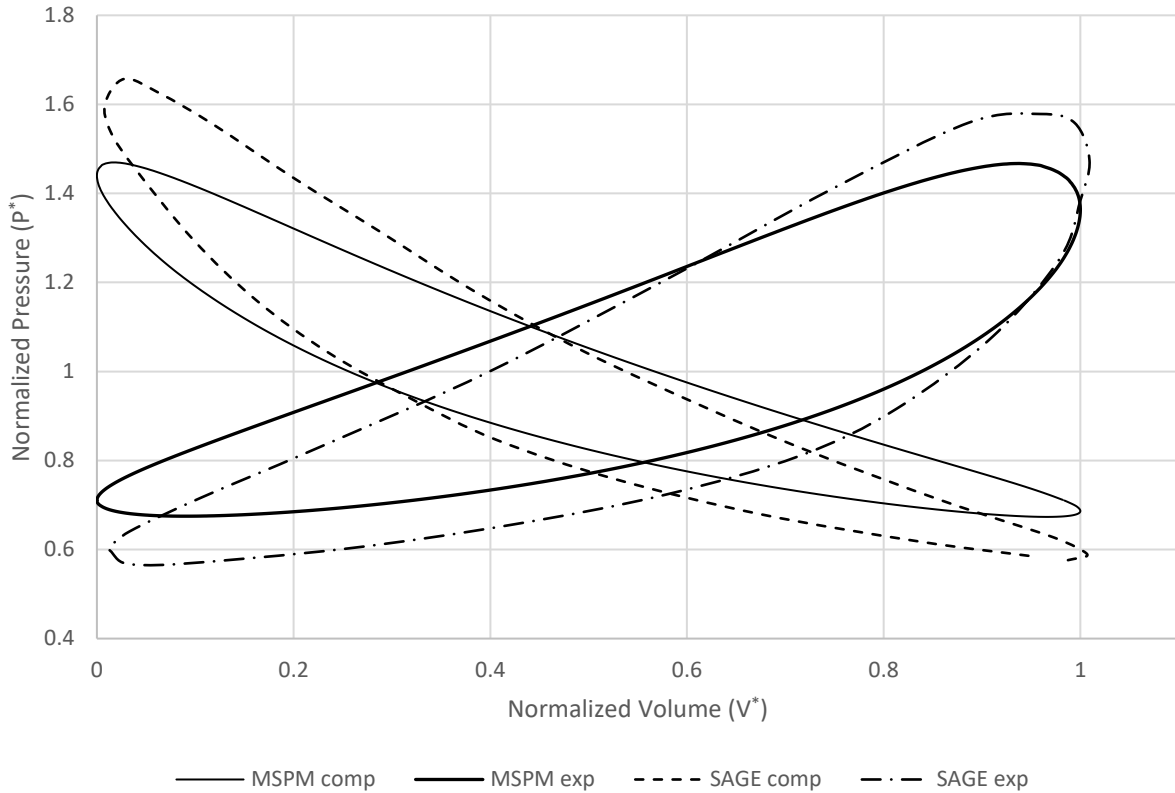
Phase 165°, Source 150 °C



Phase 90°, Source 750 °C



Phase  $135^\circ$ , Source  $750^\circ\text{C}$



Phase 165°, Source 750 °C

**Figure 7.15: Pressure – volume diagram comparison between MSPM and SAGE, data extracted from: [44]**

There are several observations that are made about the above figures. Firstly, the pressure maximums are reduced, which indicates that MSPM underpredicts – relative to SAGE – the temperature performance of the engine given that both models should have the same motions. This is likely due to a combination of two things: different Nusselt number correlation in the heat exchangers and the neglect of tortuosity. MSPM uses the same turbulent correlations as SAGE but could not adapt SAGE’s complex scheme for determining the laminar Nusselt number of its tube bank heat exchangers which involved compression and advection driven flows [35] in complex form. Also, it is not known what values SAGE uses for minor losses at component transitions as these are not shared in the documentation. In addition, MSPM does not include tortuosity in its analysis, which approximates the curviness of a fluid streamlines through a medium, this alone would result in lower regenerator performance, as incorporating tortuosity would result in a higher effective speed, yet similar residence time in the regenerator. Additionally, SAGE uses a real gas representation [35], which under high pressures approximates the divergence



of real gases from ideal ones, in particular on the higher temperature and higher phase runs where the limiting factor in the short regenerators may be the local Nusselt number.

### 7.3.2 In Low-Temperature, Low-Speed Context

The following tests expand upon the previous tests, which though useful are not running at speeds or temperatures for laboratory engines produced by DTECL which MSPM was designed for. These new experiments feature the following modified properties, properties not listed are the same as those found in Table 7.6. It should be said that, though results are compared against SAGE, it is not clear whether SAGE models correctly the low-temperature, low-speed regime. Results from Table 7.10 indicate that MSPM and SAGE predict a similar trend but diverge at higher speeds.

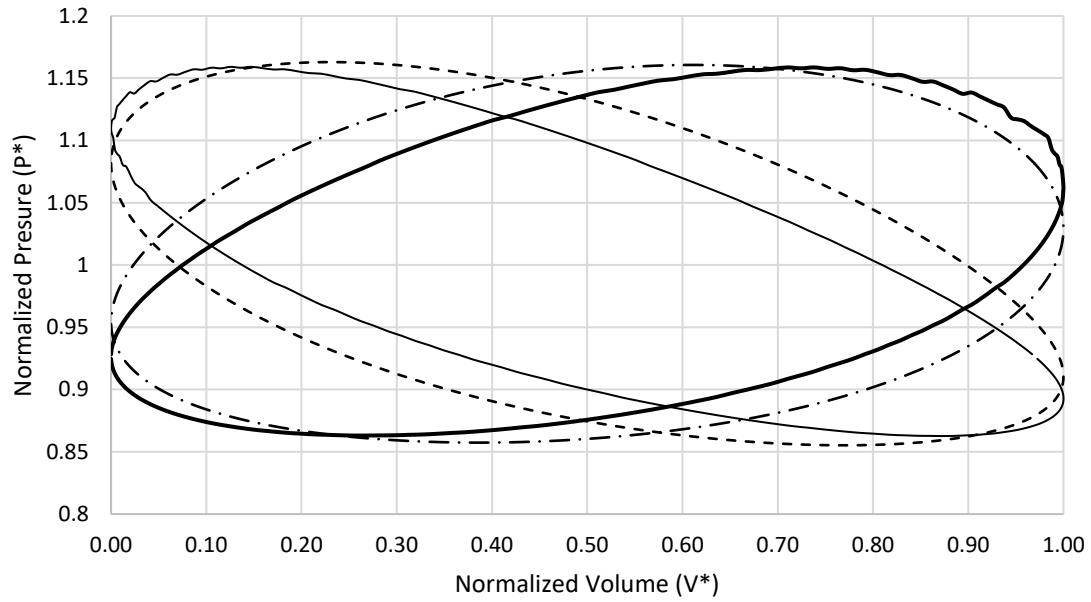
**Table 7.9: Unique properties for low-speed/low-temperature alpha engine SAGE comparisons**

Component	Value	Unit
Phase Angle	160	degrees
Angular Speed	0.5-5	Hz
Heat Source Temperature	95	°C
Heat Sink Temperature	5	°C
Mean Pressure	101325	Pa
HX Tube Number	459.6	
HX Length	0.2348	m
Regenerator Porosity	76.47	%
Regenerator Length	0.001350	m

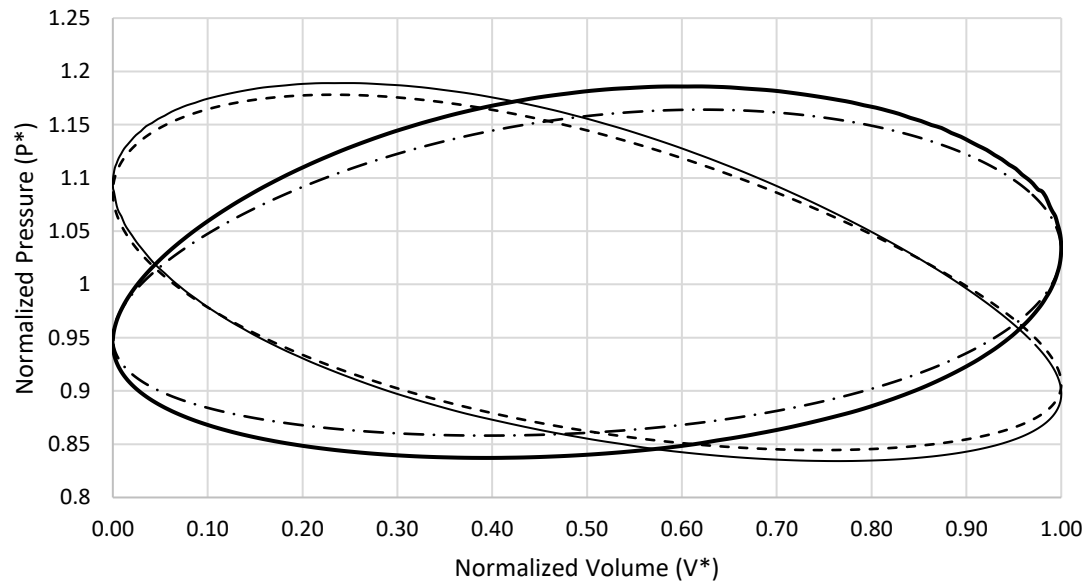
**Table 7.10: Test results for low-speed/low-temperature alpha engine SAGE comparisons**

Test (Hz)	Non-Dimensional Power (W)		Error (%)
	SAGE	MSPM	
0.5	0.015	0.006	59.9%
1.0	0.027	0.028	2.5%
1.5	0.053	0.054	3.0%
2.0	0.079	0.083	4.0%
2.5	0.101	0.107	6.3%
3.0	0.111	0.123	10.2%
3.5	0.107	0.125	16.4%
4.0	0.078	0.107	36.9%
4.5	0.016	0.066	301.5%
5.0	-0.077	-0.004	94.3%

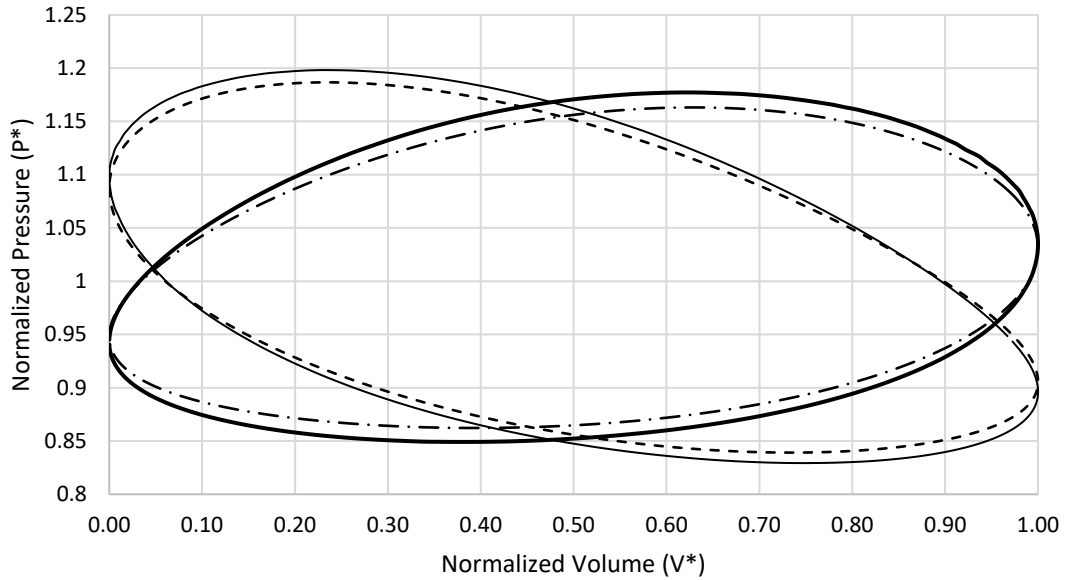
MSPM comp     MSPM exp    Normalized Pressure:    Normalized Volume:  
 SAGE comp     SAGE exp     $P/P_{mean}$      $V/V_{max}$



(a) 0.5 Hz



(b) 3 Hz



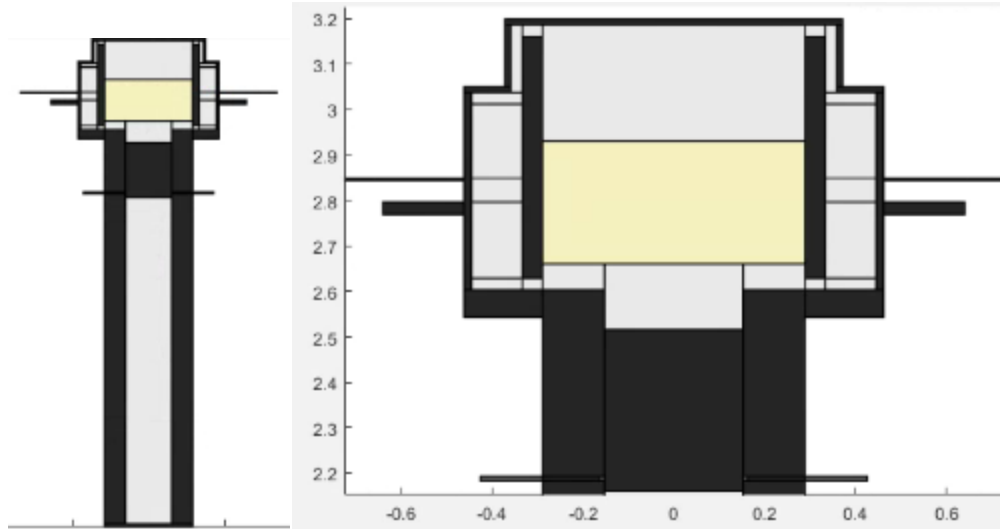
(c) 5 Hz

**Figure 7.16: Pressure – volume diagram comparison between MSPM and SAGE for low-speed, low-temperature, low-pressure case.**

The results indicate that MSPM, relative to SAGE, sometimes overestimates the temperature ratio that the gas achieves – via the broadness of the indicator diagram. At the same time MSPM underestimates, by a factor of roughly 2 the pressure loss through the components – via the pressure difference between the peaks of expansion and compression curves. Here the differences between SAGE’s representation of the laminar Nusselt and friction loss coefficient are seen to a greater extent.

## 7.4 Optimization Studies

MSPM was tasked with optimizing a new engine to be created by the laboratory. The proposed engine would be in a beta-style configuration, as shown on Figure 7.17, with the power piston residing on the cold side of the engine. During optimizations only the stroke of the pistons and size of the power piston was kept constant while the diameter of the displacer and heat exchangers was modified. During these tests the target parameter was volumetric power density  $\dot{e} = \dot{E}_{shaft}/V_{engine}$  which gave rise to reasonable engines. The properties of the running conditions and heat exchangers for an engine without a regenerator are included in Table 7.11.



**Figure 7.17: Depiction of beta-layout engine to be optimized. Large volume beneath engine represents the crankcase, the additional bodies jutting out of the engine are added to prevent the optimizer from making certain features too small leading to instabilities in the solution. (these bodies will overlap before the gas body becomes too small)**

**Table 7.11 Test properties for Beta-Engine Optimization**

<b>Running Properties</b>	<b>Value</b>
<b>Running Speed</b>	1 Hz
<b>Phase</b>	90°
<b>PP &amp; DP Stroke</b>	0.254 m
<b>PP Diameter</b>	0.3048 m
<b>Fill Pressure</b>	1 → 6 atm
<b>Mechanism Efficiency</b>	0.8 → 0.9
<b>Heat Exchanger Properties</b>	<b>Bell AW450085060</b>
<b>Material</b>	Aluminium Alloy
<b>Air Gap Between Water Channels</b>	0.00635 m
<b>Water Channel Width</b>	0.003175 m
<b>Wall Thickness</b>	0.0008128 m
<b>Surface Roughness</b>	1e-6 m
<b>Fin Period</b>	0.0023314 m
<b>Fin Thickness</b>	0.00021195 m

**Table 7.12 Geometrical properties for Beta-Engine Optimization, Optimized for Maximum Power vs Gas Volume (1 Hz, filled with air)**

<b>Design Variables</b>	<b>Post Optimal Values</b>					
Charge Pressure (atm) / Mechanism Efficiency	Heater / Cooler Volume	Swept / PP Volume	HX / Swept Volume	Compression Ratio	PV Power / Shaft Power	Flow Loss / PV Power
2 atm / 0.8	1.088	5.755	0.434	1.116	0.455	0.058
4 atm / 0.8	1.142	4.010	1.140	1.104	0.475	0.050
6 atm / 0.8	1.179	5.894	1.339	1.065	0.470	0.117
2 atm / 0.9	1.095	5.261	0.696	1.099	0.783	0.090
4 atm / 0.9	1.037	4.879	0.938	1.094	0.755	0.073
6 atm / 0.9	1.178	5.860	1.347	1.091	0.678	0.089
2 atm / 1.0	1.071	5.204	0.824	1.094	0.985	0.085
4 atm / 1.0	1.091	4.760	1.071	1.090	0.986	0.067
6 atm / 1.0	1.069	4.722	1.069	1.090	0.978	0.083

From Table 7.12, it can be observed several clear trends: Optimal Heater/Cooler volume is not strongly related to either mechanism efficiency or fill pressure. Optimal swept volume over power piston volume hovers around 4-6. Optimal heat exchanger volume vs swept volume ratio hovers around 1 at low temperatures. Compression ratio is slightly greater than which is predicted by Kolin [65] equal to 1.082 for temperatures of 95 and 5 Celsius. The ratio of PV power / shaft power is strongly dependent on the mechanism efficiency falling on the line  $2.5831\eta_{mech} - 1.5957$  with an R value of 0.999. The extra mechanism losses are due to energy transmitting back and forth to the engine to compress or expand against the pressure regime or fight flow losses. With regards to flow losses, it appears to be good practice to ensure that flow losses never amount to more than 10% of PV.

An improvement was proposed to optimize the engine for maximum efficiency as the above tests featured an average efficiency on the order of 1% and would overload the available heating supply within the test laboratory. As a reference the Carnot Efficiency is on the order of 24%. As the expected range of fill pressures is between 1 to 10 atmospheres an engine will be designed to be optimum at 5 atmospheres. It was determined that a mechanism effectiveness of 80% could be used to conservatively model the mechanism currently in the design phase, though it is likely to be in the range of 85%. The addition of a regenerator and insulation of the engine is known to strongly effect the efficiency. Also, it is proposed that thin, large cross-section heat exchangers are desirable for maintaining good heat exchange without much pressure drop. Thus, the new engine

features a HX stack which is detached from the engine such that the length can be optimized independent of the displacer cylinder length, at the introduction of some additional dead volume. Through previous investigations a regenerator porosity of 95% is expected to be close to the optimum value, the stainless steel randomly oriented fibres are 0.1 mm in diameter, in line with coarse steel wool. The results of these further optimizations are shown on Table 7.13.

**Table 7.13 Test properties for Beta-Engine Optimization (listed component volumes are for available gas volume only, heat exchangers have a surface area to volume ratio of about 1.52 m<sup>2</sup>/Litre, 1 Hz, 80% efficient mechanism, wire diameter of 0.1 mm in regenerator, HX volume is for heater and cooler only, with air if not otherwise indicated)**

Details	Power (W)	Heater / Cooler Volume	HX / Swept Volume	Regen. / Swept Volume	Swept / PP Volume	Eff. (%)
<b>Power Density</b>						
90% porosity regen.	328.4	1.074	0.809	0.0363	6.372	2.18%
95% porosity regen.	336.3	1.054	0.706	0.0680	6.579	2.12%
Insulated, 95% porosity regen.	431.7	1.053	0.660	0.0653	6.673	3.70%
As above with 24" DP	416.0	0.958	0.554	0.0965	5	4.95%
Insulated, offset HX, 95% por. regen.	427.5	1.003	0.478	0.0882	5.462	4.72%
<b>Efficiency</b>						
Insulated, offset HX, 95% por. regen.	407.5	0.9488	0.567	0.142	4.925	5.55%
As above with 24" DP	329.0	0.838	0.842	0.383	5	6.20%
Insulated, offset HX, 95% por. regen. (Optimized with Helium)	486.15	0.930	0.5515	0.180	4.975	8.02%

Based on the results in Table 7.13 several observations can be made. Firstly, efficiency strongly correlates with the size of the regenerator. Secondly, well insulating the engine body can improve the efficiency by several percent and will reduce the requirements on HX size vs swept volume. Relative size of both heat exchangers is consistent among trials. Peak power density and peak efficiency occur at similar but notably different conditions. The final test, which featured an engine filled with helium resulted in increased power with very similar geometry. The power increase is largely attributed to an increase in efficiency from reduced flow losses; 33% lower than the optimal point for air. This is created by reduced viscosity in similarly sized heat exchangers and regenerators.

# CHAPTER 8. CONCLUSIONS

## 8.1 Conclusions

---

The model presented here predicts the complex phenomenon present in Stirling engines. The following subsections will discuss the model through the lens of the project goals.

### 8.1.1 Create a Combined Mechanical and Thermodynamic Model for Low-Temperature Stirling Engines

This was accomplished through the alternating coupling of the gas and kinematic mechanical system. The gas system was solved as a one-dimensional pipe network around the assumption of equal gas pressures, which prevented the emergence of acoustic instabilities and allowed each step to be solved using a larger timestep than the compressible simulation would allow. The mechanical system using an alternating coupling allows the mechanical system to respond quickly to changes in the gas system, while allowing the gas network to use deterministic positions and velocities.

### 8.1.2 Ensure the model is User-Friendly and Intuitive

While not an engineering issue, ease of use is an important component of the selection and further development of any software tool, this model is no exception. To support the user, this model features a graphical user interface (GUI) which allows the user to define engine-geometry in a physically viable context. The model may include as much detail as required, and internally develops the connectivity of the user defined blocks. Other features such as: a drop-down menu for editing object properties, an animation tool for visualizing motions and mesh visualization are included to support the user. Additionally the MSPM software package uses no external MATLAB toolboxes and requires only the MATLAB editor to run.

MSPM natively creates animations of the view-screen after simulation, which are much easier to inspect than data-structures, which it also produces. The user can also place generic point and line sensors in addition to indicator diagram generating sensors. These output features may allow the model to serve as a explanation or presentation tool.

### **8.1.3 Validate the Model against Experimental and a well Established Numerical Model**

The model is tested against a variety of classical experiments to check the correctness of different aspects of the code in isolation. In these tests MSPM matched up well with analytical results, these tests show that the fundamental aspects of the model are correct, and that the numerical method used converges to the expected trend.

When compared against the EP-1 (in lab engine) under-steady-state conditions without calibration, MSPM developed a maximum error of 43.1% with an average error of 30.6% over the 12 tests with speeds ranging from 0.56 to 2.26 Hz. The model was calibrated through modification of the compression to match the observed variance in pressure ranges which reduced the maximum and average error to 35.2% and 21.9% respectively. The remaining error was systematic and consistent among all the speeds, this indicated that the actual error may be in the effectiveness of the heat exchangers to expose the air to actual source temperatures. This fine perspective on the heat exchangers is possible in MSPM but was not explored in detail here.

When compared against SAGE, a well established and accurate numerical code, 5 tests were extracted from Hoegel [44], which modelled an opposed alpha type engine. Over the 5 tests, in which error is calculated relative to SAGE's output indicated power, MSPM developed a maximum discrepancy of 59.1% with an average of 33.5%. The cycle speed for these tests was 16.7 Hz. It was concluded that the error was largely due to differences in convection and the exclusion of tortuosity from MSPM as it stands currently. Each of these tests were tested with different geometrical and phasing parameters.

A second batch of SAGE tests were included, which were modelled in-house but at the reduced speeds and temperatures common to low-temperature Stirling engines. The two models followed a similar trend in power however diverged at higher speeds.

Finally, MSPM was used to optimize a beta engine. The resulting engines displayed consistency in the relative size of certain components and informed the design of a new engine project. The information gained from the model included optimal geometrical sizes, expected power, anticipated supply power and expected losses due to friction.



## 8.2 Sources of Error

---

The following sub-sections outline specific areas where for a variety of reasons errors are introduced in the name of simplicity or computational efficiency. The possible sources of error include:

### 8.2.1 Decoupling of Flow Friction and Volumetric Flow Rate

This will have the effect of incorrectly determining the phase lag and dampening of volumetric flow rates as well as incorrectly determining of the density of the gas.

### 8.2.2 Constant Properties

Material properties such as solid conductivity, solid heat capacity as gas-constants is assumed constant given that these properties did not vary significantly over the temperature range considered. However, this introduces a bias error for any point that is not equal to the calibration point.

### 8.2.3 Ideal Gas Representation of the Fill Gas

The ideal gas representation is valid only for low-pressures and high-temperatures. Of course, these statements are relative, and error increases the moment the gas is no longer at the point for which the gas constant was derived. At atmospheric pressure the ideal gas law for air is accurate to within 2% from 100 K with an upper bound exceeding 1000 K, at a pressure of 20 bar, the lower bound rises to 250 K and at a pressure of 100 bar, this range shrinks to within 300 to 350 K, outside of which error increases rapidly [66].

### 8.2.4 Radiation Heat Transfer is Ignored

In the annular gap in particular radiation heat transfer plays an important role in the heat transfer from 2 surfaces separated by a very small air gap. In MSPM this effect is ignored. This was justified because of the low temperature assumption which states that because the magnitude of radiation is based on temperature to the power of 4, that for low-temperature Stirling engines the

magnitude would be very small. While very small, the radiation effect would still produce a loss and thus results in a small bias error.

### **8.2.5 No Contact Resistance**

Built into the conduction algorithm is an assumption that all interfaces have zero contact resistance. This is particularly unlikely in cases where a cylinder and wall are closely mated but run against each other, the model assumes that the two surfaces are bonded perfectly, when in reality a thin layer of air and other fluids would offer a measurable resistance.

### **8.2.6 Nusselt Number is Node Based, not Surface Based**

The Nusselt number is intended and derived for convection between a gas and a surface with a particular geometry. This non-dimensional number was expanded to having a single value of Nusselt number per node due to the computational complexity of solving a Nusselt number for each face. This simplification was justified based on the concept that within a Stirling engine surfaces affect each other. Thus, turbulence created by one surface would throw off the Nusselt number calculation for another, therefore a single value from the most dominant surface would be the best guess at a global Nusselt number without ugly area weighted anonymous functions.

### **8.2.7 Constant Friction Coefficients in Mechanism**

Currently, the system takes only normal force in the calculation of friction coefficients. More realistic models would incorporate temperature buildup effects, speed, lubricant film thickness and could introduce drag from the air around the mechanism and flywheel.

### **8.2.8 One Dimensional Flow Assumption**

The one-dimensional flow assumption has its drawbacks, including the inability to evaluate the following:

- Recirculation regions [35] or complexities within open chambers
- Preferential flows [67] either due to offset designs or due to manufacturing inconsistencies

Not modelling these phenomena means that generic approximations are required. Recirculation regions are approximated by a combination of minor loss coefficients and turbulence in variable volume spaces, which are general at best, especially when combined with the mono-Nusselt number assumption. Minor loss coefficients are used out of scope as no minor loss coefficients could be found for annular interfaces. There is no generic approximation for preferential flows as it is assumed that the designer avoids it as much as possible and that features are uniform all the way around the engine, even in cases where a bridge is applied to an offset or side position, the inflow and outflow is assumed to be distributed or smeared all the way around the engine. The problem of preferential flows becomes particularly problematic with thin, HX's, which are optimal with regards to flow losses, but may result in inefficient use of HX volume or lower than expected exit temperatures.

### **8.2.9 Minor Loss Coefficients are Naively Applied**

The model does not incorporate support for laminar minor loss coefficients. Ideally these would be calibrated through CFD or real-world experiments for a range of flow through rates, but likely the designer will be restricted to generalized formulas such as the Hooper 2-K [68] or Darby 3-K [69] which provide loss coefficients for fittings in the laminar regime.

Turbulent minor loss coefficients are calculated simply by determining the change in flow area between one zone and the next. No modifications are made for turns, for the number or distribution of openings or for the effect of annularity.

### **8.2.10 Fluid Inertia and Acoustics, are Ignored**

Neglecting fluid inertia will prevent any inertia effects, such as the pressure increase/decrease in response to the speed of a moving boundary, or acoustic resonance at higher speeds. Neglecting acoustics will have the effect of improving the simulated performance of the engine due to reduced friction losses. Acoustics were neglected here to improve computational efficiency using gas speed instead of sound speed as the timestep limiter and reduce anomalies at low speeds.

### **8.2.11 Steady-State Convergence**

Currently, the model uses the deviation from cycle to cycle as the assessment criterion for convergence to steady state, for particularly large engines it may take a great deal of time to converge. Thus, the model may recognize an engine as converged even if the slope is not asymptotic but slow enough to fall within the tolerance bounds. In these cases, it is recommended that tolerance be reduced until the model shows truly convergent behavior, thankfully any follow up experiments can use the end of the previous experiment, the snapshot, as the starting point.

### **8.2.12 Calculation Errors**

Computers work within a limited scope of possible values, therefore any operations conducted using a computer has limitations. MATLAB natively uses double precision values (15 decimal digits) in its calculations, which results in very little calculation error through basic operations. Therefore, the only meaningful sources of error lie within areas where these values are magnified. These areas are matrix inversion and large values. These areas are often combined, as was the case with the flow rate solving. The error of matrix inversion is said to be on the order of the condition number of the matrix. For a typical matrix in the flow rate solving loop of the EP-1 model the condition number was on the order of  $1 \cdot 10^4$  which means that the matrix is somewhat ill-conditioned, and a small error in any of the properties may be magnified by roughly this amount.

## **8.3 Future Opportunities**

---

### **8.3.1 Real Gases**

Due to the use of Stirling engines in cryogenic fields, under high pressure and with large changes in pressure and temperature it would be beneficial to represent the gas as a non-ideal gas. The Van der Waals or Redlich-Kwong model of gases would be sufficient for this and even would allow prediction of the phase change behavior of the constituent gas.

### **8.3.2 Interface for Simulating Control System**

An improvement would introduce smart valves, localized sensors and modifiable transmission systems driven by a control system designed by the user. Ideally the system would control the mechanism phasing, which would involve a modification of how the mechanism is interpolated, fill pressure through an orifice and source temperatures.

### **8.3.3 Multi-Phase simulations**

Simulating multiphase transport, condensation, evaporation, and mixed properties would allow this to simulate more complex systems. This would involve a condensation / evaporation rate series of equations based on presence of a liquid film and flow properties, thermodynamic properties of mixtures, species transport as particulate, flow due to gravity and assisted liquid transport. Phase change is a promising angle for Stirling engines, typically incorporating phase change reduces the efficiency of a Stirling engine [70] but it is known to increase the power density substantially. With controlled evaporation it may be possible to significantly improve the performance of Stirling engines and with modification, MSPM could assist in that goal.

### **8.3.4 Source/Sink Simulation**

Simulating the temperature regime within the thermal sources and sinks along the length of the heat exchanger might be of interest in cases where significant amounts of heat are transferred. This could involve defining the source fluid path through the heat exchanger and adding more detail to the source's convection behavior, which is currently not defined.

### **8.3.5 Material Distortion**

Within the laboratory, there have been many observations of substantial flexing of members in response to pressure, the two most notable are the 1" thick acrylic blind flange of the EP-1 flexing close to 2 mm in either direction and the flexible bellow expanders which notably balloon outward or collapse under negative pressure. In short, modeling of flexible interfaces could prove valuable in cases where the pressure is contained by soft materials or when the engine body is to be analyzed for structural stability. With material distortion the simulation of more abstract structures may be

possible, including diagrams which are often used as frictionless pistons where high efficiency is required.

### **8.3.6 Improved modelling of Entrance Turbulence and Swirl in Open Volumes**

The current formulation of open space turbulence is extracted from SAGE. It would be of interest to investigate this topic further to ensure that these values are true, as it is known that interfaces between gases and solids support a laminar, near-wall, region which must be accounted for. This is not accounted for in the current implementation.

### **8.3.7 String or Text File Based Test Set Run Files**

To take this code out of the MATLAB environment it would be prudent to add an interface for the construction of test sets or allow the user to submit them as a standard text file.

### **8.3.8 Modelling of explicit faces**

When two regions become exposed to each other the resulting outflow is explosive. Such a flow process would be modelled as a compressible process. After a short period though, if the two regions remain connected this explicit face would continue to slow the simulation down as the flow would oscillate. The solution would be to combine the two regions into one during this connected period and handle the opening or closing event as a regulated transition between the separated and combined states. As such connections were not required for simulation of DTECL's Stirling engines it was not within the scope of this project to implement the projected complexity that this transition system would entail.

### **8.3.9 Improvements to Geometry Optimizer**

The current geometry optimizer does not consider that an improvement might occur through the modification of one or more properties simultaneously where independent modification of those same properties may result in a loss. This would be very common in cases where increasing the diameter of the displacer causes the heat exchangers to shrink simultaneously reducing their effectiveness and inducing more friction losses, while increasing both the displacer and heat

exchangers might result in a gain. To avoid the added cost of gradient calculation, the edits could be designed such that the cross-section of components that are downstream is maintained. This would involve turning on and off relationships, which could be made part of the degree of freedom definition.

In its current state it may be prudent then to optimize in stages, have one model broken up so that features can be as independent as possible, then reassemble into a compact interconnected engine and optimize all the features again.

### **8.3.10 Parallelization**

The program produced as part of this thesis runs entirely on a single core. A parallel implementation could, on a 4-core computer, be capable of operating at speeds close to 4 times as fast. Alternatively, this could allow the code to run more than one simulation at a time.

Alternatively, the model for steady-state cases could be discretized in both the spatial and time grid and a parallel non-linear solver would use to solve the resulting matrix. This would present a much better alternative to solving each step individually as it is done in this model and the resulting matrix updates would be calculated in parallel, preferably on the GPU as with modern CFD simulations. In such simulation the multi-grid consideration would work both in the time and space domain, the snapshot would serve as an initial guess and the steady-state solid temperatures assumption used to increase the convergence rate would replace the nodes deep within the bodies.

### **8.3.11 Other Programming Languages**

MATLAB is primarily a prototyping language and is very strong at vectorizable operations and in matrix operations, in these areas the built-in technology is almost unmatched. However, MATLAB is typically slow at conducted sequential operations, an area where C++ is more suited for. Replacing sections of the code base with C files could offer dramatic improvements.

## References

- [1] S. Stricker, T. Strack, L. F. Monier, and R. Clayton, “Market report on waste heat and requirements for cooling and refrigeration in Canadian industry.,” 2006.
- [2] J. Banks and N. B. Harris, “Geothermal Potential of Foreland Basins: A Case Study from the Western Canadian Sedimentary Basin,” *Geothermics*, vol. 76, no. May, pp. 74–98, 2018.
- [3] Alberta Energy Regulator, “Why are Wells Abandoned.” <https://www.aer.ca/abandonment-and-reclamation/why-are-wells-abandoned> (accessed Apr. 02, 2016).
- [4] B. Zinchuk, “DEEP advances field work for first geothermal power to Saskatchewan grid,” *jwnenergy*, 2019. <https://www.jwnenergy.com/article/2019/11/deep-advances-field-work-first-geothermal-power-saskatchewan-grid/> (accessed Mar. 15, 2020).
- [5] R. A. Kishore and S. Priya, “A Review on Low-Grade Thermal Energy Harvesting : Materials , Methods and Devices,” *Materials (Basel)*., vol. 11, no. 8, p. 1433, 2018, doi: 10.3390/ma11081433.
- [6] T. Tartièrè and M. Astolfi, “A World Overview of the Organic Rankine Cycle Market The Overview the Organic Rankine Assessing the feasibility of the heat,” *Energy Procedia*, vol. 129, pp. 2–9, 2017, doi: 10.1016/j.egypro.2017.09.159.
- [7] T. Knudsen, L. Røngaard, F. Haglind, and A. Modi, “Energy and exergy analysis of the Kalina cycle for use in concentrated solar power plants with direct steam generation,” *Energy Procedia*, vol. 57, pp. 361–370, 2014, doi: 10.1016/j.egypro.2014.10.041.
- [8] C. B. Vining, “An inconvenient truth about thermoelectrics TL - 8,” *Nat. Mater.*, vol. 8, no. 2, pp. 83–85, 2009.
- [9] C. M. Hargreaves, *The Philips Stirling Engine*, 1st ed. Michigan: Elsevier, 1991.
- [10] C. P. Speer, “Modifications to Reduce the Minimum Thermal Source Temperature of the ST05G-CNC Stirling Engine,” University of Alberta, 2018.
- [11] I. Urieli, “Stirling Cycle Machine Analysis,” *Russ College of Engineering and Technology Mechanical Engineering Department*, 2018. <https://www.ohio.edu/mechanical/stirling/>



(accessed May 09, 2018).

- [12] C. J. A. Stumpf, “Parameter Optimization of a Low Temperature Difference Gamma-Type Stirling Engine to Maximize Shaft Power,” University of Alberta, 2018.
- [13] D. A. Miller, “Experimental Investigation of Stirling Engine Modelling Techniques at Reduced Source Temperatures,” University of Alberta, 2019.
- [14] J. P. Michaud, “Low Temperature Difference Alpha-Type Stirling Engine for the Experimental Determination of Optimal Parameters to Maximize Shaft Power,” University of Alberta, 2020.
- [15] R. Stirling, “Stirling air engine and heat regenerator,” Patent no. 4081, 1816.
- [16] I. Urieli and D. M. Berchowitz, *Stirling cycle engine analysis*. Bristol: A. Hilger, 1984.
- [17] T. Finkelstein, A. J., and Organ, *Air Engines*. New York, 2001.
- [18] SAAB Group, “The secret to the world’s most silent submarine,” 2015. <https://saabgroup.com/media/stories/stories-listing/2015-02/the-secret-to-the-worlds-most-silent-submarine/> (accessed Jul. 29, 2020).
- [19] Microgen engine corporation, “Free Piston Stirling Engine,” 2020. <https://www.microgen-engine.com/> (accessed Jul. 29, 2020).
- [20] Office of Energy Efficiency & Renewable Energy, “Solar Dish Sets World-Record Efficiency,” *Solar Energy Technologies Office*, 2008. <https://www.energy.gov/eere/solar/downloads/solar-dish-sets-world-record-efficiency> (accessed Jul. 29, 2020).
- [21] C. C. Lloyd, “A low temperature differential Stirling engine for power generation,” p. 132, 2009.
- [22] C. D. West, *Principles and applications of Stirling Engines*. New York: Van Nostrand Reinhold, 1986.
- [23] W. Martini, “Stirling Engine Design Manual Conservation and Renewable Energy,” *Methods*, p. 412, 1983.

- [24] Y. A. Çengel and M. A. Boles, *Thermodynamics: An Engineering Approach.*, 7th ed. New York: McGraw-Hill, 2011.
- [25] G. Schmidt, “The Theory of Lehmann’s Calorimetric Machine,” *Z. ver. Dtsch. Ing.*, vol. 15, 1871.
- [26] R. Clausius, “On a Modified Form of the Second Fundamental Theorem in the Mechanical Theory of Heat,” *Phil. Mag.*, vol. 12, no. 77, pp. 81–98, 1856, doi: 10.1080/14786445608642141.
- [27] J. R. Senft, *Mechanical efficiency of heat engines*. Cambridge: Cambridge University Press, 2007.
- [28] S. Petrescu, M. Costea, C. Harman, and T. Florea, “Applications of the Direct Method to irreversible Stirling cycles with finite speed,” *Int. J. Energy Res.*, vol. 26, no. 7, pp. 589–609, 2002, doi: 10.1002/er.806.
- [29] C. G. Scheck, “Thermal Hysteresis Loss in Gas Springs,” Ohio University, 1988.
- [30] A. J. Organ, *Stirling Cycle Engines: Inner Workings and Design*. Chichester: John Wiley and Sons Ltd., 2014.
- [31] J. Weisbach, *Lehrbuch der Ingenieur- und Maschinen-Mechanik*, 1st ed. Braunschweig: Vieweg und Sohn, 1845.
- [32] C. F. Colebrook, “Turbulent flow in pipes with particular reference to the transition region between the smooth and rough pipe laws,” *J. Inst. Civ. Eng.*, vol. 11, pp. 133–156, 1939, [Online]. Available: <https://dx.doi.org/10.1680/ijoti.1939.13150>.
- [33] K. Peter, H. Janos, S. Krisztian, B. Attila, and T. Peter, “Models of Friction,” in *Robot Applications*, BME MOGI, Ed. BME MOGI, 2014.
- [34] K. Lewotsky, “Understanding Lubricants - Part I,” 2013. [https://www.motioncontrolonline.org/content-detail.cfm/Motion-Control-News/Understanding-Lubricants-Part-I/content\\_id/316](https://www.motioncontrolonline.org/content-detail.cfm/Motion-Control-News/Understanding-Lubricants-Part-I/content_id/316) (accessed Jun. 15, 2020).
- [35] D. Gedeon, “Sage User ’s Guide. Sage v11 Edition,” 2016, [Online]. Available: <http://www.sageofathens.com/Documents/SageStlxHyperlinked.pdf>.

- [36] B. Thomas, "PROSA - software for evaluation of Stirling cycle machines," in *ISEC, International Stirling Engine Conference - 10*, 2001, pp. 67–74.
- [37] A. Altman, "SNAPpro Stirling Numerical Analysis Program," 2018. <https://sites.google.com/site/snapburner/snappro-1> (accessed Feb. 13, 2020).
- [38] G. Walker, *Stirling Engine*. New York: Oxford University Press, 1980.
- [39] "MATLAB." The Mathworks, Inc., Natick, Massachusetts, 2020.
- [40] M. Babaelahi and H. Sayyaadi, "Simple-II: A new numerical thermal model for predicting thermal performance of Stirling engines," *Energy*, vol. 69, pp. 873–890, 2014, doi: 10.1016/j.energy.2014.03.084.
- [41] M. Babaelahi and H. Sayyaadi, "A new thermal model based on polytropic numerical simulation of Stirling engines," *Appl. Energy*, vol. 141, pp. 143–159, 2015, doi: 10.1016/j.apenergy.2014.12.033.
- [42] D. M. Berchowitz, "Linear Dynamics of Free-Piston Stirling Engines," in *Proceedings Institution of Mechanical Engineers*, 1985, vol. 199.
- [43] T. Finkelstein, "Computer Analysis of Stirling Engines," in *Intersociety Energy Conversion Engineering Conference*, 1975.
- [44] B. Hoegel, D. Pons, M. Gschwendtner, A. Tucker, and M. Sellier, "Thermodynamic peculiarities of alpha-type Stirling engines for low-temperature difference power generation: Optimisation of operating parameters and heat exchangers using a third-order model," *Proc. Inst. Mech. Eng. Part C J. Mech. Eng. Sci.*, vol. 228, no. 11, pp. 1936–1947, 2014, doi: 10.1177/0954406213512120.
- [45] B. Hoegel, "Thermodynamics-Based Design of Stirling Engines for Low-Temperature Heat Sources," 2014.
- [46] M. Hooshang, R. Askari Moghadam, S. Alizadeh Nia, and M. T. Masouleh, "Optimization of Stirling engine design parameters using neural networks," *Renewable Energy*, vol. 74, pp. 855–866, 2015, doi: 10.1016/j.renene.2014.09.012.
- [47] M. Hooshang, R. Askari Moghadam, and S. AlizadehNia, "Dynamic response simulation

- and experiment for gamma-type Stirling engine,” *Renew. Energy*, vol. 86, pp. 192–205, 2016, doi: 10.1016/j.renene.2015.08.018.
- [48] H. Carlsen and P. Grove, “Preliminary results from simulations of temperature oscillations in Stirling engine regenerator matrices,” *Energy*, vol. 31, pp. 1371–1383, 2006, doi: 10.1016/j.energy.2005.05.008.
- [49] K. Mahkamov, “An axisymmetric computational fluid dynamics approach to the analysis of the working process of a solar Stirling engine,” *J. Sol. Energy Eng.*, vol. 128, pp. 45–53, 2006.
- [50] R. W. Dyson, S. D. Wilson, R. C. Tew, and R. Demko, “Tech. Report TM-2005-213960 Fast whole-engine Stirling Analysis,” 2005.
- [51] N. Kwatra, J. Su, J. T. Grétarsson, and R. Fedkiw, “A method for avoiding the acoustic time step restriction in compressible flow,” *J. Comput. Phys.*, vol. 228, no. 11, pp. 4146–4161, 2009, doi: 10.1016/j.jcp.2009.02.027.
- [52] R. K. Shah, “Extended Surface Heat Transfer,” *Thermopedia*, 2011. .
- [53] A. Lambert, “What Do We Know About Pressure: Leakage Relationships in Distribution Systems?,” *IWA Conf. Syst. Approach to Leakage Control Water Distrib. Syst.*, pp. 1–8, 2000.
- [54] G. D. van Albada, B. van Leer, and J. W. W. Roberts, “A Comparative Study of Computational Methods in Cosmic Gas Dynamics,” *J. Astron. Astrophys.*, vol. 108, pp. 76–84, 1982.
- [55] W. H. Hayt and J. E. Kemmerly, *Engineering Circuit Analysis*, 5th ed. New York: McGraw Hill, 1993.
- [56] D. Gedeon, “A Cylinder Heat Transfer Model.” NASA-Lewis, 1989.
- [57] F. J. Cantelmi, “Measurement and Modelling of In-Cylinder Heat Transfer with Inflow-Produced Turbulence,” Virginia Polytechnic Institute and State University, 1995.
- [58] C. Koren *et al.*, “An Acceleration Method for Numerical Studies of Conjugate Heat Transfer With a Self-Adaptive Coupling Time Step Method: Application to a Wall-Impinging

- Flame,” 2018.
- [59] M. T. Heath, “Section 11.5.7 Multigrid Methods,” in *Scientific Computing: An Introductory Survey*, McGraw-Hill, 2002, p. 478.
- [60] Z. Wang, “Mechanistic Modeling of Nucleate Boiling,” Rensselaer Polytechnic Institute, 2019.
- [61] J. Hensen and A. E. Nakhi, “Fourier and Biot numbers and the accuracy of conduction modelling,” in *Fourier and Biot numbers and the accuracy of conduction modelling*, 1994, no. January 1994.
- [62] “Courant-Friedrichs-Lewy condition,” *Encyclopedia Of Mathematics*, 2014. [https://encyclopediaofmath.org/wiki/Courant-Friedrichs-Lewy\\_condition](https://encyclopediaofmath.org/wiki/Courant-Friedrichs-Lewy_condition) (accessed Aug. 18, 2020).
- [63] R. Karwa, *Heat and Mass Transfer*, 1st ed. Singapore: Springer, 2017.
- [64] M. Nicol-Seto, “Investigation of Drive Mechanism Modification to Increase Thermodynamic Performance of a Low Temperature Difference Gamma Stirling Engine,” University of Alberta, 2021.
- [65] I. Kolin, *Stirling Motor: History, Theory, Practice*. Zagreb: Zagreb University, 1991.
- [66] “Ideal Gas Law,” *Engineering Toolbox*, 2003. [https://www.engineeringtoolbox.com/ideal-gas-law-d\\_157.html](https://www.engineeringtoolbox.com/ideal-gas-law-d_157.html) (accessed Jul. 29, 2020).
- [67] C. P. Speer, “Modifications to Reduce the Minimum Thermal Source Temperature of the ST05G-CNC Stirling Engine,” University of Alberta, 2018.
- [68] W. B. Hooper, “The two-K method predicts head losses in pipe fittings,” *Chem. Eng.*, pp. 96–100, 1981.
- [69] R. Darby, “Correlate Pressure Drops through Fittings,” *Chem. Eng.*, vol. 106, pp. 101–104, 1999.
- [70] G. Walker and J. R. Senft, “Free Piston Stirling Engines,” in *Free Piston Stirling Engines: Liquid Piston Stirling Engines*, Berlin, Heidelberg: Springer, 1985, pp. 235–261.

- [71] E. Fried and I. E. Idelchik, *Flow Resistance: A Design Guide for Engineers*. Philadelphia: Taylor & Francis, 1989.
- [72] R. K. Shah and A. L. London, *Laminar Flow Forced Convection in Ducts*. Elsevier, 1978.
- [73] F. P. Incropera, D. P. Dewitt, T. L. Bergman, and A. S. Lavine, *Fundamentals of Heat and Mass Transfer*, 6th ed. Hoboken: Wiley, 2007.
- [74] R. L. Webb, *Principles of Enhanced Heat Transfer*. New York: John Wiley and Sons Ltd., 1994.
- [75] A. Kays, WM; London, "Compact Heat Exchangers," 1964, [Online]. Available: <http://www.amazon.com/Compact-Heat-Exchangers-W-Kays/dp/1575240602>.
- [76] H. C. Chai, "A Simple Pressure Drop Correlation Equation For Low Finned Tube Crossflow Heat Exchangers," *Int. Commun. Heat Mass Transf.*, vol. 15, pp. 95–101, 1988.
- [77] A. Ganguli and S. B. Yilmaz, *New heat transfer and pressure drop correlations for crossflow over low-finned tube banks*. Pittsburgh: American Institute of Chemical Engineers, 1987.

# Appendices

## Appendix A. Mechanisms

The mechanisms described below are only a subset of the myriad of linear to rotational mechanisms proposed for Stirling Engines. Each of them assumes the following:

- A piston force is considered positive if it pushes the piston away from the driveshaft.
- Friction is excluded except for a modification on the output torque which takes the rotational speed and formulates a moment such that the energy loss from each source of friction is represented in the modified torque.
- To ensure that the bearing load on the driveshaft is properly oriented in space the orientation of the mechanism, rotates the coordinate system of the output normal loads before outputting.
- As a piston can be based either as coming from above the cylinder or below, each mechanism has an orientation property, in addition to its rotation, which changes the sign of the piston forces and inverts the returned position offsets.

### A.1. Slider-crank Mechanism

Each component of the Slider-crank is solved using Newton's law, resulting in 7 semi-explicit equations.

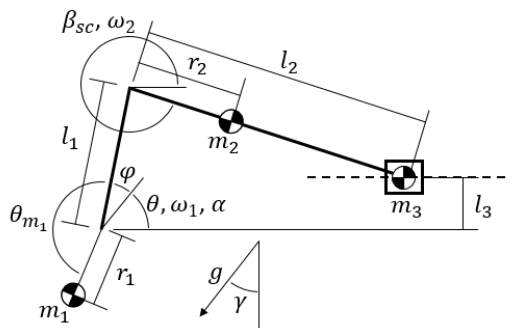
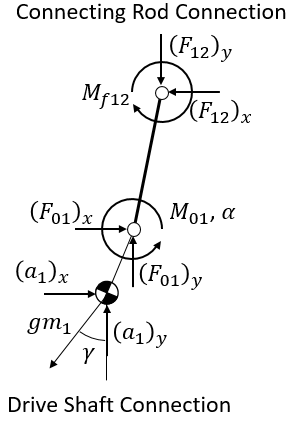


Figure 8.1. Slider-crank Mechanism: dimensions, masses, and gravity  
Crank Arm



**Figure 8.2. Free Body Diagram of Crank Arm**

$$\theta_{sc} = \theta + \varphi$$

$$\theta_g = \theta + \varphi + \theta_{m_1}$$

$$I_{m_1} = m_1 r_1^2$$

The moment generated around the driveshaft

$$\sum M = I_{m_1} \alpha = (F_{12})_x l_1 \sin(\theta_{sc}) - (F_{12})_y l_1 \cos(\theta_{sc}) + M_{01} - gm_1 r_1 \cos(\theta_g)$$

Forces in the local x-direction

$$\sum F_x = m_1 (a_1)_x = (F_{01})_x - (F_{12})_x - gm_1 \sin(\gamma)$$

Forces in the local y-direction

$$\sum F_y = m_1 (a_1)_y = (F_{01})_y - (F_{12})_y - gm_1 \cos(\gamma)$$

Linking acceleration to relevant parameters

$$(a_1)_x = (-r_1 \cos(\theta_{sc} + \theta_{m_1})) \omega^2 + (-r_1 \sin(\theta_{sc} + \theta_{m_1})) \alpha = B_{a1x} \omega^2 + C_{a1x} \alpha$$

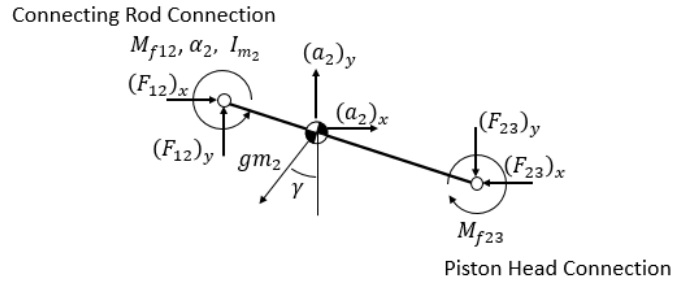
$$(a_1)_y = (-r_1 \sin(\theta_{sc} + \theta_{m_1})) \omega^2 + (r_1 \cos(\theta_{sc} + \theta_{m_1})) \alpha = B_{a1y} \omega^2 + C_{a1y} \alpha$$

Loss associated with  $M_{f12}$ :

$$L = |M_{f12} \omega_1 (1 - C_{\omega 2})|$$



## Connecting Rod



**Figure 8.3. Free Body Diagram of Connecting Rod**

$$\beta_{sc} = \sin^{-1} \left( \frac{l_3 - l_1 \sin(\theta_{sc})}{l_2} \right)$$

$$\beta_g = \beta_{sc} + \gamma$$

$$I_{m_2} = m_2 r_2^2$$

The moment generated around Crank Arm – Connecting Rod joint

$$\sum M = I_{m_2} \alpha_2 = (F_{23})_x l_2 \sin(\beta_{sc}) - (F_{23})_y l_2 \cos(\beta_{sc}) - gm_2 r_2 \cos(\beta_g)$$

Forces in the local x-direction

$$\sum F_x = m_2 (a_2)_x = (F_{12})_x - (F_{23})_x - gm_2 \sin(\gamma)$$

Forces in the local y-direction

$$\sum F_y = m_2 (a_2)_y = (F_{12})_y - (F_{23})_y - gm_2 \cos(\gamma)$$

Linking acceleration to relevant parameters

$$\omega_2 = \left( -\frac{l_1 \cos(\theta_{sc})}{l_2 \cos(\beta_{sc})} \right) \omega_1 = C_{\omega_2} \omega_1$$

$$\alpha_2 = \left( \frac{l_1 \sin(\theta_{sc}) + l_2 \sin(\beta_{sc}) C_{\omega_2}^2}{l_2 \cos(\beta_{sc})} \right) \omega_1^2 + \left( -\frac{l_1 \cos(\theta_{sc})}{l_2 \cos(\beta_{sc})} \right) \alpha_1 = B_{\alpha_2} \omega_1^2 + C_{\alpha_2} \alpha_1$$

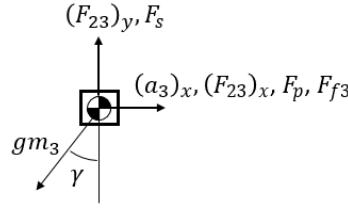
$$(a_2)_x = (-l_1 \cos(\theta_{sc}) - r_2 \cos(\beta_{sc}) C_{\omega_2}^2 - r_2 \sin(\beta_{sc}) B_{\alpha_2}) \omega^2 \\ + (-l_1 \sin(\theta_{sc}) - r_2 \sin(\beta_{sc}) C_{\omega_2}) \alpha = B_{a2x} \omega^2 + C_{a2x} \alpha$$

$$(a_2)_y = (-l_1 \sin(\theta_{sc}) - r_2 \sin(\beta_{sc}) C_{\omega_2}^2 + r_2 \cos(\beta_{sc}) B_{\alpha_2}) \omega^2 \\ + (l_1 \cos(\theta_{sc}) + r_2 \cos(\beta_{sc}) C_{\omega_2}) \alpha = B_{a2y} \omega^2 + C_{a2y} \alpha$$

Loss associated with  $M_{f23}$ :

$$L = |M_{f23} \omega_1 C_{\omega_2}|$$

### Piston Head



**Figure 8.4. Free Body Diagram of Piston Head**

Forces in the local  $x$ -direction

$$\sum F_x = m_3 (a_3)_x = F_p + (F_{23})_x - gm_3 \sin(\gamma)$$

$$\sum F_y = 0 = (F_{23})_y + F_s - gm_3 \cos(\gamma)$$

Linking acceleration to relevant parameters

$$(a_3)_x = (-l_1 \cos(\theta_{sc}) - l_2 \cos(\beta_{sc}) C_{\omega_2}^2 - l_2 \sin(\beta_{sc}) B_{\alpha_2}) \omega^2 \\ + (-l_1 \sin(\theta_{sc}) - l_2 \sin(\beta_{sc}) C_{\omega_2}) \alpha = B_{a3x} \omega^2 + C_{a3x} \alpha$$

Piston Motion and position

$$x_p = l_1 \cos(\theta_{sc}) + l_2 \cos(\beta_{sc}) - \sqrt{(l_2 - l_1)^2 - l_3^2}$$

$$v_p = (-l_1 \sin(\theta_{sc}) - l_2 \sin(\beta_{sc}) C_{\omega_2}) \omega$$

Loss associated with  $F_{f3}$ :

$$L = |F_{f3} v_p|$$

**Deriving  $M_{01}$ ,  $(F_{01})_x$  and  $(F_{01})_y$  as  $F = A\alpha + B\omega^2 + G + E$**

$$\begin{aligned}(F_{23})_x &= (m_3 C_{a3x})\alpha + (m_3 B_{a3x})\omega^2 + (gm_3 \sin(\gamma)) - F_p \\ &= A_1\alpha + B_1\omega^2 + G_1 + E_1\end{aligned}$$

$$\begin{aligned}(F_{12})_x &= (m_2 C_{a2x} + A_1)\alpha + (m_2 B_{a2x} + B_1)\omega^2 + (gm_2 \sin(\gamma) + G_1) + (E_1) \\ &= A_2\alpha + B_2\omega^2 + G_2 + E_2\end{aligned}$$

$$\begin{aligned}(F_{23})_y &= \left( \frac{-I_{m_2} C_{\omega_2}}{l_2 \cos(\beta_{sc})} + \tan(\beta_{sc}) A_1 \right) \alpha + \left( \frac{-I_{m_2} B_{\alpha_2}}{l_2 \cos(\beta_{sc})} + \tan(\beta_{sc}) B_1 \right) \omega^2 \\ &\quad + \left( \frac{-gm_2 r_2 \cos(\beta_g)}{l_2 \cos(\beta_{sc})} + \tan(\beta_{sc}) G_1 \right) + (\tan(\beta_{sc}) E_1) \\ &= A_3\alpha + B_3\omega^2 + G_3 + E_3\end{aligned}$$

$$\begin{aligned}(F_{12})_y &= (m_2 C_{a2y} + A_3)\alpha + (m_2 B_{a2y} + B_3)\omega^2 + (gm_2 \cos(\beta_g) + G_3) + (E_3) \\ &= A_4\alpha + B_4\omega^2 + G_4 + E_4\end{aligned}$$

$$\begin{aligned}(F_{01})_x &= (m_1 C_{a1x} + A_2)\alpha + (m_1 B_{a1x} + B_2)\omega^2 + (gm_1 \sin(\gamma) + G_2) + (E_2) \\ &= A_5\alpha + B_5\omega^2 + G_5 + E_5\end{aligned}$$

$$\begin{aligned}(F_{01})_y &= (m_1 C_{a1y} + A_4)\alpha + (m_1 B_{a1y} + B_4)\omega^2 + (gm_1 \cos(\gamma) + G_4) + (E_4) \\ &= A_6\alpha + B_6\omega^2 + G_6 + E_6\end{aligned}$$

**Output Variables (converted to global coordinates)**

Torque from the driveshaft to the slider-crank mechanism

$$\sum M = I_{m_1} \alpha = (F_{12})_x l_1 \sin(\theta_{sc}) - (F_{12})_y l_1 \cos(\theta_{sc}) + M_{01} - gm_1 r_1 \cos(\theta_g)$$

$$\begin{aligned}
M_0 = & (I_{m1} - l_1 \sin(\theta_{sc}) A_2 + l_1 \cos(\theta_{sc}) A_4) \alpha + (-l_1 \sin(\theta_{sc}) B_2 + l_1 \cos(\theta_{sc}) B_4) \omega^2 \\
& + (gm_1 r_1 \cos(\theta_g) - l_1 \sin(\theta_{sc}) G_2 + l_1 \cos(\theta_{sc}) G_4) \\
& + (F_p l_1 (\cos(\theta_{sc}) \tan(\beta_{sc}) - \sin(\theta_{sc}))) = A_M \alpha + B_M \omega^2 + G_M + E_M
\end{aligned}$$

Horizontal force as felt by the driveshaft.

$$\begin{aligned}
F_x = & (-\cos(\gamma) A_5 + \sin(\gamma) A_6) \alpha + (-\cos(\gamma) B_5 + \sin(\gamma) B_6) \omega^2 + (-\cos(\gamma) G_5 + \sin(\gamma) G_6) \\
& + (-\cos(\gamma) E_5 + \sin(\gamma) E_6) = A_x \alpha + B_x \omega^2 + G_x + E_x
\end{aligned}$$

Vertical force as felt by the driveshaft.

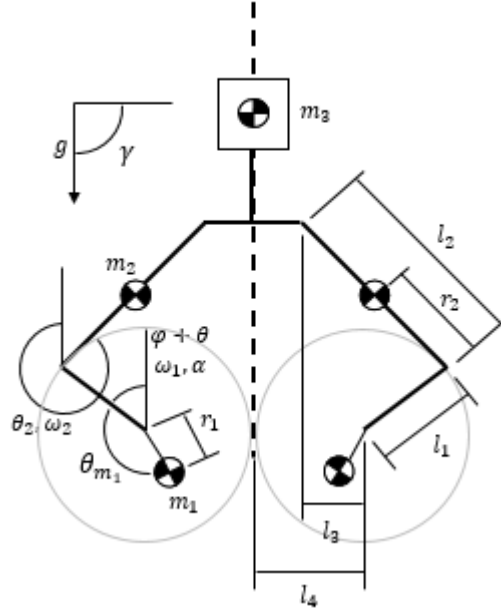
$$\begin{aligned}
F_y = & (-\sin(\gamma) A_5 - \cos(\gamma) A_6) \alpha + (-\sin(\gamma) B_5 - \cos(\gamma) B_6) \omega^2 + (-\sin(\gamma) G_5 - \cos(\gamma) G_6) \\
& + (-\sin(\gamma) E_5 - \cos(\gamma) E_6) = A_y \alpha + B_y \omega^2 + G_y + E_y
\end{aligned}$$

Losses:

$$M_{Loss} = |M_{f12}(F_{12x}, F_{12y}) \omega_1 (1 - C_{\omega 2})| + |M_{f23}(F_{23x}, F_{23y}) \omega_1 C_{\omega 2}| + |F_{f3}(F_{23y}) v_p|$$

## A.2. Rhombic Drive Mechanism

Half of the Rhombic Drive mechanism is very similar to the slider-crank mechanism except that the friction for the piston is only a product of side-load free seal friction. Also, half of the bearing load on the shaft is placed on an auxiliary shaft that can be entirely contained in the engine body without a seal.



**Figure 8.5: Rhombic Drive Mechanism: dimensions, masses, and gravity**  
**Output Variables (converted to global coordinates)**

Torque from the driveshaft to a slider-crank mechanism

$$\begin{aligned}
 M_0 = & (I_{m1} - l_1 \sin(\theta_{sc}) A_2 + l_1 \cos(\theta_{sc}) A_4) \alpha + (-l_1 \sin(\theta_{sc}) B_2 + l_1 \cos(\theta_{sc}) B_4) \omega^2 \\
 & + (g m_1 r_1 \cos(\theta_g) - l_1 \sin(\theta_{sc}) G_2 + l_1 \cos(\theta_{sc}) G_4) \\
 & + (F_p l_1 (\cos(\theta_{sc}) \tan(\beta_{sc}) - \sin(\theta_{sc}))) = A_M \alpha + B_M \omega^2 + G_M + E_M
 \end{aligned}$$

Horizontal force as felt by the driveshaft.

$$\begin{aligned}
 F_x = & 0.5((- \cos(\gamma) A_5 + \sin(\gamma) A_6) \alpha + (- \cos(\gamma) B_5 + \sin(\gamma) B_6) \omega^2 \\
 & + (- \cos(\gamma) G_5 + \sin(\gamma) G_6) + (- \cos(\gamma) E_5 + \sin(\gamma) E_6)) \\
 = & A_x \alpha + B_x \omega^2 + G_x + E_x
 \end{aligned}$$

Vertical force as felt by the driveshaft.

$$\begin{aligned}
 F_y &= 0.5((- \sin(\gamma) A_5 - \cos(\gamma) A_6)\alpha + (- \sin(\gamma) B_5 - \cos(\gamma) B_6)\omega^2 \\
 &\quad + (- \sin(\gamma) G_5 - \cos(\gamma) G_6) + (- \sin(\gamma) E_5 - \cos(\gamma) E_6)) \\
 &= A_y\alpha + B_y\omega^2 + G_y + E_y
 \end{aligned}$$

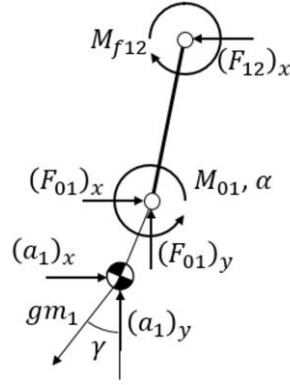
Losses:

$$\begin{aligned}
 M_{Loss} &= \left| 2M_{f12} \left( \frac{F_{12x}}{2}, \frac{F_{12y}}{2} \right) \omega_1 (1 - C_{\omega 2}) \right| + \left| 2M_{f23} \left( \frac{F_{23x}}{2}, \frac{F_{23y}}{2} \right) \omega_1 C_{\omega 2} \right| + |F_{f3}(0)v_p| \\
 &\quad + |F_{aux}(F_x, F_y)\omega_1| + \left| F_{gear} \left( \omega_1 \frac{M_0}{2} \right) \right|
 \end{aligned}$$

### A.3. Scotch Yoke Mechanism

#### Crank Arm

Connecting Rod Connection



Drive Shaft Connection

**Figure 8.6. Free Body Diagram of Crank Arm**

$$\theta_{sy} = \theta + \varphi$$

$$\theta_g = \theta + \varphi + \theta_{m_1}$$

The moment generated around the driveshaft

$$\sum M = I_{m_1} \alpha = (F_{12})_x l_1 \sin(\theta_{sy}) - (F_{12})_y l_1 \cos(\theta_{sy}) + M_{01} - gm_1 r_1 \cos(\theta_g)$$

Forces in the local  $x$ -direction

$$\sum F_x = m_1 (a_1)_x = (F_{01})_x - (F_{12})_x - gm_1 \sin(\gamma)$$

Forces in the local  $y$ -direction

$$\sum F_y = m_1 (a_1)_y = (F_{01})_y - gm_1 \cos(\gamma)$$

Linking acceleration to relevant parameters

$$(a_1)_x = (-r_1 \cos(\theta_{sy} + \theta_{m_1}))\omega^2 + (-r_1 \sin(\theta_{sy} + \theta_{m_1}))\alpha = B_{a1x}\omega^2 + C_{a1x}\alpha$$

$$(a_1)_y = (-r_1 \sin(\theta_{sy} + \theta_{m_1}))\omega^2 + (r_1 \cos(\theta_{sy} + \theta_{m_1}))\alpha = B_{a1y}\omega^2 + C_{a1y}\alpha$$

Loss associated with  $M_{f12}$ :

$$L = |M_{f12}\omega_1(1 - C_{\omega2})|$$

### Piston Assembly

The piston for the scotch yoke mechanism is supported by a linear bearing assembly, which has a moment dependent and constant component to friction loss. The forces in the local  $x$ -direction are as follows:

$$\begin{aligned}\sum F_x &= m_p(a_p)_x = F_p + (F_{12})_x \\ \sum F_y &= (F_{BAsm})_y - gm_p \cos(\gamma) \\ \sum M &= 0 = M_{BAsm} - (F_{12})_x l_1 \sin(\theta_{sy}) - gm_p \sin(\gamma)\end{aligned}$$

Where:

$$\begin{aligned}x_p &= C_{offset} + l_1 \cos(\theta_{sy}) \\ v_p &= -l_1 \omega \sin(\theta_{sy}) \\ a_p &= -l_1 \alpha \sin(\theta_{sy}) - l_1 \omega^2 \cos(\theta_{sy})\end{aligned}$$

Thus:

$$\begin{aligned}(F_{12})_x &= -m_p l_1 (\alpha \sin(\theta_{sy}) + \omega^2 \cos(\theta_{sy})) - F_p \\ (F_{BAsm})_y &= gm_p \cos(\gamma) \\ M_{BAsm} &= -(m_p l_1^2 \sin^2(\theta_{sy}) \alpha + m_p l_1^2 \cos(\theta_{sy}) \sin(\theta_{sy}) \omega^2 + l_1^2 \sin^2(\theta_{sy}) F_p) - gm_p \sin(\gamma) \\ (F_{01})_x &= m_1(a_1)_x + (F_{12})_x + gm_1 \sin(\gamma) \\ (F_{01})_x &= m_1(B_{a1x}\omega^2 + C_{a1x}\alpha) - m_p l_1 (\alpha \sin(\theta_{sy}) + \omega^2 \cos(\theta_{sy})) - F_p + gm_1 \sin(\gamma) \\ &= A_1 \alpha + B_1 \omega^2 + E_1 F_p + G_1 \\ (F_{01})_y &= m_1(a_1)_y + gm_1 \cos(\gamma)\end{aligned}$$



$$(F_{01})_y = m_1 B_{a1y} \omega^2 + m_1 C_{a1y} \alpha + g m_1 \cos(\gamma) = A_2 \alpha + B_2 \omega^2 + G_2$$

$$M_{01} = I_{m_1} \alpha + (m_p l_1^2 \sin^2(\theta_{sy}) \alpha + m_p l_1^2 \cos(\theta_{sy}) \sin(\theta_{sy}) \omega^2 + l_1^2 \sin^2(\theta_{sy}) F_p) \\ + g m_1 r_1 \cos(\theta_g) = -A_M \alpha - B_M \omega^2 - E_M F_p - G_M$$

Torque from the driveshaft to a slider-crank mechanism

$$M_0 = A_M \alpha + B_M \omega^2 + G_M + E_M$$

$$A_M = -I_{m_1} - m_p l_1^2 \sin^2(\theta_{sy})$$

$$B_M = -m_p l_1^2 \cos(\theta_{sy}) \sin(\theta_{sy})$$

$$G_M = -g m_1 r_1 \cos(\theta_g)$$

$$E_M = -l_1^2 \sin^2(\theta_{sy})$$

Horizontal force as felt by the driveshaft.

$$F_x = ((-\cos(\gamma) A_1 + \sin(\gamma) A_2) \alpha + (-\cos(\gamma) B_1 + \sin(\gamma) B_2) \omega^2 \\ + (-\cos(\gamma) G_1 + \sin(\gamma) G_2) + (-\cos(\gamma) E_1)) = A_x \alpha + B_x \omega^2 + G_x + E_x$$

$$A_x = -\cos(\gamma) (m_1 C_{a1x} - m_p l_1 \sin(\theta_{sy})) + \sin(\gamma) (m_1 C_{a1y})$$

$$B_x = -\cos(\gamma) (m_1 B_{a1x} - m_p l_1 \cos(\theta_{sy})) + \sin(\gamma) (m_1 B_{a1y})$$

$$G_x = 0$$

$$E_x = \cos(\gamma)$$

Vertical force as felt by the driveshaft.

$$F_y = ((-\sin(\gamma) A_1 - \cos(\gamma) A_2) \alpha + (-\sin(\gamma) B_1 - \cos(\gamma) B_2) \omega^2 \\ + (-\sin(\gamma) G_1 - \cos(\gamma) G_2) + (-\sin(\gamma) E_1)) = A_y \alpha + B_y \omega^2 + G_y + E_y$$

$$A_y = -\sin(\gamma) (m_1 C_{a1x} - m_p l_1 \sin(\theta_{sy})) - \cos(\gamma) (m_1 C_{a1y})$$

$$B_y = -\sin(\gamma) (m_1 B_{a1x} - m_p l_1 \cos(\theta_{sy})) - \cos(\gamma) (m_1 B_{a1y})$$

$$G_y = g(m_1 + m_p)$$

$$E_y = \sin(\gamma)$$

#### A.4. Ideal Sinusoidal Mechanism

The ideal sinusoidal mechanism is simply a simplified Scotch Yoke mechanism

$$\theta_{sc} = \theta + \varphi$$

The moment generated around the driveshaft

$$\sum M = I_{m_1} \alpha = (F_{12})_x l_1 \sin(\theta_{sc}) + M_{01}$$

Forces in the local  $x$ -direction

$$\sum F_x = 0 = (F_{01})_x - (F_{12})_x - gm_1 \sin(\gamma)$$

Forces in the local  $y$ -direction

$$\sum F_y = 0 = (F_{01})_y - gm_1 \cos(\gamma)$$

Piston

$$\sum F_x = \alpha_p m_p = (F_{12})_x + F_p - gm_p \sin(\gamma)$$

$$\sum F_y = 0 = (F_3)_y - gm_p \cos(\gamma)$$

$$x_p = l_1 \cos(\theta_{sc}) + C$$

$$v_p = -l_1 \sin(\theta_{sc}) \omega_{sc}$$

$$a_p = -l_1 \cos(\theta_{sc}) \omega_{sc}^2 - l_1 \sin(\theta_{sc}) \alpha_{sc}$$

Separating the root forces  $F_{01}$  and  $M_0$

$$(F_{12})_x = -m_p l_1 \cos(\theta_{sc}) \omega_{sc}^2 - m_p l_1 \sin(\theta_{sc}) \alpha_{sc} + gm_p \sin(\gamma) - F_p$$

$$(F_{01})_x = -m_p l_1 \cos(\theta_{sc}) \omega_{sc}^2 - m_p l_1 \sin(\theta_{sc}) \alpha_{sc} + gm_p \sin(\gamma) + gm_1 \sin(\gamma) - F_p$$

$$(F_{01})_y = gm_1 \cos(\gamma)$$

$$M_{01} = I_{m_1} \alpha_{sc} + m_p l_1^2 \cos(\theta_{sc}) \sin(\theta_{sc}) \omega_{sc}^2 + m_p l_1^2 \sin^2(\theta_{sc}) \alpha_{sc} + F_p l_1 \sin(\theta_{sc}) \\ - g m_p l_1 \sin(\gamma) \sin(\theta_{sc})$$

$$A_x = -\cos(\gamma) (-m_p l_1 \sin(\theta_{sc}))$$

$$B_x = -\cos(\gamma) (-m_p l_1 \cos(\theta_{sc}))$$

$$G_x = 0$$

$$E_x = -\cos(\gamma) (-F_p)$$

$$A_y = -\sin(\gamma) (-m_p l_1 \sin(\theta_{sc}))$$

$$B_y = -\sin(\gamma) (-m_p l_1 \cos(\theta_{sc}))$$

$$G_y = g(m_1 + m_p)$$

$$E_y = -\sin(\gamma) (-F_p)$$

$$A_M = (I_{m_1} + m_p l_1^2 \sin^2(\theta_{sc}))$$

$$B_M = m_p l_1^2 \cos(\theta_{sc}) \sin(\theta_{sc})$$

$$G_M = -g m_p l_1 \sin(\gamma) \sin(\theta_{sc})$$

$$E_M = F_p l_1 \sin(\theta_{sc})$$

## A.5. Custom Profile Mechanism

The following mathematics determines a generic representation of a custom profile mechanism.

The custom motion mechanism is simply a modified Scotch Yoke mechanism

$$\theta' = \theta + \varphi$$

Forces in the local  $x$ -direction

$$\sum F_x = m_p a_p \cos(\gamma) = -F_{x,DS} - g(m_1 + m_p) \sin(\gamma) + F_p \cos(\gamma)$$

$$F_{x,DS} = -g(m_1 + m_p) \sin(\gamma) + (F_p - m_p a_p) \cos(\gamma)$$

Forces in the local  $y$ -direction

$$\sum F_y = m_p a_p \sin(\gamma) = -F_{y,DS} - g(m_1 + m_p) \cos(\gamma) + F_p \sin(\gamma)$$

$$F_{y,DS} = -g(m_1 + m_p) \cos(\gamma) + (F_p - m_p a_p) \sin(\gamma)$$

Inertia of piston

$$KE = \frac{1}{2} m_p v_p^2, \quad \frac{dKE}{dt} = m_p v_p a_p = \omega T_p$$

$$T_p = \frac{v_p}{\omega} m_p a_p$$

This torque is positive concerning the driveshaft when the acceleration is opposite to the current velocity, as in kinetic energy is leaving the piston and being transmitted to the driveshaft.

When the velocity and acceleration have the same sign, this should be negative. Thus:

$$T_{p,DS} = -\frac{v_p}{\omega} m_p a_p$$

Work against forces

$$\frac{dE}{dt} = (F_p - g \sin(\gamma) m_p) v_p = \omega T_f$$

$$T_f = \frac{v_p}{\omega} (F_p - g \sin(\gamma) m_p)$$

This torque is positive when the force is in the same direction as the velocity, thus:

$$T_{f,DS} = -\frac{v_p}{\omega} (F_p - g \sin(\gamma) m_p)$$

Work into rotating inertia

$$\frac{dE}{dt} = I_{m_1} \alpha$$

When acceleration is positive will reduce the torque sent to the driveshaft.

$$T_{m,DS} = -I_{m_1} \alpha$$

Moment around driveshaft

$$M_{DS} = -I_{m_1} \alpha - \frac{v_p}{\omega} m_p a_p + \frac{v_p}{\omega} (F_p - g \sin(\gamma) m_p)$$

The Piston Motion

$$x_p = x(\theta)$$

$$v_p = \omega \frac{dx}{d\theta}$$

$$a_p = \alpha \frac{dx}{d\theta} + \omega^2 \frac{d^2x}{d\theta^2}$$

Separating the root forces  $F_{01}$

$$F_{x,DS} = -g(m_1 + m_p) \sin(\gamma) + (F_p - m_p a_p) \cos(\gamma)$$

$$F_{x,DS} = -m_p \left( \alpha \frac{dx}{d\theta} + \omega^2 \frac{d^2x}{d\theta^2} \right) \cos(\gamma) - g(m_1 + m_p) \sin(\gamma) + F_p \cos(\gamma)$$

$$F_{y,DS} = -g(m_1 + m_p) \cos(\gamma) + (F_p - m_p a_p) \sin(\gamma)$$

$$F_{y,DS} = -m_p \left( \alpha \frac{dx}{d\theta} + \omega^2 \frac{d^2x}{d\theta^2} \right) \sin(\gamma) - g(m_1 + m_p) \cos(\gamma) + F_p \sin(\gamma)$$

$$M_{DS} = -I_{m_1} \alpha + \frac{dx}{d\theta} \left( F_p - g \sin(\gamma) m_p - m_p \left( \alpha \frac{dx}{d\theta} + \omega^2 \frac{d^2x}{d\theta^2} \right) \right)$$

$$A_x = -\cos(\gamma) \left( m_p \frac{dx}{d\theta} \right) \quad (121)$$

$$B_x = -\cos(\gamma) \left( m_p \frac{d^2x}{d\theta^2} \right) \quad (122)$$

$$G_x = 0 \quad (123)$$

$$E_x = \cos(\gamma) (F_p) \quad (124)$$

$$A_y = -\sin(\gamma) \left( m_p \frac{dx}{d\theta} \right) \quad (125)$$

$$B_y = -\sin(\gamma) \left( m_p \frac{d^2x}{d\theta^2} \right) \quad (126)$$

$$G_y = g(m_1 + m_p) \quad (127)$$

$$E_y = \sin(\gamma) (F_p) \quad (128)$$

$$A_M = -I_{m_1} - m_p \left( \frac{dx}{d\theta} \right)^2 \quad (129)$$

$$B_M = -m_p \frac{dx}{d\theta} \frac{d^2x}{d\theta^2} \quad (130)$$

$$G_M = -g \sin(\gamma) m_p \frac{dx}{d\theta} \quad (131)$$

$$E_M = F_p \frac{dx}{d\theta} \quad (132)$$

## Appendix B. Property Correlations, Surface Area & Resistance of Matrix Elements

**Table 8.1: User inputs and correlations for various properties based on regenerator type. Correlations from (Gedeon, SAGE users manual [35]).**

	Regenerator Type			
	Woven Screen	Random Fiber	Packed Sphere	Stacked Foil
Inputs	Porosity ( $\beta$ ) Wire Diameter ( $d_o$ )	Porosity ( $\beta$ ) Wire Diameter ( $d_o$ )	Porosity ( $\beta$ ) Sphere Diameter ( $d_o$ )	Gap Width ( $l_g$ ) Thickness ( $l_t$ ) Roughness ( $l_r$ )
Hydraulic Diameter ( $d_h$ )	$\frac{d_o}{1 - \beta}$	$\frac{d_o}{1 - \beta}$	$\frac{d_o \cdot \beta}{6(1 - \beta)}$	$2l_g$
Laminar Friction Factor ( $N_f$ )	$\frac{129}{N_{Re}} + \frac{2.91}{N_{Re}^{0.103}}$	$\frac{25.7c + 79.8}{N_{Re}} + \frac{0.146c + 3.76}{N_{Re}^{0.00283c + 0.0748}}$	$\left(\frac{157}{N_{Re}} + \frac{5.15}{N_{Re}^{0.137}}\right) \left(\frac{\beta}{0.39}\right)^{3.48}$	$\frac{96}{N_{Re}}$
Laminar Nusselt Number ( $N_{Nu}$ )	$(1 + 0.99N_{Pe}^{0.66})\beta^{1.79}$	$1 + 0.186cN_{Pe}^{0.55}$	$1 + 0.48N_{Pe}^{0.65}$	8.23
Laminar Conduction Enhancement Factor ( $N_k$ )	$1 + \frac{N_{Pe}^{0.66}}{2\beta^{2.91}}$	$1 + N_{Pe}^{0.55}$	$1 + 3N_{Pe}^{0.65}$	1
Turbulent Friction Factor ( $N_f$ )	---	---	---	$0.121 \left(\frac{l_r}{d_h}\right)^{0.25} + \frac{68}{N_{Re}}$
Turbulent Nusselt Number ( $N_{Nu}$ )	---	---	---	$0.025N_{Re}^{0.79} \cdot N_{Pr}^{0.33}$
Turbulent Conduction	---	---	---	$0.022N_{Re}^{0.75} \cdot N_{Pr}$



---

Enhancement  
Factor ( $N_k$ )

---

Extra Equations:  $N_{Pe} = N_{Re} \cdot N_{Pr}$        $c = \frac{\beta}{1 - \beta}$        $\beta = \frac{l_g}{l_g + l_t}$

---

## WOVEN SCREEN & RANDOM FIBER REGENERATORS

---

For a long cylindrical element

$$\text{Surface Area} = 2\pi rL$$

$$\text{Volume} = \pi r^2 L$$

$$\text{Average Radius} = \frac{\int_0^r 2\pi r^2 dr}{\pi r^2} = \frac{\frac{2}{3}\pi r^3}{\pi r^2} = \frac{2}{3}r$$

$$\text{Resistance from Surface to Average Radius} = \frac{\ln\left(\frac{r}{\frac{2}{3}r}\right)}{2\pi Lk} = \frac{\ln\left(\frac{3}{2}\right)}{2\pi Lk}$$

$$\text{Surface Area per unit Volume} = (1 - \beta) \frac{2}{r} = 4 \frac{(1-\beta)}{d_w}$$

$$\text{Resistance times Area} = \frac{\ln\left(\frac{3}{2}\right)r}{k} = \frac{\ln\left(\frac{3}{2}\right)d_w}{2k}$$

## PACKED SPHERE REGENERATORS

---

For spherical elements

$$\text{Surface Area} = 4\pi r^2$$

$$\text{Volume} = \frac{4}{3}\pi r^3$$

$$\text{Average Radius} = \frac{\int_0^r 4\pi r^3 dr}{\frac{4}{3}\pi r^3} = \frac{\frac{4}{4}\pi r^4}{\frac{4}{3}\pi r^3} = \frac{3}{4}r$$

$$\text{Resistance from Surface to Average Radius} = \frac{r(1-\frac{3}{4})}{4\pi k(\frac{3}{4})r^2} = \frac{1}{12\pi kr}$$

$$\text{Surface Area per unit Volume} = (1 - \beta) \frac{4\pi r^2}{\frac{4}{3}\pi r^3} = 6 \frac{(1-\beta)}{d_s}$$

$$\text{Resistance times Area} = \frac{4\pi r^2}{12\pi kr} = \frac{d_s}{6k}$$

## STACKED FOIL REGENERATORS

---

For Planar Elements

$$\text{Surface Area} = 2dxdy$$

$$\text{Volume} = dxdy(l_t + l_g)$$

$$\text{Average Radius} = \frac{1}{4}l_t$$

$$\text{Resistance from Surface to Average Radius} = \frac{\frac{1}{4}l_t}{k2dxdy} = \frac{l_t}{8kdxdy}$$

$$\text{Surface Area per unit Volume} = \frac{2}{l_t+l_g}$$

$$\text{Resistance times Area} = \frac{l_t 2dxdy}{8kdxdy} = \frac{l_t}{4k}$$

**Table 8.2: User inputs and correlations for various properties based on heat exchanger type: Fin Enhanced Surface, Fin Connected Channels. Correlations from [35] unless otherwise indicated.**

	Heat Exchanger Type		
	Fin Enhanced Surface	Fin Connected Channels (Rectangular)	Fin Connected Channels (Triangular)
Inputs	Fin Separation ( $l_g$ ) Fin Thickness ( $l_{th}$ ) Roughness ( $l_r$ ) Surface to build off of	Gas space between source channels ( $l_{c,g}$ ) Source channel width ( $l_{c,w}$ ) Source channel wall thickness ( $l_{c,wth}$ ) Surface roughness ( $l_r$ ) Base Width / Fin Separation ( $l_{f,g}$ ) Fin Thickness ( $l_{th}$ )	
Porosity ( $\beta$ )	$\frac{l_g}{l_g + l_{th}}$	$\frac{l_f}{l_f + l_{c,w}} \frac{l_{f,g}}{l_{f,g} + l_{th}}$	$\frac{l_{c,g}}{l_{c,g} + l_{c,w}} \frac{l_{f,g}}{l_{f,g} + l_{eff}}$
Hydraulic Diameter ( $d_h$ )	$\frac{2l_g}{1 + \frac{l_g}{l_f}}$	$\frac{2l_{f,g}}{1 + \frac{l_{f,g}}{l_f}}$	$\frac{0.5l_{f,g}}{1 + \sqrt{1 + \frac{1}{\tan^2 \Phi}}}$ *
Laminar Friction Factor ( $N_f$ )	$\frac{-43.94\alpha^3 + 123.2\alpha^2 - 118.31\alpha + 96}{N_{Re}}$ *		$\frac{-6.1181\Phi^2 + 8.8371\Phi + 49.433}{N_{Re}}$ *
Laminar Nusselt Number ( $N_{Nu}$ )	8.23		$2.66\theta\Phi^5 - 12.19\Phi^4 + 21.63\Phi^3 - 19.9\Phi^2 + 8.92\Phi + 0.956$ **
Lam. Cond. Enhancement Factor ( $N_k$ )		1	
Turbulent Friction Factor ( $N_f$ )	$(-0.0086c^3 + 0.0223c^2 - 0.0247c + 0.121) \left(\frac{l_r}{d_h} + \frac{68}{N_{Re}}\right)^{0.25}$ *		$(-0.0184\theta^2 + 0.0414\theta + 0.0847) \left(\frac{l_r}{d_h} + \frac{68}{N_{Re}}\right)^{0.25}$ *
Turbulent Nusselt Number ( $N_{Nu}$ )	$0.025N_{Re}^{0.79} \cdot N_{Pr}^{0.33}$		$\frac{\left(\frac{N_f}{8}\right) N_{Re} N_{Pr}}{1.07 + 12.7 \left(\frac{2}{N_{Pr}^3} - 1\right) \sqrt{\frac{N_f}{8}}}$ ***
Turbulent Conduction Enhancement Factor ( $N_k$ )		$0.022N_{Re}^{0.75} N_{Pr}$	
Extra Equations:	Depending on the orientation of the fin, Fin Length ( $l_f$ ) is $\begin{cases} x_o - x_i \\ or \\ \min(y_{o\theta} - y_{i\theta}) \end{cases}$ $c = \min\left(\frac{l_f}{l_g}, \frac{l_g}{l_f}\right)$	$l_f = l_{c,g} - l_{th}$ $c = \min\left(\frac{l_{f,g}}{l_f}, \frac{l_f}{l_{f,g}}\right)$	$l_f = l_{c,g} - l_{th}$ $\Phi = \tan^{-1}\left(\frac{0.5l_{f,g}}{l_f}\right)$ $l_{eff} = \frac{l_{th}}{\cos(\Phi)}$

\*Polynomial Fit to Table Data of weights [71] multiplied onto equation for circular pipe [35]

\*\*Constant Wall Temperature inflow and peripheral directions [72]

\*\*\*Gnielinski correlation [73]

**Table 8.3: User inputs and correlations for various properties based on heat exchanger type: Tube and Plate Heat Exchangers.**

Continuously Finned	
Staggered – always turbulent	
Inputs	Spacing Perpendicular to Flow ( $l_{perp}$ )
	Spacing Parallel to Flow ( $l_{para}$ )
	Fin Thickness ( $l_{th}$ )
	Fin Separation ( $l_g$ )
	Tube Outer Diameter ( $d_o$ ) Tube Inner Diameter ( $d_i$ )
Porosity ( $\beta$ )	$\left(1 - \frac{\frac{\pi}{4} d_o^2}{l_{perp} \cdot l_{para}}\right) \left(\frac{l_g}{l_g + l_{th}}\right)$
Hydraulic Diameter ( $d_h$ )	$\frac{4\beta \cdot l_{perp} \cdot l_{para} \cdot l_{th} \cdot l_g}{\pi \cdot d_o \cdot l_g + 2(l_{perp} \cdot l_{para} - \pi \cdot d_o)}$
Friction Factor ( $N_f$ )	$c'_4 \cdot N_{Re}^{-0.521} + c'_5 \left(\frac{d_o}{d_h} \cdot N_{Re}\right)^{-0.18} *$
Nusselt Number ( $N_{Nu}$ )	If $N_r \geq 4$ $0.14 \left(\frac{l_{para}}{l_{perp}}\right)^{0.502} \left(\frac{l_g}{d_o}\right)^{0.031} N_{Re}^{0.672} \cdot N_{Pr}^{0.333} *$ Else $c_2 \cdot N_{Re}^{c_3} \cdot N_{Pr}^{0.333} *$
Conduction Enhancement Factor ( $N_k$ )	1
Equations: = $\frac{N_r \cdot \text{total streamwise distance}}{l_{para}}$	$c_1 = 0.14 \left(\frac{l_{para}}{l_{perp}}\right)^{0.502} \left(\frac{l_g}{d_o}\right)^{0.031}$ $c_2 = c_1 \cdot 0.991 \left(2.24 \left(\frac{4}{N_r}\right)^{0.031}\right)^{(-0.607(4-N_r))}$ $c_3 = 1 + (-0.092 * 0.607(4 - N_r) - 0.328)$ $c_4 = \frac{S_f}{S_{tube} + S_f}$ $c_5 = (1 - c_4)(1 - \beta_f)$ $c'_4 = 2.032 c_4 \left(\frac{l_{para}}{d_o}\right)^{1.318}$ $c'_5 =$ $4c_5 \left[\left(\frac{l_{perp}}{l_{para}}\right)^2 \frac{l_{perp}}{l_{para}} \ 1\right] \begin{bmatrix} -0.108 & 0.730 & -0.213 \\ 0.314 & -1.296 & 0.561 \\ -0.234 & 1.034 & -0.747 \end{bmatrix} \cdot \left[\left(\frac{l_{para}}{d_o}\right)^2 \frac{l_{para}}{d_o} \ 1\right] **$

\* [74]

\*\* Combined formula of [74], with component representing pressure drop of bare staggered tube banks derived from data of [75].

**Table 8.4: User inputs and correlations for various properties based on heat exchanger type: Individually Finned Tube Heat Exchangers.**

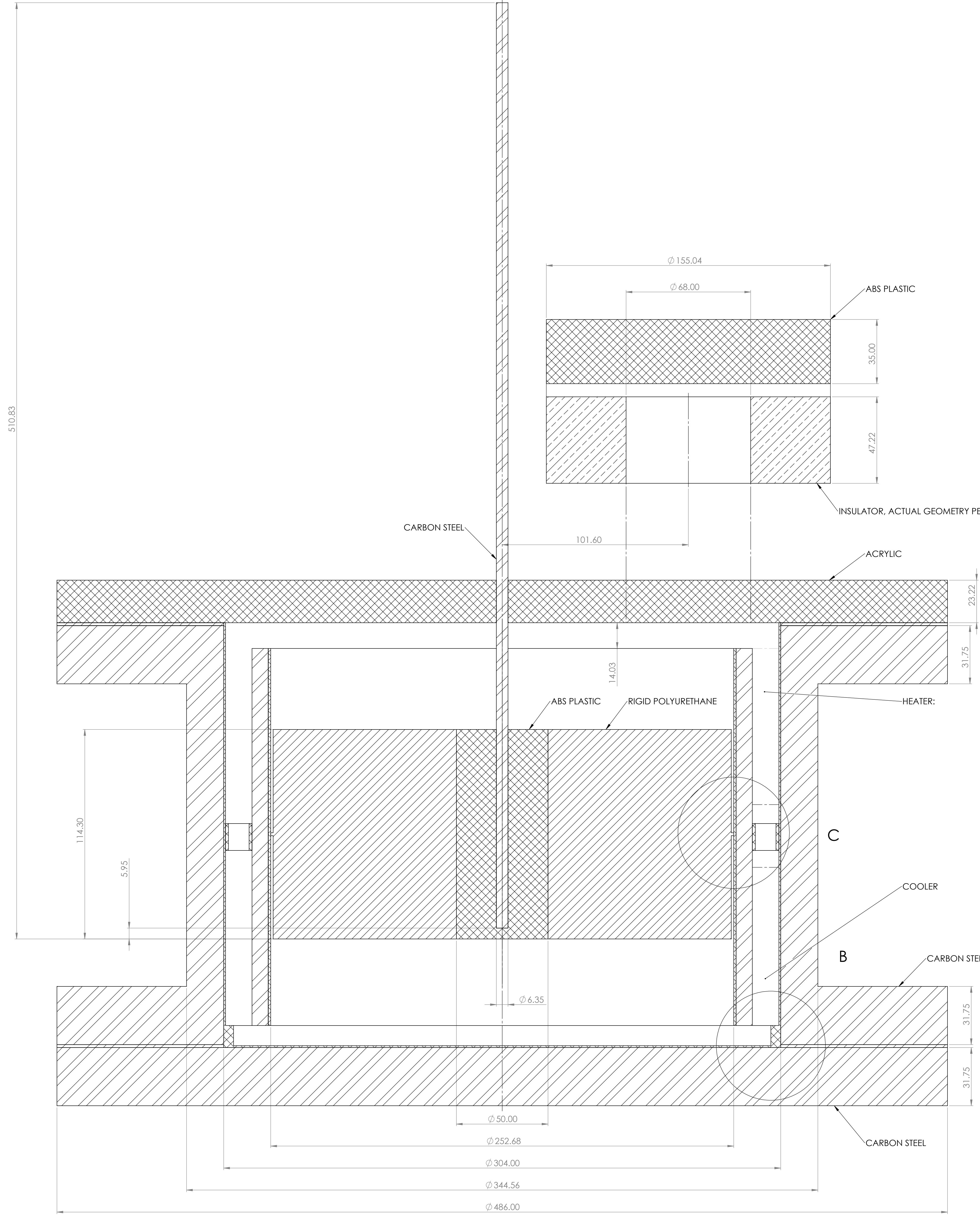
Individually Finned	
Staggered	
Inputs	Fin Length ( $l_f$ )
Porosity ( $\beta$ )	$1 - \pi \frac{\left(\frac{d_o}{2} + l_f\right)^2 - \left(\frac{l_g}{l_{th} + l_g}\right) \left(\left(\frac{d_o}{2} + l_f\right)^2 - \left(\frac{d_o}{2}\right)^2\right)}{l_{perp} \cdot l_{para}}$
Hydraulic Diameter ( $d_h$ )	$\frac{4\beta \cdot l_{perp} \cdot l_{para} \cdot l_{th} \cdot l_g}{\pi \cdot d_o \cdot l_g + 2\pi \left(\left(\frac{d_o}{2} + l_f\right)^2 - \left(\frac{d_o}{2}\right)^2\right) + \pi \cdot l_f (d_o + l_f)}$
Laminar Friction Factor ( $N_f$ )	Assumed to be always turbulent
Laminar Nusselt Number ( $N_{Nu}$ )	Assumed to be always turbulent
Laminar Conduction Enhancement Factor ( $N_k$ )	$1 + 0.5(0.5113^{-2.91} (N_{Re} N_{Pr})^{0.66})$
Turbulent Friction Factor ( $N_f$ )	<p>If <math>l_f/d_o &lt; 0.09</math> (Low Finned Tubes)</p> $\frac{4 \left(1.748 \left(\frac{l_f \cdot d_o}{l_g \cdot l_{para}}\right)^{0.1738} \left(\frac{d_o}{l_{perp}}\right)^{0.599}\right)}{N_{Re}^{0.233}} *$ <p>Else (High Finned Tubes)</p> $\frac{4 \left(9.465 \left(\frac{d_o}{l_{perp}}\right)^{0.927} \left(\frac{l_{perp}}{\sqrt{l_{perp}^2 + l_{para}^2}}\right)^{0.515}\right)}{N_{Re}^{0.316}} **$
Turbulent Nusselt Number ( $N_{Nu}$ )	<p>If <math>l_f/d_o &lt; 0.09</math> (Low Finned Tubes)</p> $0.255 \left(\frac{2l_f + d_o}{l_g}\right) N_{Re}^{0.7} \cdot N_{Pr}^{0.333***}$ <p>Else (High Finned Tubes)</p> $0.134 \left(\frac{l_g}{l_f}\right)^{0.2} \left(\frac{l_g}{l_{th}}\right)^{0.11} N_{Re}^{0.681} \cdot N_{Pr}^{0.333***}$
Turbulent Conduction Enhancement Factor ( $N_k$ )	1

\* Chai [76]  
\*\* Webb [74]  
\*\*\* Ganguli & Yilmaz [77]

**Table 8.5: User inputs and correlations for various properties based on heat exchanger type: Bare Tube Banks (internal). Correlations from [35] unless otherwise indicated.**

Bare Tube Banks (internal)	
Staggered	
Inputs	Tube Spacing ( $l_{tube,s}$ ) (Circle Packed Arrangement)
	Tube Outer Diameter ( $d_o$ )
	Tube Inner Diameter ( $d_i$ )
Porosity ( $\beta$ )	$\frac{\pi \cdot d_i^2}{\sqrt{3}l_{tube,s}^2}$
Hydraulic Diameter ( $d_h$ )	$d_i$
Laminar Friction Factor ( $N_f$ )	$\frac{64}{N_{Re}}$
Laminar Nusselt Number ( $N_{Nu}$ )	6.0
Laminar Conduction Enhancement Factor ( $N_k$ )	1
Turbulent Friction Factor ( $N_f$ )	$0.11 \left( \frac{\epsilon}{d_h} + \frac{68}{N_{Re}} \right)^{0.25}$
Turbulent Nusselt Number ( $N_{Nu}$ )	$0.036 \left( \frac{L}{d_h} \right)^{-0.055} N_{Re}^{0.8} \cdot N_{Pr}^{0.33}$
Turbulent Conduction Enhancement Factor ( $N_k$ )	$0.022 N_{Re}^{0.75} \cdot N_{Pr}$

## **Appendix C. EP-1 Geometry**



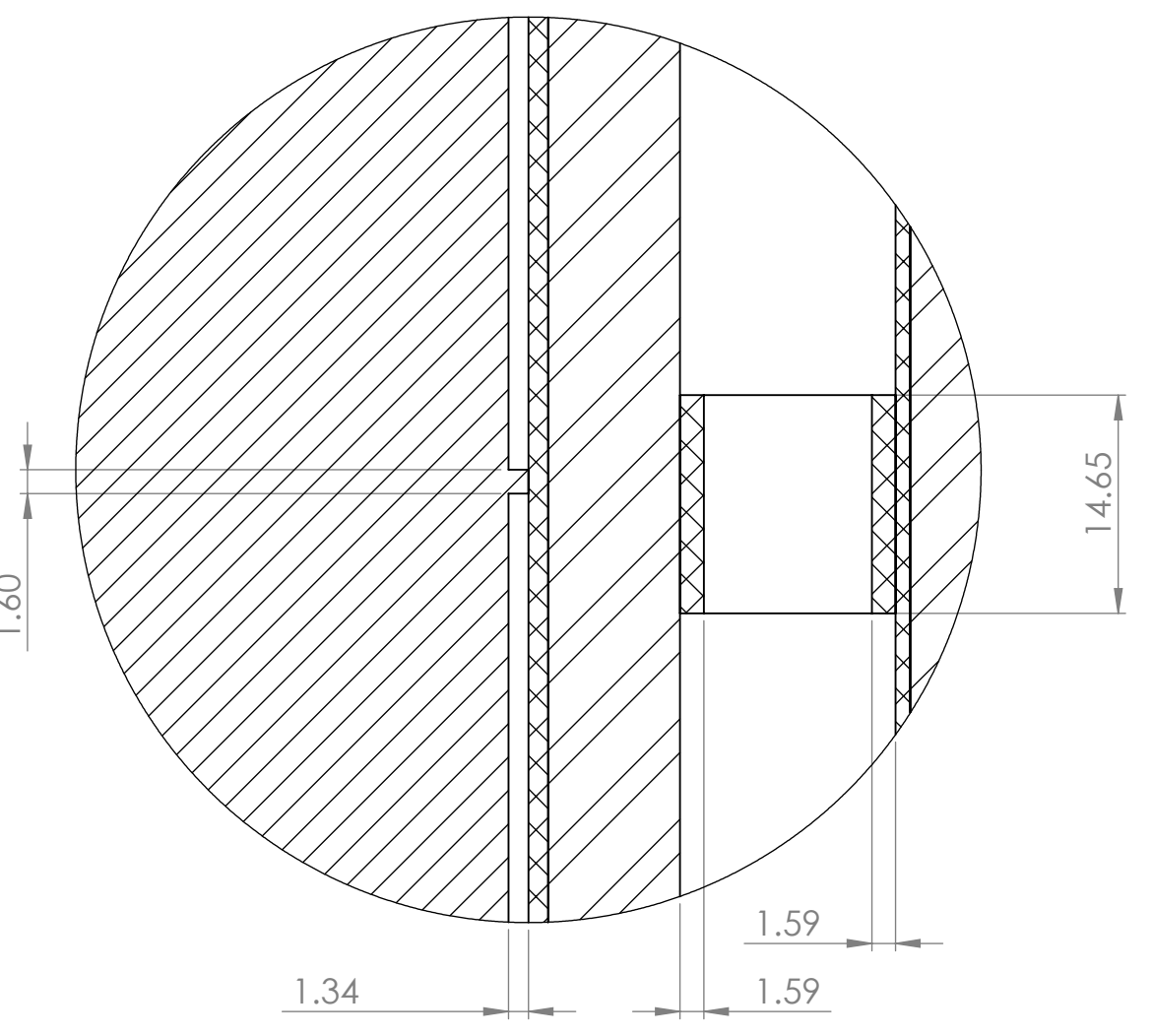
**HEATER: STAGGERED TUBE BANK**  
 MATERIAL: COPPER  
 PERPENDICULAR SPACING: 0.014575 m (BETWEEN ADJACENT NEIGHBOURS)  
 PARALLEL SPACING: 0.00826 m (FROM ROW TO ROW)  
 FIN THICKNESS: 0.000506 m  
 FIN SEPARATION: 0.001306 m  
 FIN LENGTH: 0.003175 m  
 TUBE OUTER DIAMETER: 0.0064 m  
 TUBE INNER DIAMETER: 0.00511 m  
 SURFACE AREA REDUCTION: 0.5 (GIVEN THAT THIS IS AN APPROXIMATION OF A STAGGERED TUBE HEAT EXCHANGER)  
 LENGTH: 0.09255 m

**COOLER: SEE HEATER**

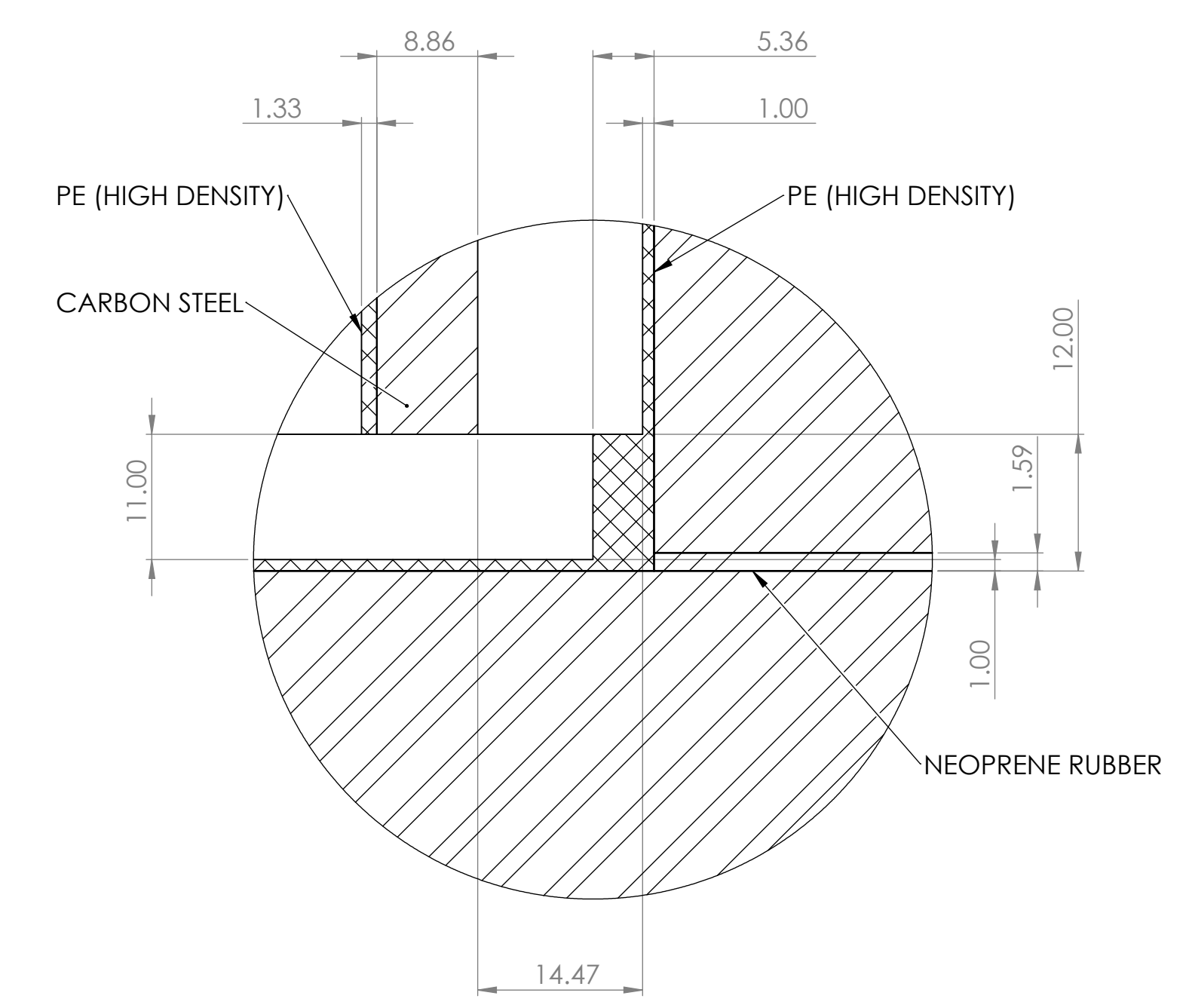
**REGENERATOR:**  
 MATERIAL: ABS PLASTIC  
 GAP: 0.0015 m  
 THICKNESS: 0.000902 m  
 ROUGHNESS: 0.0003 m  
 LENGTH: 0.01465 m

**DISPLACER PISTON:**  
 STROKE: 0.115 m  
 CROSS-SECTIONAL AREA: 0.0490874 m<sup>2</sup>  
 CONNECTING ROD / CRANK ARM LENGTH RATIO: 6

**POWER PISTON:**  
 STROKE: 0.09 m  
 CROSS-SECTIONAL AREA: 0.018878 m<sup>2</sup>  
 CONNECTING ROD / CRANK ARM LENGTH RATIO: 2



**DETAIL C**  
 SCALE 2 : 1



**DETAIL B**  
 SCALE 2 : 1



## Appendix D. Motion Profile Mathematics for EP-1 Studies

For all elliptical profiles the following equations convert the number of lobes ( $n$ ) and the Elliptical Factor ( $e$ ) and lobe phase ( $\phi_{lobe}$ ) into the resulting series of angles for input into the attached slider crank.

$$C = \frac{\sqrt{1 + (n^2 - 1)(1 - e^2)} + e}{n(1 - e)}$$

$$\theta' = \text{atan}(C \cdot \tan(\theta + \phi_{off})) + \phi_{lobe}$$

For all of the following profiles the conversion of the translated angle is as follows. This converts it into the distorted harmonic motion produced by slider crank mechanisms.

$$\theta'_2 = \text{asin}\left(-\frac{l_{cr} \sin(\theta')}{l_{con}}\right)$$

$$x = l_{con}(\cos(\theta'_2) - 1) + l_{cr}(\cos(\theta') + 1)$$

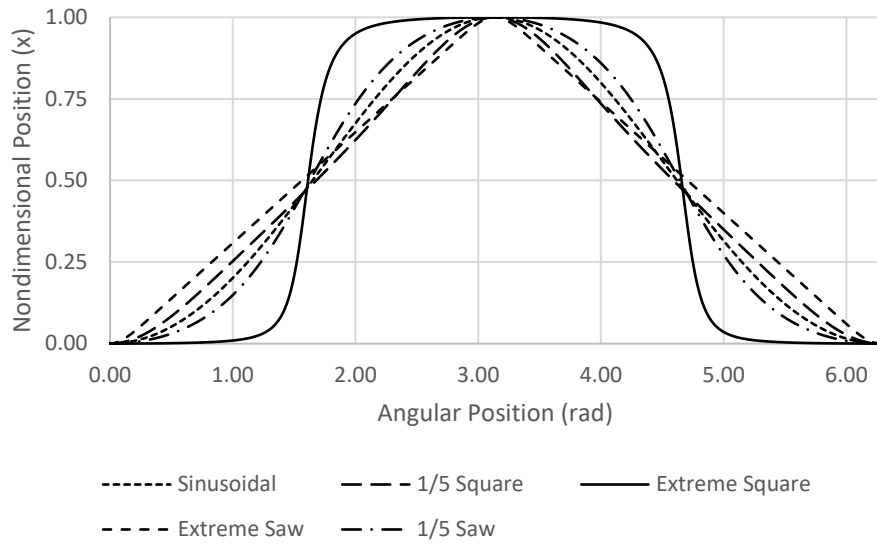
The final profile is shifted numerically post calculation.

Motion	Number of Lobes ( $n$ )	Elliptical Factor ( $e$ )	Phase Shift ( $\phi_{off}$ )	Lobe Phase ( $\phi_{lobe}$ )
Sinusoidal	1	0	0	0
1/5 Elliptical for Square Wave	2	0.2	$\frac{\pi}{2}$	$\frac{\pi}{2}$
1/5 Elliptical for Saw Wave	2	0.2	$\pi$	0
Extreme Square Wave	2	0.8	$\frac{\pi}{2}$	$\frac{\pi}{2}$

Extreme Saw Wave Due to the extreme necking that occurs with a highly elliptical set of gears set into the saw wave configuration, a synthesized version was created, which was originally seeded as a pure saw wave, followed by several iterations of smoothing filter.

The Extreme Saw Wave is defined by a smoothed and normalized saw wave to avoid the extreme necking that occurs when a saw wave is constructed from a set of highly elliptical gears.

The non-phase-shifted profiles are displayed below.



**Figure D.1: Different motion profiles tested – using a connecting arm to crank arm ratio of 6.**

When the mechanism orientation is stated as downwards the profile will appear inverted as well.

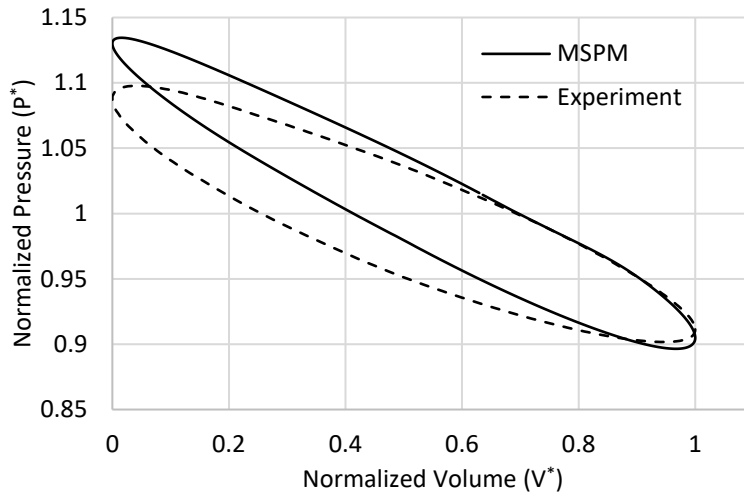
## Appendix E. EP-1 Studies PV Diagrams

Included in the power curves produced for each mechanical arrangement, a series of simulations were taken to line up with experiments conducted by Nicol-Seto [61]. These test cases and the simulation-experiment percent error are as follows. The percent error is calculated by comparing the area under the curve associated with the power piston, that is, the pressure measured at the power piston, plotted against the volume of the power piston.

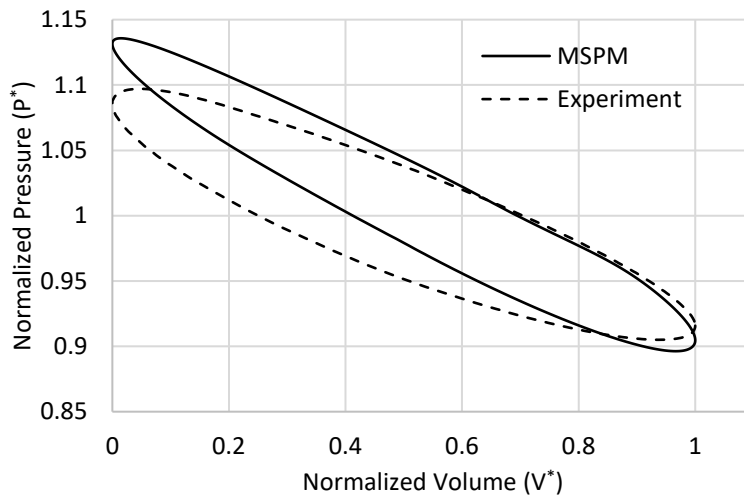
Test #	Gearing (DP, PP)	Speed (Hz)	Exp. Normalized Cycle Energy	MSPM: Normalized Cycle Energy	Power Piston PV error (%)
1	Sinusoidal	1.1055	11.01	8.21	25.4%
2		1.5257	11.26	8.27	26.6%
3		1.8735	11.57	7.83	32.3%
4		2.2606	12.01	8.13	32.3%
5	1/5 Elliptical for Square Wave Sinusoidal	0.582	10.21	8.14	20.3%
6		0.8818	10.65	8.21	22.9%
7		1.2407	10.89	8.24	24.3%
8		1.6444	11.43	7.46	34.7%
9	1/5 Elliptical for Square Wave	0.5558	11.45	7.79	32.0%
10		0.9051	12.24	7.80	36.3%
11	1/5 Elliptical for Square Wave	1.1992	12.54	7.83	37.6%
12		1.5825	13.05	7.43	43.1%

The following plots include the simulated as well as experimental PV diagrams for the 12 tests.

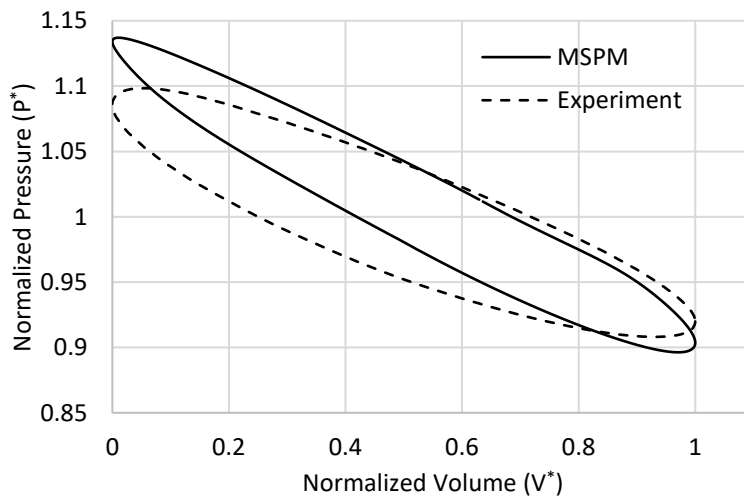
Test 1:



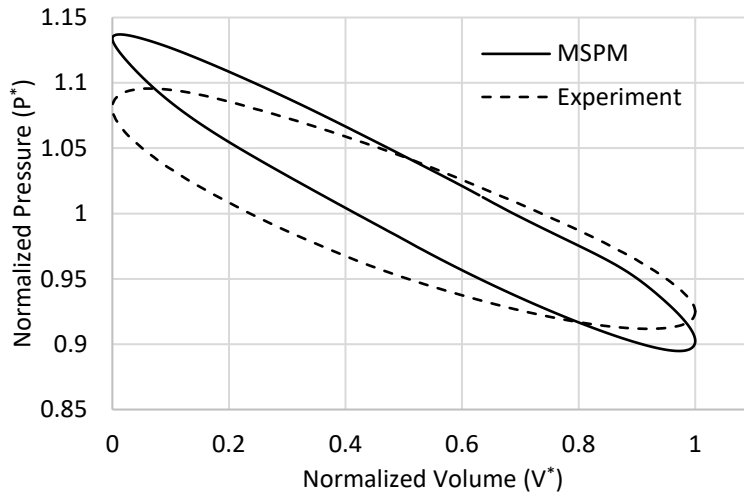
Test 2:



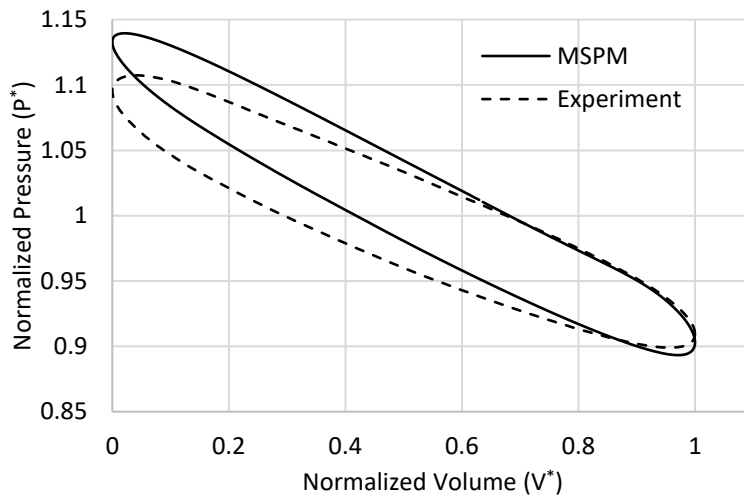
Test 3:



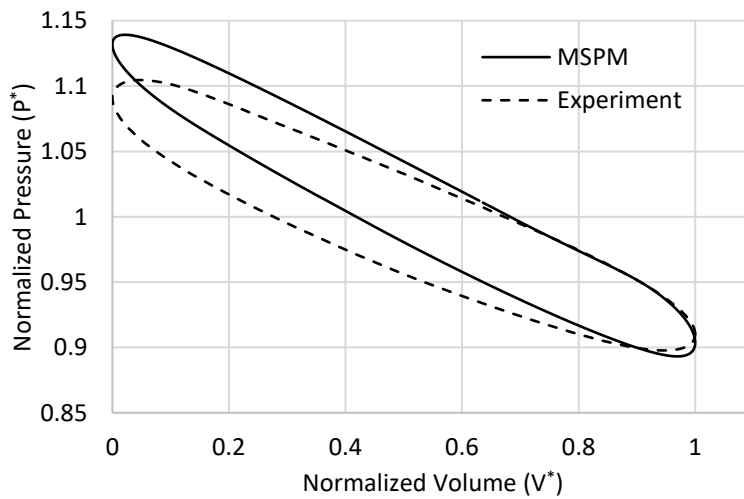
Test 4:



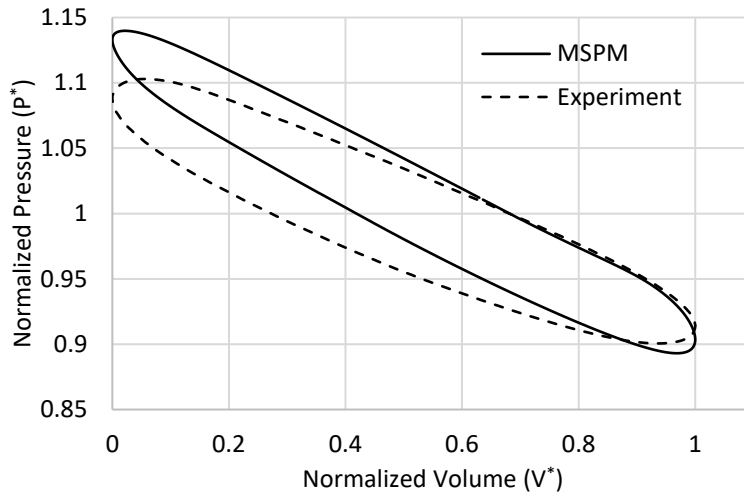
Test 5:



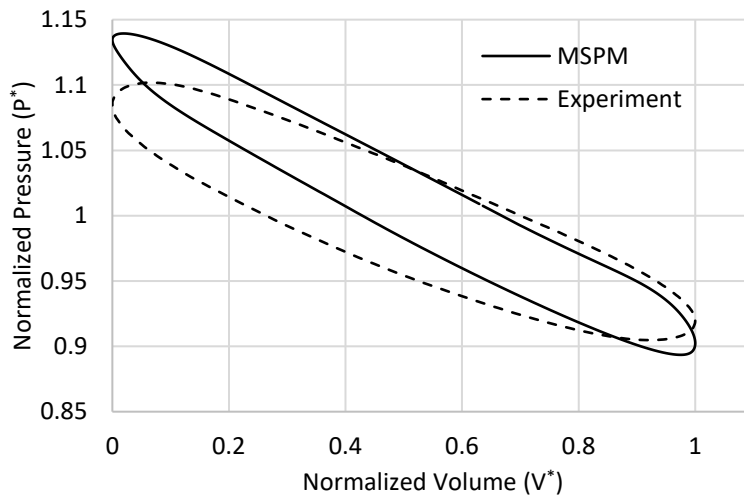
Test 6:



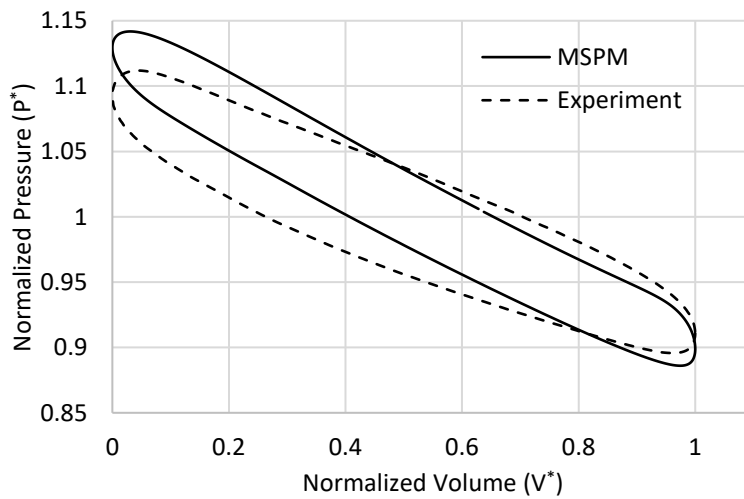
Test 7:



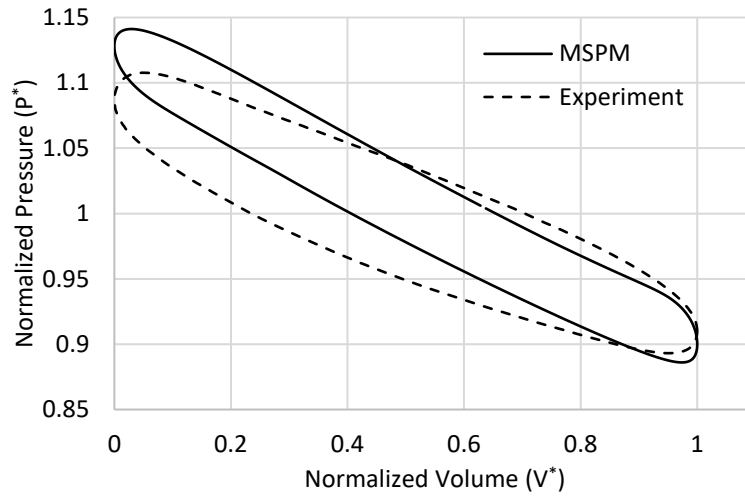
Test 8:



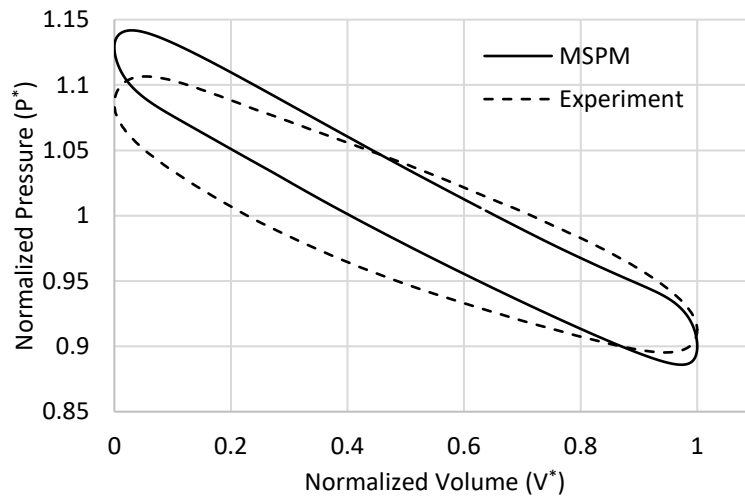
Test 9:



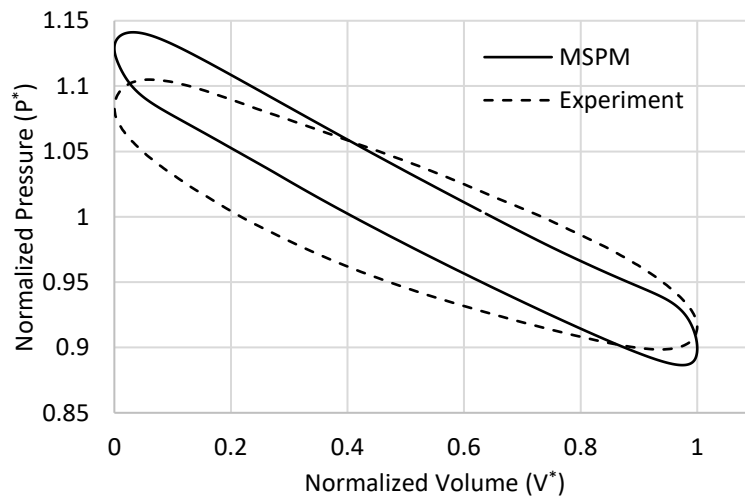
Test 10:



Test 11:



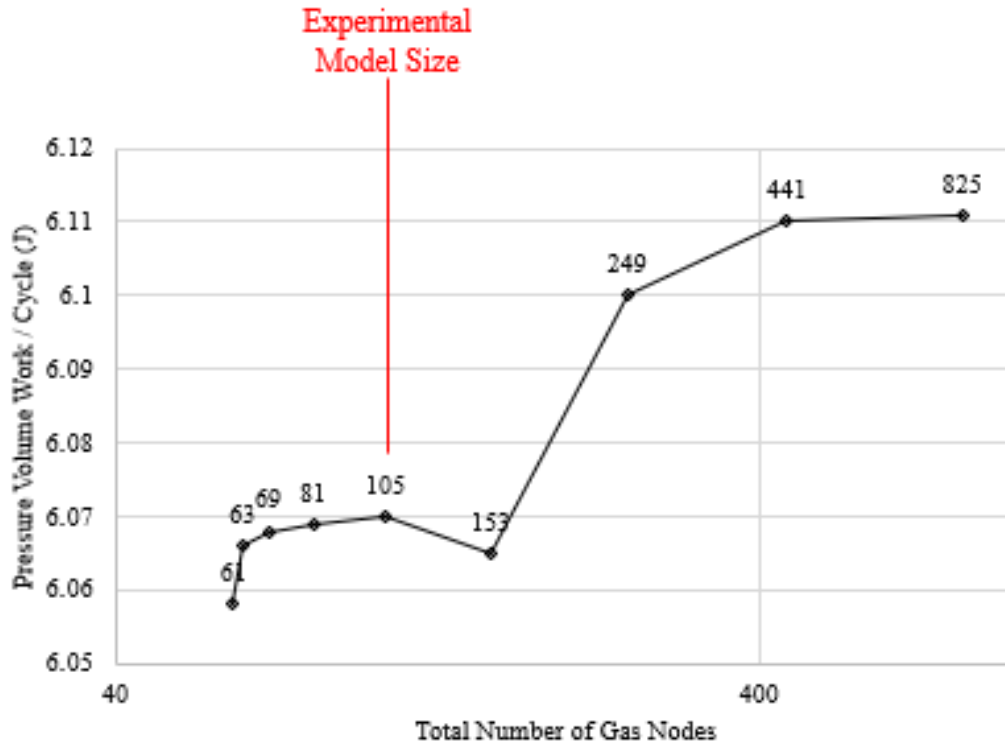
Test 12:



## Appendix F. Mesh Sensitivity

### G.1. EPM-1 Model

A correctly performed mesh independency test should capture a series of tests in which the model decays on a predictable, mono-directional rate towards the exact value (given the assumptions made to construct the model). Sensitivity of the results on the mesh are plotted on Figure G.1.



**Figure G.1: Sensitivity of indicated cycle energy on Test Set 1 at 1 Hz**

The amount of error between the highest mesh size and the mesh which was used for the experiments is equal to 0.8%. The final order of convergence,  $p$ , is equal to 3.84. This is calculated via the following formula:

$$p = \frac{\ln\left(\frac{y_N - y_{N-1}}{y_{N-1} - y_{N-2}}\right)}{\ln\left(\frac{x_{N-1}}{x_N}\right)} = 3.84$$

A value in the range of 3 would be expected, given that MSPM uses a 3<sup>rd</sup> order polynomial for interpolation.



## **Appendix G. MATLAB CODE**

The MATLAB code can found in this appendix or hosted on GITHUB.

### **G.1. GUI**

## Simulation Interface V5

The following code is the main GUI of the software, this is paired with the actual controls which are stored as a .fig file. There are several main sections of this code:

1. Required Header Components
2. Insert, Select, Delete and Animate buttons, which all use the “GUI\_ButtonDownFcn” under different modes to provide different functionality.
3. Green Highlighting of active buttons and initialization of the modes is handled by ButtonCore
4. Save, Load Functionality
5. Show Option set of functions
6. Box & Recenter Zooms
7. Recording & Animation Options
8. Other Simulation Options
9. Run Options (Run & Run Test Set)

```
function varargout = SimulationInterfaceV5(varargin)
% SIMULATIONINTERFACEV5 MATLAB code for SimulationInterfaceV5.fig
%     SIMULATIONINTERFACEV5, by itself, creates a new SIMULATIONINTERFACEV5 or raises the
existing
%     singleton*.
%
%     H = SIMULATIONINTERFACEV5 returns the handle to a new SIMULATIONINTERFACEV5 or the handle
to
%     the existing singleton*.
%
%     SIMULATIONINTERFACEV5('CALLBACK',hObject,eventData,handles,...) calls the local
%     function named CALLBACK in SIMULATIONINTERFACEV5.M with the given input arguments.
%
%     SIMULATIONINTERFACEV5('Property','Value',...) creates a new SIMULATIONINTERFACEV5 or
raises the
%     existing singleton*. Starting from the left, property value pairs are
%     applied to the GUI before SimulationInterfaceV5_OpeningFcn gets called. An
%     unrecognized property name or invalid value makes property application
%     stop. All inputs are passed to SimulationInterfaceV5_OpeningFcn via varargin.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help SimulationInterfaceV5

% Last Modified by GUIDE v2.5 16-Oct-2020 09:07:43

% Begin initialization code - DO NOT EDIT
```

```

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @SimulationInterfaceV5_OpeningFcn, ...
    'gui_OutputFcn',  @SimulationInterfaceV5_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before SimulationInterfaceV5 is made visible.
function SimulationInterfaceV5_OpeningFcn(hObject, ~, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to SimulationInterfaceV5 (see VARARGIN)

% UIWAIT makes SimulationInterfaceV5 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

%% Choose default command line output for SimulationInterfaceV5
handles.output = hObject;

%% Generate space for mouse events
handles.MODE = '';
handles.SelectCon = Connection.empty;
handles.IndexC = 1;
handles.SelectBod = Body();
handles.IndexB = 1;
handles.SelectGroup = Group.empty;
handles.IndexG = 1;

%% Set Initial Values for Display Options
handles.InterGroupDistance = 0.05;
handles.ClickTolerance = 0.1;

%% Generate Default Model
handles.Model = Model();
handles.Model.AxisReference = handles.GUI;
handles.corner_points = [];
handles.SimulationParameters = cell(0);
DistributeGroup(handles);
show_Model(handles);

%% Object Properties
handles.SData = SelectionListData();
handles.SData.Code = '';
handles.SData.ListObjs = ListObj.empty;
handles.DropDownMode = '';
updateSelectionList(handles);

%% Optimization Stuff
handles.OptimizationStudyIndex = 0;

%% Show Options
set(handles.showGroups, 'Value', handles.Model.showGroups);
set(handles.showBodies, 'Value', handles.Model.showBodies);
set(handles.showConnections, 'Value', handles.Model.showConnections);
set(handles.showLeaks, 'Value', handles.Model.showLeaks);
set(handles.showBridges, 'Value', handles.Model.showBridges);
set(handles.showInterConnections, 'Value', handles.Model.showInterConnections);
set(handles.showEnvironmentConnections, 'Value', handles.Model.showEnvironmentConnections);

```

```

set(handles.showNodes,'Value',handles.Model.showNodes);
set(handles.showSensors,'Value',handles.Model.showSensors);
set(handles.showRelations,'Value',handles.Model.showRelations);
set(handles.RelationMode,'String','On')
handles.Model.RelationOn = true;

%% Update handles structure
guidata(hObject, handles);

% --- Outputs from this function are returned to the command line.
function varargout = SimulationInterfaceV5_OutputFcn(hObject, ~, handles) %#ok<INUSL>
% Get default command line output from handles structure
varargout{1} = handles.output;

%% General button codes
function GUI_ButtonDownFcn(hObject, ~, h)
C = get(hObject,'Currentpoint');
C = C(1,1:2);
if isempty(h.Model.ActiveGroup)
    % Select the group based on where the user clicked
    h.Model.FindGroup(C);
end
switch h.MODE
    case 'InsertBody'
        % Select 4 connections
        switch get(gcf,'SelectionType')
            case 'normal'
                L = true;
            case 'alt'
                L = false;
            case 'extend'
                L = false;
            otherwise
                L = true;
        end
        found = false;
        %% Get round specific information
        switch h.IndexC
            case 1
                DIR = enumOrient.Vertical;
                prompt = 'New Body Inner Radius 1: ';
                OFFSET = 0;
            case 2
                OFFSET = h.SelectCon(1).x;
                if h.SelectCon(1).Orient == enumOrient.Vertical
                    DIR = enumOrient.Vertical;
                    prompt = 'New Body Radial Thickness: ';
                else
                    DIR = enumOrient.Horizontal;
                    prompt = 'New Body Vertical Thickness: ';
                end
            case 3
                OFFSET = 0;
                if h.SelectCon(2).Orient == enumOrient.Vertical
                    DIR = enumOrient.Horizontal;
                    prompt = 'New Body Lower Vertical Position: ';
                else
                    DIR = enumOrient.Vertical;
                    prompt = 'New Body Inner Radius: ';
                end
            case 4
                OFFSET = h.SelectCon(3).x;
                if h.SelectCon(2).Orient == enumOrient.Vertical
                    DIR = enumOrient.Horizontal;
                    prompt = 'New Body Thickness: ';
                else
                    DIR = enumOrient.Vertical;
                    prompt = 'New Body Radial Thickness: ';
                end
            otherwise % We are done here
                found = true;
        end
    end
end

```

```

end

%% Find connection at click
if L && found == false % Left Click
    if h.IndexC == 1
        h.SelectCon(h.IndexC) = ...
        h.Model.ActiveGroup.FindConnection(C);
        found = true;
        fprintf(['Selected Connection: ' ...
            h.SelectCon(h.IndexC).name '.\n']);
    else
        Con = h.Model.ActiveGroup.FindConnection(...
            C,DIR,h.SelectCon(h.IndexC-1));
        if ~isempty(Con)
            h.SelectCon(h.IndexC) = Con;
            fprintf(['Selected Connection: ' ...
                h.SelectCon(h.IndexC).name '.\n']);
            found = true;
        else
            found = false;
        end
    end
end
end

%% Get user Input
if found == false
    % Get User Radius Submission
    DIM = '';
    while ~isnumeric(DIM)
        answer = inputdlg(prompt,'Specify Dimension Window');
        if ~isempty(answer)
            DIM = str2double(answer{1});
        else
            return;
        end
    end
    % If this does not match any Group Connection then CreateNew
    found = false;
    for iCon = h.Model.ActiveGroup.Connections
        if iCon.Orient == DIR && iCon.x == DIM+OFFSET
            h.SelectCon(h.IndexC) = iCon;
            found = true;
        end
    end
    if ~found
        h.SelectCon(h.IndexC) = Connection(DIM+OFFSET,DIR,h.Model.ActiveGroup);
    end
end

%% Iterate or finish up
if h.IndexC == 4
    % Define the body
    matl = [];
    show_Model(h);
    while isempty(matl)
        [matl, tf] = listdlg(...
            'PromptString','Select a material type for this new body:',...
            'SelectionMode','single',...
            'ListString',Material.Source);
    end
    if tf
        newBody = Body(...
            h.Model.ActiveGroup,...
            h.SelectCon,...
            Material(Material.Source{matl}));
        if handles.Model.ActiveGroup.isOverlapping(newBody)
            fprintf('XXX New Body overlaps, creation cancelled XXX\n');
            handles.Model.clearHighLighting();
        else
            h.Model.ActiveGroup.addBody(newBody);
        end
    end
end
end

```

```

else
    fprintf('XXX You must select a material, creation cancelled XXX\n');
    h.Model.clearHighLighting();
end
h.IndexC = 1;
else
    h.Model.HighLight(h.SelectCon(1:h.IndexC));
    h.IndexC = h.IndexC + 1;
end
case 'InsertGroup'
    % Will simply define a vertical Group at the next slot
    % Determine where the user clicked
    C = get(gca,'Currentpoint'); C = C(1,1:2);
    h.Model.addGroup(Group(h.Model,Position(C(1),0,pi/2),[]));
    h.Model.distributeGroup(h.InterGroupDistance);
case 'InsertBridge'
    % Select 2 horizontal connections and 2 bodies
    if h.IndexC == 1
        if h.IndexB == 1
            % Picking the first Connection
            if isempty(h.Model.ActiveGroup)
                ChangeGroup_Callback(hObject, [], h);
            end
            Con = h.Model.ActiveGroup.FindConnection(C);
            if ~isempty(Con)
                h.SelectCon(h.IndexC) = Con;
                h.Model.HighLight(Con);
                h.IndexC = 2;
                set(h.message,'String','[click] Select the Foundation Body');
            end
        end
    elseif h.IndexC == 2
        if h.IndexB == 1
            % Picking the first Body
            Bod = h.SelectCon(1).findConnectedBody(C);
            if ~isempty(Bod)
                h.SelectBody(h.IndexB) = Bod;
                h.Model.HighLight(Bod);
                h.IndexB = 2;
                set(h.message,'String','[click] Select the Connection for the other body (the
Body that may shift)');
            end
        else
            % Picking the second Connection
            ChangeGroup_Callback(hObject, [], h);
            Con = h.Model.ActiveGroup.FindConnection(C);
            if ~isempty(Con)
                h.SelectCon(h.IndexC) = Con;
                h.Model.HighLight(Con);
                h.IndexC = 3;
                set(h.message,'String','[click] Select the associated body');
            end
        end
    end
else
    if h.IndexB == 2
        % Picking the second Body
        Bod = h.SelectCon(2).findConnectedBody(C);
        if ~isempty(Bod)
            % Finish and Create
            if h.SelectCon(1).Orient == h.SelectCon(2).Orient
                if h.SelectCon(1).Orient == enumOrient.Vertical
                    prompt = 'Select the height adjustment for body 2 as it is placed
around body 1';
                    [~,~,defaultval,~] = h.SelectBody(1).limits(enumOrient.Vertical);
                else
                    prompt = 'Select the radial offset distance';
                    defaultval = 0;
                end
            end
        else
            prompt = 'Select the vertical center offset for the horizontal face to be
up the vertical face';
        end
    end
end

```

```

        if h.SelectCon(1).Orient == enumOrient.Vertical
            [~,~,defaultval,~] = h.SelectBody(1).limits(enumOrient.Vertical);
        else
            [~,~,defaultval,~] = h.SelectBody(2).limits(enumOrient.Vertical);
        end
    end
    x = inputdlg(prompt,'Specify Bridge Offset',[1 35],[num2str(defaultval)]);
    % Define the Bridge
    if ~isempty(x)
        h.SelectBody(h.IndexB) = Bod;
        h.Model.HighLight(Bod);
        h.Model.addBridge(Bridge(...
            h.SelectBody(1),h.SelectBody(2),...
            h.SelectCon(1),h.SelectCon(2),str2double(x{1})));
        h.IndexC = 1;
        h.IndexB = 1;
        set(h.message,'String','---');
    end
end
end
end
case 'InsertLeakConnection'
% Select 2 horizontal connections and 2 bodies/Environments
case 'InsertSensor'
% Select a group
C = C(1,1:2);
% Select a body
[~, objects] = h.Model.findNearest(C,h.ClickTolerance);
if ~isempty(objects)
    for obj = objects
        if isa(obj{1},'Body')
            h.Model.HighLight(obj{1});
            h.Model.addSensor(Sensor(h.Model,obj{1}));
        end
    end
end
case 'InsertPVoutput'
% Find, within a radius of confidence, the nearest Body
C = C(1,1:2);
[~, objects] = h.Model.findNearest(C,h.ClickTolerance);
if ~isempty(objects)
    for obj = objects
        if isa(obj{1},'Body')
            if obj{1}.matl.Phase == enumMaterial.Gas
                h.Model.addPVoutput(PVoutput(obj{1}));
                set(h.message,'String',['PVoutput added to Body: ' obj{1}.name]);
            else
                set(h.message,'String','Must select a Gas Body');
            end
        end
    end
end
case 'InsertNonConnection'
% Select 2 horizontal connections and 2 bodies
Bod = Body.empty;
if h.IndexB == 1
    % Picking the first Body
    [~, objects] = h.Model.findNearest(C,h.ClickTolerance);
    for obj = objects; if isa(obj{1},'Body'); Bod = obj{1}; break; end; end
    if ~isempty(Bod)
        h.SelectBody(h.IndexB) = Bod;
        h.Model.HighLight(Bod);
        h.IndexB = 2;
        set(h.message,'String','[click] Select the second body, or click in open space to
select the environment');
    end
elseif h.IndexB == 2
    % Picking the first Body
    [~, objects] = h.Model.findNearest(C,h.ClickTolerance);
    for obj = objects; if isa(obj{1},'Body'); Bod = obj{1}; break; end; end
    if Bod ~= h.SelectBody(1)

```

```

        if ~isempty(Bod)
            h.SelectBody(h.IndexB) = Bod;
            h.Model.HighLight(Bod);
            set(h.message,'String','---');
            h.Model.addNonConnection(...
                NonConnection(h.SelectBody(1),h.SelectBody(2)));
        end
        return;
    end
    % No object was selected, select the environment instead
    set(h.message,'String','---');
    h.Model.addNonConnection(...
        NonConnection(h.SelectBody(1),h.Model.Surroundings));
end
case 'InsertCustomMinorLoss'
    % Find, within a radius of confidence, the nearest body
    C = C(1,1:2);
    [~, objects] = h.Model.findNearest(C,h.ClickTolerance);
    if ~isempty(objects)
        for obj = objects
            if isa(obj,'Body')
                h.SelectBody(h.IndexB) = obj;
                if (h.IndexB == 2)
                    % Finalize Custom Minor Loss
                    h.IndexB = 1;
                    h.Model.AddCustomMinorLoss(...
                        CustomMinorLoss(...
                            h.SelectBody(1),...
                            h.SelectBody(2)));
                end
                h.IndexB = 2;
            end
        end
    end
end
case 'Select'
    % Find, within a radius of confidence, the nearest...
    % Body, Group, Connection, Bridge and Leak Connection
    C = C(1,1:2);
    [names, objects] = h.Model.findNearest(C,h.ClickTolerance);
    if ~isempty(names)
        if length(names) > 1
            [index,tf] = listdlg(...
                'PromptString','Which Object did you select?',...
                'ListString',names,...
                'SelectionMode','single',...
                'ListSize',[400 100]);
        else
            index = 1;
            tf = true;
        end
        if tf
            h.Model.switchHighLighting(objects{index});
        end
    end
end
case 'MultiSelect'
    % Find, within a radius of confidence, the nearest...
    % Body, Group, Connection, Bridge and Leak Connection
    C = C(1,1:2);
    [names, objects] = h.Model.findNearest(C,h.ClickTolerance);
    if ~isempty(names)
        if length(names) > 1
            [index,tf] = listdlg(...
                'PromptString','Which Object did you select?',...
                'ListString',names,...
                'SelectionMode','single',...
                'ListSize',[400 100]);
        else
            index = 1;
            tf = true;
        end
        if tf

```



```

        h.Model.HighLight(objects{index});
    end
end
case 'InsertRelation'
% Find, within a radius of confidence, the nearest connection
C = C(1,1:2);
[~, objects] = h.Model.findNearest(C,h.ClickTolerance);
if ~isempty(objects)
    for objc11 = objects
        obj = objc11{1};
        if isa(obj,'Connection')
            if h.IndexC == 1 || ...
                (obj.Orient == h.SelectCon(1).Orient && ...
                 obj.Group == h.SelectCon(1).Group)
                h.SelectCon(h.IndexC) = obj;
            if (h.IndexC == 2)
                % Finalize the new relation
                % Ask the user about the type
                names = {
                    'Constant Offset', ...
                    'Cross-Section Maintaining', ...
                    'Zero x Based Scale', ...
                    'Smallest x Based Scale', ...
                    'Width Set'};
                if obj.Orient == enumOrient.Horizontal
                    names(end+1) = 'Defines Stroke Length';
                    names(end+1) = 'Defines Piston Length';
                end
                for RMan = obj.Group.RelationManagers
                    if RMan.Orient == obj.Orient; break; end
                end
                if ~isempty(RMan)
                    [Type, tf] = listdlg(...
                        'PromptString','What type of relationship?',...
                        'ListString',names,...
                        'SelectionMode','single',...
                        'ListSize',[400 100]);
                    switch names{Type}
                        case 'Constant Offset'
                            EnumType = enumRelation.Constant;
                        case 'Cross-Section Maintaining'
                            EnumType = enumRelation.AreaConstant;
                        case 'Zero x Based Scale'
                            EnumType = enumRelation.Scaled;
                        case 'Smallest x Based Scale'
                            EnumType = enumRelation.LowestScaled;
                        case 'Width Set'
                            EnumType = enumRelation.Width;
                        case 'Defines Stroke Length'
                            EnumType = enumRelation.Stroke;
                        case 'Defines Piston Length'
                            EnumType = enumRelation.Piston;
                    end
                end
                if tf
                    Label = RMan.getLabel(EnumType, ...
                        h.SelectCon(1), h.SelectCon(2));
                    if isempty(Label)
                        Label = getProperName([names{Type} ' Relation']);
                    end
                    if isempty(Label); return; end
                    if EnumType == enumRelation.Stroke || ...
                        EnumType == enumRelation.Piston
                        % Ask which mechanism?
                        objs = h.Model.Converters;
                        mecs = cell(0);
                        for index = length(objs):-1:1
                            mecs{index} = objs(index).name;
                        end
                        index = listdlg(...
                            'ListString',mecs,...
                            'SelectionMode','single');
                    end
                end
            end
        end
    end
end

```

```

        if isempty(index)
            break;
        else
            Mech = objs(index).Frames(1);
        end
    end
    switch EnumType
        case {enumRelation.Constant, ...
            enumRelation.AreaConstant, ...
            enumRelation.Scaled, ...
            enumRelation.LowestScaled, ...
            enumRelation.Width}
            success = RMan.addRelation(...
                Label, ...
                EnumType, ...
                h.SelectCon(1), ...
                h.SelectCon(2));
        case {enumRelation.Stroke, ...
            enumRelation.Piston}
            % Ask which mechanism?
            success = RMan.addRelation(...
                Label, ...
                EnumType, ...
                h.SelectCon(1), ...
                h.SelectCon(2), ...
                Mech);
        otherwise
            msgbox(['Selected relation type' ...
                ' is not implemented']);
            h.IndexC = 1;
            break;
        end
    end
    if ~success
        msgbox(['Relationship was not ' ...
            'added successfully']);
    end
    h.IndexC = 1;
end
end
end
h.IndexC = 1;
end
h.IndexC = 2;
else
    msgbox(['The two connections must have the ' ...
        'same orientation.']);
end
end
end
end
otherwise
end
show_Model(h);
hP = pan(h.output);
hP.ModeHandle.Blocking = false;
hP.Enable = 'off';
updateSelectionList(h);
guidata(hObject,h);
drawnow(); pause(0.05);

function objs = getButtonObjs(handles)
    objs = [...
        handles.InsertBody ...
        handles.InsertGroup ...
        handles.InsertBridge ...
        handles.InsertLeakConnection ...
        handles.InsertSensor ...
        handles.InsertPVoutput ...
        handles.NonConnection ...
        handles.CustomMinorLoss ...
        handles.InsertRelation ...
        handles.SelectObjects ...

```

```

        handles.MultiSelectObjects];

function ButtonCore(hObject,Mode,handles,message)
    inactivated = hObject.UserData(1) == 0;
    handles = clearButtons(handles);
    if inactivated
        handles.MODE = Mode;
        show_Model(handles);
        set(handles.message,'String',message);
        hObject.BackgroundColor = [0.33 0.67 0.33];
        hObject.UserData(1) = 1;
    else

    end
    show_Model(handles);
    updateSelectionList(handles);
    guidata(hObject, handles);
    drawnow(); pause(0.05);

function handles = clearButtons(handles)
    hObjects = getButtonObjs(handles);
    handles.Model.clearHighLighting();
    set(handles.message,'String','---');
    handles.MODE = '';
    for obj = hObjects
        if obj.UserData(1) == 1
            obj.UserData(1) = 0;
            obj.BackgroundColor = [0.94 0.94 0.94];
            break;
        end
    end
    handles.IndexC = 1;
    handles.IndexG = 1;
    handles.IndexB = 1;
    handles.SelectCon = Connection.empty;
    handles.SelectBody = Body.empty;

%% Individual button codes
function InsertBody_CreateFcn(hObject, ~, ~)
    hObject.UserData(1) = 0;
function InsertBody_Callback(hObject, ~, handles)
    ButtonCore(hObject,'InsertBody',handles,{'[left click] To select a connection.','[right click] to
    prescribe a dimension.'});

function InsertGroup_CreateFcn(hObject, ~, ~)
    hObject.UserData(1) = 0;
function InsertGroup_Callback(hObject, ~, handles)
    ButtonCore(hObject,'InsertGroup',handles,'[click] To select a position to place a new group.');
```

```

function InsertBridge_CreateFcn(hObject, ~, ~)
    hObject.UserData(1) = 0;
function InsertBridge_Callback(hObject, ~, handles)
    ButtonCore(hObject,'InsertBridge',handles,'[click] To select the connection associated with the
    foundation body');
```

```

function InsertLeakConnection_CreateFcn(hObject, ~, ~)
    hObject.UserData(1) = 0;
function InsertLeakConnection_Callback(hObject, ~, handles)
    ButtonCore(hObject,'InsertLeakConnection',handles,'[click] To select Connection 1');
```

```

function SelectObjects_CreateFcn(hObject, ~, ~)
    hObject.UserData(1) = 0;
function SelectObjects_Callback(hObject, ~, handles)
    ButtonCore(hObject,'Select',handles,'[click] To select a single object');
```

```

function MultiSelectObjects_CreateFcn(hObject, ~, ~)
    hObject.UserData(1) = 0;
function MultiSelectObjects_Callback(hObject, ~, handles)
    ButtonCore(hObject,'MultiSelect',handles,'[click] To add to select objects');
```

```

function InsertSensor_CreateFcn(hObject, ~, ~)

```

```

hObject.UserData(1) = 0;
function InsertSensor_Callback(hObject,~,handles)
ButtonCore(hObject,'InsertSensor',handles,'[click] To select a body');

function InsertPVoutput_CreateFcn(hObject, ~, ~)
hObject.UserData(1) = 0;
function InsertPVoutput_Callback(hObject, ~, handles)
ButtonCore(hObject,'InsertPVoutput',handles,'[click] To select a body');

function CustomMinorLoss_CreateFcn(hObject, ~, ~)
hObject.UserData(1) = 0;
function CustomMinorLoss_Callback(hObject, ~, handles)
ButtonCore(hObject,'InsertCustomMinorLoss',handles,'[click] To select a body');

function NonConnection_CreateFcn(hObject, ~, ~)
hObject.UserData(1) = 0;
function NonConnection_Callback(hObject, ~, handles)
ButtonCore(hObject,'InsertNonConnection',handles,'[click] To select a body');

function InsertRelation_CreateFcn(hObject, ~, ~)
hObject.UserData(1) = 0;
function InsertRelation_Callback(hObject, ~, handles)
ButtonCore(hObject,'InsertRelation',handles,'[click] To select a connection');

function ChangeGroup_Callback(hObject, ~, handles)
[x,y] = ginput(1);
Pnt = [x y];
backupMessage = get(handles.message,'String');
set(handles.message,'String','[click] Select A group');
handles.Model.switchHighLightedGroup(...
    handles.Model.findNearestGroup(Pnt,handles.ClickTolerance^2) );
set(handles.message,'String',backupMessage);
guidata(hObject,handles);
drawnow(); pause(0.05);

%% Selection Properties
function updateSelectionList(h,index)
switch h.DropDownMode
case ''
    if nargin == 2
        if index > length(h.SData.ListObjs) || index < 1
            fprintf('Index Exceeds Matrix Dimensions: This may be caused by severe lag');
            return;
        else
            if strcmp(h.SData.ListObjs(index).MODE,'Deleteobj')
                % Close all
                Code = '';
            else
                Code = MakeCode(h.SData.ListObjs,index);
            end
        end
    else
        Code = MakeCode(h.SData.ListObjs);
        Code = ResetCode(Code);
    end
    n = 1 + length(h.Model.Selection);
    SelectedObjs(n,1) = ListObj();
    for Obj = [h.Model.Selection {h.Model}]
        SelectedObjs(n) = ListObj('Expandobj',0,[],Obj{1});
        n = n - 1;
    end
    h.SData.ListObjs = ReadCode(Code, SelectedObjs);
    ListString = cell(length(h.SData.ListObjs),1);
    for i = 1:length(h.SData.ListObjs)
        ListString{i} = h.SData.ListObjs(i).getString();
    end
    if nargin < 2; index = get(h.SelectionProps,'Value'); end
    set(h.SelectionProps,'Value',max([1 min([index length(ListString)])]));
    set(h.SelectionProps,'String',ListString);
case 'Optimizer'
    h.DropDownMode = '';

```

```

    if h.OptimizationStudyIndex == 0
        % Create a new study
        h.Model.OptimizationSchemes(end+1) = ...
            OptimizationScheme(h.Model);
        h.OptimizationStudyIndex = ...
            h.Model.OptimizationSchemes(end).ID;
    end
    % This appends the object and field to the optimization study
    if h.OptimizationStudyIndex > 0
        for scheme = h.Model.OptimizationSchemes
            if h.OptimizationStudyIndex == scheme.ID
                break;
            end
        end
        if scheme.ID == h.OptimizationStudyIndex
            if nargin > 1
                obj = h.SData.ListObjs(index).Parent;
                child = h.SData.ListObjs(index).Child;
                if isa(obj,'Connection')
                    scheme.AddObj(obj,'x');
                elseif isa(child,'LinRotMechanism')
                    scheme.AddObj(child,'Stroke');
                elseif isa(child,'Connection')
                    scheme.AddObj(child,'x');
                end
            end
        end
    end
end

end

function SelectionProps_Callback(hObject, ~, h)
% The user has clicked on the SelectionProp's listbox
index = get(hObject,'Value');
if index <= length(h.SData.ListObjs)
    h.SData.ListObjs(index).on_click();
end
updateSelectionList(h,index);

function SelectionProps_CreateFcn(hObject, ~, ~)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%% Optimization
function SwitchDropdownMode_Callback(hObject, ~, h)
% Interfaces with the drop down menu when selecting parameters
if strcmp(h.DropDownMode,'')
    h.DropDownMode = 'Optimizer';
    set(h.DropDownModeUI,'String',h.DropDownMode);
else
    h.DropDownMode = '';
    set(h.DropDownModeUI,'String',h.DropDownMode);
end
guidata(hObject,h);

% --- Executes on button press in SwitchOptimizationStudy.
function SwitchOptimizationStudy_Callback(hObject, ~, h)
% Find the optimization scheme after the current one
% Find the current one
if h.OptimizationStudyIndex == 0
    if ~isempty(h.Model.OptimizationSchemes)
        h.OptimizationStudyIndex = h.Model.OptimizationSchemes(1).ID;
        set(h.OptStudyName,'String',h.Model.OptimizationSchemes(1).name);
    else
        set(h.OptStudyName,'String','Create New Study');
    end
    end
    guidata(hObject,h);
    return;
end
for i = 1:length(h.Model.OptimizationSchemes)
    if h.Model.OptimizationSchemes(i).ID == ...

```

```

        h.OptimizationStudyIndex
        set(h.OptStudyName,'String',h.Model.OptimizationSchemes(i).name);
    end
end
i = i + 1;
if i > length(h.Model.OptimizationSchemes)
    h.OptimizationStudyIndex = 0;
    set(h.OptStudyName,'String','Create New Study');
else
    h.OptimizationStudyIndex = h.Model.OptimizationSchemes(i).ID;
end
guidata(hObject,h);

function RunStudy_Callback(~,~,h)
if h.OptimizationStudyIndex ~= 0
    found = false;
    for i = 1:length(h.Model.OptimizationSchemes)
        if h.Model.OptimizationSchemes(i).ID == h.OptimizationStudyIndex
            found = true;
            break;
        end
    end
    if found
        History = GradientAscent(h.Model,h.OptimizationStudyIndex);
        if ~isempty(History)
            save([h.Model.OptimizationSchemes(i).name ' - History','History']);
        end
    end
end
end

%% Visual Appearance
function DistributeGroup(handles)
% Look at handles.Model.Bridges
% Simultaneously minimize the distance that things move, as well as the
% bridge horizontal distance
handles.Model.distributeGroup(handles.InterGroupDistance);
show_Model(handles);

function GUI_CreateFcn(hObject,~,handles) %#ok<INUSD>
%% Create a figure that has zoom & pan capabilities
set(hObject,'NextPlot','add');
% pan off;
% mouse_figure(gcf);

%% Dynamics
function CreateMechanism_Callback(hObject, eventdata, handles) %#ok<INUSL>
% Open up user form asking for
% ... Type from Source (mechanism type)
% ... ... Stroke (m) (double)
% ... ... Weight (kg) (double)
% ... ... Phases (rad) (double)
% ... ... TiltAngle (rad) (double)
% ... ... MaximumCrankArmAngle (rad) (double)
% ... ... CustomProfile Fcn
Data = Holder({});
[h] = CreateMechanismInterface(Data);
uiwait(h);
handles.Model.addConverter(LinRotMechanism(handles.Model,...
    Data.vars{1},Data.vars{2}));

% --- Executes on button press in Animate.
function Animate_Callback(hObject, ~, handles)
% Temporarily turn off connections, ghosts, groups... etc.
if handles.Model.isAnimating
    hObject.BackgroundColor = [0.94 0.94 0.94];
    handles.Model.isAnimating = false;
    if handles.ViewOptionBackup(1); showConnections_Callback(handles.showConnections, 0,
handles); end
    if handles.ViewOptionBackup(2); showBodyGhosts_Callback(handles.showBodyGhosts, 0, handles);
end
end

```

```

        show_Model(handles);
    else
        hObject.BackgroundColor = [0.33 0.67 0.33];
        handles.ViewOptionBackup = false(2,1);
        handles.ViewOptionBackup(1) = handles.Model.showConnections;
        handles.ViewOptionBackup(2) = handles.Model.showBodyGhosts;
        if handles.ViewOptionBackup(1); showConnections_Callback(handles.showConnections, 0,
handles); end
        if handles.ViewOptionBackup(2); showBodyGhosts_Callback(handles.showBodyGhosts, 0, handles);
end
        handles.Model.isAnimating = true;
        guidata(hObject,handles);
        drawnow(); pause(0.05);
        handles.Model.Animate(); % ANIMATE IT!
        if handles.Model.isAnimating
            hObject.BackgroundColor = [0.94 0.94 0.94];
            handles.Model.isAnimating = false;
            if handles.ViewOptionBackup(1); showConnections_Callback(handles.showConnections, 0,
handles); end
            if handles.ViewOptionBackup(2); showBodyGhosts_Callback(handles.showBodyGhosts, 0,
handles); end
            show_Model(handles);
        end
    end
end

% --- Executes on button press in Delete.
function Delete_Callback(~, ~, handles)
% Delete Selection
if length(handles.Model.Selection) == 1
    if handles.Model.ActiveGroup == handles.Model.Selection{1}
        handles.Model.ActiveGroup(:) = [];
        handles.Model.Selection{1}.deReference();
        handles.Model.Selection = cell(0);
        return;
    end
end
for obj = handles.Model.Selection
    if ~isa(obj{1},'Group')
        obj{1}.deReference();
    end
end
handles.Model.Selection = cell(0);
% For all the selected items

% --- Executes on button press in Revive.
function Revive_Callback(hObject, eventdata, handles) %#ok<INUSD>
% Open up the recycle bin, Full of Bodies and Special Components that
% have handles and dependencies

%% Save Functionality
function save_Callback(~, ~, handles)
saveModel(false,handles);
function saveas_Callback(~,~,handles)
saveModel(true,handles);
function saveModel(savenew,h)
% The Model name is by default used, if the model name is blank, then the
% userform asks for a name.
if isempty(h.Model.name) || savenew

    notdone = true;
    while notdone
        if ~isempty(h.Model.name)
            name = inputdlg('Save as...', 'Save Model', 1, {h.Model.name});
        else
            name = inputdlg('Save as...', 'Save Model');
        end
        if isempty(name); return; else; name = name{1}; end
        if ~isempty(regexp(name, '[/\*:? "<>|]', 'once'))
            fprintf(['XXX Invalid File name, a file name cannot contain ' ...
                'the characters [/*:? "<>|] XXX\n']);
        else

```

```

        if all(ismember(name(1),'0123456789'))
            fprintf(['XXX Invalid File name, a file name cannot start ' ...
                'with a number. XXX\n']);
        else
            notdone = false;
        end
    end
end
if length(name) > 4 && strcmp(name(end-3:end),'.mat')
    name = name(1:end-4);
end
ogname = name;
else
    name = h.Model.name;
    ogname = name;
end
% If the name is already an existing file, it asks to overwrite, if false,
% then asks for a new name, suggesting a variation.
SavedModels = dir('Saved Files');
start = 3;
dupfound = false;
notdone = true;
naming = true;
while naming
    while notdone
        for i = start:length(SavedModels)
            if strcmp(SavedModels(i).name,[name '.mat'])
                % Devise an alternative
                if strcmp(SavedModels(i).name(end-4,','))
                    offset = 1;
                    while ...
                        all(ismember(SavedModels(i).name(end-4-offset),'0123456789')) ||...
                            SavedModels(i).name(end-4-offset) == '.'
                        offset = offset + 1;
                    end
                    offset = offset - 1;
                    num = str2double(SavedModels(i).name(end-4-offset:end-5));
                    num = num + 1;
                    name = [SavedModels(i).name(1:end-5-offset) num2str(num) ''];
                    notdone = true;
                    dupfound = true;
                    break;
                else
                    name = [SavedModels(i).name(1:end-4) ' (1)'];
                    dupfound = true;
                end
            end
        end
    end
    if notdone
        notdone = false;
        % Double check that there are no duplicates
        for i = 3:length(SavedModels)
            if strcmp(SavedModels(i).name,[name '.mat'])
                notdone = true;
                start = i;
                break;
            end
        end
    end
end
end
% We have the new unique name
if dupfound
    switch questdlg(['Do you want to overwrite the existing file: ' ogname])
        case 'Yes'
            name = ogname;
            naming = false;
        case 'No'
            cellname = inputdlg('Name: ', ['Rename: ' ogname], 1, {name});
            if isempty(cellname); return; end
            newname = cellname{1};
            if strcmp(newname,name)

```



```

                naming = false;
            else
                oiname = newname;
                notdone = true;
                dupfound = false;
            end
            name = newname;
        case {'Cancel',''}
            return;
        end
    end
    else
        naming = false;
    end
end
h.Model.name = name;
backupAxis = h.Model.AxisReference;
h.Model.AxisReference(:) = [];
newfile = ['Saved Files\' name '.mat'];
Model = h.Model; %#ok<NASGU>
Model.saveME();
h.Model.AxisReference = backupAxis;
fprintf('Model Saved\n');

%% Load Functionality
function h = load_sub(name, h)
newfile = [pwd '\Saved Files\' name];
File = load(newfile,'Model');
h.Model = File.Model;
h.Model.AxisReference = h.GUI;

h.Model.showInterConnections = false;
h.Model.showNodes = false;
h.Model.RelationOn = true; set(h.RelationMode,'String','On');
h.Model.showGroups = get(h.showGroups,'Value');
h.Model.showBodies = get(h.showBodies,'Value');
h.Model.showBodyGhosts = get(h.showBodyGhosts,'Value');
h.Model.showConnections = get(h.showConnections,'Value');
h.Model.showLeaks = get(h.showLeaks,'Value');
h.Model.showBridges = get(h.showBridges,'Value');
h.Model.showSensors = get(h.showSensors,'Value');
h.Model.showRelations = get(h.showRelations,'Value');
h.Model.showInterConnections = get(h.showInterConnections,'Value');
h.Model.showEnvironmentConnections = get(h.showEnvironmentConnections,'Value');
h.Model.showNodes = get(h.showNodes,'Value');

h.Model.showPressureAnimation = get(h.ShowPressureAnimation,'Value');
h.Model.recordPressure = get(h.RecordPressure,'Value');
h.Model.showTemperatureAnimation = get(h.ShowTemperatureAnimation,'Value');
h.Model.recordTemperature = get(h.RecordTemperature,'Value');
h.Model.showVelocityAnimation = get(h.ShowVelocityAnimation,'Value');
h.Model.recordVelocity = get(h.RecordVelocity,'Value');
h.Model.showTurbulenceAnimation = get(h.ShowTurbulenceAnimation,'Value');
h.Model.recordTurbulence = get(h.RecordTurbulence,'Value');
h.Model.recordOnlyLastCycle = get(h.RecordOnlyLastCycle,'Value');
h.Model.outputPath= get(h.OutputPath,'String');
h.Model.warmUpPhaseLength = str2double(get(h.WarmUpPhaseLength,'String'));
h.Model.animationFrameTime = str2double(get(h.AnimationFrameTime,'String'));

cla;
show_Model(h);
drawnow(); pause(0.05);

function load_Callback(hObject, ~, h)
% Asks the user if they want to save the current model
% if True. Call save_Callback.
switch questdlg('Do you want to save the current model?')
    case 'Yes'
        if ~isempty(h.Model.name)
            switch questdlg('Do you want to save as a new Model?')
                case 'Yes'
                    saveModel(true,h);
            end
        end
    end
end

```

```

        case 'No'
            saveModel(false,h);
        case {'Cancel',''}
            return;
        end
    else
        saveModel(true,h);
    end
    case 'No'
        % Do nothing
    case {'Cancel',''}
        return;
    end

end

% Then provide the user with a list of saved models in the Saved Files
% folder.
SavedModels = dir('Saved Files');
names = {SavedModels.name};
i = 1;
while names{i}(1) == '.'
    i = i + 1;
end
[selection, tf] = listdlg('ListString',names(i:end),...
    'SelectionMode','single');
if tf
    name = names(selection+i-1);
else
    return;
end

% if the user selects one, then replace current model with the loaded one
% and reset the userform.
[h] = load_sub(name, h);
guidata(h.load,h);

%% Show Options
function showGroups_Callback(hObject, ~, h) %#ok<*DEFNU>
value = get(hObject,'Value');
if (value ~= h.Model.showGroups)
    h.Model.showGroups = value;
    show_Model(h);
end
function showBodies_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~=h.Model.showBodies)
    h.Model.showBodies = value;
    show_Model(h);
end
function showConnections_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showConnections)
    h.Model.showConnections = value;
    show_Model(h);
end
function showLeaks_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showLeaks)
    h.Model.showLeaks = value;
    show_Model(h);
end
function showBridges_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showBridges)
    h.Model.showBridges = value;
    show_Model(h);
end
function showInterConnections_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showInterConnections)
    h.Model.showInterConnections = value;
    show_Model(h);
end

```

```

end
function showEnvironmentConnections_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showEnvironmentConnections)
    h.Model.showEnvironmentConnections = value;
    show_Model(h);
end
function showBodyGhosts_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showBodyGhosts)
    h.Model.showBodyGhosts = value;
    show_Model(h);
end
function showNodes_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showNodes)
    h.Model.showNodes = value;
    show_Model(h);
end
function showSensors_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showSensors)
    h.Model.showSensors = value;
    show_Model(h);
end
function showRelations_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.showRelations)
    h.Model.showRelations = value;
    show_Model(h);
end

function BoxZoom_Callback(hObject, eventdata, h) %#ok<INUSL>
%handles.corner_points = ginput(2);
show_Model(h,ginput(2));

function show_Model(h,cornerpoints)
h.Model.show();
if nargin == 2
    % Preserve aspect ratio
    axes = gca;
    width = abs(cornerpoints(1,1) - cornerpoints(2,1));
    height = abs(cornerpoints(1,2) - cornerpoints(2,2));
    r_new = width/height;

    % Get current aspect ratio
    r_old = axes.PlotBoxAspectRatio(1)/axes.PlotBoxAspectRatio(2);

    if r_old > r_new
        width = width*r_old/r_new;
    else
        height = height*r_new/r_old;
    end

    % Determine the center
    c_x = 0.5*(cornerpoints(1,1) + cornerpoints(2,1));
    c_y = 0.5*(cornerpoints(1,2) + cornerpoints(2,2));
    % Adjust the axes
    axes.XLim = [c_x-width/2 c_x+width/2];
    axes.YLim = [c_y-height/2 c_y+height/2];
end
drawnow(); pause(0.05);

function RecenterView_Callback(~, ~, h)
axes = gca;
xlim = h.Model.getXLim();
ylim = h.Model.getYLim();
ar = abs(ylim(1)-ylim(2))/abs(xlim(1)-xlim(2));
cur_xlim = axes.XLim;
cur_ylim = axes.YLim;
cur_ar = abs(cur_ylim(1)-cur_ylim(2))/abs(cur_xlim(1)-cur_xlim(2));

```

```

if ar > cur_ar
    % ylim is the base
    cx = mean(xlim);
    dx = 0.5*abs(ylim(1)-ylim(2))/cur_ar;
    xlim = [cx - dx, cx + dx];
else
    % xlim is the base
    cy = mean(ylim);
    dy = 0.5*cur_ar*abs(xlim(1)-xlim(2));
    ylim = [cy - dy, cy + dy];
end
if any(isnan(xlim)) || any(isinf(xlim)); return; end
if any(isnan(ylim)) || any(isinf(ylim)); return; end
axes.XLim = xlim;
axes.YLim = ylim;
show_Model(h);

%% RunTime Show Options
function showLivePV_Callback(hObject, ~, h)
value = get(hObject, 'Value');
if (value ~= h.Model.showLivePV)
    h.Model.showLivePV = value;
end
function stopSimulation_Callback(~, ~, h)
h.Model.stopSimulation();

function Run_Callback(~, ~, h)
h.Model.Run();

function CreateMechanism_CreateFcn(~, ~, ~)
function Animate_CreateFcn(~, ~, ~)

%% Simulation Options
function Reset_Discretization_Callback(~, ~, h)
h.Model.resetDiscretization();
show_Model(h);
function DispNumbers_Callback(~, ~, h)
h.Model.dispNodeIndexes();
function clearAxes_Callback(~, ~, ~)
cla;

function ShowPressureAnimation_Callback(hObject, ~, h)
value = get(hObject, 'Value');
if (value ~= h.Model.showPressureAnimation)
    h.Model.showPressureAnimation = value;
end
if value
    set(h.RecordPressure, 'Value', value);
    RecordPressure_Callback(h.RecordPressure, [], h);
end
function RecordPressure_Callback(hObject, ~, h)
value = get(hObject, 'Value');
if (value ~= h.Model.recordPressure)
    h.Model.recordPressure = value;
end
if ~value
    set(h.ShowPressureAnimation, 'Value', value);
    ShowPressureAnimation_Callback(h.ShowPressureAnimation, [], h);
end

function ShowTemperatureAnimation_Callback(hObject, ~, h)
value = get(hObject, 'Value');
if (value ~= h.Model.showTemperatureAnimation)
    h.Model.showTemperatureAnimation = value;
end
if value
    set(h.RecordTemperature, 'Value', value);
    RecordTemperature_Callback(h.RecordTemperature, [], h);
end
function RecordTemperature_Callback(hObject, ~, h)
value = get(hObject, 'Value');

```

```

if (value ~= h.Model.recordTemperature)
    h.Model.recordTemperature = value;
end
if ~value
    set(h.ShowTemperatureAnimation,'Value',value);
    ShowTemperatureAnimation_Callback(h.ShowTemperatureAnimation,[],h);
end

function ShowVelocityAnimation_Callback(hObject,~,h)
value = get(hObject,'Value');
if (value ~= h.Model.showVelocityAnimation)
    h.Model.showVelocityAnimation = value;
end
if value
    set(h.RecordVelocity,'Value',value);
    RecordVelocity_Callback(h.RecordVelocity,[],h);
end
function RecordVelocity_Callback(hObject,~,h)
value = get(hObject,'Value');
if (value ~= h.Model.recordVelocity)
    h.Model.recordVelocity = value;
end
if ~value
    set(h.ShowVelocityAnimation,'Value',value);
    ShowVelocityAnimation_Callback(h.ShowVelocityAnimation,[],h);
end

function ShowTurbulenceAnimation_Callback(hObject,~,h)
value = get(hObject,'Value');
if (value ~= h.Model.showTurbulenceAnimation)
    h.Model.showTurbulenceAnimation = value;
end
if value
    set(h.RecordTurbulence,'Value',value);
    RecordTurbulence_Callback(h.RecordTurbulence,[],h);
end
function RecordTurbulence_Callback(hObject,~,h)
value = get(hObject,'Value');
if (value ~= h.Model.recordTurbulence)
    h.Model.recordTurbulence = value;
end
if ~value
    set(h.ShowTurbulenceAnimation,'Value',value);
    ShowTurbulenceAnimation_Callback(h.ShowVelocityAnimation,[],h);
end

function ShowConductionAnimation_Callback(hObject,~,h)
value = get(hObject,'Value');
if (value ~= h.Model.showConductionAnimation)
    h.Model.showTurbulenceAnimation = value;
end
if value
    set(h.RecordConductionFlux,'Value',value);
    RecordConductionFlux_Callback(h.RecordConductionFlux,[],h);
end
function RecordConductionFlux_Callback(hObject,~,h)
value = get(hObject,'Value');
if (value ~= h.Model.recordConductionFlux)
    h.Model.recordTurbulence = value;
end
if ~value
    set(h.ShowConductionAnimation,'Value',value);
    ShowConductionAnimation_Callback(h.ShowConductionAnimation,[],h);
end

function PressureDropAnimation_Callback(hObject,~,h)
value = get(hObject,'Value');
if (value ~= h.Model.showPressureDropAnimation)
    h.Model.showPressureDropAnimation = value;
end
if value

```

```

        set(h.recordPressureDrop,'Value',value);
        recordPressureDrop_Callback(h.recordPressureDrop,[],h);
    end
function recordPressureDrop_Callback(hObject,~,h)
value = get(hObject,'Value');
if (value ~= h.Model.recordPressureDrop)
    h.Model.recordPressureDrop = value;
end
if ~value
    set(h.PressureDropAnimation,'Value',value);
    PressureDropAnimation_Callback(h.PressureDropAnimation,[],h);
end

function RecordOnlyLastCycle_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.recordOnlyLastCycle)
    h.Model.recordOnlyLastCycle = value;
end

function RecordStatistics_Callback(hObject, ~, h)
value = get(hObject,'Value');
if (value ~= h.Model.recordStatistics)
    h.Model.recordStatistics = value;
end

function OutputPath_CreateFcn(~, ~, ~)
function OutputPath_ButtonDownFcn(hObject, ~, h)
value = uigetdir;
set(hObject,'String',value);
h.Model.outputPath = value;

function WarmUpPhaseLength_Callback(hObject, ~, h)
value = get(hObject,'String');
if isempty(value); value = '0'; end
if all(ismember(value,'.0123456789'))
    set(hObject,'UserData',value);
    h.Model.warmUpPhaseLength = str2double(value);
else
    msgbox('The length must be a number, the units are already defined as seconds');
    set(hObject,'String',get(hObject,'UserData'));
end

function WarmUpPhaseLength_CreateFcn(hObject, ~, ~)
set(hObject,'UserData','0');
set(hObject,'String','0');
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function AnimationFrameTime_Callback(hObject, ~, h)
value = get(hObject,'String');
if isempty(value); value = '0.05'; end
if all(ismember(value,'.0123456789'))
    set(hObject,'UserData',value);
    h.Model.animationFrameTime = str2double(value);
else
    msgbox('The length must be a number, the units are already defined as seconds');
    set(hObject,'String',get(hObject,'UserData'));
end

function AnimationFrameTime_CreateFcn(hObject, ~, ~)
set(hObject,'UserData','0.05');
set(hObject,'String','0.05');
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function RecordSnapshot_Callback(~, ~, handles)
if ~isempty(handles.Model.Result)
    name = getProperName('Snapshot');
    handles.Model.Result.getSnapshot(this,handles.Model,name)

```

```

end

function RunTestSet_Callback(~, ~, h)
% Find the Folder "Test_Running"
files = dir('Test_Running');
names = {files.name};
names(1:2) = [];
if ~iscell(names)
    names = {names};
end
for index = size(names,1):-1:1
    names{index} = names{index}(1:end-2);
end
index = listdlg('ListString',names,...
    'SelectionMode','single',...
    'InitialValue',index);
if ~isempty(index)
    if strfind(names{index},'.m')
        func = str2func(names{index}(1:end-2));
    else
        func = str2func(names{index});
    end
    Test_Set = func();

    % Chunk the test set into groups that have the same model
    group_start = 1;
    group_end = 1;
    while group_end <= length(Test_Set)
        Model = Test_Set(group_start).Model;
        while group_end <= length(Test_Set) && ...
            strcmp(Model,Test_Set(group_end).Model)
            group_end = group_end + 1;
        end
        group_end = group_end - 1;
        h = load_sub(Model, h);
        h.Model.Run(Test_Set(group_start:group_end));
        group_start = group_end + 1;
        group_end = group_start;

        % The Model name is the default name used, it overwrites automatically
        % name = h.Model.name;
        % newfile = ['Saved Files\' name '.mat'];
        % Model = h.Model;
        % save(newfile,'Model');
        % fprintf('Model Saved.\n');
    end
end

function DerefinementFactor_Callback(hObject, ~, handles)
value = str2double(get(hObject,'String'));
if isnan(value)
    set(hObject,'String','1');
    return;
end
if value >= 0.01 && value <= 100
    handles.Model.deRefinementFactorInput = value;
else
    if value < 0.01
        set(hObject,'String','0.01');
        handles.Model.deRefinementFactorInput = 0.01;
    else
        set(hObject,'String','100');
        handles.Model.deRefinementFactorInput = 100;
    end
end
handles.Model.resetDiscretization();
guidata(hObject,handles);

function DerefinementFactor_CreateFcn(hObject, ~, ~)
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

```

```
        set(hObject,'BackgroundColor','white');
    end

% --- Executes on button press in SwitchRelationMode.
function SwitchRelationMode_Callback(~, ~, handles)
if strcmp(get(handles.RelationMode,'String'),'On')
    set(handles.RelationMode,'String','Off');
    handles.Model.RelationOn = false;
else
    set(handles.RelationMode,'String','On');
    handles.Model.RelationOn = true;
end
```



## Create Mechanism Interface

This module pops up when the user creates a new linear to rotational mechanism. The interface includes a type selection drop-down. This dropdown then switches what is displayed in the property editor, in the form of an editable table. The table contains a column for each property and multiple rows in the case where multiple mechanisms attached to the same point, such as a 90 degree gamma or beta type engine.

```
function varargout = CreateMechanismInterface(varargin)
% CREATEMECHANISMINTERFACE MATLAB code for CreateMechanismInterface.fig
% CREATEMECHANISMINTERFACE, by itself, creates a new CREATEMECHANISMINTERFACE or raises the
existing
% singleton*.
%
% H = CREATEMECHANISMINTERFACE returns the handle to a new CREATEMECHANISMINTERFACE or the
handle to
% the existing singleton*.
%
% CREATEMECHANISMINTERFACE('CALLBACK',hObject,eventData,handles,...) calls the local
function named CALLBACK in CREATEMECHANISMINTERFACE.M with the given input arguments.
%
% CREATEMECHANISMINTERFACE('Property','Value',...) creates a new CREATEMECHANISMINTERFACE or
raises
% the existing singleton*. Starting from the left, property value pairs are
% applied to the GUI before CreateMechanismInterface_OpeningFcn gets called. An
% unrecognized property name or invalid value makes property application
% stop. All inputs are passed to CreateMechanismInterface_OpeningFcn via varargin.
%
% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
% instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help CreateMechanismInterface

% Last Modified by GUIDE v2.5 13-Dec-2018 14:29:58

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
                  'gui_Singleton', gui_Singleton, ...
                  'gui_OpeningFcn', @CreateMechanismInterface_OpeningFcn, ...
                  'gui_OutputFcn', @CreateMechanismInterface_OutputFcn, ...
                  'gui_LayoutFcn', [] , ...
                  'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before CreateMechanismInterface is made visible.
function CreateMechanismInterface_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
```

```

% varargin    command line arguments to CreateMechanismInterface (see VARARGIN)

% Choose default command line output for CreateMechanismInterface
handles.output = hObject;
switch length(varargin{1}.vars)
case 0 % Create New
    handles.iType = [];
    handles.iData = [];
case 2 % Modify Existing
    % assume it is a "Holder"
    handles.iType = varargin{1}.vars{1};
    handles.iData = varargin{1}.vars{2};
case 1 % ???
    handles.iType = varargin{1}.vars{1};
    handles.iData = [];
end
handles.outData = varargin{1};
handles.DataEstablished = false;

% Setup MechType
if ~isempty(handles.iType)
    % Find the index
    i = FindStringInCell(LinRotMechanism.Source,handles.iType);
    if i ~= 0
        set(handles.MechType,'Value',i);
    else
        % Type not found, erase handles.iData & handles.iType
        fprintf(['XXX Type not found in registry, make sure to include ' ...
            'support for "' handles.iType '" if you want to use it. XXX\n']);
        handles.iType = [];
        handles.iData = [];
    end
end

% Setup Data
if ~isempty(handles.iData)
    % Make sure iType is valid
    i = FindStringInCell(LinRotMechanism.Source,handles.iType);
    if i ~= 0
        set(handles.PropertiesTable,'Data',handles.iData);
    else
        % Type not found, erase handles.iData & handles.iType
        fprintf(['XXX Type not found in registry, make sure to include ' ...
            'support for "' handles.iType '" if you want to use it. XXX\n']);
        handles.iType = [];
        handles.iData = [];
    end
    handles.DataEstablished = true;
else
    set(handles.PropertiesTable,'Visible','off');
    handles.DataEstablished = false;
end

% Other things
handles.MODE = '';

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes CreateMechanismInterface wait for user response (see UIRESUME)
% uiwait(handles.TheWindow);

% --- Outputs from this function are returned to the command line.
function varargout = CreateMechanismInterface_OutputFcn(hObject, eventdata, handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject     handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

```

```

% contents = cellstr(get(handles.MechType,'String'));
% varargout{1} = contents{get(handles.MechType,'Value')};
% varargout{2} = get(handles.PropertiesTable,'Data');

% --- Executes on selection change in MechType.
function MechType_Callback(hObject, eventdata, handles)
contents = cellstr(get(hObject,'String'));
Type = contents{get(hObject,'Value')};
if ~handles.DataEstablished
    [Data, Instructions] = ...
        LinRotMechanism.GetPropertyTableSource(Type);
    handles.DataEstablished = true;
else
    [Data, Instructions] = ...
        LinRotMechanism.GetPropertyTableSource(...
            Type,...
            get(handles.PropertiesTable,'Data'));
end
set(handles.PropertiesTable,'Visible','on');
set(handles.PropertiesTable,'Data',Data);
handles.PropertiesTable.ColumnEditable = true(1,size(Data,2));
handles.PropertiesTable.ColumnFormat = cell(1,size(Data,2));
set(handles.Instructions,'String',Instructions);
EstablishWidths(handles);
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function MechType_CreateFcn(hObject, eventdata, handles)
set(hObject,'String',LinRotMechanism.Source);
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all properties.
function PropertiesTable_CreateFcn(hObject, eventdata, handles)

% --- Executes when entered data in editable cell(s) in PropertiesTable.
function PropertiesTable_CellEditCallback(hObject, eventdata, handles)
% hObject    handle to PropertiesTable (see GCBO)
% eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
%     Indices: row and column indices of the cell(s) edited
%     PreviousData: previous data for the cell(s) edited
%     EditData: string(s) entered by the user
%     NewData: EditData or its converted form set on the Data property. Empty if Data was not
changed
%     Error: error string when failed to convert EditData to appropriate value for Data
% handles    structure with handles and user data (see GUIDATA)
if eventdata.Indices(1) == 1
    Data = get(hObject,'Data');
    Data{eventdata.Indices(1),eventdata.Indices(1)} = eventdata.PreviousData;
    set(hObject,'Data',Data);
    fprintf('XXX You cannot edit column headers, no matter how hard you try. XXX\n');
end

function EstablishWidths(handles)
Source = get(handles.PropertiesTable,'Data');
for col = size(Source,2):-1:1
    Widths{col} = length(Source{1,col})*6;
end
set(handles.PropertiesTable,'ColumnWidth',Widths);

% Sum of widths
totalWidth = 0;
for i = 1:length(Widths)
    totalWidth = totalWidth + Widths{i};
end

PosInst = get(handles.Instructions,'Position');
PosTable = get(handles.PropertiesTable,'Position');
PosFrame = get(handles.PropertiesFrame,'Position');
PosWin = get(handles.TheWindow,'Position');

```

```

% Table Size
PosTable(3) = totalWidth+32;
set(handles.PropertiesTable, 'Position', PosTable);

% Instructions Size
PosInst(3) = min([400 PosTable(3)]);
set(handles.Instructions, 'Position', PosInst);

% Frame Size
PosFrame(3) = PosTable(3) + 2*PosTable(1);
set(handles.PropertiesFrame, 'Position', PosFrame);

% Window Size
PosWin(3) = PosTable(3) + 4*PosTable(1);
set(handles.TheWindow, 'Position', PosWin);
guidata(handles.TheWindow, handles);

function Ok_Callback(~, ~, handles)
Types = get(handles.MechType, 'String');
Type = Types{get(handles.MechType, 'Value')};
Source = get(handles.PropertiesTable, 'Data');
handles.outData.vars = {Type, Source};
close(handles.TheWindow);
% Close it.

% --- Executes when selected cell(s) is changed in PropertiesTable.
function PropertiesTable_CellSelectionCallback(hObject, eventdata, handles)
if ~isempty(eventdata.Indices)
    row = eventdata.Indices(1);
    if row ~= 1
        switch handles.MODE
            case 'delete'
                Data = get(handles.PropertiesTable, 'Data');
                NewData = cell(size(Data)-[1 0]);
                k = 0;
                for i = 1:size(Data,1)
                    if i ~= row
                        for j = 1:size(Data,2)
                            NewData{i-k,j} = Data{i,j};
                        end
                    else
                        k = 1;
                    end
                end
                set(handles.PropertiesTable, 'Data', NewData);
            case 'copy'
                Data = get(handles.PropertiesTable, 'Data');
                NewData = cell(size(Data)+[1 0]);
                for i = 1:size(Data,1)
                    for j = 1:size(Data,2)
                        NewData{i,j} = Data{i,j};
                    end
                end
                for i = 1:size(Data,2)
                    NewData{size(Data,1)+1,i} = Data{row,i};
                end
                set(handles.PropertiesTable, 'Data', NewData);
            otherwise
                end
        end
    end
end

% --- Executes on button press in DeleteOnClick.
function DeleteOnClick_Callback(hObject, eventdata, handles)
if strcmp(handles.MODE, 'delete')
    handles.MODE = '';
    set(hObject, 'BackgroundColor', [0.94 0.94 0.94]);
else
    handles.MODE = 'delete';
    set(hObject, 'BackgroundColor', [0 1 0]);
end

```

```

    set(handles.CopyOnClick,'BackgroundColor',[0.94 0.94 0.94]);
end
guidata(hObject, handles);

% --- Executes on button press in CopyOnClick.
function CopyOnClick_Callback(hObject, eventdata, handles)
if strcmp(handles.MODE,'copy')
    handles.MODE = '';
    set(hObject,'BackgroundColor',[0.94 0.94 0.94]);
else
    handles.MODE = 'copy';
    set(hObject,'BackgroundColor',[0 1 0]);
    set(handles.DeleteOnClick,'BackgroundColor',[0.94 0.94 0.94]);
end
guidata(hObject, handles);

% --- Executes on button press in AddBlankRow.
function AddBlankRow_Callback(hObject, eventdata, handles)
handles.MODE = '';
set(handles.CopyOnClick,'BackgroundColor',[0.94 0.94 0.94]);
set(handles.DeleteOnClick,'BackgroundColor',[0.94 0.94 0.94]);
Data = get(handles.PropertiesTable,'Data');
Data = AddRow(Data,1);
set(handles.PropertiesTable,'Data',Data);
guidata(hObject, handles);

```

## **G.2. Major Elements**

## Body

The body is a class that includes:

A creation function

Functions used to append internal lists of other classes.

Function which is called when it is destroyed: to properly disable its dependents.

A get/set interface, used by the property drop-down on the main GUI.

A set of utility functions which calculate the body orientation, sort connections, detect overlaps and update the bodies derived properties. Including its bounds, its validity, its name, translation reference, moving mode, discretization status, default temperatures, default pressures.

A function to discretize it.

A set of functions that get its color from the material, remove it from the figure and add it to the figure.

```
classdef Body < handle
    % body Summary of this class goes here
    % Detailed explanation goes here

    properties (Constant)
        MaterialUndefinedColor = [1 0.5569 1];
        ActiveColor = [0 1 0];
        NormalColor = [0 0 0];
        InvalidColor = [1 0 0];
    end

    properties (Hidden)
        GUIObjects;
        isStateValid logical = false;
        isStateDiscretized logical = false;
        StateMovingStatus enumMove;
        s_lb_Vert double;
        s_ub_Vert double;
        s_lb_Hor double;
        d_lb_Hor double;
        s_ub_Hor double;
        d_ub_Hor double;
        customTemperature = [];
        customPressure = [];
    end

    properties (Dependent)
        name;
        isValid;
        MovingStatus;
        RefFrame;
        Temperature;
        Pressure;
        isDiscretized;
```

```

end

properties
    customname = '';
    ID;
    nodeIndex int32;
    Group Group;
    Connections Connection;
    PVoutputs PVoutput;
    Sensors Sensor;
    matl Material;
    divides = [1 1]; % [Nx, Ny]
    NuFunc function_handle;
    fFunc function_handle;
    Matrix Matrix;
    Mesher Mesher;
    DiscretizationFunctionRadial;
    DiscretizationFunctionAxial;

    % Boolean Values
    isActive logical = false;
    isChanged logical = true;

    % Discretization
    Nodes Node;
    Faces Face;
end

methods
    %% Constructor
    function this = Body(Group,Connections,matl)
        if nargin == 3
            % Get name from
            this.Group = Group;
            this.Connections = Connections;
            for iCon = this.Connections; iCon.addBody(this); end
            this.isChanged = true;
            this.matl = matl;
            fprintf(['Body created in Group ' Group.name '.\n']);
        end
    end
    function addPVoutput(this,PVoutputToAdd)
        this.PVoutputs = PVoutputToAdd;
    end
    function addSensor(this,SensorToAdd)
        for iS = SensorToAdd
            found = false;
            for i = 1:length(this.Sensors)
                if this.Sensors(i) == iS
                    found = true;
                    break;
                end
            end
            if ~found
                this.Sensors(end+1) = iS;
                this.Group.Model.addSensor(iS);
            end
        end
    end
    function deReference(this)
        % Remove Reference from connections
        for iCon = this.Connections
            for i = length(iCon.Bodies):-1:1
                if iCon.Bodies(i) == this
                    iCon.Bodies(i) = [];
                    iCon.change();
                    break;
                end
            end
        end
    end
    for i = length(this.Connections):-1:1

```



```

        if isempty(this.Connections(i).Bodies)
            this.Connections(i).deReference();
        end
    end
end
% Remove Reference from Group
iGroup = this.Group;
for i = length(iGroup.Bodies):-1:1
    if iGroup.Bodies(i) == this
        iGroup.Bodies(i) = [];
        iGroup.isChanged = true;
        break;
    end
end
% Remove Reference from any Bridges
iModel = iGroup.Model;
for i = length(iModel.Bridges):-1:1
    if iModel.Bridges(i).Body1 == this || iModel.Bridges(i).Body2 == this
        iModel.Bridges(i).deReference();
    end
end
% Remove Reference from any Leaks
for i = length(iModel.LeakConnections):-1:1
    if (isa(iModel.LeakConnections(i).obj1,'Body') ...
        && iModel.LeakConnections(i).obj1 == this) ...
        || (isa(iModel.LeakConnections(i).obj2,'Body') ...
            && iModel.LeakConnections(i).obj2 == this)
        iModel.LeakConnections(i).deReference();
    end
end
% Remove Reference from any Custom Minor Losses
for i = length(iModel.CustomMinorLosses):-1:1
    if iModel.CustomMinorLosses(i).Body1 == this || ...
        iModel.CustomMinorLosses(i).Body2 == this
        iModel.CustomMinorLosses(i) = [];
    end
end
% Remove Reference from any NonConnections
for i = length(iModel.NonConnections):-1:1
    if iModel.NonConnections(i).Body1 == this || ...
        iModel.NonConnections(i).Body2 == this
        iModel.NonConnections(i) = [];
    end
end
% Remove Reference from any PVoutputs
for i = length(iModel.PVoutputs):-1:1
    if iModel.PVoutputs(i) == this.PVoutputs
        iModel.PVoutputs(i).deReference();
    end
end
% Remove Reference from any Sensors
for i = length(iModel.Sensors):-1:1
    for j = 1:length(this.Sensors)
        if iModel.Sensors(i) == this.Sensors(j)
            iModel.Sensors(i).deReference();
            break;
        end
    end
end
this.Nodes(:) = [];
this.Faces(:) = [];
% Remove any visual remenant
this.removeFromFigure(gca);
this.delete();
end

%% get/set
function Item = get(this,PropertyName)
    switch PropertyName
        case 'Name'
            Item = this.name;
        case 'Bottom Connection'

```

```

        miny = inf;
        for iCon = this.Connections
            if iCon.Orient == enumOrient.Horizontal && iCon.x < miny
                miny = iCon.x;
                Item = iCon;
            end
        end
    case 'Top Connection'
        maxy = -inf;
        for iCon = this.Connections
            if iCon.Orient == enumOrient.Horizontal && iCon.x > maxy
                maxy = iCon.x;
                Item = iCon;
            end
        end
    case 'Inner Connection'
        minx = inf;
        for iCon = this.Connections
            if iCon.Orient == enumOrient.Vertical && iCon.x < minx
                minx = iCon.x;
                Item = iCon;
            end
        end
    case 'Outer Connection'
        maxx = -inf;
        for iCon = this.Connections
            if iCon.Orient == enumOrient.Vertical && iCon.x > maxx
                maxx = iCon.x;
                Item = iCon;
            end
        end
    case 'Material'
        Item = this.matl;
    case 'Temperature'
        Item = this.Temperature;
    case 'Pressure'
        Item = this.Pressure;
    case 'Radial Divides'
        Item = this.divides(1);
    case 'Axial Divides'
        Item = this.divides(2);
    case 'RefFrame'
        Item = Frame.empty;
        for iCon = this.Connections
            if iCon.Orient == enumOrient.Horizontal && isempty(iCon.RefFrame)
                if isempty(Item)
                    Item = iCon.RefFrame;
                else
                    if Item ~= iCon.RefFrame
                        Item = Frame.empty;
                        break;
                    end
                end
            end
        end
    case 'Change Matrix'
        if isempty(this.Matrix)
            Item = Matrix(this); %#ok<PROPLC>
            this.Matrix = Item;
        else
            Item = this.Matrix;
        end
    case 'Expand Matrix'
        Item = this.Matrix;
    case 'Radial Discretization Function'
        Item = this.DiscretizationFunctionRadial;
    case 'Axial Discretization Function'
        Item = this.DiscretizationFunctionAxial;
    otherwise
        fprintf(['XXX Body GET Inteface for ' PropertyName ' is not found XXX\n']);
end
end

```

```

end
function set(this,PropertyName,Item)
switch PropertyName
case 'Name'
this.customname = Item;
case 'Radial Divides'
this.divides(1) = Item;
case 'Axial Divides'
this.divides(2) = Item;
case 'Temperature'
if Item ~= this.Group.Model.engineTemperature
this.customTemperature = Item;
else
this.customTemperature = [];
end
case 'Pressure'
if Item ~= this.Group.Model.enginePressure
this.customPressure = Item;
else
this.customPressure = [];
end
case 'Change Matrix'
this.Matrix = Item;
this.Matrix.Body = this;
case 'Radial Discretization Function'
this.DiscretizationFunctionRadial = Item;
case 'Axial Discretization Function'
this.DiscretizationFunctionAxial = Item;
case 'RefFrame'
if isempty(Item)
for iCon = this.Connections
if iCon.Orient == enumOrient.Horizontal
iCon.set('RefFrame',Item);
end
end
else
for iCon = this.Connections
if iCon.Orient == enumOrient.Horizontal
if isempty(iCon.RefFrame) || iCon.RefFrame ~= Item
iCon.set('RefFrame',Item);
end
end
end
end
otherwise
fprintf(['XXX Body SET Inteface for ' PropertyName ' is not found XXX\n']);
return;
end
this.change();
end

%% Utility
function sortConnections(this)
% Sort the connections in an order that is xmin,xmax,ymin,ymax
for i = 1:length(this.Connections)-1
for j = i+1:length(this.Connections)
if this.Connections(i).Orient == this.Connections(j).Orient
if this.Connections(i).x > this.Connections(j).x
% swap the two
tempCon = this.Connections(i);
this.Connections(i) = this.Connections(j);
this.Connections(j) = tempCon;
end
elseif this.Connections(i).Orient == enumOrient.Horizontal
% swap the two
tempCon = this.Connections(i);
this.Connections(i) = this.Connections(j);
this.Connections(j) = tempCon;
end
end
end
end
end
end

```

```

end
function dir = getBodyDirection(this)
    if this.divides(1) > this.divides(2)
        dir = 1;
    elseif this.divides(2) > this.divides(1)
        dir = 2;
    else
        cons = zeros(1,2);
        for iCon = this.Connections
            switch iCon.Orient
                case enumOrient.Vertical
                    [b1,b2,~,~] = this.limits(enumOrient.Horizontal);
                    for iBody = iCon.Bodies
                        [y1,y2,~,~] = iBody.limits(enumOrient.Horizontal);
                        if ~(all(y1 > b2) || all(y2 < b1))
                            cons(1) = cons(1) + 1;
                        end
                    end
                case enumOrient.Horizontal
                    [b1,b2,~,~] = this.limits(enumOrient.Vertical);
                    for iBody = iCon.Bodies
                        [x1,x2,~,~] = iBody.limits(enumOrient.Vertical);
                        if ~(all(x1 > b2) || all(x2 < b1))
                            cons(2) = cons(2) + 1;
                        end
                    end
            end
        end
        if cons(1) == 0
            if cons(2) == 0
                dir = 2;
            else
                dir = 2;
            end
        else
            if cons(2) == 0
                dir = 1;
            else
                if cons(1) > cons(2)
                    dir = 1;
                else
                    dir = 2;
                end
            end
        end
    end
end

end

%% Creation Tests
function isit = overlaps(thisBody,otherBody)
    isit = false;
    if thisBody ~= otherBody
        % Test x-coords
        [~,~,xmin1,xmax1] = thisBody.limits(enumOrient.Vertical);
        [~,~,xmin2,xmax2] = otherBody.limits(enumOrient.Vertical);
        if xmin1 >= xmax2 || xmin2 >= xmax1
            isit = false;
            return;
        end
        [ymin1,ymax1,~,~] = thisBody.limits(enumOrient.Horizontal);
        [ymin2,ymax2,~,~] = otherBody.limits(enumOrient.Horizontal);
        N = max([1 length(ymin1) length(ymax1)]);
        if N ~= 1
            if (~isscalar(ymin2) && N ~= length(ymin2)) || ...
                (~isscalar(ymax2) && N ~= length(ymax2))
                otherBody.update();
                [ymin2,ymax2,~,~] = otherBody.limits(enumOrient.Horizontal);
            end
        end
        if all(ymin1 >= ymax2) || all(ymin2 >= ymax1)
            isit = false;
        end
    end
end

```

```

        return;
    end
    isit = true;
end
end
function [doesit, orient, xmin, xmax, y] = touches(thisBody,otherBody)
    if thisBody.Connections(1) == otherBody.Connections(2) || ...
        thisBody.Connections(2) == otherBody.Connections(1)
        % Vertical Connections
        orient = thisBody.Connections(1).Orient;
        [~,~,xmin1,xmax1] = thisBody.limits(enumOrient.Vertical);
        [~,~,xmin2,xmax2] = otherBody.limits(enumOrient.Vertical);
        xmin = xmin1; xmin(xmin<xmin2) = xmin2(xmin<xmin2);
        xmax = xmax1; xmax(xmax>xmax2) = xmax2(xmax>xmax2);
        if thisBody.Connections(1) == otherBody.Connections(2)
            y = thisBody.Connections(1).x;
        else
            y = thisBody.Connections(2).x;
        end
        doesit = any(xmin<xmax);
    elseif thisBody.Connections(3) == otherBody.Connections(4) || ...
        thisBody.Connections(4) == otherBody.Connections(3)
        % Horizontal Connections
        orient = thisBody.Connections(3).Orient;
        [ymin1,ymax1,~,~] = thisBody.limits(enumOrient.Horizontal);
        [ymin2,ymax2,~,~] = otherBody.limits(enumOrient.Horizontal);
        xmin = ymin1;
        if isscalar(ymin2)
            xmin(xmin<ymin2) = ymin2;
        else
            if isscalar(ymin1)
                xmin = ymin1*ones(size(ymin2));
            end
            xmin(xmin<ymin2) = ymin2(xmin<ymin2);
        end
        xmax = ymax1;
        if isscalar(ymax2)
            xmax(xmax>ymax2) = ymax2;
        else
            if isscalar(ymin1)
                xmax = ymax1*ones(size(ymax2));
            end
            xmax(xmax>ymax2) = ymax2(xmax>ymax2);
        end
        if thisBody.Connections(3) == otherBody.Connections(4)
            y = thisBody.Connections(3);
        else
            y = thisBody.Connections(4);
        end
        doesit = any(xmin<xmax);
    else
        doesit = false;
        orient = enumOrient.Vertical;
        xmin = inf;
        xmax = inf;
        y = inf;
    end
end
end

%% Update on Demand
function update(this)
    if isempty(this.ID); this.ID = this.Group.Model.getBodyID(); end
    if isempty(this.Connections)
        this.isChanged = false;
        return;
    end
    if any(~isvalid(this.Sensors))
        this.Sensors = this.Sensors(isvalid(this.Sensors));
    end
    if any(~isvalid(this.PVoutputs))
        this.PVoutputs = this.PVoutputs(isvalid(this.PVoutputs));
    end
end

```

```

end

if ~isempty(this.Matrix)
    if isempty(this.Matrix.matl) || isempty(this.Matrix.Dh)
        delete(this.Matrix);
        this.Matrix(:) = [];
    elseif ~(this.Matrix.Body == this)
        this.Matrix.Body = this;
    end
end

end

this.isChanged = false;

% Update Connections
for iCon = this.Connections
    found = false;
    iCon.CleanUpBodies;
    for iBody = iCon.Bodies
        if iBody == this; found = true; end
    end
    if ~found; iCon.addBody(this); end
end

this.sortConnections();
%% Update Limits
% Find vertical connections
nv = 2; nh = 2;
arrConV(2) = Connection();
arrConH(2) = Connection();
for iCon = this.Connections
    if iCon.Orient == enumOrient.Vertical
        arrConV(nv) = iCon; nv = 1;
    else
        arrConH(nh) = iCon; nh = 1;
    end
end
if arrConV(1).x > arrConV(2).x
    this.s_lb_Vert = arrConV(2).x;
    this.s_ub_Vert = arrConV(1).x;
else
    this.s_lb_Vert = arrConV(1).x;
    this.s_ub_Vert = arrConV(2).x;
end
if arrConH(1).x > arrConH(2).x
    this.s_lb_Hor = arrConH(2).x;
    if arrConH(2).get('isStationary')
        this.d_lb_Hor = this.s_lb_Hor;
    else
        this.d_lb_Hor = this.s_lb_Hor + arrConH(2).RefFrame.Positions;
    end
    this.s_ub_Hor = arrConH(1).x;
    if arrConH(1).get('isStationary')
        this.d_ub_Hor = this.s_ub_Hor;
    else
        this.d_ub_Hor = this.s_ub_Hor + arrConH(1).RefFrame.Positions;
    end
else
    this.s_lb_Hor = arrConH(1).x;
    if arrConH(1).isStationary
        this.d_lb_Hor = this.s_lb_Hor;
    else
        this.d_lb_Hor = this.s_lb_Hor + arrConH(1).RefFrame.Positions;
    end
    this.s_ub_Hor = arrConH(2).x;
    if arrConH(2).isStationary
        this.d_ub_Hor = this.s_ub_Hor;
    else
        this.d_ub_Hor = this.s_ub_Hor + arrConH(2).RefFrame.Positions;
    end
end
end
end

```

```

%% Update MovingStatus
found = false;
frame = [];
varenum = enumMove.Stretching;
for i = 1:length(this.Connections)
    if ~this.Connections(i).get('isStationary')
        frame = this.Connections(i).RefFrame;
        found = true;
        break;
    end
end
if ~found
    varenum = enumMove.Static;
else
    found = false;
    for j = i+1:length(this.Connections)
        if ~this.Connections(j).get('isStationary') ...
            && this.Connections(i).RefFrame == frame
            varenum = enumMove.Moving;
            found = true;
        end
    end
    if ~found
        varenum = enumMove.Stretching;
    end
end
this.StateMovingStatus = varenum;

%% Update isValid
varb = true;
isSolid = (this.matl.Phase == enumMaterial.Solid);
% Gas bodies do not support multiple dimensions
if isSolid
    % SOLIDS MUST HAVE FINITE VOLUME
    [~,~,dim1, dim2] = limits(this,enumOrient.Vertical);
    if dim1 == dim2
        fprintf(...
            ['Solid volumes must have finite volumes, please ' ...
            'define a x-dimension for ' ...
            this.name '.\n']);
        varb = false;
    end
    [~,~,dim1, dim2] = limits(this,enumOrient.Horizontal);
    if dim1 == dim2
        fprintf(...
            ['Solid volumes must have finite volumes, please ' ...
            'define a y-dimension for ' ...
            this.name '.\n']);
        varb = false;
    end
    % SOLIDS CANNOT STRETCH
    if this.MovingStatus == enumMove.Stretching
        fprintf(...
            ['Solid volumes cannot be stretched, please define ' ...
            'the same frame to both lateral surfaces of ' ...
            this.name '.\n']);
        varb = false;
    end
else
    % GASES CANNOT HAVE MULTIPLE DIMENSIONS
    if min(this.divides) ~= 1
        fprintf(...
            ['Gas volumes are restricted to single dimensional discretization,' ...
            'please review ' this.name "'s definition.\n']);
        varb = false;
    end
end

% Check with interference from other bodies
if this.Group.isOverlapping(this)

```

```

        varb = false;
    end
    fprintf(['Update Body: ' this.name '\n']);
    this.isStateValid = varb;
end
function resetDiscretization(this)
    for iCon = this.Connections
        iCon.resetDiscretization();
    end
    this.Nodes(:) = [];
    this.Faces(:) = [];
    this.isStateDiscretized = false;
end
function change(this)
    this.isChanged = true;
    this.resetDiscretization();
    this.Group.change();
end
function name = get.name(this)
    if isempty(this.customname)
        [~,~,x1,x2] = this.limits(enumOrient.Vertical);
        [~,~,y1,y2] = this.limits(enumOrient.Horizontal);
        name = [this.matl.name ' Body ' ...
            '(' num2str(x1) ', ' num2str(x2) ') ' ...
            '(' num2str(y1) ', ' num2str(y2) ') vol:' ...
            num2str(pi*(x2^2-x1^2)*(y2(1)-y1(1))) ];
    else
        name = this.customname;
    end
end
function [d_lb, d_ub, s_lb, s_ub] = limits(this, Orient)
    if this.isChanged; this.update(); end
    switch Orient
        case enumOrient.Vertical
            d_lb = 0;
            d_ub = 0;
            s_lb = this.s_lb_Vert;
            s_ub = this.s_ub_Vert;
        case enumOrient.Horizontal
            d_lb = this.d_lb_Hor;
            d_ub = this.d_ub_Hor;
            s_lb = this.s_lb_Hor;
            s_ub = this.s_ub_Hor;
    end
end
function isValid = get.isValid(this)
    this.update();
    % if this.isChanged; this.update(); end
    isValid = this.isStateValid;
end
function frame = get.RefFrame(this)
    if this.isChanged; this.update(); end
    frame = [];
    if this.MovingStatus == enumMove.Moving
        for iCon = this.Connections
            if iCon.Orient == enumOrient.Horizontal && ~iCon.get('isStationary')
                frame = iCon.RefFrame;
            end
        end
    end
end
function MovingStatus = get.MovingStatus(this)
    if this.isChanged; this.update(); end
    MovingStatus = this.StateMovingStatus;
end
function Discretized = get.isDiscretized(this)
    if this.isChanged; this.update(); end
    if isempty(this.Nodes)
        this.isStateDiscretized = false;
    end
    Discretized = this.isStateDiscretized;
end

```



```

end

%% Property Parameters
function Temp = get.Temperature(this)
    if isempty(this.customTemperature)
        Temp = this.Group.Model.engineTemperature;
    else
        Temp = this.customTemperature;
    end
end

function Press = get.Pressure(this)
    if isempty(this.customPressure)
        Press = this.Group.Model.enginePressure;
    else
        Press = this.customPressure;
    end
end

%% Node Generation
function discretize(this)
    this.update();
    if this.isDiscretized || ~this.isValid
        return;
    end
    isSolid = (this.matl.Phase == enumMaterial.Solid);
    if isSolid; FType = enumFType.Solid; else; FType = enumFType.Gas; end
    if this.isChanged
        this.update();
    end
    %% DETERMINE THE NODE TYPE
    if isSolid
        NType = enumNType.SN; % SN - Solid Node
    else
        % SVGN - Static Volume Gas Node
        % VVGN - Variable Volume Gas Node
        % SAGS - Shearing Annular Gas Node
        switch this.MovingStatus
            case enumMove.Static
                % Decide, is it shearing or just moving?
                % Looking at the two vertical connections
                for iCon = this.Connections
                    NType = enumNType.SVGN;
                    if iCon.Orient == enumOrient.Vertical
                        % Find a body that shares that
                        % connection and scope of x
                        for iBody = this.Group.Bodies
                            if iBody ~= this
                                if ~isempty(iBody.RefFrame)
                                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                                    NType = enumNType.SAGN;
                                    %frame = iBody.RefFrame;
                                    break;
                                end
                            end
                        end
                    end
                end
            case enumMove.Moving
                % Decide, is it shearing or just moving?
                % Looking at the two vertical connections
                for iCon = this.Connections
                    NType = enumNType.SVGN;
                    if iCon.Orient == enumOrient.Vertical
                        % Find a body that shares that
                        % connection and scope of x
                        for iBody = this.Group.Bodies
                            if iBody ~= this
                                if isempty(iBody.RefFrame)
                                    NType = enumNType.SAGN;
                                    frame = this.RefFrame;
                                    break;
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end

```

```

                                end
                            end
                        end
                    end
                end
            case enumMove.Stretching
                NType = enumNType.VVGN;
            end
        end
    end

%% Y LIMITS
[ymin,ymax,~,~] = this.limits(enumOrient.Horizontal);
if ~prod(ymax>=ymin) % Will give false if this is not true everywhere
    changed_registered = false;
    for iCon = this.Group.Connections
        if iCon.Orient == this.Connections(3).Orient && ...
            iCon.x == this.Connections(3).x
            if length(iCon.RefFrame) ~= length(this.Connections(3).RefFrame)
                this.Connections(3) = iCon.x;
                this.update();
                changed_registered = true;
            end
        elseif iCon.Orient == this.Connections(4).Orient && ...
            iCon.x == this.Connections(4).x
            if length(iCon.RefFrame) > length(this.Connections(4).RefFrame)
                this.Connections(4).RefFrame = iCon.RefFrame;
                this.update();
                changed_registered = true;
            end
        end
    end
end
if changed_registered
    fprintf(...
        ['XXX A memory error occured for Body ' this.name ...
        ' in which a connection reference was duplicated,' ...
        ' this has been mitigated but will require a restart of' ...
        ' the discretization. XXX\n']);
    return;
else
    fprintf(...
        ['XXX Calculated maximum and minimum positions ' this.name ...
        ' for will result in a case of negative area, consider' ...
        ' readjusting gas volume or start positions to mitigate' ...
        ' this overlap. XXX\n']);
    return;
end
end
%% X LIMITS
[~,~,xmin,xmax] = this.limits(enumOrient.Vertical);
if isempty(this.DiscretizationFunctionRadial)
    x = transpose(linspace(xmin,xmax,this.divides(1)+1));
else
    if isSolid
        [x] =
this.DiscretizationFunctionRadial(this,this.Group.Model.Mesher,enumOrient.Vertical);
        if isempty(x); return; end
        deltas = diff(x);
        if ~(all(sign(deltas) > 0) || all(sign(deltas) < 0))
            fprintf('XXX x generation issue in Body\m');
            [x] =
this.DiscretizationFunctionRadial(this,this.Group.Model.Mesher,enumOrient.Vertical);
        end
        if x(end,1) < x(1,1); x = flip(x,1); end
    else
        if isempty(this.Matrix)
            fprintf(...
                ['XXX Smart Discretization functions currently cannot' ...
                ' be used for matrixless gas nodes. Problem found in radial direction
of Body:' ...
                this.name ' . XXX\n']);
            return;
        end
    end
end

```

```

else
    if this.divides(1) > 1
        [x] =
this.DiscretizationFunctionRadial(this,this.Group.Model.Mesh,enumOrient.Vertical);
        if isempty(x); return; end
        deltas = diff(x);
        if ~(all(sign(deltas) > 0) || all(sign(deltas) < 0))
            fprintf('XXX x generation issue in Body\m');
            [x] =
this.DiscretizationFunctionRadial(this,this.Group.Model.Mesher,enumOrient.Vertical);
        end
        if x(end,1) < x(1,1); x = flip(x,1); end
    else
        x = [xmin; xmax];
    end
end
end
end

%% Y LIMITS
LEN = this.divides(2)+1;
if isempty(this.DiscretizationFunctionAxial)
    if isscalar(ymin)
        if isscalar(ymax)
            % SCALAR-SCALAR CASE
            y = transpose(linspace(ymin,ymax,LEN));
        else % only ymin is scalar - stretching
            y = zeros(LEN,Frame.NTheta);
            for i = 1:length(ymin)
                y(:,i) = transpose(linspace(ymin,ymax(i),LEN));
            end
        end
    elseif isscalar(ymax) % only ymax is scalar - stretching
        y = zeros(this.divides(2)+1,Frame.NTheta);
        for i = 1:length(ymin)
            y(:,i) = transpose(linspace(ymin(i),ymax,LEN));
        end
    else % both are stretching or moving
        y = zeros(this.divides(2)+1,Frame.NTheta);
        for i = 1:length(ymin)
            y(:,i) = transpose(linspace(ymin(i),ymax(i),LEN));
        end
    end
else
    if isSolid
        [y] =
this.DiscretizationFunctionAxial(this,this.Group.Model.Mesher,enumOrient.Horizontal);
        if isempty(y); return; end
        deltas = diff(y);
        try
            if ~(all(all(sign(deltas) > 0)) || all(all(sign(deltas) < 0)))
                fprintf('XXX y generation issue in Body\m');
            end
        catch
            fprintf('err');
        end
        if y(end,1) < y(1,1); y = flip(y,1); end
    else
        if isempty(this.Matrix)
            fprintf(...
                ['XXX Smart Discretization functions currently cannot' ...
                ' be used for matrixless gas nodes. Problem found in axial direction
of Body:' ...
                this.name '. XXX\n']);
            return;
        else
            if this.divides(2) > 1

```

```

        [y] =
this.DiscretizationFunctionAxial(this,this.Group.Model.Mesher,enumOrient.Horizontal);
        if isempty(y); return; end
        deltas = diff(y);
        try
            if ~(all(all(sign(deltas) > 0)) || all(all(sign(deltas) < 0)))
                fprintf('XXX y generation issue in Body\m');
            end
        catch
            fprintf('err');
        end
        if y(end,1) < y(1,1); y = flip(y,1); end
    else
        if isscalar(ymin)
            if isscalar(ymax)
                % SCALAR-SCALAR CASE
                y = transpose(linspace(ymin,ymax,LEN));
            else % only ymin is scalar - stretching
                y = zeros(LEN,Frame.NTheta);
                for i = 1:length(ymin)
                    y(:,i) = transpose(linspace(ymin,ymax(i),LEN));
                end
            end
        elseif isscalar(ymin) % only ymax is scalar - stretching
            y = zeros(LEN,Frame.NTheta);
            for i = 1:length(ymin)
                y(:,i) = transpose(linspace(ymin(i),ymax,LEN));
            end
        else % both are stretching or moving
            y = zeros(LEN,Frame.NTheta);
            for i = 1:length(ymin)
                y(:,i) = transpose(linspace(ymin(i),ymax(i),LEN));
            end
        end
    end
end
end
end

if strcmp(this.matl.name,'Perfect Insulator') || ...
    strcmp(this.matl.name,'Constant Temperature')
    x = [x(1,:); x(end,:)];
    y = [y(1,:); y(end,:)];
end
divx = size(x,1) - 1;
divy = size(y,1) - 1;

this.Nodes = Node.empty;
this.Faces = Face.empty;

%% INITIALIZE
sendtoConnections{4} = NodeContact.empty;
ncount = divx*divy;
fcount = (divx-1)*divy + divx*(divy-1);
%fcount = prod([divx divy]-[1 0])+prod(this.divides-[0 1]);

%% FOR EACH DISTINCT NODE WITHIN BODY
for iy = size(y,1) - 1:-1:1
    % loop initialization
    starty = y(iy,:);
    endy = y(iy+1,:);
    starty = CollapseVector(starty);
    endy = CollapseVector(endy);

    for ix = size(x,1) - 1:-1:1
        %% Define this.Nodes
        CurrentNode =
Node(NType,x(ix),x(ix+1),starty,endy,Face.empty,Node.empty,this,0);
        this.Nodes(ncount) = CurrentNode;
    end
end
end

```

```

        ncount = ncount - 1;
    end
end

for i = 1:length(this.Nodes)
    nd = this.Nodes(i);
    if nd.xmin == xmin
        sendtoConnections{1}(end+1) = ...
            NodeContact(nd,nd.ymin,nd.ymax,FType,this.Connections(1));
    end
    if nd.xmax == xmax
        sendtoConnections{2}(end+1) = ...
            NodeContact(nd,nd.ymin,nd.ymax,FType,this.Connections(2));
    else
        % Make Vertical connection
        this.Faces(fcount) = ...
            Face([this.Nodes(i+1) nd],FType,enumOrient.Vertical);
        fcount = fcount - 1;
    end
    if nd.ymin(1) == ymin(1)
        sendtoConnections{3}(end+1) = ...
            NodeContact(nd,nd.xmin,nd.xmax,FType,this.Connections(3));
    end
    if nd.ymax(1) == ymax(1)
        sendtoConnections{4}(end+1) = ...
            NodeContact(nd,nd.xmin,nd.xmax,FType,this.Connections(4));
    else
        % Make Horizontal connection
        this.Faces(fcount) = ...
            Face([this.Nodes(i+divx) nd],FType,enumOrient.Horizontal);
        fcount = fcount - 1;
    end
end

end

%% SEND THE COMPILED LIST TO CONNECTIONS FOR PROCESSING
for i = 1:length(this.Connections)
    this.Connections(i).addNodeContacts(sendtoConnections{i});
end

if ~isempty(this.Matrix) && ~isempty(this.Matrix.Geometry)
    % Pass Nodes to Matrix for generation
    [nodes, faces] = this.Matrix.discretize(this.Nodes);
    this.Nodes = [this.Nodes nodes];
    this.Faces = [this.Faces faces];
end

end

this.isStateDiscretized = true;
% fprintf(['Body ' this.name ' is discretized, but this.Nodes still need to reference
their this.Faces.\n']);
end

%% GRAPHICS FUNCTIONS
function color = getColor(this)
    if this.isActive
        color = Body.ActiveColor;
    else
        color = Body.NormalColor;
    end
end

function updateColor(this)
    if ~isempty(this.GUIObjects)
        for iGraphicsObject = this.GUIObjects
            set(iGraphicsObject,'FaceColor',this.getColor());
        end
    end
end

function removeFromFigure(this,AxisReference)
    if ~isempty(this.GUIObjects)
        children = get(AxisReference,'Children');
        for obj = this.GUIObjects

```

```

        if isgraphics(obj)
            for i = length(children):-1:1
                if isgraphics(children(i)) && children(i) == obj
                    children(i).delete;
                    break;
                end
            end
        end
        end
        end
        end
        this.GUIObjects = [];
    end
end
function show(this,AxisReference,Inc)
    if this.isChanged; this.update(); end
    % fprintf(['Plotted Body ' this.name '.\n']);
    % Remove object from plot
    % this.removeFromFigure(AxisReference);

    if this.isValid
        if ~isempty(this.matl) && ~isempty(this.matl.Color)
            fillcolor = this.matl.Color;
        else
            fillcolor = Body.MaterialUndefinedColor;
        end
    else
        fillcolor = Body.InvalidColor;
    end
    edgecolor = this.getColor();
    % Find the extents of the body and position the rectangle(s)
    % accordingly

    %% Case 1: If it has 6 connections it is a cuboid
    if length(this.Connections) == 6
        % Render as cuboid

        return;
    end

    %% Case 2: It is a cylinder
    % If one connection is vertical and x = 0
    for iConnection = this.Connections
        if iConnection.Orient == enumOrient.Vertical && iConnection.x == 0
            % Treat it as a cylinder
            [~,~,maxx] = this.limits(enumOrient.Vertical);
            if nargin > 2 % Inc Exists
                [miny, maxy,~,~] = this.limits(enumOrient.Horizontal);
                if length(miny) > 1; miny = miny(Inc); end
                if length(maxy) > 1; maxy = maxy(Inc); end
            else
                % plot a motion ghost
                if this.Group.Model.showBodyGhosts && this.MovingStatus ==
enumMove.Moving

                    [y1,y2,miny,maxy] = this.limits(enumOrient.Horizontal);
                    gminy = max(y1);
                    gmaxy = max(y2);
                    OffsetRot = this.Group.Position.Rot;
                    R = RotMatrix(OffsetRot);
                    RootPosition = [this.Group.Position.x; this.Group.Position.y];
                    p = [R*[gminy;maxx]+RootPosition ...
                        R*[gmaxy;maxx]+RootPosition ...
                        R*[gmaxy;-maxx]+RootPosition ...
                        R*[gminy;-maxx]+RootPosition];

                    this.Group.Model.GhostGUIObjects(end+1) = fill(p(1,:),p(2,:),...
                        fillcolor,...
                        'EdgeColor',edgecolor,...
                        'LineWidth',1,...
                        'HitTest','off',...
                        'FaceAlpha',0.25,...
                        'EdgeAlpha',0.75);
                else

```

```

        [~,~,miny,maxy] = this.limits(enumOrient.Horizontal);
    end
end

OffsetRot = this.Group.Position.Rot;
R = RotMatrix(OffsetRot);
RootPosition = [this.Group.Position.x; this.Group.Position.y];
p = [R*[miny;maxx]+RootPosition ...
     R*[maxy;maxx]+RootPosition ...
     R*[maxy;-maxx]+RootPosition ...
     R*[miny;-maxx]+RootPosition];

this.removeFromFigure(AxisReference)
this.GUIObjects = fill(p(1,:),p(2,:),...
    fillcolor,...'FaceColor',fillcolor,...
    'EdgeColor',edgecolor,...
    'LineWidth',1,...
    'HitTest','off');
return;
end
end

%% Case 3: It is an annulus
% Get extents of body
[~,~,minx, maxx] = this.limits(enumOrient.Vertical);
if nargin > 2 % Inc exists
    [miny, maxy,~,~] = this.limits(enumOrient.Horizontal);
    if length(miny) > 1; miny = miny(Inc); end
    if length(maxy) > 1; maxy = maxy(Inc); end
else
    % plot a motion ghost
    if this.Group.Model.showBodyGhosts && this.MovingStatus == enumMove.Moving
        [y1,y2,miny,maxy] = this.limits(enumOrient.Horizontal);
        gminy = max(y1);
        gmaxy = max(y2);
        OffsetRot = this.Group.Position.Rot;
        R = RotMatrix(OffsetRot);
        RootPosition = [this.Group.Position.x; this.Group.Position.y];
        p = [R*[gminy;maxx]+RootPosition ...
             R*[gmaxy;maxx]+RootPosition ...
             R*[gmaxy;minx]+RootPosition ...
             R*[gminy;minx]+RootPosition];

        this.Group.Model.GhostGUIObjects(end+1) = fill(p(1,:),p(2,:),...
            fillcolor,...
            'EdgeColor',edgecolor,...
            'LineWidth',1,...
            'HitTest','off',...
            'FaceAlpha',0.25,...
            'EdgeAlpha',0.75);

        p = [R*[gminy;-minx]+RootPosition ...
             R*[gminy;-maxx]+RootPosition ...
             R*[gmaxy;-maxx]+RootPosition ...
             R*[gmaxy;-minx]+RootPosition];

        this.Group.Model.GhostGUIObjects(end+1) = fill(p(1,:),p(2,:),...
            fillcolor,...'FaceColor',fillcolor,...
            'EdgeColor',edgecolor,...
            'LineWidth',1,...
            'HitTest','off',...
            'FaceAlpha',0.25,...
            'EdgeAlpha',0.75);
    else
        [~,~,miny,maxy] = this.limits(enumOrient.Horizontal);
    end
end
end

OffsetRot = this.Group.Position.Rot;
R = RotMatrix(OffsetRot);
RootPosition = [this.Group.Position.x; this.Group.Position.y];

```

```

p1 = [R*[miny;maxx]+RootPosition ...
      R*[maxy;maxx]+RootPosition ...
      R*[maxy;minx]+RootPosition ...
      R*[miny;minx]+RootPosition];
p2 = [R*[miny;-minx]+RootPosition ...
      R*[miny;-maxx]+RootPosition ...
      R*[maxy;-maxx]+RootPosition ...
      R*[maxy;-minx]+RootPosition];

if length(this.GUIObjects) == 2 && ...
    isgraphics(this.GUIObjects(1)) && ...
    isgraphics(this.GUIObjects(2))
    set(this.GUIObjects(1), 'XData', p1(1,:));
    set(this.GUIObjects(1), 'YData', p1(2,:));
    set(this.GUIObjects(1), 'FaceColor', fillcolor);
    set(this.GUIObjects(1), 'EdgeColor', edgecolor);
    set(this.GUIObjects(2), 'XData', p2(1,:));
    set(this.GUIObjects(2), 'YData', p2(2,:));
    set(this.GUIObjects(2), 'FaceColor', fillcolor);
    set(this.GUIObjects(2), 'EdgeColor', edgecolor);
else
    this.removeFromFigure(AxisReference)
    this.GUIObjects(2) = fill(p2(1,:), p2(2,:), ...
        fillcolor, ...
        'EdgeColor', edgecolor, ...
        'LineWidth', 1, ...
        'HitTest', 'off');
    this.GUIObjects(1) = fill(p1(1,:), p1(2,:), ...
        fillcolor, ...
        'EdgeColor', edgecolor, ...
        'LineWidth', 1, ...
        'HitTest', 'off');
end

end

end

end
end

```



## Bridge

The bridge is a class that includes:

A creation function.

A function that is called when the bridge is deleted, to clean up the references.

A get/set interface used by the property editor on the main GUI.

A set of functions for updating, the name, validity and discretization status.

The discretization function which may return faces depending on the mode.

A set of functions for displaying or not displaying the bridge on the GUI.

```
classdef Bridge < handle
    %UNTITLED Summary of this class goes here
    % Detailed explanation goes here

    properties
        Body1 Body;
        Body2 Body;
        Connection1 Connection;
        Connection2 Connection;
        x double;

        GUIObjects;
        isActive logical = false;

        isChanged logical = true;
        isDiscretized logical = false;

        Faces Face;
    end

    properties (Dependent)
        isValid;
        name;
    end

    methods
        %% Constructor
        function this = Bridge(Body1,Body2,C1,C2,x)
            if nargin > 3
                this.Body1 = Body1;
                this.Body2 = Body2;
                this.Connection1 = C1;
                this.Connection2 = C2;
                if nargin > 4
                    this.x = x;
                else
                    this.x = 0;
                end
                fprintf('Bridge Created Successfully.\n');
            end
        end
        function deReference(this)
            if isValid(this.Body1)
                iModel = this.Body1.Group.Model;
            end
        end
    end
end
```

```

elseif isValid(this.Body2)
    iModel = this.Body2.Group.Model;
end
for i = length(iModel.Bridges):-1:1
    if iModel.Bridges(i) == this
        iModel.Bridges(i) = [];
        break;
    end
end
for iBody = [this.Body1 this.Body2]
    if isValid(iBody); iBody.change(); end
end
for iCon = [this.Connection1 this.Connection2]
    if isValid(iCon); iCon.change(); end
end
this.Faces(:) = [];
if isValid(gca)
    this.removeFromFigure(gca);
end
this.delete();
end

%% Get/Set Interface
function Item = get(this,PropertyName)
    switch PropertyName
        case 'Connection 1'
            Item = this.Connection1;
        case 'Connection 2'
            Item = this.Connection2;
        case 'Body 1'
            Item = this.Body1;
        case 'Body 2'
            Item = this.Body2;
        otherwise
            fprintf(['XXX Bridge GET Inteface for ' PropertyName ' is not found XXX\n']);
        end
    end
end
function set(~,PropertyName,~)
    switch PropertyName
        otherwise
            fprintf(['XXX Bridge SET Inteface for ' PropertyName ' is not found XXX\n']);
            return;
        end
    end
    this.change();
end

%% (Update on Demand)
function change(this)
    this.isChanged = true;
    this.isDiscretized = false;
    if this.isDiscretized
        this.Connection1.change();
        this.Connection2.change();
    end
end
function name = get.name(this)
    if this.Connection1.Orient == enumOrient.Vertical
        [~,~,x1,~] = this.Body1(1).limits(enumOrient.Vertical);
        if this.Connection1.x == x1; descriptor1 = 'Inside';
        else; descriptor1 = 'Outside'; end
    else
        [~,~,y1,~] = this.Body1(1).limits(enumOrient.Horizontal);
        if this.Connection1.x == y1; descriptor1 = 'Bottom';
        else; descriptor1 = 'Top'; end
    end
    if this.Connection2.Orient == enumOrient.Vertical
        [~,~,x1,~] = this.Body2(1).limits(enumOrient.Vertical);
        if this.Connection2.x == x1; descriptor2 = 'Inside';
        else; descriptor2 = 'Outside'; end
    else
        [~,~,y1,~] = this.Body2(1).limits(enumOrient.Horizontal);

```

```

        if this.Connection2.x == y1; descriptor2 = 'Bottom';
    else; descriptor2 = 'Top'; end
end
name1 = [];
for iBody = this.Body1
    for i = 1:length(iBody.Group.Bodies)
        if iBody.Group.Bodies(i) == iBody
            break;
        end
    end
    name1 = [name1 num2str(i) ' '];
end
name1 = ['Bodies ' name1 ' of Group' this.Body1(1).Group.name];
name2 = [];
for iBody = this.Body2
    for i = 1:length(iBody.Group.Bodies)
        if iBody.Group.Bodies(i) == iBody
            break;
        end
    end
    name2 = [name2 num2str(i) ' '];
end
name1 = ['Bodies ' name1 ' of Group' this.Body1(1).Group.name];
name = ['Bridge btwn. ' descriptor1 ' of ' name1 ' and ' ...
        descriptor2 ' of ' name2];
end
function Valid = get.isValid(this)
    Valid = true;
    if isempty(this.Body1) ...
        || isempty(this.Body2) ...
        || isempty(this.Connection1) ...
        || isempty(this.Connection2)
        Valid = false;
        fprintf('XXX Bridge is created but not fully defined XXX');
        return;
    end
    for iBody = this.Body1
        if ~any(iBody.Connections == this.Connection1)
            Valid = false;
            fprintf(['XXX Bridge ' this.name ...
                    'has invalid, body and connection pairs']);
            return;
        end
    end
    for iBody = this.Body2
        if ~any(iBody.Connections == this.Connection2)
            Valid = false;
            fprintf(['XXX Bridge ' this.name ...
                    'has invalid, body and connection pairs']);
            return;
        end
    end
end
end

%% Face Generation
function resetDiscretization(this)
    this.Faces(:) = [];
    this.isDiscretized = false;
    this.isChanged = true;
end
function discretize(this)
    this.isDiscretized = false;
    Con1 = this.Connection1;
    Con2 = this.Connection2;
    for iBody = [Con1.Bodies Con2.Bodies]
        if ~iBody.isDiscretized
            iBody.discretize();
            if ~iBody.isDiscretized
                fprintf(['XXX Exited Discretization at Body: ' iBody.name '.XXX\n']);
                return;
            end
        end
    end
end

```

```

end
end

if Con1.Orient == Con2.Orient && this.x == 0
    %% Standard, same Orientation
    % Validity Check
    if Con1.Orient == enumOrient.Vertical
        if Con1.x ~= Con2.x
            fprintf(['XXX Bridge: ' this.name ...
                ' Failed to discretize due to incompatible radii']);
            this.isDiscretized = false;
            return;
        end
    end
end

% Occlude non-B1 Con1 with B2 Con2
i = 1;
keep = true(size(Con1.NodeContacts));
for Others = Con1.NodeContacts
    if Others.Node.Body ~= this.Body1
        for B2 = Con2.NodeContacts
            if B2.Node.Body == this.Body2
                keep(i) = B2.AlignedMask(Others,-inf,inf);
            end
            if ~keep(i); break; end
        end
    end
    i = i + 1;
end
Con1.NodeContacts = Con1.NodeContacts(keep);

% Add B2 Con2 copies to Con1
% ... Copy B2 Con2
B2C2 = NodeContact.empty;
for NC = Con2.NodeContacts
    if NC.Node.Body == this.Body2
        B2C2(end+1) = CopyClass(NC);
    end
end

% Occlude B2 Con2 with B1 Con1
i = 1;
keep = true(size(Con2.NodeContacts));
for B2 = Con2.NodeContacts
    if B2.Node.Body == this.Body2
        for B1 = Con1.NodeContacts
            if B1.Node.Body == this.Body1
                keep(i) = B1.AlignedMask(B2,-inf,inf);
            end
            if ~keep(i); break; end
        end
    end
    i = i + 1;
end
Con2.NodeContacts = Con2.NodeContacts(keep);

% ... Add to Con1
Con1.addNodeContacts(B2C2);

elseif Con1.Orient == enumOrient.Vertical && Con1.Orient == Con2.Orient
    %% Both Vertical, Offset

    % Validity Check
    if Con1.x ~= Con2.x
        fprintf(['XXX Bridge: ' this.name ...
            ' Failed to discretize due to incompatible radii']);
        this.isDiscretized = true;
        return;
    end

    %% Both Vertical

```

```

% Get node contacts from Con2 and shift them
for NC = Con2.NodeContacts
    NC.Start = NC.Start + this.x;
    NC.End = NC.End + this.x;
end

% Con1 mask other of Con2 within bounds of B2
keep = true(size(Con2.NodeContacts));
switch Con1.Orient
    case enumOrient.Vertical
        [b1,b2,~,~] = this.Body2.limits(enumOrient.Horizontal);
    case enumOrient.Horizontal
        [b1,b2,~,~] = this.Body2.limits(enumOrient.Vertical);
end
for mask = Con1.NodeContacts
    if mask.Node.Body == this.Body1
        for i = 1:length(Con2.NodeContacts)
            if keep(i)
                target = Con2.NodeContacts(i);
                if target.Node.Body ~= this.Body2
                    keep(i) = mask.AlignedMask(target,b1,b2);
                end
            end
        end
    end
end
Con2.NodeContacts = Con2.NodeContacts(keep);

% Con2 mask other of Con1 within bounds of B1
keep = true(size(Con1.NodeContacts));
switch Con1.Orient
    case enumOrient.Vertical
        [b1,b2,~,~] = this.Body1.limits(enumOrient.Horizontal);
    case enumOrient.Horizontal
        [b1,b2,~,~] = this.Body1.limits(enumOrient.Vertical);
end
for mask = Con2.NodeContacts
    if mask.Node.Body == this.Body2
        for i = 1:length(Con1.NodeContacts)
            if keep(i)
                target = Con1.NodeContacts(i);
                if target.Node.Body ~= this.Body1
                    keep(i) = mask.AlignedMask(target,b1,b2);
                end
            end
        end
    end
end
Con1.NodeContacts = Con1.NodeContacts(keep);

% Copy NContacts of B1 from C1 onto C2
MoveContacts = NodeContact.empty;
for NC = Con1.NodeContacts
    if NC.Node.Body == this.Body1
        MoveContacts(end+1) = NodeContact(...
            NC.Node,NC.Start,NC.End,NC.Type,NC.Connection);
    end
end
Con2.addNodeContacts(MoveContacts);

% Unshift Node Contacts in Con2
for NC = Con2.NodeContacts
    NC.Start = NC.Start - this.x;
    NC.End = NC.End - this.x;
end

elseif Con1.Orient == enumOrient.Horizontal && ...
    Con2.Orient == enumOrient.Horizontal
    %% Both Horizontal, Offset
    % Determine which one to take from, it would be the smaller of the

```

```

% two
r1 = 0;
r2 = 0;
for NContact = this.Connection1.NodeContacts
    if any(NContact.Node.Body == this.Body1)
        if r1 < NContact.End
            r1 = NContact.End;
        end
    end
end
for NContact = this.Connection2.NodeContacts
    if any(NContact.Node.Body == this.Body2)
        if r2 < NContact.End; r2 = NContact.End; end
    end
end
if r1 > r2
    Source = this.Connection2;
    Destination = this.Connection1;
    DestinationBody = this.Body1;
    SourceBody = this.Body2;
    max_r = r2;
else
    Source = this.Connection1;
    Destination = this.Connection2;
    DestinationBody = this.Body2;
    SourceBody = this.Body1;
    max_r = r1;
end
min_r = 10000;
for NContact = Source.NodeContacts
    if NContact.Node.Body == SourceBody
        if min_r > NContact.Start
            min_r = NContact.Start;
            if min_r == 0; break; end
        end
    end
end
end

% Gather Node Contacts from Source for comparison with Destination
SContacts(length(Source.NodeContacts)) = NodeContact; n = 1;
keep = true(size(Source.NodeContacts));
for i = 1:length(Source.NodeContacts)
    NContact = Source.NodeContacts(i);
    if NContact.Node.Body == SourceBody
        SContacts(n) = NContact; n = n + 1;
        keep(i) = false;
    end
end
Source.NodeContacts = Source.NodeContacts(keep);
SContacts = SContacts(1:n-1);
Ss = zeros(size(SContacts));
Es = zeros(size(SContacts));
i = 1;
for NContact = SContacts
    Es(i) = NContact.End;
    Ss(i) = NContact.Start;
    i = i + 1;
end

keep = true(size(Destination.NodeContacts));
keep2 = true(size(SContacts));
for i = 1:length(Destination.NodeContacts)
    if Destination.NodeContacts(i).Node.Body == DestinationBody
        DCont = Destination.NodeContacts(i);
        s = DCont.Start;
        e = DCont.End;
        for j = 1:length(SContacts)
            if keep2(j)
                % Calculate Percentange that the segment covers
                P = ...

```

```

        GetAreaPercentHorizontal(this.x,s,e,2*Es(j)) - ...
        GetAreaPercentHorizontal(this.x,s,e,2*Ss(j));
    if P == 0; continue; end
    if isempty(DCont.data)
        DCont.data = struct('Perc',1);
    end
    if isfield(DCont.data,'Perc')
        DCont.data.Perc = DCont.data.Perc - P;
    else; DCont.data.Perc = 1 - P;
    end

    % Calculate the Percentage of the source that the segment
    % ... covers
    P2 = ...
        GetAreaPercentHorizontal(this.x,Ss(j),Es(j),2*e) - ...
        GetAreaPercentHorizontal(this.x,Ss(j),Es(j),2*s);
    if isempty(SContacts(j).data)
        SContacts(j).data = struct('Perc',1);
    end
    if isfield(SContacts(j).data,'Perc')
        SContacts(j).data.Perc = SContacts(j).data.Perc - P2;
    else; SContacts(j).data.Perc = 1 - P2;
    end

    % Make Faces
    P1 = DCont.data.Perc;
    DCont.data.Perc = 1;
    NewFace = Face(...
        NodeContact(SContacts(j).Node,...
        SContacts(j).Start + this.x,SContacts(j).End + this.x,...
        SContacts(j).Type,SContacts(j).Connection),DCont,true);
    DCont.data.Perc = P1;

    % Modify Properties
    if isfield(NewFace.data,'Area')
        NewFace.data.Area = NewFace.data.Area*P;
        if isfield(NewFace.data,'R')
            NewFace.data.R = NewFace.data.R/P;
        elseif isfield(NewFace.data,'Dh')
            NewFace.data.Dh = 2*(max_r - min_r);
        end
    elseif isfield(NewFace.data,'U')
        NewFace.data.U = NewFace.data.U*P;
    end
    this.Faces = [this.Faces NewFace];

    if ~keep(i); break; end
end
end
end
end
for i = 1:length(Destination.NodeContacts)
    if isfield(Destination.NodeContacts(i).data,'Perc')
        if Destination.NodeContacts(i).data.Perc <= 1e-6
            keep(i) = false;
        end
    end
end
Destination.NodeContacts = Destination.NodeContacts(keep);
for i = 1:length(SContacts)
    if isfield(SContacts(i).data,'Perc')
        if SContacts(i).data.Perc <= 1e-6
            keep2(i) = false;
        end
    end
end
Source.addNodeContacts(SContacts(keep2));
else
    fprintf(['XXX The Bridge Discretization method has not been ' ...
        'updated to improved standards. It may not work as expected XXX\n']);
end

```

```

%% Mix, Offset
% Move Node Contacts from Connection2 that are associated with
% Body 2 and add them to Connection1 in range of Body1
if this.Connection1.Orient == enumOrient.Horizontal
    Source = this.Connection1;
    SourceBody = this.Body1;
    Destination = this.Connection2;
    DestinationBody = this.Body2;
else
    Source = this.Connection2;
    SourceBody = this.Body2;
    Destination = this.Connection1;
    DestinationBody = this.Body1;
end
end
max_r = 0;
min_r = 10000;
for NContact = Source.NodeContacts
    if max_r < NContact.End
        max_r = NContact.End;
    end
    if min_r > NContact.Start
        min_r = NContact.Start;
    end
end
Dh = 2*max_r - 2*min_r;
DontKeep = false(size(Source.NodeContacts));
for i = 1:length(Source.NodeContacts)
    if Source.NodeContacts(i).Node.Body == SourceBody
        SContacts = Source.NodeContacts(i);
        DontKeep(i) = true;
    end
end
Source.NodeContacts(DontKeep) = [];
for i = 1:length(Destination.NodeContacts)
    if Destination.NodeContacts(i).Node.Body == DestinationBody
        r = Destination.x;
        DCont = Destination.NodeContacts(i);
        s = DCont.Start;
        e = DCont.End;
        for j = 1:length(SContacts)
            SCont = SContacts(j);
            % Calculate Percentange that the segment covers
            if isscalar(s)
                if isscalar(e)
                    % Both scalars
                    P = GetAreaPercentMix(r,this.x,s,e,SCont.End) - ...
                        GetAreaPercentMix(r,this.x,s,e,SCont.Start);
                else
                    % just "s" is a scalar
                    for k = 1:length(e)
                        P(k) = GetAreaPercentMix(r,this.x,s,e(k),SCont.End) - ...
                            GetAreaPercentMix(r,this.x,s,e(k),SCont.Start);
                    end
                end
            else
                if isscalar(e)
                    % just "e" is a scalar
                    for k = 1:length(s)
                        P(k) = GetAreaPercentMix(r,this.x,s(k),e,SCont.End) - ...
                            GetAreaPercentMix(r,this.x,s(k),e,SCont.Start);
                    end
                else
                    % Both vectors
                    for k = 1:length(s)
                        P(k) = GetAreaPercentMix(r,this.x,s(k),e(k),SCont.End) - ...
                            GetAreaPercentMix(r,this.x,s(k),e(k),SCont.Start);
                    end
                end
            end
        end
        if ~isempty(DCont.data) && isfield(DCont.data,'Perc')
            DCont.data.Perc = DCont.data.Perc - P;
        end
    end
end

```



```

else
    DCont.data.Perc = 1 - P;
end
if any(P > 0)
    % Make Faces
    % Precondition
    SCont.Start = this.x - max_r;
    SCont.End = this.x + max_r;
    P1 = DCont.data.Perc;
    DCont.data.Perc = 1;
    NewFace = Face(SCont,DCont);
    % Recondition
    DCont.data.Perc = P1;
    if isfield(NewFace.data,'Area')
        NewFace.data.Area = NewFace.data.Area.*P;
        if isfield(NewFace.data,'Dh')
            NewFace.data.Dh = Dh;
        elseif isfield(NewFace.data,'R')
            NewFace.data.R = NewFace.data.R./P;
        end
    elseif isfield(NewFace.data,'U')
        NewFace.data.U = NewFace.data.U.*P;
    end
    this.Faces = [this.Faces NewFace];
end
if DontKeep(i)
    break;
end
end
end
end
Destination.NodeContacts(DontKeep) = [];

end
this.isDiscretized = true;
end

%% Graphics
function removeFromFigure(this,AxisReference)
    if ~isempty(this.GUIObjects)
        children = get(AxisReference,'Children');
        for obj = this.GUIObjects
            if isgraphics(obj)
                for i = length(children):-1:1
                    if isgraphics(children(i)) && children(i) == obj
                        children(i).delete;
                        break;
                    end
                end
            end
        end
        this.GUIObjects = [];
    end
end
function show(this,AxisReference)
    this.removeFromFigure(AxisReference);
    % Plot a dotted line between the middle of the Connection1's Overlap
    % with Body1 to the middle of Connection2's Overlap with Body2

    % Find P1;
    Ax = this.Connection1.Group;
    R = RotMatrix(Ax.Position.Rot - pi/2);
    d = this.Connection1.x;
    switch this.Connection1.Orient
        case enumOrient.Vertical
            [~,~,y1,y2] = this.Body1.limits(enumOrient.Horizontal);
            A = [Ax.Position.x; Ax.Position.y] + R*[d; (y1+y2)/2];
            B = [Ax.Position.x; Ax.Position.y] + R*[-d; (y1+y2)/2];
        case enumOrient.Horizontal
            [~,~,x1,x2] = this.Body1.limits(enumOrient.Vertical);
            A = [Ax.Position.x; Ax.Position.y] + R*[(x1+x2)/2; d];
    end
end

```

```

        B = [Ax.Position.x; Ax.Position.y] + R*[-(x1+x2)/2; d];
    end

    % Find P2;
    Ax = this.Connection2.Group;
    R = RotMatrix(Ax.Position.Rot - pi/2);
    d = this.Connection2.x;
    switch this.Connection1.Orient
        case enumOrient.Vertical
            [~,~,y1,y2] = this.Body2.limits(enumOrient.Horizontal);
            C = [Ax.Position.x; Ax.Position.y] + R*[d; (y1+y2)/2];
            D = [Ax.Position.x; Ax.Position.y] + R*[-d; (y1+y2)/2];
        case enumOrient.Horizontal
            [~,~,x1,x2] = this.Body2.limits(enumOrient.Vertical);
            C = [Ax.Position.x; Ax.Position.y] + R*[(x1+x2)/2; d];
            D = [Ax.Position.x; Ax.Position.y] + R*[-(x1+x2)/2; d];
    end

    % Find minimum pair
    pair = zeros(2,2);
    dAC = Dist4Compare(A,C);
    dAD = Dist4Compare(A,D);
    dmin = Dist4Compare(B,D);
    if dAC < dmin; pair = [A C]; dmin = dAC;
    else; pair = [B D];
    end
    if dAD < dmin; pair = [A D]; dmin = dAD; end
    if Dist4Compare(B,C) < dmin; pair = [B C]; end

    % Find the closest blank space in the model and drag the label there
    [d, y, h] = this.Body1.Group.Model.findInterSpace(pair);
    %newpair = [pair(:,1) [d; y+h/2] [d; y-h/2] pair(:,2)];

    % Two points in pair are minimum distance
    this.GUIObjects = line(...
        pair(1,:),pair(2,:),...
        'Color',[0.5 0.5 0.5]);
    end
end
end
end

```

## Connection

The connection is a class that includes the following functions:

A constructor / creation function.

A function that is called before it is deleted to clean up the external references.

A get/set interface used by the property dropdown editor on the main GUI.

A set of functions to append the internal lists of other classes.

A series of functions to update the name, list of node contacts

The discretization function.

A series of functions used to display or not display the connection on the main GUI.

```
classdef Connection < handle
%UNTITLED Summary of this class goes here
% Detailed explanation goes here
properties (Constant)
    Extension = 1.1;
    MinimumDisplayLength = 0.4;
    ActiveColor = [0 1 0];
    NormalColor = [0.2 0.2 0.2];
end

properties (Access = public)
    ID;
    Group Group;
    Bodies Body;
    isActive = false;
    x double = 0;
    Orient enumOrient = enumOrient.Horizontal;

    RefFrame Frame;
    GUIObjects;
    NodeContacts NodeContact;
    Faces Face;

    isDiscretized logical = false;
    isChanged logical = true;

    BodiesToNotJoin Body;
end

properties (Dependent)
    name;
    index;
    id;
end

methods
%% Constructor
function this = Connection(x,Orient,Group)
    switch nargin
        % case 0
        % obj.x = 0;
    end
end
```

```

    % obj.Orient = enumOrient.Horizontal;
    case 1
        this.x = x;
        % this.Orient = enumOrient.Horizontal;
    case 2
        this.x = x;
        this.Orient = Orient;
    case 3
        this.x = x;
        this.Orient = Orient;
        this.Group = Group;
        this.ID = Group.Model.getConID();
    end
end
end

function deReference(this)
    % Remove Reference from Group
    iGroup = this.Group;
    for i = length(iGroup.Connections):-1:1
        if iGroup.Connections(i) == this
            iGroup.Connections(i) = [];
            iGroup.isChanged = true;
        end
    end
end

% Remove relations from the relationship managers
for RMan = iGroup.RelationManagers
    if RMan.Orient == this.Orient
        RMan.isChanged = true;
        for i = length(RMan.Relations):-1:1
            if RMan.Relations(i).con1 == this || ...
                RMan.Relations(i).con2 == this
                RMan.Relations(i).deReference();
            end
        end
        RMan.update();
    end
end

if ~isempty(this.Bodies) % i.e. Body/deReference has not been called already
    % Remove Reference from any Bodies
    for iBody = this.Bodies
        iBody.deReference();
        iBody.delete();
    end
    % Remove Reference from any Bridges
    iModel = iGroup.Model;
    for i = length(iModel.Bridges):-1:1
        if iModel.Bridges(i).Connection1 == this || iModel.Bridges(i).Connection2 == this
            iModel.Bridges(i).deReference();
            iModel.Bridge(i).delete();
            iModel.Bridges(i) = [];
        end
    end
    % Remove Reference from any Leaks
    for i = length(iModel.LeakConnections):-1:1
        if iModel.LeakConnections(i).Connection1 == this ...
            || iModel.LeakConnections(i).Connection2 == this
            iModel.LeakConnections(i).deReference();
            iModel.LeakConnections(i).delete();
            iModel.LeakConnections(i) = [];
        end
    end
    % Remove any visual remenant
    this.removeFromFigure(gca);
end
this.delete();
end

function change(this)
    this.isChanged = true;
    this.Faces(:) = [];
end

```

```

    this.NodeContacts(:) = [];
    for iBody = this.Bodies; iBody.change(); end
    this.isDiscretized = false;
end
function CleanUpBodies(this)
    for i = length(this.Bodies):-1:1
        if ~isvalid(this.Bodies(i))
            this.Bodies(i) = [];
        end
    end
end
function update(this)
    if isempty(this.ID)
        this.ID = this.Group.Model.getConID();
    end
    if any(~isvalid(this.Bodies))
        this.Bodies = this.Bodies(isvalid(this.Bodies));
    end
    if ~isvalid(this.RefFrame)
        this.RefFrame = [];
    end
    this.isChanged = false;
end
function [yesno] = IsFixedTo(this, other)
    yesno = false;
    for iRM = this.Group.RelationManagers
        if iRM.Orient == this.Orient && iRM.Orient == other.Orient
            iRM.update();
            for row = 1:size(iRM.Grid,1)
                if iRM.Grid(row,this.index) && ...
                    iRM.Grid_modes{row} == enumRelation.Constant
                    if iRM.Grid(row,other.index)
                        yesno = true;
                        return;
                    end
                end
            end
        end
    end
end
end
end

%% Get/Set Interface
function Item = get(this,PropertyName)
    if this.isChanged; this.update(); end
    switch PropertyName
        case 'x'
            Item = this.x;
        case 'RefFrame'
            Item = this.RefFrame;
        case 'Bodies'
            Item = this.Bodies;
        case 'Isolated Bodies'
            Item = this.BodiesToNotJoin;
        case 'isStationary'
            if isempty(this.RefFrame)
                Item = true;
            else
                Item = false;
            end
        otherwise
            fprintf(['XXX Connection GET Inteface for ' PropertyName ' is not found XXX\n']);
    end
end

%% (Update on Demand) Triggers
function name = get.name(this)
    switch this.Orient
        case enumOrient.Vertical
            name = ['Vertical Connection at x = ' num2str(this.x,3)];
        case enumOrient.Horizontal
            name = ['Horizontal Connection at y = ' num2str(this.x,3)];
    end
end

```

```

    end
end
function index = get.index(this)
    i = 1;
    for iCon = this.Group.Connections
        if iCon == this
            index = i;
            return;
        end
        i = i + 1;
    end
    index = 0;
end

function set(this,PropertyName,Item)
    switch PropertyName
        case 'x'
            % Check all Relationships
            for iCon = this.Group.Connections
                if iCon.x == Item && ...
                    iCon.Orient == this.Orient
                    % This kind of shift can't result in a merge.
                    return;
                end
            end
            for RMan = this.Group.RelationManagers
                if RMan.Orient == this.Orient
                    RMan.Edit(this,Item-this.x);
                    break;
                end
            end
        case 'RefFrame'
            if isempty(Item); this.RefFrame = Frame.empty;
            else; this.RefFrame = Item;
            end
        otherwise
            fprintf(['XXX Connection SET Inteface for ' ...
                PropertyName ' is not found XXX\n']);
            return;
        end
    end
    this.change();
end

function set.isActive(this,value)
    if islogical(value)
        if isvalid(this)
            this.isActive = value;
        end
    else
        fprintf('Input to isActive must be a boolean value.\n');
    end
end

function functions(this,FunctionName)
    switch FunctionName
        case 'Add Bodies To Not Join'
            [cx,cy] = ginput(1);
            TheBody = findConnectedBody(this,[cx cy]);
            this.addBodyToNotJoin(TheBody);
        case 'Remove Bodies To Not Join'
            if ~isempty(this.BodiesToNotJoin)
                objects = cell(1,length(this.BodiesToNotJoin));
                names = objects;
                i = 1;
                for iBody = this.BodiesToNotJoin
                    objects{i} = iBody;
                    names{i} = iBody.name;
                    i = i + 1;
                end
                [indx,tf] = listdlg(...

```

```

        'PromptString','Which one are you going to remove?','...
        'SelectionMode','single',...
        'ListString',names);
    if tf; this.BodiesToNotJoin(indx) = []; end
    end
    otherwise
end
end

%% Operators
function isequal = isFunctionallyEqualTo(this,otherConnection)
    if this.x == otherConnection.x && ...
        this.Orient == otherConnection.Orient && ...
        ((isempty(this.RefFrame) && isempty(otherConnection.RefFrame)) || ...
        (isempty(this.RefFrame) == isempty(otherConnection.RefFrame) && ...
        this.RefFrame == otherConnection.RefFrame))
        isequal = true;
    else
        isequal = false;
    end
end

%% Interating
function addBody(this,BodiesToAdd)
    try
        count = length(this.Bodies);
        if isrow(BodiesToAdd)
            this.Bodies = [this.Bodies BodiesToAdd];
        else
            this.Bodies = [this.Bodies BodiesToAdd'];
        end
        this.Bodies = unique(this.Bodies);
        if count ~= length(this.Bodies)
            this.removeFaces();
        end
        return;
    catch
        fprintf('XXX Error in Connection/AddBody XXX\n');
    end
end

function addBodyToNotJoin(this,BodiesToNotJoin)
    len = length(this.BodiesToNotJoin);
    this.BodiesToNotJoin(len+length(BodiesToNotJoin)) = Body();
    this.BodiesToNotJoin(length(len+1:end)) = BodiesToNotJoin;
    this.BodiesToNotJoin = unique(this.BodiesToNotJoin);
end

function TheBody = findConnectedBody(this,Pnt)
    if this.isChanged; this.update(); end
    % Find a body in this Group that is selected and closest
    Pntmod = (RotMatrix(pi/2-this.Group.Position.Rot)*Pnt)...
        - [this.Group.Position.x; this.Group.Position.y];
    mindist = inf;
    for iBody = this.Bodies
        % Establish Rectangle of iBody
        [~,~,x1,x2] = iBody.limits(enumOrient.Vertical);
        [~,~,y1,y2] = iBody.limits(enumOrient.Horizontal);

        R.Width = x2-x1;
        R.Height = y2-y1;
        R.Cx = (x1+x2)/2;
        R.Cy = (y1+y2)/2;
        dist = Dist2Rect(Pntmod(1),Pntmod(2),R.Cx,R.Cy,R.Width,R.Height);
        if dist < mindist
            mindist = dist;
            TheBody = iBody;
        else
            R.Cx = -R.Cx;
            dist = Dist2Rect(Pntmod(1),Pntmod(2),R.Cx,R.Cy,R.Width,R.Height);
            if dist < mindist
                if dist == 0

```

```

        TheBody = iBody;
        return;
    end
    mindist = dist;
    TheBody = iBody;
end
end
end
end

%% Working with nodes
function deleteNodeContactsFromObj(this,Obj)
    for iBridge = this.Group.Model.Bridges
        if iBridge.Connection1 == this
            iBridge.change();
        elseif iBridge.Connection2 == this
            iBridge.change();
        end
    end
    this.cleanUpNodeContacts();
    LEN = length(this.NodeContacts);
    if LEN == 0; return; end
    if this.isDiscretized; this.removeFaces(); end
    i = 1;
    while (i < LEN+1 && this.NodeContacts(i).Node.Body ~= Obj); i = i + 1; end
    START = i - 1;
    while (i < LEN+1 && this.NodeContacts(i).Node.Body == Obj); i = i + 1; end
    END = i;
    if START ~= 0
        if END ~= LEN+1
            this.NodeContacts = [this.NodeContacts(1:START) this.NodeContacts(END:LEN)];
        else
            this.NodeContacts = this.NodeContacts(1:START);
        end
    else
        if END ~= LEN+1
            this.NodeContacts = this.NodeContacts(END:LEN);
        else
            this.NodeContacts = NodeContact.empty;
        end
    end
end
end
function addNodeContacts(this,newContacts)
    this.NodeContacts = [this.NodeContacts newContacts];
    if this.isDiscretized
        this.removeFaces();
    end
end
function removeFaces(this)
    this.Faces = Face.empty;
    this.isDiscretized = false;
end
function cleanUpNodeContacts(this)
    dontkeep = true(size(this.NodeContacts));
    for i = 1:length(this.NodeContacts)
        if isvalid(this.NodeContacts(i)) && ...
            isvalid(this.NodeContacts(i).Node) && ...
            isvalid(this.NodeContacts(i).Node.Body)
            dontkeep(i) = false;
        end
    end
    if any(dontkeep)
        this.NodeContacts = this.NodeContacts(~dontkeep);
    end
end
function resetDiscretization(this)
    if isvalid(this)
        this.NodeContacts(:) = [];
        this.Faces(:) = [];
        this.isDiscretized = false;
        this.isChanged = true;
    end
end

```



```

end
end
function discretize(this)
    if this.isChanged; this.update(); end
    this.Faces = Face.empty;
    if isempty(this.Bodies) || ...
        (this.Orient == enumOrient.Vertical && this.x == 0)
        this.isDiscretized = true;
        this.Faces = Face.empty;
        return;
    end

    % Remove Bodies that should not conduct
    keep = true(1,length(this.NodeContacts));
    for iBody = this.BodiesToNotJoin
        for i = 1:length(this.NodeContacts)
            if this.NodeContacts(i).Node.Body == iBody
                keep(i) = false;
            end
        end
    end
    this.NodeContacts = this.NodeContacts(keep);

    if ~this.isDiscretized
        for iBody = this.Bodies
            if ~iBody.isDiscretized
                iBody.discretize();
                if ~iBody.isDiscretized
                    return;
                end
            end
        end
    end
    if this.Group.isChanged
        this.Group.isEnvironmentCasted = true;
        this.Group.update();
    elseif ~this.Group.isEnvironmentCasted
        this.Group.isEnvironmentCasted = true;
        this.Group.updateBorder(true);
    end
    %% INITIALIZE
    this.Faces(2*length(this.NodeContacts)) = Face();
    n = 1;
    % Clean Up
    keep = true(size(this.NodeContacts));
    for i = 1:length(this.NodeContacts)
        if ~isvalid(this.NodeContacts(i)) || ...
            ~isvalid(this.NodeContacts(i).Node) || ...
            ~isvalid(this.NodeContacts(i).Node.Body)
            keep(i) = false;
        end
    end
    if any(~keep); this.NodeContacts = this.NodeContacts(keep); end

    % Sort the environmental connections to the end
    members = false(size(this.NodeContacts)); i = 1;
    for nc = this.NodeContacts
        members(i) = this.NodeContacts(i).Node.Type == enumNType.EN;
        i = i + 1;
    end
    envNC = this.NodeContacts(members);
    this.NodeContacts(members) = [];
    this.addNodeContacts(envNC);

    %% GO THROUGH EACH NODE COMBINATION
    keep = true(length(this.NodeContacts),1);
    len = length(keep);
    for i = 1:length(this.NodeContacts)
        if length(keep) >= i && keep(i)
            for j = i+1:length(this.NodeContacts)
                if length(keep) >= j && keep(j)
                    if this.NodeContacts(i).Node.Body ~= this.NodeContacts(j).Node.Body

```



```

        end
    end
    if ~found
        fprintf(['XXX Connection '
            this.name ' does not know what its group is XXX\n']);
        return;
    end
end
end
if this.Group.Model.showRelations
    for RMan = this.Group.RelationManagers
        if RMan.Orient == this.Orient
            ind = this.index;
            if isempty(RMan.Grid) || ind > size(RMan.Grid,2)
                RMan.update();
            end
            rows = find(RMan.Grid(:,ind) == true);
            if ~isempty(rows)
                max_count = 0;
                max_index = 0;
                for i = rows
                    count = sum(RMan.Grid(i,:));
                    if count > max_count
                        max_index = i;
                        max_count = count;
                    end
                end
                color = RMan.getColor(max_index);
                break;
            end
            if this.isActive
                color = Connection.ActiveColor;
            else
                color = Connection.NormalColor;
            end
            break;
        end
    end
end
else
    if this.isActive
        color = Connection.ActiveColor;
    else
        color = Connection.NormalColor;
    end
end
end
function updateColor(this)
    if ~isempty(this.GUIObjects)
        for iGraphicsObject = this.GUIObjects
            set(iGraphicsObject,'Color',this.getColor());
        end
    end
end
function show(this, AxisReference)
    if this.isChanged; this.update(); end
    color = this.getColor();
    switch this.Orient
        case enumOrient.Vertical
            % Plot two lines on equal sides of the Group
            % Find vertical extent of the Group
            VectorLength = Connection.Extension*max(this.Group.Height,0.1);
            OffsetRot = this.Group.Position.Rot;
            Offset = (VectorLength-this.Group.Height)/2;
            % Make a template vector
            R = RotMatrix(OffsetRot);
            templateVector = R * [VectorLength; 0];
            Trans = [this.Group.Position.x; this.Group.Position.y];
            LeftStart = R * [-Offset; this.x] + Trans;
            RightStart = R * [-Offset; -this.x] + Trans;

            % Plot line

```

```

if this.x == 0
%
%   if length(this.GUIObjects) == 1 && isgraphics(this.GUIObjects(1))
%   % Plot line
%   try
%       set(this.GUIObjects,'Color',color);
%       set(this.GUIObjects,'XData',...
%           [LeftStart(1) LeftStart(1)+templateVector(1)]);
%       set(this.GUIObjects,'YData',...
%           [LeftStart(2) LeftStart(2)+templateVector(2)]);
%   catch
%       this.removeFromFigure(AxisReference);
%       this.GUIObjects(1) = line(...
%           [LeftStart(1) LeftStart(1)+templateVector(1)],...
%           [LeftStart(2) LeftStart(2)+templateVector(2)],...
%           'Userdata',this,'Color',color,'LineStyle','--',...
%           'HitTest','off');
%   end
%   else
%       this.removeFromFigure(AxisReference);
%       this.GUIObjects(1) = line(...
%           [LeftStart(1) LeftStart(1)+templateVector(1)],...
%           [LeftStart(2) LeftStart(2)+templateVector(2)],...
%           'Userdata',this,'Color',color,'LineStyle','--',...
%           'HitTest','off');
%   end
%   else
%       if length(this.GUIObjects) == 2 && ...
%           isgraphics(this.GUIObjects(1)) && isgraphics(this.GUIObjects(2))
%           set(this.GUIObjects(1),'Color',color);
%           set(this.GUIObjects(1),'XData',...
%               [LeftStart(1) LeftStart(1)+templateVector(1)]);
%           set(this.GUIObjects(1),'YData',...
%               [LeftStart(2) LeftStart(2)+templateVector(2)]);
%           set(this.GUIObjects(2),'Color',color);
%           set(this.GUIObjects(2),'XData',...
%               [RightStart(1) RightStart(1)+templateVector(1)]);
%           set(this.GUIObjects(2),'YData',...
%               [RightStart(2) RightStart(2)+templateVector(2)]);
%       else
%           this.removeFromFigure(AxisReference);
%           this.GUIObjects(2) = line(...
%               [RightStart(1) RightStart(1)+templateVector(1)],...
%               [RightStart(2) RightStart(2)+templateVector(2)],...
%               'Userdata',this,'Color',color,'LineStyle','--',...
%               'HitTest','off');
%           this.GUIObjects(1) = line(...
%               [LeftStart(1) LeftStart(1)+templateVector(1)],...
%               [LeftStart(2) LeftStart(2)+templateVector(2)],...
%               'Userdata',this,'Color',color,'LineStyle','--',...
%               'HitTest','off');
%       end
%   end
case enumOrient.Horizontal
% Plot a single line
% Find horizontal extent of the Group
VectorLength = Connection.Extension*this.Group.Width/2;
OffsetRot = this.Group.Position.Rot;
% Make a template vector
R = RotMatrix(OffsetRot);
Trans = [this.Group.Position.x ;
        this.Group.Position.y ];
LeftPoint = R * [this.x; 0.5*VectorLength] + Trans;
RightPoint = R * [this.x; -0.5*VectorLength] + Trans;

% Plot line
if length(this.GUIObjects) == 1 && isgraphics(this.GUIObjects)
    set(this.GUIObjects,'Color',color);
    set(this.GUIObjects,'XData',[LeftPoint(1) RightPoint(1)]);
    set(this.GUIObjects,'YData',[LeftPoint(2) RightPoint(2)]);
else
    this.GUIObjects = line(...

```

```

        [LeftPoint(1) RightPoint(1)], [LeftPoint(2) RightPoint(2)], ...
        'Userdata', this, 'Color', color, 'LineStyle', '--', ...
        'HitTest', 'off');
    end
end
% fprintf(['Plotted Connection ' this.name '.\n']);
end
end
end

% Helper functions - UNUSED
function face = appendDynamicFaceVert(face,k,T1,s,e,Area)
% Face.
% .isDynamic - DONE
% .Node1 - DONE
% .Node2 - DONE
% .A - Mix/Gas Append
% .dh - Static
% .Type - DONE
% .value - Solid/Mix/Gas Append
% .K - Gas Append
% .ActiveTimes - Always Append
if s < e
    switch face.Type
    case enumFType.Solid
        % Combine resistances and store as a conductance
        face.value = [face.value Area(n,s,e)/(n1.value+n2.value)];
    case enumFType.Mix
        % Store only the resistance as a conductance
        if T1 == enumFType.Solid
            face.A = [face.A Area(n,s,e)];
            face.value = [face.value (face.A(end)/(n1.value))];
        else
            face.A = [face.A Area(n,s,e)];
            face.value = [face.value (face.A(end)/(n2.value))];
        end
    case enumFType.Gas
        % Record the combined distance stored in Ri
        face.A = [face.A Area(n,s,e)];
        face.value = [face.value n1.value + n2.value];
    end
    face.ActiveTimes = [face.ActiveTimes k];
end
end
function face = genStaticFace(n1,n2,Area)
face.Node1 = n1.node;
face.Node2 = n2.node;
face.isDynamic = false;
face.K = 0;
face.ActiveTimes = [];
face.Type = getFaceType(n1.Type,n2.Type);
switch face.Type
    case enumFType.Solid
        % Combine resistances and store as a conductance
        face.value = ...
            (Area(n,max([n1.Start n2.Start]),min([n1.End n2.End]))...
            / (n1.value + n2.value));
    case enumFType.Mix
        % Store only the resistance as a conductance
        if n1.Type == enumFType.Solid
            face.dh = n2.dh;
            face.A = Area(n,max([n1.Start n2.Start]),min([n1.End n2.End]));
            face.value = (face.A/(n1.value));
        else
            face.dh = n1.dh;
            face.A = Area(n,max([n1.Start n2.Start]),min([n1.End n2.End]));
            face.value = (face.A/(n2.value));
        end
    case enumFType.Gas
        % Record the combined distance stored in value
        face.value = n1.value + n2.value;
end

```

end  
end

# Environment

The environment is a class that contains the following functions:

A constructor / creation function.

A get / set interface used by the property drop-down editor on the main GUI.

A short discretize function.

A function to remove its graphical representation from the figure, show functionality is within group.

```
classdef Environment < handle
    %ENVIRONMENT Summary of this class goes here
    % Detailed explanation goes here

    properties (Constant)
        StdPressure = 101325; % Pa
        StdTemperature = 298; % K
        Stdh = 20; % W/m*K
        StdGas = 'AIR';
    end

    properties
        Pressure double;
        Temperature double;
        h double;
        matl Material;
        nodeIndex double;
        name char;

        GUIObjects = [];

        isDiscretized logical = false;
        Node Node;
    end

    properties (Dependent)
        Group
    end

    methods
        %% Constructor
        function this = Environment(Pressure, Temperature, h, MaterialRef)
            switch nargin
                case 0
                    this.Pressure = Environment.StdPressure;
                    this.Temperature = Environment.StdTemperature;
                    this.h = Environment.Stdh;
                    this.matl = Material(Environment.StdGas);
                    this.name = 'Standard AIR Environment';
                case 1
                    this.Pressure = Pressure;
                    this.Temperature = Environment.StdTemperature;
                    this.h = Environment.Stdh;
                    this.matl = Material(Environment.StdGas);
                    this.name = 'Untitled Environment';
                case 2
                    this.Pressure = Pressure;
                    this.Temperature = Temperature;
            end
        end
    end
end
```

```

        this.h = Environment.Stdh;
        this.matl = Material(Environment.StdGas);
        this.name = 'Untitled Environment';
    case 3
        this.Pressure = Pressure;
        this.Temperature = Temperature;
        this.h = h;
        this.matl = Material(Environment.StdGas);
        this.name = 'Untitled Environment';
    case 4
        this.Pressure = Pressure;
        this.Temperature = Temperature;
        this.h = h;
        this.matl = MaterialRef;
        this.name = 'Untitled Environment';
    end
end

%% Get/Set Interface
function Item = get(this,PropertyName)
    switch PropertyName
    case 'Pressure'
        Item = this.Pressure;
    case 'Temperature'
        Item = this.Temperature;
    case 'h'
        Item = this.h;
    case 'Gas'
        Item = this.matl;
    case 'Name'
        Item = this.name;
    otherwise
        fprintf(['XXX Environment GET Inteface for ' PropertyName ' is not found XXX\n']);
    end
end

function set(this,PropertyName,Item)
    switch PropertyName
    case 'Pressure'
        this.Pressure = Item;
        if this.isDiscretized
            this.Node.data.Pressure = Item;
        end
    case 'Temperature'
        this.Temperature = Item;
        if this.isDiscretized
            this.Node.data.Temperature = Item;
        end
    case 'h'
        this.h = Item;
        if this.isDiscretized
            this.Node.data.h = Item;
        end
    case 'Name'
        this.customname = Item;
    otherwise
        fprintf(['XXX Environment SET Inteface for ' PropertyName ' is not found XXX\n']);
    end
end

%% Node Management
function resetDiscretization(this)
    this.Node(:) = [];
    this.isDiscretized = false;
end

function discretize(this)
    this.Node = Node.empty;
    this.Node = Node(enumNType.EN,0,0,0,0,Face.empty,Node.empty,this,0);
    this.isDiscretized = true;
    this.Node.data.Dh = 1e8;
    if ~this.isDiscretized
        delete(this.Node);
    end
end

```



```

%         this.Node = Node(enumNType.EN,0,0,0,0,Face.empty,Node.empty,this,0); %#ok<PROP>
%         this.isDiscretized = true;
%     end
end

%% Graphics
function removeFromFigure(this,AxisReference)
if ~isempty(this.GUIObjects)
    children = get(AxisReference,'Children');
    for obj = this.GUIObjects
        if isgraphics(obj)
            for i = length(children):-1:1
                if isgraphics(children(i)) && children(i) == obj
                    children(i).delete;
                    break;
                end
            end
        end
    end
    this.GUIObjects = [];
end
end

function igroup = get.Group(this)
    igroup = Group([],Position(0,0,pi/2),Body.empty);
end
end
end

```

## Frame

The frame is a class that contains an array of positions, a reference to a mechanism and a method used to create a name.

```
classdef Frame < handle
    %FRAME Summary of this class goes here
    % Detailed explanation goes here
    properties (Constant)
        NTheta = 200;
        DecimateFactor = 10;
    end

    properties
        % Kinematic frames can be precalculated
        isKinematic = true;
        % = false; is for free piston designs
        % In these cases the position array simply defines a
        % uniformly spaced position array between the motion
        % extents

        Positions double = []; % no negative positions, pistons should be sketched at minimum, not
        center.
        Mechanism LinRotMechanism; % as MechanicalSystem; % Defines a reference to the mechanism
        output that defines the motion of this frame
        MechanismIndex int8 = 1; % By Default
        CustomName char = [];
    end

    properties (Dependent)
        CurrentPosition;
        name;
    end

    methods
        function name = get.name(this)
            if isvalid(this)
                if isempty(this.CustomName)
                    ii = this.MechanismIndex;
                    name = [this.Mechanism.Type ...
                        ' L= ' num2str(this.Mechanism.Stroke(ii)) ...
                        ' m , P= ' num2str(this.Mechanism.Phase(ii)) ' rad.\n'];
                end
            else
                name = '...';
            end
        end
    end
end

end
```

## Group

The group is a class that contains the following functions:

A constructor.

A function called prior to its deletion, to clean up other objects.

A get / set interface used by the property drop-down editor on the main GUI.

A series of functions for managing the internal lists of other classes.

A series of functions of managing the derived properties of this class.

A function used to calculate the exposed surface of the child bodies.

A function for finding the nearest connection to a point.

A function for rotating the local coordinate to the world coordinates.

A series of functions for displaying the group on the GUI.

```
classdef Group < handle
    %Group Summary of this class goes here
    % Detailed explanation goes here

    properties (Constant)
        ConnectionTolerance = 1e-6; % 0.001 mm plenty small enough for films
        Extension = 1.33;
        MinimumDisplayLength = 0.1;
        MinimumDisplayWidth = 0.1;
        HighlightedColor = [0 1 0];
        NormalColor = [0 0 0];
    end

    properties (Dependent)
        isValid;
        Width;
        Height;
        ValidBorder;
        InvalidBorder;
        isDiscretized;
    end

    properties (Hidden)
        isStateValid logical = false;
        WidthState double;
        HeightState double;
        ValidBorderState Line2DChain;
        InvalidBorderState Line2DChain;
        isStateDiscretized logical = false;
        isEnvironmentCasted logical = false;
    end

    properties
        isChanged logical = true;
        Bodies Body;
    end
end
```

```

Connections Connection;
RelationManagers RelationManager;

GUIObjects;
isActive = true;
name = 'Default Group';
Model Model;
Position Position;

Nodes Node;
Faces Face;
end

methods

%% Constructor Function
function this = Group(inputModel,inputPosition,inputBodies)
    if nargin == 0; return; end
    this.RelationManagers(2) = ...
        RelationManager(this, enumOrient.Horizontal);
    this.RelationManagers(1) = ...
        RelationManager(this, enumOrient.Vertical);
    switch nargin
        case 1
            % Only Model Provided
            this.Model = inputModel;
        case 2
            % A Model and position is provided
            this.Model = inputModel;
            this.Position = inputPosition;
        case 3
            % A Model, position and a bunch of bodies are provided
            this.Model = inputModel;
            this.Position = inputPosition;
            this.addBody(inputBodies);
            for iBody = inputBodies
                this.addConnection(iBody.Connections);
            end
        end
    this.Connections = [Connection(0,enumOrient.Vertical,this) ...
        Connection(0,enumOrient.Horizontal,this)];
    if ~isempty(this.Model)
        this.isActive = true;
        this.Model.switchHighLighting(this);
    end
    this.isChanged = true;
    fprintf('Created Group.\n');
end
function deReference(this)
    iModel = this.Model;
    iModel.isStateDiscretized = false;
    for i = length(iModel.Groups):-1:1
        if iModel.Groups(i) == this
            iModel.Groups(i) = [];
        end
    end
    for iBody = this.Bodies
        iBody.deReference();
    end
    this.Bodies = [];
    for iCon = this.Connections
        iCon.deReference();
    end
    this.Connections = [];
    % Remove any visual remenant
    this.removeFromFigure(gca);
end

%% Get/Set Interface
function Item = get(this,PropertyName)
    switch PropertyName

```

```

        case 'Name'
            Item = this.name;
        case 'Position'
            Item = this.Position;
        case 'Bodies'
            Item = this.Bodies;
        case 'Connections'
            Item = this.Connections;
        case 'Leak Connections'
            Item = this.LeakConnections;
        case 'Relation Managers'
            Item = this.RelationManagers;
        otherwise
            fprintf(['XXX Group GET Inteface for ' PropertyName ...
                ' is not found XXX\n']);
    end
end
function set(this,PropertyName,Item)
    switch PropertyName
        case 'Name'
            this.name = Item;
        otherwise
            fprintf(['XXX Group SET Inteface for ' PropertyName ...
                ' is not found XXX\n']);
    end
end

%% Add Objects
function addBody(this,inputBodies)
    if isrow(inputBodies)
        this.Bodies = [this.Bodies inputBodies];
    else
        this.Bodies = [this.Bodies inputBodies'];
    end
    for iBody = inputBodies
        if isempty(iBody.ID)
            iBody.ID = this.Model.getBodyID();
        end
        this.addConnection(iBody.Connections);
        fprintf(['Added ' iBody.name ' to ' this.name '.\n']);
    end
    this.Bodies = unique(this.Bodies,'rows');
    this.isChanged = true;
    this.fixDatum();
end
function addConnection(this,inputobj)
    for RMan = this.RelationManagers
        RMan.isChanged = true;
    end
    if isrow(inputobj)
        this.Connections = [this.Connections inputobj];
    else
        this.Connections = [this.Connections inputobj'];
    end
    for iCon = inputobj
        if isempty(iCon.ID)
            iCon.ID = this.Model.getConID();
        end
    end
    this.Connections = unique(this.Connections,'rows');
end

%% Clean up and Organization
function cleanUpConnections(this)
    % Ensure Group.Connections Reflects the bodies within it
    keep = false(size(this.Connections));
    keep(1:2) = true;
    for iBody = this.Bodies
        for iCon = iBody.Connections
            found = false;
            for i = 1:length(this.Connections)

```

```

        if iCon == this.Connections(i)
            found = true;
            keep(i) = true;
            break;
        end
    end
    if ~found
        this.addConnection(iCon);
        keep(length(this.Connections)) = true;
    end
end
end
if any(~keep); this.Connections = this.Connections(keep);
end

keep = true(size(this.Connections));
for i = 1:length(this.Connections)
    if ~keep(i)
        iCon = this.Connections(i);
        if isempty(iCon.Bodies); keep(i) = false; end
        for j = i+1:length(this.Connections)
            if ~keep(j)
                jCon = this.Connections(j);
                if iCon.isFunctionallyEqualTo(jCon)
                    for jBody = jCon.Bodies
                        % Replace all references of j with i
                        for k = 1:length(jBody.Connections)
                            if jBody.Connections(k) == jCon
                                jBody.Connections(k) = iCon;
                            end
                        end
                    end
                    iCon.addBody(jBody);
                end
                for iBridge = this.Model.Bridges
                    if iBridge.Connection1 == jCon
                        iBridge.Connection1 = iCon;
                    elseif iBridge.Connection2 == jCon
                        iBridge.Connection2 = iCon;
                    end
                end
                for iLeak = this.Model.LeakConnections
                    if iLeak.Connection1 == jCon
                        iLeak.Connection1 = iCon;
                    elseif iLeak.Connection2 == jCon
                        iLeak.Connection2 = iCon;
                    end
                end
                jCon.removeFromFigure(gca);
            end
        end
    end
end
end
if any(~keep)
    for i = 1:length(this.Connections)
        if ~keep(i)
            this.Connections(i).delete();
        end
    end
    this.Connections = this.Connections(keep);
end

for iCon = this.Connections
    keep = true(size(iCon.Bodies));
    for i = 1:length(iCon.Bodies)
        if ~any(this.Bodies == iCon.Bodies(i))
            keep(i) = false;
        end
    end
end

```

```

        end
    end
    if any(~keep)
        iCon.Bodies = iCon.Bodies(keep);
    end
end

count1 = false;
count2 = false;
for iCon = this.Connections
    if iCon.x == 0
        if iCon.Orient == enumOrient.Vertical; count1 = true;
        elseif iCon.Orient == enumOrient.Horizontal; count2 = true;
        end
    end
end
if ~count1; this.addConnection(Connection(0,enumOrient.Vertical,this)); end
if ~count2; this.addConnection(Connection(0,enumOrient.Horizontal,this)); end
fprintf(['Cleaned up Connections in Group ' this.name '.\n']);
end
function fixDatum(this)
    offset = 0;
    for iCon = this.Connections
        if iCon.Orient == enumOrient.Horizontal && iCon.x < offset
            offset = iCon.x;
        end
    end
    for iCon = this.Connections
        if iCon.Orient == enumOrient.Horizontal
            iCon.x = iCon.x - offset;
        end
    end
end
function isit = isOverlapping(this,TheBody)
    % Determine if TheBody is interfering with any other body
    % Determine if its in the same column
    isit = false;
    for iBody = this.Bodies
        if TheBody.overlaps(iBody)
            isit = true;
            return;
        end
    end
end
end

%% Update on Demand
function change(this)
    this.isChanged = true;
    this.Model.change();
end
function update(this)
    if isempty(this.RelationManagers)
        this.RelationManagers(2) = ...
            RelationManager(this, enumOrient.Horizontal);
        this.RelationManagers(1) = ...
            RelationManager(this, enumOrient.Vertical);
    end
    if length(this.Connections) > 2
        this.cleanUpConnections();
    end
    %% Update isValid
    varb = true;
    % Test to see if any bodies overlap
    for iBody = this.Bodies
        for jBody = this.Bodies
            if iBody ~= jBody
                [Ax1,Ax2,~,~] = iBody.limits(enumOrient.Vertical);
                [Bx1,Bx2,~,~] = jBody.limits(enumOrient.Vertical);
                if (Ax1 < Bx2 || Ax2 > Bx1) % overlap x's
                    [Ay1,Ay2,~,~] = iBody.limits(enumOrient.Horizontal);
                    [By1,By2,~,~] = jBody.limits(enumOrient.Horizontal);
                end
            end
        end
    end
end

```





```

Lines = Line2DChain.empty;
for iBody = this.Bodies
    [~, ~, x1, x2] = iBody.limits(enumOrient.Vertical);
    [~, ~, y1, y2] = iBody.limits(enumOrient.Horizontal);
    if x1 > 0
        Lines(end+4) = Line2DChain(x1, y1, x1, y2);
        Lines(end-1) = Line2DChain(x1, y2, x2, y2);
        Lines(end-2) = Line2DChain(x2, y1, x2, y2);
        Lines(end-3) = Line2DChain(x1, y1, x2, y1);
    else
        Lines(end+3) = Line2DChain(x1, y2, x2, y2);
        Lines(end-1) = Line2DChain(x2, y1, x2, y2);
        Lines(end-2) = Line2DChain(x1, y1, x2, y1);
    end
end
j = 0;
i = 0;
while (i < length(Lines))
    i = i + 1;
    while (j < length(Lines))
        j = j + 1;
        if i ~= j
            [Lines,i,j] = intersects(i,j, Lines);
        end
    end
    j = i + 1;
end

%% Decimate duplicate points and merge
Finished = Line2DChain.empty;
old_n = inf;
while ~isempty(Lines)
    % Combine Step
    n = length(Lines);

    Eliminated = zeros(1,n);
    for i = length(Lines):-1:2
        for j = i-1:-1:1
            if ~Eliminated(j)
                Eliminated(j) = Lines(i).attemptToMerge(Lines(j));
            end
        end
    end

    % Decimate Lines that have been added to others
    for i = length(Lines):-1:1
        if Eliminated(i)
            Lines(i) = [];
        end
    end

    % Pick out finished Lines
    isDone = false(1,length(Lines));
    for i = 1:length(Lines)
        isDone(i) = Lines(i).isFinished;
    end
    Finished = [Finished Lines(isDone)];
    Lines(isDone) = [];

    if old_n == n
        fprintf('XXX Infinite Loop detected, exiting XXX\n');
        Finished = [Finished Lines];
        Lines = [];
    end
    old_n = n;
end

if length(Finished) > 1
    % There can only be one valid border
    % Pick the one with the largest value of x
    maxx = 0;
    for i = 1:length(Finished)
        ix = max(Finished(i).XData);
    end
end

```

```

        if ix > maxx
            index = i;
            maxx = ix;
        end
    end
    this.ValidBorderState = Finished(index);
    this.InvalidBorderState = Finished(Finished~=Finished(index));
    if castToConnections
        this.isEnvironmentCasted = false;
        fprintf(['XXX Environmental Shell generation failed, ' ...
            'there are internal volumes XXX\n']);
    end
else
    this.ValidBorderState = Finished;
    this.InvalidBorderState = Line2DChain.empty;
    if castToConnections
        this.castEnvironmentToConnections();
    end
end
end
%% fprintf(['Group ' this.name ' has been scanned for contact with
surroundings.\n']);
end
end
end
function Valid = get.isValid(this)
    if ischanged
        this.update();
    end
    Valid = this.isStateValid;
end
function Width = get.Width(this)
    if this.isChanged; this.update(); end
    Width = this.WidthState;
end
function Height = get.Height(this)
    if this.isChanged
        this.update();
    end
    Height = this.HeightState;
end
function ValidBorder = get.ValidBorder(this)
    if this.isChanged
        this.update();
    end
    ValidBorder = this.ValidBorderState;
end
function InvalidBorder = get.InvalidBorder(this)
    if this.isChanged
        this.update();
    end
    InvalidBorder = this.InvalidBorderState;
end
function castEnvironmentToConnections(this)
    if ~this.Model.surroundings.isDiscretized
        this.Model.surroundings.discretize();
    end

    for iCon = this.Connections
        % Remove existing environment connections
        k = 1; keep = true(size(iCon.NodeContacts));
        for iNC = iCon.NodeContacts
            if ~isvalid(iNC.Node) || ...
                ~isvalid(iNC.Node.Body) || ...
                isa(iNC.Node.Body, 'Environment')
                keep(k) = false;
            end
            k = k + 1;
        end
        iCon.NodeContacts = iCon.NodeContacts(keep);
    end
end

```

```

% For each segment of pnts
for i = 1:length(this.ValidBorderState.Pnts)-1
    Start = this.ValidBorderState.Pnts(i);
    End = this.ValidBorderState.Pnts(i+1);
    if Start.x ~= End.x % Horizontal
        for iCon = this.Connections
            if iCon.Orient == enumOrient.Horizontal && iCon.x == Start.y
                iCon.addNodeContacts(NodeContact( ...
                    this.Model.surroundings.Node, ...
                    min([Start.x End.x]), ...
                    max([End.x Start.x]), ...
                    enumFType.Environment,iCon));
            end
        end
    else
        for iCon = this.Connections
            if iCon.Orient == enumOrient.Vertical && iCon.x == Start.x
                iCon.addNodeContacts(NodeContact( ...
                    this.Model.surroundings.Node, ...
                    min([Start.y End.y]), ...
                    max([End.y Start.y]), ...
                    enumFType.Environment,iCon));
            end
        end
    end
end
end
function Discretized = get.isDiscretized(this)
    if this.isChanged
        this.update();
    end
    Discretized = this.isStateDiscretized;
end

%% Discretizing
function resetDiscretization(this)
    for iBody = this.Bodies
        iBody.resetDiscretization();
    end
    for iCon = this.Connections
        iCon.resetDiscretization();
    end
    this.Nodes(:) = [];
    this.Faces(:) = [];
    this.isChanged = true;
    this.isStateDiscretized = false;
end
function discretize(this, derefinement_factor)
    this.isStateDiscretized = false;
    this.Nodes(:) = [];
    this.Faces(:) = [];
    nn = 0;
    nf = 0;
    if isempty(this.Bodies)
        this.isStateDiscretized = true;
        return;
    end
    for iBody = this.Bodies
        if ~iBody.isDiscretized
            if nargin == 2
                backup_divisions = iBody.divides;
                if iBody.matl.Phase == enumMaterial.Solid
                    iBody.divides = ceil(iBody.divides*derefinement_factor);
                else
                    if any(iBody.divides ~= 1)
                        if iBody.divides(1) == 1
                            iBody.divides(2) = ...
                                max(2,ceil(iBody.divides(2)*derefinement_factor));
                        elseif iBody.divides(2) == 1
                            iBody.divides(1) = ...
                                max(2,ceil(iBody.divides(2)*derefinement_factor));
                        end
                    end
                end
            end
        end
    end
end

```

```

        end
    end
end
iBody.discretize();
if ~iBody.isDiscretized
    fprintf(['XXX Exited Discretization at Body: ' iBody.name '.XXX\n']);
    if nargin == 2; iBody.divides = backup_divisions; end
    return;
end
keep = true(size(iBody.Nodes));
for i = length(iBody.Nodes):-1:1
    if ~isvalid(iBody.Nodes(i)) || ...
        iBody.Nodes(i).xmin == iBody.Nodes(i).xmax
        keep(i) = false;
    end
end
if any(~keep); iBody.Nodes = iBody.Nodes(keep); end
keep = true(size(iBody.Faces));
for i = length(iBody.Faces):-1:1
    if ~isvalid(iBody.Faces(i)) || isempty(iBody.Faces(i).Nodes)
        keep(i) = false;
    end
end
if any(~keep); iBody.Faces = iBody.Faces(keep); end
if nargin == 2; iBody.divides = backup_divisions; end
end
nn = nn + length(iBody.Nodes);
nf = nf + length(iBody.Faces);
end
% Discretize the surroundings
this.updateBorder(true);

for iCon = this.Connections
    if ~iCon.isDiscretized
        iCon.discretize();
        if ~iCon.isDiscretized
            fprintf(['XXX Exited Discretization at Connection: ' ...
                iCon.name '.XXX\n']);
            return;
        end
    end
    keep = true(size(iCon.Faces));
    for i = length(iCon.Faces):-1:1
        if ~isvalid(iCon.Faces(i))
            keep(i) = false;
        else
            if isempty(iCon.Faces(i).Nodes)
                keep(i) = false;
            end
        end
    end
    if any(~keep)
        iCon.Faces = iCon.Faces(keep);
    end
    nf = nf + length(iCon.Faces);
end
if nn == 0
    return;
end
for i = nn:-1:1; this.Nodes(i) = Node(); end
for i = nf:-1:1; this.Faces(i) = Face(); end
if nf == 0
    nn = 1;
    for iBody = this.Bodies
        this.Nodes(nn:nn-1+length(iBody.Nodes)) = iBody.Nodes;
        nn = nn + length(iBody.Nodes);
    end
else
    nn = 1; nf = 1;
    for iBody = this.Bodies

```

```

        this.Nodes(nn:nn-1+length(iBody.Nodes)) = iBody.Nodes;
        this.Faces(nf:nf-1+length(iBody.Faces)) = iBody.Faces;
        nn = nn + length(iBody.Nodes);
        nf = nf + length(iBody.Faces);
    end
    for iCon = this.Connections
        this.Faces(nf:nf-1+length(iCon.Faces)) = iCon.Faces;
        nf = nf + length(iCon.Faces);
    end
end
this.isStateDiscretized = true;
end

%% Finding things
function Con = FindConnection(this, Pos, Orient, notCon)
    Pos = Pos(1,1:2);
    Pos(1) = Pos(1) - this.Position.x;
    Pos(2) = Pos(2) - this.Position.y;
    Con = [];
    distance = inf;
    if nargin == 4
        if Orient == enumOrient.Vertical
            C = RotMatrix(pi/2 - this.Position.Rot)*Pos';
            for iCon = this.Connections
                if iCon ~= notCon && iCon.Orient == Orient
                    if abs(C(1,1)-iCon.x) < distance
                        distance = abs(C(1,1)-iCon.x);
                        Con = iCon;
                    end
                    if abs(C(1,1)+iCon.x) < distance
                        distance = abs(C(1,1)+iCon.x);
                        Con = iCon;
                    end
                end
            end
        else % Horizontal
            C = RotMatrix(-this.Position.Rot)*Pos';
            for iCon = this.Connections
                if iCon ~= notCon && iCon.Orient == Orient
                    if abs(C(1,1)-iCon.x) < distance
                        distance = abs(C(1,1)-iCon.x);
                        Con = iCon;
                    end
                end
            end
        end
    end
    if nargin == 2
        CV = RotMatrix(pi/2 - this.Position.Rot)*Pos';
        CH = RotMatrix(-this.Position.Rot)*Pos';
        for iCon = this.Connections
            switch iCon.Orient
                case enumOrient.Vertical
                    if abs(CV(1,1)-iCon.x) < distance
                        distance = abs(CV(1,1)-iCon.x);
                        Con = iCon;
                    end
                    if abs(CV(1,1)+iCon.x) < distance
                        distance = abs(CV(1,1)+iCon.x);
                        Con = iCon;
                    end
                case enumOrient.Horizontal
                    if abs(CH(1,1)-iCon.x) < distance
                        distance = abs(CH(1,1)-iCon.x);
                        Con = iCon;
                    end
            end
        end
    end
end
function Pnt = TranslatePnt2D(this, center)

```

```

    Rot = RotMatrix(this.Position.Rot-pi/2);
    x = center.x*Rot(1,1) + center.y*Rot(1,2) + this.Position.x;
    y = center.x*Rot(2,1) + center.y*Rot(2,2) + this.Position.y;
    Pnt = Pnt2D(x,y);
end

%% Graphics
function color = getColor(this)
    if this.isActive; color = Group.HighlightedColor;
    else; color = Group.NormalColor;
    end
end
function removeFromFigure(this,AxisReference)
    if ~isempty(this.GUIObjects)
        children = get(AxisReference,'Children');
        for obj = this.GUIObjects
            if isgraphics(obj)
                for i = length(children):-1:1
                    if isgraphics(children(i)) && children(i) == obj
                        children(i).delete;
                        break;
                    end
                end
            end
        end
        this.GUIObjects = [];
    end
end
function show(this,CODE,AxisReference,Inc,showOptions)
    switch CODE
        case 'all'
            % Show everything in base state
            this.removeFromFigure(AxisReference); % Group and Environmental
            if showOptions(1) % Show Groups
                color = this.getColor();

                % Plot a single line
                % Find horizontal extent of the Group
                VectorLength = this.Width;
                TotalVectorLength = 1.2*max(VectorLength,0.1);
                OffsetRot = this.Position.Rot;
                % Make a template vector
                R = RotMatrix(OffsetRot);
                VStart = [this.Position.x ; this.Position.y ] - ...
                    R * [(TotalVectorLength-VectorLength)/2; 0];
                VEnd = R * [TotalVectorLength; 0] + VStart;

                % Plot line
                this.GUIObjects = line(...
                    [VStart(1) VEnd(1)],...
                    [VStart(2) VEnd(2)],...
                    'Userdata',this,...
                    'Color',color,...
                    'LineWidth',3,...
                    'LineStyle','--',...
                    'HitTest','off');
            end
            if showOptions(2) % Show Bodies
                for iBody = this.Bodies
                    iBody.show(AxisReference);
                end
            else
                for iBody = this.Bodies
                    iBody.removeFromFigure(AxisReference);
                end
            end
            if showOptions(3) % Show Connections
                for iCon = this.Connections
                    iCon.show(AxisReference);
                end
            else

```

```

        for iCon = this.Connections
            iCon.removeFromFigure(AxisReference);
        end
    end
end
if showOptions(7) % Show Environment Connections
    shift = [this.Position.x; this.Position.y];
    rotate = RotMatrix(this.Position.Rot - pi/2);
    % Show the validBorder
    if ~isempty(this.ValidBorder)
        if ~isvalid(this.ValidBorder)
            this.update();
        end
        [XData, YData] = ...
            DistortPositionVectors(...
                this.ValidBorder.XData, this.ValidBorder.YData, ...
                shift, rotate);

        this.GUIObjects(end+1) = line(...
            'XData',XData,...
            'YData',YData,...
            'LineWidth',2,...
            'Color',[0 0 1]);
    end
    % Show the invalidBorder
    if ~isempty(this.InvalidBorder)
        for LineChain = this.InvalidBorder
            [XData, YData] = ...
                DistortPositionVectors(...
                    LineChain.XData, LineChain.YData, ...
                    shift, rotate);
            this.GUIObjects(end+1) = line(...
                'XData',XData,...
                'YData',YData,...
                'LineWidth',2,...
                'LineStyle','--',...
                'Color',[1 0 0]);
        end
    end
end
end
case 'Dynamic'
    if showOptions(2) % Show Bodies
        for iBody = this.Bodies
            if iBody.MovingStatus ~= enumMove.Static
                iBody.show(AxisReference,Inc);
            end
        end
    end
end
case 'Static'
    if showOptions(1) % Show Groups
        color = this.getColor();

        % Plot a single line
        % Find horizontal extent of the Group
        VectorLength = max(...
            [Group.MinimumDisplayLength this.Width]);
        TotalVectorLength = VectorLength*Group.Extension;
        OffsetRot = this.Position.Rot;
        % Make a template vector
        R = RotMatrix(OffsetRot);
        VStart = [this.Position.x ;
            this.Position.y ] - ...
            R * [(TotalVectorLength-VectorLength)/2; 0];
        VEnd = R * [TotalVectorLength; 0] + VStart;

        % Plot line
        this.GUIObjects = line(...
            [VStart(1) VEnd(1)],...
            [VStart(2) VEnd(2)],...
            'Userdata',this,...
            'Color',color,...
            'LineWidth',3,...

```

```

        'LineStyle','--',...
        'HitTest','off');
    end
    if showOptions(2) % Show Bodies
        for iBody = this.Bodies
            if iBody.MovingStatus == enumMove.Static
                iBody.show(AxisReference);
            end
        end
    end
end
end
end
end

function [Lines,i,j] = intersects(i,j,Lines)
if i < 1 || j < 1 || i > length(Lines) || j > length(Lines)
    return;
end
kill_i = false;
kill_j = false;
istart = Lines(i).Pnts(1);
iend = Lines(i).Pnts(end);
jstart = Lines(j).Pnts(1);
jend = Lines(j).Pnts(end);
if all(Lines(i).XData == Lines(j).XData) && ~(istart.y == iend.y)
    % Both Vertical and may overlap
    if iend.y < jstart.y || istart.y > jend.y
        return;
    end
    x = istart.x;
    if istart.y <= jstart.y
        % i starts before j
        if iend.y <= jend.y
            % i is staggered with j
            temp = iend.y;
            if (istart == jstart)
                kill_i = true;
            else
                Lines(i).Pnts(end).y = jstart.y;
            end
            if (jend.y == temp)
                kill_j = true;
            else
                Lines(j).Pnts(1).y = temp;
            end
        else
            % j is within i
            if (iend ~= jend)
                Lines(end+1) = Line2DChain(x,jend.y,x,iend.y);
            end
            if (istart == jstart)
                kill_i = true;
            else
                Lines(i).Pnts(end).y = jstart.y;
            end
            kill_j = true;
        end
    else
        % j starts before i
        if jend.y <= iend.y
            % j is staggered with i
            temp = jend.y;
            if (istart == jstart)
                kill_j = true;
            else
                Lines(j).Pnts(end).y = istart.y;
            end
            if (iend.y == temp)
                kill_i = true;
            else

```



```

        Lines(i).Pnts(1).y = temp;
    end
else
    % i is within j
    if (iend ~= jend)
        Lines(end+1) = Line2DChain(x,iend.y,x,jend.y);
    end
    if (istart == jstart)
        kill_j = true;
    else
        Lines(j).Pnts(end).y = istart.y;
    end
    kill_i = true;
end
end
elseif all(Lines(i).YData == Lines(j).YData) && ~(istart.x == iend.x)
    % Both Horizontal and may overlap
    if iend.x < jstart.x || istart.x > jend.x
        return;
    end
    if istart.x == jstart.x && iend.x == jend.x
        kill_i = true;
        kill_j = true;
    else
        y = istart.y;
        if istart.x <= jstart.x
            % i starts before j
            if iend.x <= jend.x
                % i is staggered with j
                % i --|-|
                % j   |-|--
                temp = iend.x;
                if (istart.x == jstart.x)
                    kill_i = true;
                else
                    Lines(i).Pnts(end).x = jstart.x;
                end
                if (temp == jend.x)
                    kill_j = true;
                else
                    Lines(j).Pnts(1).x = temp;
                end
            else
                % j is within i
                % i -|--|-
                % j   |--|
                kill_j = true;
                if (iend.x ~= jend.x)
                    Lines(end+1) = Line2DChain(jend.x,y,iend.x,y);
                end
                if (istart.x == jstart.x)
                    kill_i = true;
                end
                Lines(i).Pnts(end).x = jstart.x;
            end
        end
    else
        % j starts before i
        if jend.x <= iend.x
            % j is staggered with i
            % i   |-|--
            % j --|-|
            temp = jend.x;
            if (istart == jstart)
                kill_j = true;
            else
                Lines(j).Pnts(end).x = istart.x;
            end
            if (iend.x == temp)
                kill_i = true;
            else
                Lines(i).Pnts(1).x = temp;
            end
        end
    end
end
end

```

```

        end
    else
        % i is within j
        % i |--|
        % j -|--|-
        kill_i = true;
        if (iend ~= jend)
            Lines(end+1) = Line2DChain(iend.x,y,jend.x,y);
        end
        if (istart == jstart)
            kill_j = true;
        else
            Lines(j).Pnts(end).x = istart.x;
        end
    end
end
end
end
if kill_i
    Lines(i) = [];
    if kill_j
        if (i > j)
            Lines(j) = [];
        else
            Lines(j-1) = [];
        end
    end
    j = i;
    i = i - 1;
else
    if kill_j
        Lines(j) = [];
        j = j - 1;
    end
end
end
end

```

## LeakConnection

The leak connection includes the following functions:

A constructor.

A destructor.

A get / set interface used by the property editor in the main GUI.

A discretize function.

A unused show function derived from the bridge component.

```
classdef LeakConnection < handle
    %UNTITLED Summary of this class goes here
    % Detailed explanation goes here

    properties
        isChanged logical = true;
        isDiscretized logical = false;

        LeakFunc function_handle;
        obj1 = []; % Body or Environment
        obj2 = []; % Body or Environment
        Connection1 Connection;
        Connection2 Connection;
        Model Model;
    end

    properties (Hidden)
        customname;
    end

    properties (Dependent)
        name;
        isValid;
    end

    methods
        %% Constructor
        function this = LeakConnection(LeakFunc,obj1,obj2,Connection1,Connection2)
            if ~isa(LeakFunc,'function_handle')
                fprintf('You must define LeakFunc as a function handle, 0 assigned');
                this.LeakFunc = @(P1,P2) 0;
            else
                this.LeakFunc = LeakFunc;
            end
            if isa(obj1,'Body')
                this.obj1 = obj1;
                this.Connection1 = Connection1;
                this.Model = this.obj1.Group.Model;
            else; this.obj1 = obj1;
            end
            if isa(obj2,'Body')
                this.obj2 = obj2;
                this.Connection2 = Connection2;
                this.Model = this.obj2.Group.Model;
            else; this.obj2 = obj2;
            end
        end
    end
end
```

```

function deReference(this)
    iModel = Body.Group.Model;
    for i = length(iModel.LeakConnections):-1:1
        if iModel.LeakConnections(i) == this
            iModel.LeakConnections(i) = LeakConnection.empty;
            break;
        end
    end
    if isa(this.obj1,'Body')
        this.obj1.change();
    end
    if isa(this.obj2,'Body')
        this.obj2.change();
    end
    this.Connection1.deleteNodeContactsFromObj(this.obj1);
    this.Connection1.deleteNodeContactsFromObj(this.obj2);
    this.Connection2.deleteNodeContactsFromObj(this.obj1);
    this.Connection2.deleteNodeContactsFromObj(this.obj2);
    this.Connection1.change();
    this.Connection2.change();
    this.removeFromFigure(gca);
    this.delete();
end

%% Get/Set Interface
function Item = get(this,PropertyName)
    switch PropertyName
        case 'Name'
            Item = this.name;
        case 'Connection 1'
            Item = this.Connection1;
        case 'Connection 2'
            Item = this.Connection2;
        case 'Object 1'
            Item = this.obj1;
        case 'Object 2'
            Item = this.obj2;
        case 'LeakFunc'
            Item = this.LeakFunc;
        otherwise
            fprintf(['XXX LeakConnection GET Inteface for ' PropertyName ' is not found XXX\n']);
    end
end

function set(this,PropertyName,Item)
    switch PropertyName
        case 'Name'
            if Item ~= this.name
                this.customname = Item;
                this.change();
            end
        case 'LeakFunc'
            this.LeakFunc = Item;
            this.change();
        otherwise
            fprintf(['XXX LeakConnection SET Inteface for ' PropertyName ' is not found XXX\n']);
    end
end

function change(this)
    this.isChanged = true;
    this.isDiscretized = false;
    fprintf('XXX Update Function for LeakConnection is Not written. XXX\n');
end

%% Generate Nodes
function leakfaces = getleakface(this)
    if this.isValid
        switch class(this.obj1)
            case 'Body'
                if this.obj1.isDiscretized
                    if this.Connection1 == false

```

```

        n1 = this.obj1.nodeIndex(end);
    else
        n1 = this.obj1.nodeIndex(1);
    end
    else
        leakfaces = [];
        return;
    end
    case 'Environment'
        if this.obj1.isValid
            n1 = this.obj1.nodeIndex;
        end
    end
end
switch class(this.obj2)
    case 'Body'
        if this.obj2.isDiscretized
            if this.Connection1 == false
                n2 = this.obj2.nodeIndex(end);
            else
                n2 = this.obj2.nodeIndex(1);
            end
        else
            leakfaces = [];
            return;
        end
    case 'Environment'
        if this.obj2.isValid
            n2 = this.obj2.nodeIndex;
        end
    end
leakfaces = struct(...
    'Node1',n1,...
    'Node2',n2,...
    'LeakFunc',this.LeakFunc);
this.isDiscretized = true;
end
end

function [Faces] = discretize(this)
    fprintf('XXX Discretize function for LeakConnection is Not written. XXX\n');
    this.isDiscretized = true;
end

%% Testing
function Valid = get.isValid(this)
    Valid = true;
    if isempty(this.obj1)
        Valid = false;
        fprintf(['Missing reference for Leak Connection: ' ...
            this.name '.\n']);
    end
    if isempty(this.obj2)
        Valid = false;
        fprintf(['Missing reference for Leak Connection: ' ...
            this.name '.\n']);
    end
    if (class(this.obj1) == 'Body' && isempty(this.Connection1))
        Valid = false;
        fprintf(['Missing end descriptor for connection 1 of Leak ' ...
            'Connection: ' this.name '.\n']);
    end
    if (class(this.obj2) == 'Body' && isempty(this.Connection2))
        Valid = false;
        fprintf(['Missing end descriptor for connection 2 of Leak ' ...
            'Connection: ' this.name '.\n']);
    end
end

%% Graphics
function removeFromFigure(this,AxisReference)
    if ~isempty(this.GUIObjects)

```

```

children = get(AxisReference,'Children');
for obj = this.GUIObjects
    if isgraphics(obj)
        for i = length(children):-1:1
            if isgraphics(children(i)) && children(i) == obj
                children(i).delete;
                break;
            end
        end
    end
end
this.GUIObjects = [];
end
end
function show(this,AxisReference)
    this.removeFromFigure(AxisReference);
    % Plot a dotted line between the middle of the Connection1's Overlap
    % with Body1 to the middle of Connection2's Overlap with Body2

    % Find P1;
    if isa(this.obj1,'Body')
        Ax = this.Connection1.Group;
        R = RotMatrix(Ax.Position.Rot);
        x = this.Connection1.x;
        switch this.Connection1.Orient
            case enumOrient.Vertical
                [~,~,y1,y2] = this.obj1.limits(enumOrient.Horizontal);
                A = [Ax.Position.x; Ax.Position.y] + ...
                    R*[(y1+y2)/2; x];
                B = [Ax.Position.x; Ax.Position.y] + ...
                    R*[(y1+y2)/2; -x];
            case enumOrient.Horizontal
                [~,~,y1,y2] = this.obj1.limits(enumOrient.Horizontal);
                A = [Ax.Position.x; Ax.Position.y] + ...
                    R*[x; (y1+y2)/2];
                B = [Ax.Position.x; Ax.Position.y] + ...
                    R*[x; -(y1+y2)/2];
        end
    elseif isa(this.obj1,'Environment')
        [x,y] = this.Model.EnvironmentPosition(this.obj1);
        A = [x; y];
        B = [x; y];
    else
        return;
    end

    % Find P2;
    if isa(this.obj2,'Body')
        Ax = this.Connection2.Group;
        R = RotMatrix(Ax.Position.Rot);
        x = this.Connection2.x;
        switch this.Connection1.Orient
            case enumOrient.Vertical
                [~,~,y1,y2] = this.obj2.limits(enumOrient.Vertical);
                C = [Ax.Position.x; Ax.Position.y] + ...
                    R*[(y1+y2)/2; x];
                D = [Ax.Position.x; Ax.Position.y] + ...
                    R*[(y1+y2)/2; -x];
            case enumOrient.Horizontal
                [~,~,y1,y2] = this.obj2.limits(enumOrient.Horizontal);
                C = [Ax.Position.x; Ax.Position.y] + ...
                    R*[x; (y1+y2)/2];
                D = [Ax.Position.x; Ax.Position.y] + ...
                    R*[x; -(y1+y2)/2];
        end
    elseif isa(this.obj2,'Environment')
        [x,y] = this.Model.EnvironmentPosition(this.obj2);
        C = [x; y];
        D = [x; y];
    else
        return;
    end
end

```

```

end

% Find minimum pair
% pair = zeros(2,2);
if Dist4Compare(A,C) < Dist4Compare(B,D)
    pair = [A C];
    dmin = Dist4Compare(A,C);
else
    pair = [B D];
    dmin = Dist4Compare(B,D);
end
if Dist4Compare(A,D) < dmin
    pair = [A D];
    dmin = Dist4Compare(A,D);
end
if Dist4Compare(B,C) < dmin
    pair = [B C];
end

% Find the closest blank space in the model and drag the label there
[x, y, h] = this.Model.findInterSpace(pair);
newpair = [pair(:,1) [x; y+h/2] [x; y-h/2] pair(:,2)];

% Two points in pair are minimum distance
this.GUIObjects = line(...
    newpair(1,:),newpair(2,:),...
    'Color',[0.5 0.5 0.5]);
end
end
end

```

# Matrix

The matrix component is a class that includes the following functionality:

A constructor, destructor.

A get / set interface.

A modify function that walks the user through a series of uniforms specific to a particular matrix type.

A discretize function.

```
classdef Matrix < handle
    %MATRIX Summary of this class goes here
    % Detailed explanation goes here
    properties (Constant)
        GeometrySource = {...
            'Woven Screen';
            'Random Fibre';
            'Packed Sphere';
            'Stacked Foil';
            'Custom Regen';
            'Heat Exchanger'};
    end

    properties
        GeometryEnum enumMatrix;

        matl Material;
        Geometry enumMatrix;

        fFunc_t function_handle;
        fFunc_l function_handle;
        NuFunc_t function_handle;
        NuFunc_l function_handle;
        NkFunc_l function_handle;
        NkFunc_t function_handle;

        Volumetric_HeatCapacity;

        Dh;
        Volumetric_SurfaceArea;
        data struct;
        isFullyLaminar logical = false;
        HasSource logical = false;

        Body Body;
        Nodes Node;
        Faces Face;
    end

    properties (Dependent)
        name;
    end

    methods
        function this = Matrix(Body)
            if nargin == 0
                return;
            end
        end
    end
end
```



```

    if nargin == 1
        this.Body = Body;
    end
    if isempty(this.GeometryEnum); this.assignGeometryEnum(); end
end
function deReference(this)
    if ~isempty(this)
        if ~isempty(this.Body)
            this.Body.Matrix = Matrix.empty;
            this.Body.change();
        end
        delete(this.Nodes);
        delete(this.Faces);
        if isfield(this.data, 'Connection')
            this.data.Connection.change();
        end
        this.delete();
    end
end
function assignGeometryEnum(this)
    this.GeometryEnum(1) = enumMatrix.WovenScreen;
    this.GeometryEnum(2) = enumMatrix.RandomFiber;
    this.GeometryEnum(3) = enumMatrix.PackedSphere;
    this.GeometryEnum(4) = enumMatrix.StackedFoil;
    this.GeometryEnum(5) = enumMatrix.CustomRegen;
    this.GeometryEnum(6) = enumMatrix.HeatExchanger;
end
function item = get(this,PropertyName)
    switch PropertyName
        case 'Material'
            if isempty(this.matl)
                item = Material();
            else
                item = this.matl;
            end
        case 'Laminar Friction Function'
            item = this.fFunc_l;
        case 'Turbulent Friction Function'
            item = this.fFunc_t;
        case 'Laminar Nusselt Function'
            item = this.NuFunc_l;
        case 'Turbulent Nusselt Function'
            item = this.NuFunc_t;
        case 'Laminar Streamwise Cond. Enhancement'
            item = this.NkFunc_l;
        case 'Turbulent Streamwise Cond. Enhancement'
            item = this.NkFunc_t;
        case 'Source Temperature'
            if isfield(this.data, 'SourceTemperature')
                item = this.data.SourceTemperature;
            else
                item = 0;
            end
        end
    end
end
function set(this,PropertyName,item)
    switch PropertyName
        case 'Material'
            this.matl = item;
        case 'Laminar Friction Function'
            this.fFunc_l = item;
        case 'Turbulent Friction Function'
            this.fFunc_t = item;
        case 'Laminar Nusselt Function'
            this.NuFunc_l = item;
        case 'Turbulent Nusselt Function'
            this.NuFunc_t = item;
        case 'Laminar Streamwise Mixing Enhancement'
            this.NkFunc_l = item;
        case 'Turbulent Streamwise Mixing Enhancement'
            this.NkFunc_t = item;
    end
end

```

```

        case 'Source Temperature'
            if isfield(this.data,'SourceTemperature')
                this.data.SourceTemperature = item;
            end
        end
    end
end
function Modify(this)
    % define Material
    if isempty(this.matl); this.matl = Material(); end
    this.matl.Modify();

    % define Geometry
    if ~isempty(this.Geometry)
        for index = 1:length(this.GeometryEnum)
            if this.GeometryEnum(index) == this.Geometry; break; end
        end
    else; index = 1;
    end
    index = listdlg('ListString',this.GeometrySource,...
        'SelectionMode','single',...
        'InitialValue',index);
    if isempty(this.GeometryEnum); this.assignGeometryEnum(); end
    this.Geometry = this.GeometryEnum(index);

    % calculate Properties
    %% Regenerators
    if isempty(this.data); this.data = struct('hasSource',false);
    else; this.data.hasSource = false; end

    switch this.Geometry
        case enumMatrix.WovenScreen
            this.isFullyLaminar = true;
            this.HasSource = false;

            % Assign Default User Inputs, from history or hardcoded values
            op = {'90','0.001'}; % Default Values
            if isfield(this.data,'Porosity'); op{1} = num2str(this.data.Porosity*100); end
            if isfield(this.data,'dw'); op{2} = num2str(this.data.dw); end

            % Get User Inputs
            firstround = true;
            while firstround || ~isStrNumeric(op{1}) || ~isStrNumeric(op{2})
                if firstround; firstround = false;
                else; msgbox('Numeric Values only'); end
                op = inputdlg(...
                    {'Porosity (%)','Wire Diameter (m)',...
                    'Generate a Woven Screen Matrix',[1 35],op);
            end
            this.data.Porosity = str2double(op{1})/100;
            %this.Volumetric_HeatCapacity = (1 -
            this.data.Porosity)*this.matl.HeatCapacity*this.matl.Density;
            this.data.dw = str2double(op{2});
            this.Dh = this.data.dw/(1-this.data.Porosity);

            % Friction Factor
            this.fFunc_l = @(Re) 129./Re+2.91*(Re.^(-0.103));
            this.fFunc_t = this.fFunc_l;

            % Nusselt Number
            this.NuFunc_l = @(Re,Pr) (1+0.99*(this.data.Porosity^1.79)*(Re.*Pr).^0.66);
            this.NuFunc_t = this.NuFunc_l;

            % Streamwise mixing enhancement
            this.NkFunc_l = @(Re,Pr) 1+0.5*(this.data.Porosity^(-2.91))*((Re.*Pr).^0.66);
            this.NkFunc_t = this.NkFunc_l;
        case enumMatrix.RandomFiber
            this.isFullyLaminar = true;
            this.HasSource = false;

            % Assign Default User Inputs, from history or hardcoded values
            op = {'90','0.001'}; % Default Values

```

```

if isfield(this.data,'Porosity'); op{1} = num2str(this.data.Porosity*100); end
if isfield(this.data,'dw'); op{2} = num2str(this.data.dw); end

% Get User Inputs
firstround = true;
while firstround || ~isStrNumeric(op{1}) || ~isStrNumeric(op{2})
    if firstround; firstround = false;
    else; msgbox('Numeric Values only'); end
    op = inputdlg(...
        {'Porosity (%)','Wire Diameter (m)'},...
        'Generate a Random Fibre Matrix',[1 35],op);
end
this.data.Porosity = str2double(op{1})/100;
%this.Volumetric_HeatCapacity = (1-
this.data.Porosity)*this.matl.HeatCapacity*this.matl.Density;
this.data.dw = str2double(op{2});
this.Dh = this.data.dw/(1-this.data.Porosity);
alpha = this.data.Porosity/(1-this.data.Porosity);

% Friction Factor
this.fFunc_l = @(Re) (25.7*alpha+79.8)./Re+...
    (0.146*alpha+3.76)*(Re.^(-0.00283*alpha-0.0748));
this.fFunc_t = this.fFunc_l;

% Nusselt Number
this.NuFunc_l = @(Re,Pr) 1+0.186*alpha*(Re.*Pr).^0.55;
this.NuFunc_t = this.NuFunc_l;

% Streamwise Mixing Enhancement
this.NkFunc_l = @(Re,Pr) (1+(Re.*Pr).^0.55);
this.NkFunc_t = this.NkFunc_l;
case enumMatrix.PackedSphere
this.isFullyLaminar = true;
this.HasSource = false;

% Assign Default User Inputs, from history or hardcoded values
op = {'90','0.001'}; % Default Values
if isfield(this.data,'Porosity'); op{1} = num2str(this.data.Porosity*100); end
if isfield(this.data,'Dp'); op{2} = num2str(this.data.dw); end

% Get User Inputs
firstround = true;Run
while firstround || ~isStrNumeric(op{1}) || ~isStrNumeric(op{2})
    if firstround; firstround = false;
    else; msgbox('Numeric Values only'); end
    op = inputdlg(...
        {'Porosity (%)','Particle Diameter (m)'},...
        'Generate a Stacked Particle Matrix',[1 35],op);
end
this.data.Porosity = str2double(op{1});
%this.Volumetric_HeatCapacity = (1-
this.data.Porosity)*this.matl.HeatCapacity*this.matl.Density;
this.data.Dp = str2double(op{2});
this.Dh = this.data.Dp*this.data.Porosity/(6*(1-this.data.Porosity));

% Friction Factor
this.fFunc_l = @(Re) (157./Re+(5.15*(this.data.Porosity/0.39)^(3.48))*(Re.^-0.137));
this.fFunc_t = @(Re,Pr) (157./Re+(5.15*(this.data.Porosity/0.39)^(3.48))*(Re.^-0.137));

% Nusselt Number
this.NuFunc_l = @(Re,Pr) (1+0.48*(Re.*Pr).^0.65);
this.NuFunc_t = this.NuFunc_l;

% Streamwise Mixing Enhancement
this.NkFunc_l = @(Re,Pr) 1+3*(Re.*Pr).^0.65;
this.NkFunc_t = this.NkFunc_l;
case enumMatrix.StackedFoil
this.isFullyLaminar = false;
this.HasSource = true;

% Assign Default User Inputs, from history or hardcoded values

```

```

op = {'0.00025','0.0001','0.0001'}; % Default Values
if isfield(this.data,'gap'); op{1} = num2str(this.data.gap); end
if isfield(this.data,'dw'); op{2} = num2str(this.data.dw); end
if isfield(this.data,'e'); op{3} = num2str(this.data.e); end

% Get User Inputs
firstround = true;
while firstround || ~isStrNumeric(op{1}) || ...
    ~isStrNumeric(op{2}) || ~isStrNumeric(op{3})
    if firstround; firstround = false;
    else; msgbox('Numeric Values only'); end
    op = inputdlg(...
        {'Gap Width (m)','Sheet Thickness (m)','Sheet Roughness (m)',...
        'Generate a Stacked Foil Matrix',[1 35],op);
end
this.data.gap = str2double(op{1});
this.data.dw = str2double(op{2});
this.data.e = str2double(op{3});
this.Dh = 2*this.data.gap;
this.data.Porosity = this.data.gap/(this.data.gap+this.data.dw);
%this.Volumetric_HeatCapacity = (1-
this.data.Porosity)*this.matl.HeatCapacity*this.matl.Density;

% Friction Factors
% E. Fried, I.E. Idelchik, Flow Resistance: A Design Guide for Engineers,
% Hemisphere, (1989)
this.fFunc_l = @(Re) 96./Re;
this.fFunc_t = @(Re) 0.121*(this.data.e/this.Dh+68./Re).^0.25;

% Nusselt Number
this.NuFunc_l = @(Re) 8.23;
this.NuFunc_t = @(Re,Pr) 0.025*(Re.^0.79).*(Pr.^0.33);

% Streamwise Mixing Enhancement
this.NkFunc_l = @(Re) 1;
this.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).*(Pr);

case enumMatrix.CustomRegen
    this.isFullyLaminar = false;
    this.HasSource = true;
    op = {'0.025','0.80','0.121','-0.25','1000','0.95'}; % Default Values
    if isfield(this.data,'C1'); op{1} = num2str(this.data.C1); end
    if isfield(this.data,'C2'); op{2} = num2str(this.data.C2); end
    if isfield(this.data,'C3'); op{3} = num2str(this.data.C3); end
    if isfield(this.data,'C4'); op{4} = num2str(this.data.C4); end
    if isfield(this.data,'SA_V'); op{5} = num2str(this.data.SA_V); end
    if isfield(this.data,'Porosity'); op{6} = num2str(this.data.Porosity); end

% Get User Inputs
firstround = true;
while firstround || ~isStrNumeric(op{1}) || ...
    ~isStrNumeric(op{2}) || ~isStrNumeric(op{3}) || ...
    ~isStrNumeric(op{4}) || ~isStrNumeric(op{5}) || ...
    ~isStrNumeric(op{6})
    if firstround; firstround = false;
    else; msgbox('Numeric Values only'); end
    op = inputdlg(...
        {'C1','C2','C3','C4','Surface area to volume ratio [m^2/m^3]','Porosity'},...
        'Provide Parameters Nu = C1*Re^C2, F = C3*Re^C4 and other properties',[1 35],op);
end
this.data.C1 = str2double(op{1});
this.data.C2 = str2double(op{2});
this.data.C3 = str2double(op{3});
this.data.C4 = str2double(op{4});
this.data.SA_V = str2double(op{5});
this.data.Porosity = str2double(op{6});
this.Dh = 4/this.data.SA_V;

% Friction Factors
this.fFunc_l = @(Re) this.data.C3.*Re.^this.data.C4;
this.fFunc_t = this.fFunc_l;

```

```

% Nusselt Number
this.NuFunc_l = @(Re,Pr) this.data.C1.*Re.^this.data.C2.*Pr.^0.33333;
this.NuFunc_t = this.NuFunc_l;

% Streamwise Mixing Enhancement
this.NkFunc_l = @(Re) 1;
this.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).*(Pr);
end
%% Heat Exchangers
switch this.Geometry
% Friction Factors
% G.W. Swift, Thermoacoustics: A Unifying Perspective for some Engines
% and Refrigerators, Fourth draft, LA-UR-99-895, 1999
% this.fFunc_l = @(Va)
% this.fFunc_t = @(Re) 0.11*(this.data.e/this.Dh+68/Re)^0.25;
% Nusselt Numbers
% this.NuFunc_l = @(Re)
% this.NuFunc_t = @(Re,Pr) 0.036*(Re^0.8)* (
case enumMatrix.HeatExchanger
%% Determine what Classification
% Fined Surface Type
% Channel Type
% Normal to Tube Type
ChoosingClassification = true;
while (ChoosingClassification)
% Select Heat Exchanger Type from List
Source = {'Fin Enhanced Surface','Fin Connected Channels','Staggered Fin Connected
Tubes','Tube Bank Internal','Custom HX'};
found = false;
if isfield(this.data,'Classification')
for index = 1:length(Source)
if strcmp(this.data.Classification,Source{index})
found = true;
break;
end
end
end
if ~found; index = 1; end
index = listdlg('ListString',Source,'SelectionMode','single','InitialValue',index);
if isempty(this.data); this.data = struct('Classification',Source{index});
else; this.data.Classification = Source{index}; end

% If the User Made a selection
if index > 0
ChoosingClassification = false;
switch index
case 1 % 'Fin Enhanced Surface'
%% Assume straight Flat fins aligned with flow direction
% Assign Values
Source = {'Fin Separation','Fin Thickness','Surface Roughness'};
op = {'0.00318', '0.00318', '0.000001'};
if isfield(this.data,'FinSeparation'); op{1} =
num2str(this.data.FinSeparation); end
if isfield(this.data,'FinThickness'); op{2} = num2str(this.data.FinThickness);
end

if isfield(this.data,'Roughness'); op{3} = num2str(this.data.Roughness); end

DeterminingFinProperties = true;

while (DeterminingFinProperties)
op = inputdlg(Source,'Determine Fin Properties',[1 35],op);
if isempty(op); ChoosingClassification = true; break; end

% If the User inputed the appropriate data
if isStrNumeric(op{1}) && isStrNumeric(op{2}) && isStrNumeric(op{3})
DeterminingFinProperties = false;
this.data.FinSeparation = str2double(op{1});
this.data.FinThickness = str2double(op{2});
this.data.Roughness = str2double(op{3});
lg = this.data.FinSeparation;

```

```

    lth = this.data.FinThickness;
    e = this.data.Roughness;

    % Get the user to select a surface that will be
    % ... enhanced
    Source = cell(1,4);
    i = 1;
    for iCon = this.Body.Connections
        Source{i} = iCon.name; i = i + 1;
    end
    index =
listdlg('ListString',Source,'SelectionMode','single','InitialValue',1);

    % If the User made a selection
    if index > 0
        this.data.Connection = this.Body.Connections(index);
        this.data.Porosity = lg/(lg+lth);

        %% Hydraulic Diameter
        if this.data.Connection.Orient == enumOrient.Vertical
            % Along the wall
            [~,~,xmin,xmax] = this.Body.limits(enumOrient.Vertical);
            this.data.FinLength = xmax - xmin;
        else
            % Along the top or bottom surface
            [ymin,ymax,~,~] = this.Body.limits(enumOrient.Horizontal);
            this.data.FinLength = min(ymax-ymin);
        end
        this.Dh = (4*lg*lth)/(2*lg + 2*lth);

        % Friction Factor
        % ... E. Fried, I.E. Idelchik, Flow Resistance: A Design Guide for
Engineers,
        % ... Hemisphere, (1989)
        x = min(lg/lf,lf/lg);
        C1 = -59.33*x^3+145.6*x^2-125.37*x+96;
        this.fFunc_l = @(Re) C1./Re;
        this.fFunc_t = @(Re) 0.121*(e/this.Dh+68./Re).^0.25;

        % Nusselt Number
        this.NuFunc_l = @(Re) 8.23;
        this.NuFunc_t = @(Re,Pr) 0.025*(Re.^0.79).*(Pr.^0.33);

        % Stramwise Mixing Enhancement
        this.NkFunc_l = @(Re) 1;
        this.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).*(Pr);

    else
        DeterminingFinProperties = true;
    end
end
end
this.data.hasSource = false;
case 2 % 'Fin Connected Channels'
    % Assume the source runs in planes coincident with flow
    % ... direction with fins weaving their way between these
    % ... channels
    % Assign Values
    Source = {'Gap Between Source Channels',...
        'Source Channel Total Width',...
        'Source Channel Wall Thickness',...
        'Surface Roughness'};
    op = {'0.01','0.002','0.0005','0.000001'};
    if isfield(this.data,'gap'); op{1} = num2str(this.data.gap); end
    if isfield(this.data,'ChannelThickness'); op{2} =
num2str(this.data.ChannelThickness); end
    if isfield(this.data,'WallThickness'); op{3} =
num2str(this.data.WallThickness); end
    if isfield(this.data,'Roughness'); op{4} = num2str(this.data.Roughness); end

    DeterminingGeneralChannelGeometry = true;

```

```

while (DeterminingGeneralChannelGeometry)
    op = inputdlg(Source,'Define General Heat Exchanger Geometry',...
        [1 35],op);
    if isempty(op); ChoosingClassification = true; break; end
    if isStrNumeric(op{1}) && isStrNumeric(op{2}) && ...
        isStrNumeric(op{3}) && isStrNumeric(op{4})
        DeterminingGeneralChannelGeometry = false;
        this.data.gap = str2double(op{1});
        this.data.ChannelThickness = str2double(op{2});
        this.data.WallThickness = str2double(op{3});
        this.data.Roughness = str2double(op{4});
        lf = this.data.gap;
        lcth = this.data.ChannelThickness;
        e = this.data.Roughness;
        DeterminingGeometry = true;

        % Pick the fin pattern, straight across or zig-zag
        Source = {'Rectangular','Triangular'};
        index = 1;
        while (DeterminingGeometry)
            index =
listdlg('ListString',Source,'SelectionMode','single','InitialValue',index);

            % If the User made a selection
            if index > 0
                DeterminingGeometry = false;
                this.data.Geometry = Source{index};

                DeterminingGeometryProperties = true;
                % Assign Defaults
                Source = {'Base Width','Fin Thickness'};
                op = {'0.002','0.0002'};
                if isfield(this.data,'BaseWidth'); op{1} =
num2str(this.data.BaseWidth); end
                if isfield(this.data,'FinThickness'); op{2} =
num2str(this.data.FinThickness); end

                while (DeterminingGeometryProperties)
                    op = inputdlg(Source,'Define In Channel Geometry',...
                        [1 35],op);
                    if isempty(op); DeterminingGeometry = true; break; end
                    this.data.BaseWidth = str2double(op{1});
                    this.data.FinThickness = str2double(op{2});
                    lb = this.data.BaseWidth;
                    lth = this.data.FinThickness;
                    switch index
                        case 1 % 'Rectangular'
                            this.data.FinLength = lf;
                            % Porosity
                            this.data.Porosity = ...
                                ((lf - lth)/(lf - lth + lcth))*...
                                (lb/(lb + lth));

                            % Hydraulic Diameter
                            this.Dh = 4*lf*lb/(2*lf + 2*lb);

                            % Friction Factor
                            % E. Fried, I.E. Idelchik, Flow Resistance: A Design Guide for
Engineers,
                            % Hemisphere, (1989)
                            x = min(lf/lb,lb/lf);
                            C1 = -59.33*x^3+145.6*x^2-125.37*x+96;
                            if C1 > 96; C1 = 96; elseif C1 < 56.92; C1 = 56.92; end
                            this.fFunc_l = @(Re) C1./Re;
                            this.fFunc_t = @(Re) 0.25*(e/this.Dh+68./Re).^0.25;

                            % Nusselt Number
                            this.NuFunc_l = @(Re) 8.23;
                            this.NuFunc_t = @(Re,Pr) 0.025*(Re.^0.79).*(Pr.^0.33);

```

```

        % Streamwise Mixing Enhancement
        this.NkFunc_l = @(Re) 1;
        this.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).* (Pr);
    case 2 % 'Triangular'
        A = atan(lb/(2*(this.data.gap-lth)));
        this.data.FinLength = this.data.gap/cos(A);
        lf = this.data.FinLength;
        % Porosity
        lth2 = lth/cos(A);
        this.data.Porosity = ...
            (this.data.gap/(this.data.gap + lcth))*...
            (lb/(lb + lth2));

        % Hydraulic Diameter
        this.Dh = (lb/2)/(1+sqrt(1/(tan(A)^2) + 1));

        % Friction Factor
        C1 = 2.263*A^3 - 7.208*A^2 + 5.738*A + 12;
        this.fFunc_l = @(Re) C1./Re;
        C2 = -0.0184*A^2 + 0.0414*A + 0.0847;
        this.fFunc_t = @(Re) C2*(e/this.Dh+68./Re).^0.25;

        % Nusselt Number
        NuT = 2.66*A^5 - 12.19*A^4 + 21.63*A^3 - 19.9*A^2 + 8.92*A +
0.956;

        this.NuFunc_l = @(Re) NuT;
        TempFunc = @(f,Re,Pr) (f./8).*Re.*Pr./(1.07+12.7*(Pr.^(2/3)-
1).*(f./8).^0.5);

        this.NuFunc_t = @(Re,Pr) TempFunc(this.fFunc_t(Re),Re,Pr);

        % Mixing
        this.NkFunc_l = @(Re) 1;
        this.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).* (Pr); % its similiar
to rectangular flows
    end
        DeterminingGeometryProperties = false;
    end
    else
        DeterminingGeneralChannelGeometry = true;
        break;
    end
end
end
end
this.data.hasSource = true;
case 3 % 'Staggered Fin Connected Tubes'
    Source = {'Spacing Perpendicular to Flow',...
        'Spacing Parallel to Flow',...
        'Fin: Thickness',...
        'Fin: Separation',...
        'Fin: Fin Radial Length (C: Continuous Plate)',...
        'Tube Outer Diameter',...
        'Tube Inner Diameter',...
        'Surface Area Factor',...
        'Heat Transfer Factor'};
    op = {'0.02',...
        '0.02',...
        '0.0002',...
        '0.002',...
        'C',...
        '0.00635',...
        '0.00508',...
        '1',...
        '1'};
    if isfield(this.data,'PerpSpacing'); op{1} = num2str(this.data.PerpSpacing);
end
    if isfield(this.data,'ParaSpacing'); op{2} = num2str(this.data.ParaSpacing);
end
    if isfield(this.data,'FinThickness'); op{3} = num2str(this.data.FinThickness);
end

```



```

        if isfield(this.data,'FinSeparation'); op{4} =
num2str(this.data.FinSeparation); end
        if isfield(this.data,'FinLength'); op{5} = num2str(this.data.FinLength); end
        if isfield(this.data,'do'); op{6} = num2str(this.data.do); end
        if isfield(this.data,'di'); op{7} = num2str(this.data.di); end
        if isfield(this.data,'SurfaceAreaFactor'); op{8} =
num2str(this.data.SurfaceAreaFactor); end
        if isfield(this.data,'HeatTransferFactor'); op{9} =
num2str(this.data.HeatTransferFactor); end

DeterminingNormalToTubeType = true;
while (DeterminingNormalToTubeType)
    op = inputdlg(Source,'Define Finned Tube HX Geometry',...
    [1 35],op);
    if isempty(op); ChoosingClassification = true; break;
    end
    if isStrNumeric(op{1}) && isStrNumeric(op{2}) && ...
        isStrNumeric(op{3}) && isStrNumeric(op{4}) && ...
        (strcmp(op{5},'C') || isStrNumeric(op{5})) && ...
        isStrNumeric(op{6}) && isStrNumeric(op{7}) && ...
        isStrNumeric(op{8})
        % Assign Base Properties
        DeterminingNormalToTubeType = false;
        this.data.PerpSpacing = str2double(op{1});
        this.data.ParaSpacing = str2double(op{2});
        this.data.FinThickness = str2double(op{3});
        this.data.FinSeparation = str2double(op{4});
        if ~strcmp(op{5},'C')
            this.data.FinLength = str2double(op{5});
            lf = this.data.FinLength;
        end
        this.data.do = str2double(op{6});
        this.data.di = str2double(op{7});
        this.data.SurfaceAreaFactor = str2double(op{8});
        this.data.HeatTransferFactor = str2double(op{9});

        lperp = this.data.PerpSpacing;
        lpara = this.data.ParaSpacing;
        lth = this.data.FinThickness;
        lg = this.data.FinSeparation;
        do = this.data.do;

        Ao = lperp*lpara;
        Vo = Ao*(lth + lg);

        % Porosity
        this.data.PercentageTube = (pi/4*do^2)/Ao;
        if isfield(this.data,'FinLength')
            ro = do/2 + lf;
            this.data.Porosity = ...
                (lg + lth)*(Ao - pi*ro^2)/Vo + ... Empty space
                lg*pi*(ro^2 - 0.25*do^2)/Vo; % Finned Areas
        else
            this.data.Porosity = ...
                lg*(Ao - pi/4*do^2)/Vo; % Finned Areas
        end
        this.Dh = do;

        % Nusselt Number
        % Aligned with theta; may be axial, may be radial
        [~,~,xmin,xmax] = this.Body.limits(enumOrient.Vertical);
        [~,~,ymin,ymax] = this.Body.limits(enumOrient.Horizontal);
        if this.Body.divides(1) > 1
            dist = abs(xmax-xmin);
        elseif this.Body.divides(2) > 1
            dist = abs(ymax-ymin);
        else
            if abs(xmax-xmin) > abs(ymax-ymin)
                dist = abs(xmax-xmin);
            else
                dist = abs(ymax-ymin);
            end
        end
    end
end

```

```

end
end
Nr = dist/this.data.ParaSpacing;

if isfield(this.data,'FinLength')
% Individual Finned Tubes - Staggered
lf_do = this.data.FinLength/this.data.do;
if lf_do < 0.09
% Low finned tubes
% if Re -> (895, 713,000)
C1 = 0.255*(2*ro/lg);
this.NuFunc_t = @(Re,Pr) ...
    this.data.HeatTransferFactor*...
    C1*(Re.^0.7).*Pr.^(0.333);
do_Xt = do/lperp;
if do_Xt < 0.25
    fprintf('XXX LateralSpacing/OuterDiameter is out friction
correlation range XXX\n');
end
if round(Nr) < 4
    fprintf('XXX Calculated # of tube rows is out friction correlation
range XXX\n');
end
C2 = 4*1.748*(lf/lg)^0.552*do_Xt^0.599*(do/lpara)^0.1738;
this.fFunc_t = @(Re) C2*Re.^(-0.233);

% Laminar Cases - assume always turbulent
this.NuFunc_l = this.NuFunc_t;
this.fFunc_l = this.fFunc_t;

else % High finned tubes
% if Re -> (1100, 18,000)
s_lf = lg/lf;
s_df = lg/lth;
lf_do = lf/do;
df_do = lth/do;
Xt_do = lperp/do;
if s_lf < 0.13 || s_lf > 0.63
    fprintf('XXX FinSeparation/FinLength is out of Nusselt correlation
range XXX\n');
end
if s_df < 1.01 || s_df > 6.62
    fprintf('XXX FinSeparation/FinThickness is out of Nusselt correlation
range XXX\n');
end
if lf_do > 0.69 || lf_do < 0.09
    fprintf('XXX FinLength/OuterDiameter is out of Nusselt correlation
range XXX\n');
end
if df_do < 0.011 || df_do > 0.15
    fprintf('XXX FinThickness/OuterDiameter is out Nusselt correlation
range XXX\n');
end
if Xt_do < 1.54 || Xt_do > 8.23
    fprintf('XXX LateralSpacing/OuterDiameter is out Nusselt correlation
range XXX\n');
end
if this.data.do < 0.0111 || this.data.do > 0.0409
    fprintf('XXX OuterDiameter is out Nusselt correlation range XXX\n');
end
C1 = 0.134*(s_lf^0.2)*(s_df^0.11);
this.NuFunc_t = @(Re,Pr) ...
    this.data.HeatTransferFactor*...
    C1*(Re.^(0.681)).*(Pr.^(0.333));
C2 = (2*this.Dh/dist)*4*9.465*(Xt_do^-0.927)*(lperp/...
    sqrt(lperp^2 + lpara^2))^0.515;
this.fFunc_t = @(Re) C2*Re.^(-0.316);

% Laminar Cases - assume always turbulent
this.NuFunc_l = this.NuFunc_t;
this.fFunc_l = this.fFunc_t;

```

```

        end
    else
        fprintf('XXX Sorry, the Flat Plan Fins on Staggered Tube Bank is
currently under development\n');
        fprintf('The Best Source is here:
http://thermopedia.com/content/750/\n');
        %{
        % Flat Plain Fins on a Staggered Tube Bank
        C1 = 0.14*...
            (lperp/lpara)^-0.502*...
            (lg/do)^0.031;
        if Nr >= 4
            this.NuFunc_t = @(Re,Pr) C1*Re^0.672*Pr^0.333;
        else
            C2 = C1*0.991*(2.24*(Nr/4)^-0.031)^(-0.607*(4-Nr));
            C3 = 1+(-0.092*0.607*(4-Nr))-0.328;
            this.NuFunc_t = @(Re,Pr) C2*Re^C3*Pr^0.333;
        end
        % Re -> (500, 24,700)
        Xt_do = lperp/do;
        Xl_do = lpara/do;
        s_do = lg/do;
        C4 = SFin/(STube+SFin);
        C5 = (1-C4)*FinVoid;
        if Xt_do < 1.97 || Xt_do > 2.55
            fprintf('XXX LateralSpacing/OuterDiameter is out of Friction
correlation range XXX\n');
        end
        if Xl_do < 1.7 || Xl_do > 2.58
            fprintf('XXX LongitudinalSpacing/OuterDimeter is out of Friction
correlation range XXX\n');
        end
        if s_do < 0.08 || s_do > 0.64
            fprintf('XXX FinSeparation/OuterDimeter is out of Friction correlation
range XXX\n');
        end
        end
        C4a = 4*C4*0.508*(Xt_do)^1.318;
        % Staggered Tube Grid Friction Factor
        % Re -> (300, 15,000)
        C5a = 4*C5*TubeBankFriction(lperp,lpara,do);
        this.fFunc_t = @(Re) C4a*Re^(-0.521) + C5a*((do/this.Dh)*Re)^(-0.18);

        % Laminar Cases - assume always turbulent due to
        % being tripped
        this.NuFunc_l = this.NuFunc_t;
        this.fFunc_l = this.fFunc_t;
        %}
    end

    %% Mixing
    % Axial Conduction Enhancement
    % Taken From woven regenerators
    this.NkFunc_l = @(Re,Pr) 1+0.5*(this.data.Porosity^(-
2.91))*((Re.*Pr).^0.66);
    this.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).*Pr;
    end
end
this.data.hasSource = true;
case 4 % Tube Bank Internal
Source = {'Number of Tubes',...
'Tube Outer Diameter',...
'Tube Inner Diameter'};
op = {'100',...
'0.01',...
'0.008'};
if isfield(this.data,'Number'); op{1} = num2str(this.data.Number); end
if isfield(this.data,'do'); op{2} = num2str(this.data.do); end
if isfield(this.data,'di'); op{3} = num2str(this.data.di); end

DeterminingNormalToTubeType = true;

```

```

while (DeterminingNormalToTubeType)
    op = inputdlg(Source,'Define Tube Bank Internal HX Geometry',...
        [1 35],op);
    if isempty(op); ChoosingClassification = true; break;
    end
    if isStrNumeric(op{1}) && isStrNumeric(op{2}) && ...
        isStrNumeric(op{3})
        DeterminingNormalToTubeType = false;
        this.data.Number = str2double(op{1});
        this.data.do = str2double(op{2});
        this.data.di = str2double(op{3});
        di = this.data.di;

        % Nusselt Number
        % Aligned with theta; may be axial, may be radial
        [~,~,xmin,xmax] = this.Body.limits(enumOrient.Vertical);
        [~,~,ymin,ymax] = this.Body.limits(enumOrient.Horizontal);
        if this.Body.divides(1) > 1
            dist = abs(xmax-xmin);
            A = 2*pi*abs(ymax-ymin)*xmin;
        elseif this.Body.divides(2) > 1
            dist = abs(ymax-ymin);
            A = pi*(xmax^2-xmin^2);
        else
            if abs(xmax-xmin) > abs(ymax-ymin)
                dist = abs(xmax-xmin);
                A = 2*pi*(ymax-ymin)*xmin;
            else
                dist = abs(ymax-ymin);
                A = pi*(xmax^2-xmin^2);
            end
        end
        this.data.Ao = A/this.data.Number;
        this.data.Spacing = sqrt(4*this.data.Ao/sqrt(3));
        this.data.Porosity = ((pi/4)*di^2)/this.data.Ao;
        this.Dh = di;

        Cturb = 0.036*(dist/di)^(-0.055);
        this.NuFunc_t = @(Re,Pr) Cturb*(Re.^0.8).*(Pr.^0.33);
        this.fFunc_t = @(Re) 0.11*(this.Body.Group.Model.roughness/di
+68./Re).^0.25;

        % Laminar Cases - assume always turbulent
        this.NuFunc_l = @(Re) 6;
        this.fFunc_l = @(Re) 64./Re;

        %% Mixing
        % Axial Conduction Enhancement
        % Taken From woven regenerators
        this.NkFunc_l = @(Re) 1;
        this.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).*Pr;
    end
    end
    this.data.hasSource = true;
case 5 % Custom HX
    Source = {'C1','C2','C3','C4','Surface are to gas volume ratio
[m^2/m^3'],'Porosity'};
    op = {'0.020', '0.8', '0.11', '-0.25', '1.5', '0.5'};
    if isfield(this.data,'C1'); op{1} = num2str(this.data.C1); end
    if isfield(this.data,'C2'); op{2} = num2str(this.data.C2); end
    if isfield(this.data,'C3'); op{3} = num2str(this.data.C3); end
    if isfield(this.data,'C4'); op{4} = num2str(this.data.C4); end
    if isfield(this.data,'SA_V'); op{5} = num2str(this.data.SA_V); end
    if isfield(this.data,'Porosity'); op{6} = num2str(this.data.Porosity); end

    DeterminingParameterSet = true;
    while (DeterminingParameterSet)
        op = inputdlg(Source,'Pick Nu = C1*Re^C2*Pr^0.33, F = C3*Re^C4, HX Surface to
Volume Ratio and porosity',...
            [1 35],op);
        if isempty(op); ChoosingClassification = true; break;
    end
end

```

```

end
if isStrNumeric(op{1}) && isStrNumeric(op{2}) && ...
    isStrNumeric(op{3}) && isStrNumeric(op{4}) && ...
    isStrNumeric(op{5}) && isStrNumeric(op{6})
    DeterminingParameterSet = false;
    this.data.C1 = str2double(op{1});
    this.data.C2 = str2double(op{2});
    this.data.C3 = str2double(op{3});
    this.data.C4 = str2double(op{4});
    this.data.SA_V = str2double(op{5});
    this.data.Porosity = str2double(op{6});

    this.Dh = 4/this.data.SA_V;

    % Nusselt Number
    % Aligned with theta; may be axial, may be radial
    this.NuFunc_t = @(Re,Pr) this.data.C1.*(Re.^this.data.C2).*(Pr.^0.33);
    this.fFunc_t = @(Re) this.data.C3.*Re.^this.data.C4;

    % Laminar Cases - assume always turbulent
    this.NuFunc_l = @(Re) 3.66;
    this.fFunc_l = @(Re) 64./Re;

    %% Mixing
    % Axial Conduction Enhancement
    % Taken From woven regenerators
    this.NkFunc_l = @(Re) 1;
    this.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).*Pr;
end
end
this.data.hasSource = true;
end
else
    fprintf('XXX Matrix not finished XXX\n');
    return;
end
end
end
if this.data.hasSource
    op = cell(1);
    if isfield(this.data, 'SourceTemperature')
        op{1} = num2str(this.data.SourceTemperature);
    else
        op{1} = '';
    end
    op = inputdlg('What will be the source Temperature?', 'Define Source Temperature', 1, op);
    if ~isnan(str2double(op{1}))
        this.data.SourceTemperature = str2double(op{1});
    else
        this.data.SourceTemperature = this.Body.Temperature();
    end
end
end
function [nodes, faces] = discretize(this, pnd)
    Np = length(pnd);
    this.Nodes = Node.empty;
    this.Faces = Face.empty;
    k = this.matl.ThermalConductivity;
    % Create Nodes to depth based on biot number
    switch this.Geometry
    case {enumMatrix.WovenScreen, ...
        enumMatrix.RandomFiber, ...
        enumMatrix.PackedSphere, ...
        enumMatrix.StackedFoil, ...
        enumMatrix.CustomRegen}
        this.data.ignore_canister = true;
        ncount = Np + 1;
        fcount = Np + 1;
        switch this.Geometry
        case {enumMatrix.WovenScreen, enumMatrix.RandomFiber}
            % Coefficient = Length of wire per volume * pi * diameter

```

```

    % ... Total_Area/Volume = pi*dw*L / (0.25*pi*dw*dw*L)
    % ... Total_Area/Volume = 1 / (0.25*dw)
    A_V = 4*(1-this.data.Porosity)/this.data.dw;
    % ... Resistance * Total_Area = log(2)/(2*pi*L*k) * pi*dw*L
    % ... = log(2)/(2) * dw/k
    RxA = 0.3466*this.data.dw/k; % ds/(2*k/ln(2));
case enumMatrix.PackedSphere
    % Coefficient = Number of spheres per volume*pi*dw^2
    % ... A/V = (pi*dw^2)/(4*pi*dw^3/(8*3)) = 6/dw
    A_V = 6*(1-this.data.Porosity)/this.data.dw;
    % Resistance * Total_Area = 0.5*dw/(4*pi*k*dw*dw*0.5) *
    % ... pi*dw^2
    % ... = pi*dw^2/(4*pi*k*dw) = dw/(4*k)
    RxA = this.data.dw/(4*k);
case enumMatrix.StackedFoil
    % Coefficient = Area/Volume = 2/(gap + dw)
    A_V = 2/(this.data.gap + this.data.dw);
    % Resistance = L/(k*A) = (dw/2) / (k*2)
    % Total_Area = 2
    RxA = this.data.dw/(2*k);
case enumMatrix.CustomRegen
    A_V = this.data.SA_V;
    dw = 4*(1-this.data.Porosity)/this.data.SA_V;
    RxA = 0.3466*dw/k;
end
this.Nodes(1,Np) = Node();
% Define Nodes
i = 1;
for nd = pnd
    newnd = this.Nodes(i); %
    newnd.Type = enumNType.SN;
    newnd.data = struct('matl',this.matl,...
        'T',nd.data.T,'dT_dU',this.matl.dT_du);
    newnd.xmin = nd.xmin; %
    newnd.xmax = nd.xmax; %
    newnd.ymin = nd.ymin; %
    newnd.ymax = (nd.ymax-nd.ymin).*(1-this.data.Porosity) + nd.ymin; %
    i = i + 1;
end

% Define Faces
this.Faces(1,Np) = Face();
i = 1;
for nd = pnd
    % Create a mixed Face
    newfc = this.Faces(i);
    newfc.Type = enumFType.Mix;
    newfc.Orient = enumOrient.Vertical;
    newfc.isEdge = false;
    newfc.ActiveTimes = true;
    newfc.Nodes = [nd this.Nodes(i)];
    newfc.data = struct('Area',nd.total_vol()*A_V,'R',RxA);
    i = i + 1;
end

case enumMatrix.HeatExchanger
    % Treat the heat exchanger material as a lumped model.
    % Heat Comes from the source, using convection, heat leaves the
    % heat exchanger by convection
    switch this.data.Classification
        case 'Fin Enhanced Surface' % Channel Fins -
            this.data.ignore_canister = false;
            % Source is a connection
            % FinSeparation, FinThickness, Roughness, Porosity, Dh
            N = this.Body.Group.Model.Mesher.HeatExchangerFinDivisions;
            for i = Np*N:-1:1; this.Nodes(i) = Node(); end
            for i = (Np + 1)*N*Np:-1:1; this.Faces(i) = Face(); end
            ncount = 1; fcount = 1;
            Con = this.data.Connection;

            % Find xs and ys from parent nodes

```

```

if length(pnd)>1 || pnd(1).xmin ~= pnd(2).xmin
    % Discretized in X
    if Con.Orient == enumOrient.Vertical
        % Discretized Across the gas path
        xs = zeros(Np+1,1); i = 1;
        for nd = pnd; xs(i) = nd.xmin; i=i+1; end
        xs(end) = pnd(end).xmax;
        ys = linspace(pnd(1).ymin(1),pnd(1).ymax(1),N+1);
    else
        % Discretized with the gas path
        xs = zeros(Np*N+1,1); i = 1; k = 1;
        for nd = pnd
            xs(i:i+N-1) = linspace(pnd(k).xmin,pnd(k).xmax,N);
            i = i + N;
            k = k + 1;
        end
        xs(end) = pnd(end).xmax;
        ys = [pnd(1).ymin(1) pnd(2).ymax(1)];
    end
else
    % Discretized in Y
    if Con.Orient == enumOrient.Horizontal
        % Discretized Across the gas path
        ys = zeros(Np+1,1); i = 1;
        for nd = pnd; ys(i) = nd.ymin(1); i=i+1; end
        ys(end) = pnd(end).ymax(1);
        xs = linspace(pnd(1).xmin,pnd(1).xmax,N+1);
    else
        % Discretized with the gas path
        ys = zeros(Np*N+1,1); i = 1; k = 1;
        for nd = pnd
            ys(i:i+N-1) = linspace(pnd(k).ymin(1),pnd(k).ymax(1),N);
            i = i + N;
            k = k + 1;
        end
        ys(end) = pnd(end).ymax(1);
        xs = [pnd(1).xmin pnd(2).xmax];
    end
end
end

% Declare Nodes
ncount = Np*N;
for j = length(ys)-1:-1:1
    for i = length(xs)-1:-1:1
        this.Nodes(ncount).xmin = xs(i);
        this.Nodes(ncount).xmax = xs(i+1);
        this.Nodes(ncount).ymin = ys(i,:);
        this.Nodes(ncount).ymax = ys(i+1,:);
        this.Nodes(ncount).Type = enumNType.SN;
        index = findmatching(pnd, this.Nodes(ncount));
        this.Nodes(ncount).data = struct(...
            'T',this.Body.Temperature(),...
            'dT_dU',this.matl.dT_du,...
            'matl',this.matl,...
            'ParentNode',index);
        ncount = ncount - 1;
    end
end

% Declare Mixed Faces
% Area / Total Volume
A_V = 2/(this.data.FinThickness + this.data.FinSeparation);
RxA = this.data.FinThickness/(2*k);
for i = 1:length(this.Nodes)
    this.Faces(fcount).Type = enumFType.Mix;
    this.Faces(fcount).Nodes = ...
        [pnd(this.Nodes(i).data.ParentNode) this.Nodes(i)];
    this.Faces(fcount).data = struct(...
        'Area',A_V*this.Nodes(i).total_vol(),...
        'R',RxA,...
        'NuFunc_1',this.NuFunc_1,...

```

```

        'NuFunc_t',this.NuFunc_t);
        this.Faces(fcount).isDynamic = false;
        fcount = fcount + 1;
    end

% Declare Internal Faces
NY = length(ys)-1;
for i = 1:length(this.Nodes)
    nd = this.Nodes(i);
    if i > 1 && this.Nodes(i-1).xmax == nd.xmin
        this.Faces(fcount) = Face([this.Nodes(i-1) nd], ...
            enumFType.Solid,enumOrient.Vertical);
        this.Faces(fcount).data.U = ...
            this.Faces(fcount).data.U*(1-this.data.Porosity);
        fcount = fcount + 1;
    end
    if i > NY
        this.Faces(fcount) = Face([this.Nodes(i-NY) nd], ...
            enumFType.Solid, enumOrient.Horizontal);
        this.Faces(fcount).data.U = ...
            this.Faces(fcount).data.U*(1-this.data.Porosity);
        fcount = fcount + 1;
    end
end

% Modify Gas Node Connections to Selected Connection
for NdCon = Con.NodeContacts
    if NdCon.Node.Body == this.Body
        if isfield(this.data,'Perc')
            this.data.Perc = this.data.Perc * this.data.Porosity;
        else
            this.data.Perc = this.data.Porosity;
        end
    end
end

% Declare Connections to Selected Connection
x = Con.x;
if Con.Orient == enumOrient.Vertical
    for i = 1:length(this.Nodes)
        if this.Nodes(i).xmin == x || this.Nodes(i).xmax == x
            Con.addNodeContacts(...
                NodeContact(this.Nodes(i),...
                    this.Nodes(i).ymin,this.Nodes(i).ymax,...
                    enumFType.Solid,Con));
            Con.NodeContacts(end).data.Perc = (1-this.data.Porosity);
        end
    end
else
    x = x(1);
    for i = 1:length(this.Nodes)
        if this.Nodes(i).ymin(1) == x || ...
            this.Nodes(i).ymax(1) == x
            Con.addNodeContacts(...
                NodeContact(this.Nodes(i),...
                    this.Nodes(i).xmin,this.Nodes(i).xmax,...
                    enumFType.Solid,Con));
            Con.NodeContacts(end).data.Perc = (1-this.data.Porosity);
        end
    end
end

% Modify Gas and Solid Node volume
for i = 1:4
    if Con == this.Body.Connections(i)
        break;
    end
end
switch i
    case {1, 2}
        for fc = this.Faces

```



```

        if fc.Type == enumFType.Mix
            % Get parent node
            if fc.Nodes(1).Type == enumNType.SN
                p = fc.Nodes(2); s = fc.Nodes(1);
            else
                p = fc.Nodes(1); s = fc.Nodes(2);
            end
            anchor = p.ymax;
            s.ymin = anchor + ...
                (s.ymin - anchor)*(1-this.data.Porosity);
            s.ymax = anchor + ...
                (s.ymax - anchor)*(1-this.data.Porosity);
            anchor = p.ymin;
            p.ymax = anchor + ...
                (p.ymax - anchor)*this.data.Porosity;
        end
    end
case {3, 4}
    for fc = this.Faces
        if fc.Type == enumFType.Mix
            % Get parent node
            if fc.Nodes(1).Type == enumNType.SN
                p = fc.Nodes(2); s = fc.Nodes(1);
            else
                p = fc.Nodes(1); s = fc.Nodes(2);
            end
            anchor = p.xmax;
            s.xmin = anchor + ...
                (s.xmin - anchor)*(1-this.data.Porosity);
            s.xmax = anchor + ...
                (s.xmax - anchor)*(1-this.data.Porosity);
            anchor = p.xmin;
            p.xmax = anchor + ...
                (p.xmax - anchor)*this.data.Porosity;
        end
    end
end
case 'Fin Connected Channels'
    this.data.ignore_canister = true;
    % Properties:
    % gap, ChannelThickness, WallThickness, Roughness, BaseWidth,
    % ... FinThickness, FinLength, sourceConvection
    % Make the source node
    N = double(this.Body.Group.Model.Mesher.HeatExchangerFinDivisions);
    for i = (Np + 1)*N + 1:-1:1; this.Nodes(i) = Node(); end
    for i = (Np + 1)*N*Np:-1:1; this.Faces(i) = Face(); end
    ncount = 1; fcount = 1;
    lcth = this.data.ChannelThickness;
    lwth = this.data.WallThickness;
    lg = this.data.gap;
    lth = this.data.FinThickness;
    lb = this.data.BaseWidth;

    % Volume of source / total volume
    SourceV_V = (lcth - lwth)/(lg + lcth);
    % Surface area of source / total volume
    SourceA_V = 2/(lg + lcth);
    % As a multiple of Remaining Volume
    switch this.data.Geometry
        case 'Rectangular'
            % Volume of fin / total volume
            FinV_V = lth*lg/((lg+lcth)*lb);
            FinA_FinV = 2/lth;
            % Linear Distance between fin nodes
            Li = lg/N;
        case 'Triangular'
            lf = this.data.FinLength;
            % Volume of fin / total volume
            FinV_V = lth*lf/((lg+lcth)*lb);
            FinA_FinV = 2/lth;
    end
end

```

```

        % Linear Distance between fin nodes
        Li = lf/N;
    end
    % Exposed surface of source skin / total volume
    SkinA_V = SourceA_V*(lb/(lth+lb));
    % Proportion of channel wall / total volume
    SkinV_V = lwth/(lg + lcth);

    % Define Source Node
    SourceNd = this.Nodes(ncount);
    SourceNd.Type = enumNType.SN;
    SourceNd.data = struct(...
        'matl',Material('Constant Temperature'),...
        'T',this.data.SourceTemperature,...
        'dT_dU',-1);
    [~,~,x1,x2] = this.Body.limits(enumOrient.Vertical);
    [~,~,y1,y2] = this.Body.limits(enumOrient.Horizontal);
    isHorizontal = this.Body.divides(1) > this.Body.divides(2);
    SourceNd.xmin = x1;
    SourceNd.ymin = y1;
    if isHorizontal
        % Discretized along the x direction
        SourceNd.xmax = x2;
        y1 = offsety(SourceV_V,Node(enumNType.SN,x1,x2,y1,y2),y1);
        SourceNd.ymax = y1;
    else
        % Discretized along the y direction
        x1 = offsetx(SourceV_V,Node(enumNType.SN,x1,x2,y1,y2),x1);
        SourceNd.xmax = x1;
        SourceNd.ymax = y2;
    end

    ncount = ncount + 1;
    j = 0;
    backup = x1;
    backup_y1 = y1;
    for nd = pnd
        j = j + 1;
        Vp = nd.total_vol();
        % Define Skin Node
        this.Nodes(ncount) = Node();
        SkinNd = this.Nodes(ncount);
        SkinNd.Type = enumNType.SN;
        SkinNd.data = struct(...
            'matl',this.matl,...
            'T',this.data.SourceTemperature,...
            'dT_dU',this.matl.dT_du);
        if isHorizontal
            SkinNd.xmin = nd.xmin;
            SkinNd.xmax = nd.xmax;
            SkinNd.ymin = backup_y1;
            front = offsety(SkinV_V,nd,backup_y1);
            SkinNd.ymax = front;
        else
            SkinNd.xmin = backup;
            front = offsetx(SkinV_V,nd,backup);
            SkinNd.xmax = front;
            SkinNd.ymin = nd.ymin(1);
            SkinNd.ymax = nd.ymax(1);
        end
        ncount = ncount + 1;

        % Define Conduction between Skin Node and Source
        if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
        newfc = this.Faces(fcount);
        newfc.Type = enumFType.Solid;
        newfc.Nodes = [SkinNd SourceNd];
        USkin2Source = SourceA_V*Vp*k*2/lwth;
        newfc.data = struct('U',USkin2Source);
        newfc.isDynamic = false;
        fcount = fcount + 1;
    end

```

```

% Define Mixed Face to Skin Node
if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
newfc = this.Faces(fcount);
newfc.Type = enumFType.Mix;
newfc.Nodes = [SkinNd nd];
RxA = this.data.WallThickness/(k*2);
newfc.data = struct(...
    'R',RxA,...
    'Area',SkinA_V*Vp,...
    'NuFunc_l',this.NuFunc_l,...
    'NuFunc_t',this.NuFunc_t);
fcount = fcount + 1;

% Define Fin Nodes
Vi = FinV_V*Vp/N;
% Li - Defined Earlier
for i = 1:N
    this.Nodes(ncount) = Node();
    newnd = this.Nodes(ncount);
    newnd.Type = enumNType.SN;
    newnd.data = struct(...
        'matl',this.matl,...
        'T',this.data.SourceTemperature,...
        'dT_dU',this.matl.dT_du);
    if isHorizontal
        newnd.xmin = nd.xmin;
        newnd.xmax = nd.xmax;
        newnd.ymin = front;
        front = offsety(FinV_V/double(N),nd,front);
        newnd.ymax = front;
    else
        newnd.xmin = front;
        front = offsetx(FinV_V/double(N),nd,front);
        newnd.xmax = front;
        newnd.ymin = nd.ymin(1);
        newnd.ymax = nd.ymax(1);
    end
    ncount = ncount + 1;
end

% Define Conduction between 1st Fin Node and Skin
if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
newfc = this.Faces(fcount);
newfc.Type = enumFType.Solid;
newfc.Nodes = [this.Nodes(ncount-N) this.Nodes(ncount-N-1)];
newfc.data = struct(...
    'U',FinV_V*SourceA_V*Vp*k*2/(Li+lwth));
newfc.isDynamic = false;
fcount = fcount + 1;

for i = 1:N-1
    % Define Conduction between Fin Nodes (1-N)
    if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
    newfc = this.Faces(fcount);
    newfc.Type = enumFType.Solid;
    newfc.Nodes = [this.Nodes(ncount-N-1+i) this.Nodes(ncount-N+i)];
    newfc.data = struct('U',FinV_V*SourceA_V*Vp*k/Li);
    newfc.isDynamic = false;
    fcount = fcount + 1;
end

for i = 1:N
    % Define Mixed Faces to Fin Nodes
    if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
    newfc = this.Faces(fcount);
    newfc.Type = enumFType.Mix;
    newfc.Nodes = [this.Nodes(ncount-N-1+i) nd];
    newfc.data = struct(...
        'Area',Vi*FinA_FinV,...
        'R',lth/(4*k),...

```

```

        'NuFunc_1',this.NuFunc_1,...
        'NuFunc_t',this.NuFunc_t);
    newfc.isDynamic = false;
    fcount = fcount + 1;
end

if j > 1
    % Define Downstream Conduction
    for i = 1:N
        if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
        newfc = this.Faces(fcount);
        newfc.Type = enumFType.Solid;
        newfc.Nodes = [this.Nodes(ncount-(N+1)+i) this.Nodes(ncount-2*(N+1)+i)];
        if isHorizontal
            newfc.data = struct('U',FinV_V*2*k*...
                (2*pi*nd.xmin*(nd.ymax(1)-nd.ymin(1)))/...
                (nd.xmax-oldnd.xmin));
        else
            newfc.data = struct('U',FinV_V*2*k*...
                (pi*(nd.xmax^2-nd.xmin^2))/...
                (nd.ymax(1)-oldnd.ymin(1)));
        end
        newfc.isDynamic = false;
        fcount = fcount + 1;
    end
    end
    oldnd = nd;
end
case 'Staggered Fin Connected Tubes'
    this.data.ignore_canister = true;
    % Source is a reservoir with h
    % TubeOrient, PerpSpacing, ParaSpacing, Alignment
    % FinThickness, FinSeparation, FinLength, do, di
    % PercentageTube, PercentageFin
    ncount = 1;
    fcount = 1;
    N = this.Body.Group.Model.Mesher.HeatExchangerFinDivisions;
    if isfield(this.data,'FinLength')
        this.Faces = Face.empty;
        for i = (2*N+2)*Np - 1:-1:1; this.Faces(i) = Face(); end
    else
        this.Faces = Face.empty;
        for i = (2*N+2)*Np - 1:-1:1; this.Faces(i) = Face(); end
        this.Faces((2*N+2)*Np - 1) = Face();
    end
    this.Nodes = Node.empty;
    for i = Np*(N+1) + 1:-1:1; this.Nodes(i) = Node(); end

    % Make the source node
    [~,~,x1,x2] = this.Body.limits(enumOrient.Vertical);
    [~,~,y1,y2] = this.Body.limits(enumOrient.Horizontal);
    lth = this.data.FinThickness;
    lg = this.data.FinSeparation;
    di = this.data.di;
    do = this.data.do;
    VTube_V = this.data.PercentageTube;
    VFin_VFinned = lth/(lth + lg);

    % Percentage of the volume that is the temperature source
    SourceV_V = VTube_V*(di/do)^2;
    SkinV_V = VTube_V - SourceV_V;

    % Surface Area of the tubular source elements
    SourceA_V = 4*SourceV_V/di;
    SkinA_V = (1-VFin_VFinned)*do/di*SourceA_V;
    isHorizontal = this.Body.divides(1) > this.Body.divides(2);

    % Create the Source Node
    SourceNd = this.Nodes(ncount);
    SourceNd.Type = enumNType.SN;
    SourceNd.xmin = x1;

```

```

SourceNd.ymin = y1;
if isHorizontal
    SourceNd.xmax = x2;
    front =
offsety(this.data.SurfaceAreaFactor*SourceV_V,Node(enumNType.SN,x1,x2,y1,y2),y1);
    SourceNd.ymax = front;
else
    front =
offsetx(this.data.SurfaceAreaFactor*SourceV_V,Node(enumNType.SN,x1,x2,y1,y2),x1);
    SourceNd.xmax = front;
    SourceNd.ymax = y2;
end
SourceNd.data = struct(...
    'matl',Material('Constant Temperature'),...
    'T',this.data.SourceTemperature,...
    'dT_du',-1);
ncount = ncount + 1;

% Define Volume, Radii and Surface Area Values
% FinRadii(i) - N + 1 length
% FinVolume(i) - N length
% FinArea(i) - N length
Ao = this.data.PerpSpacing*this.data.ParaSpacing;
if isfield(this.data,'FinLength')
    % FinRadii
    ri = linspace(do,do+this.data.FinLength,N+1);
    % FinVolume
    FinV_V = VFin_VFinned*pi*(ri(2:end).^2 - ri(1:end-1).^2)/Ao;
    % FinArea
    FinA_V = FinV_V*2/lth;
    FinA_V(N) = (FinA_V(N) + 2*pi*ri(N+1)*VFin_VFinned/Ao);
else
    % FinRadii
    Rmax = sqrt(Ao/pi);
    ri = linspace(do,Rmax,N+1);
    % FinVolume
    FinV_V = VFin_VFinned*pi*(ri(2:end)^2 - ri(1:end-1)^2)/Ao;
    % FinArea
    FinA_V = FinV_V*2/lth;
end
RxA_Fin = lth/(4*k);

backup = front;
for nd = pnd
    front = backup;

    Vp = nd.total_vol();
    Vp = Vp(1);
    Lpipe = SourceV_V*Vp/(pi/4*(di^2));

    % Generate Skin Node
    SkinNd = this.Nodes(ncount);
    SkinNd.Type = enumNType.SN;

    if isHorizontal
        SkinNd.xmin = nd.xmin;
        SkinNd.xmax = nd.xmax;
        SkinNd.ymin = front;
        front = offsety(this.data.SurfaceAreaFactor*SkinV_V,...
            nd,front);
        SkinNd.ymax = front;
    else
        SkinNd.xmin = front;
        front = offsetx(this.data.SurfaceAreaFactor*SkinV_V,...
            nd,front);
        SkinNd.xmax = front;
        SkinNd.ymin = nd.ymin;
        SkinNd.ymax = nd.ymax;
    end
    SkinNd.data = struct(...
        'matl',this.matl,...

```

```

    'T',this.data.SourceTemperature,...
    'dT_du',this.matl.dT_du);
ncount = ncount + 1;

% Generate Conduction Face Between Source and Skin
if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
newfc = this.Faces(fcount);
newfc.Type = enumFType.Solid;
newfc.Nodes = [SkinNd SourceNd];
ro = sqrt(di*do/4);
newfc.data = struct('U',...
    this.data.SurfaceAreaFactor*...
    2*pi*Lpipe*k/log(ro/(di/2)));
newfc.isDynamic = false;
fcount = fcount + 1;

% Generate Mixed Face Between Skin and Gas
if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
newfc = this.Faces(fcount);
newfc.Type = enumFType.Mix;
newfc.Nodes = [SkinNd nd];
RxA = log(2*do/(di + do))*(do/2)/k;
newfc.data = struct(...
    'Area',this.data.SurfaceAreaFactor*Vp*SkinA_V,...
    'R',RxA,...
    'NuFunc_l',this.NuFunc_l,...
    'NuFunc_t',this.NuFunc_t);
newfc.isDynamic = false;
fcount = fcount + 1;

for i = 1:N
    % Define Node
    newnd = this.Nodes(ncount);
    newnd.Type = enumNType.SN;
    if isHorizontal
        newnd.xmin = nd.xmin;
        newnd.xmax = nd.xmax;
        newnd.ymin = front;
        front = offsety(this.data.SurfaceAreaFactor*FinV_V(i),nd,front);
        newnd.ymax = front;
    else
        newnd.xmin = front;
        front = offsetx(this.data.SurfaceAreaFactor*FinV_V(i),nd,front);
        newnd.xmax = front;
        newnd.ymin = nd.ymin;
        newnd.ymax = nd.ymax;
    end
    newnd.data = struct(...
        'matl',this.matl,...
        'T',this.Body.Temperature,...
        'dT_du',this.matl.dT_du);
    ncount = ncount + 1;

    % Define Mixed Face
    if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
    newfc = this.Faces(fcount);
    newfc.Type = enumFType.Mix;
    newfc.Nodes = [this.Nodes(ncount-1) nd];
    newfc.data = struct(...
        'Area',this.data.SurfaceAreaFactor*Vp*FinA_V(i),...
        'R',RxA_Fin,...
        'NuFunc_l',this.NuFunc_l,'NuFunc_t',this.NuFunc_t);
    newfc.isDynamic = false;
    fcount = fcount + 1;

    if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
    newfc = this.Faces(fcount);
    newfc.Type = enumFType.Solid;
    newfc.Nodes = [this.Nodes(ncount-1) this.Nodes(1)];
    outside = sqrt(ri(i)*ri(i+1));

```

```

        if i > 1
            % Define Internal Conduction
            inside = sqrt(ri(i)*ri(i-1));
            newfc.data = struct(...
                'U',this.data.SurfaceAreaFactor*VFin_VFinned*...
                2*pi*Lpipe*k/log(outside/inside));
        else
            % Define Skin-Fin Conduction
            inside = sqrt(sqrt(di*do/4)*ri(i));
            newfc.data = struct(...
                'U',this.data.SurfaceAreaFactor*VFin_VFinned*...
                2*pi*Lpipe*k/log(outside/inside));
        end
        newfc.isDynamic = false;
        fcount = fcount + 1;
    end
end
%% Testing Outputs
AreaSum = 0;
CondSum = 0;
for i = 1:length(this.Faces)
    fc = this.Faces(i);
    if ~isempty(fc.data)
        if isfield(fc.data,'Area')
            AreaSum = AreaSum + fc.data.Area;
        end
        if isfield(fc.data,'U')
            CondSum = CondSum + fc.data.U;
        end
    end
end
VolumeSum = 0;
for i = 1:length(this.Nodes)
    nd = this.Nodes(i);
    VolumeSum = VolumeSum + nd.vol();
end
fprintf(['...
    'Area Sum: ' num2str(AreaSum) ...
    ' - Cond Sum: ' num2str(CondSum) ...
    ' - Vol Sum: ' num2str(VolumeSum) '\n']);
case 'Tube Bank Internal'
    this.data.ignore_canister = true;
    % Properties:
    % gap, ChannelThickness, WallThickness, Roughness, BaseWidth,
    % ... FinThickness, FinLength, sourceConvection
    % Make the source node
    for i = Np + 1:-1:1; this.Nodes(i) = Node(); end
    for i = 3*Np - 1:-1:1; this.Faces(i) = Face(); end
    ncount = 1; fcount = 1;
    % Volume of source / total volume
    SourceV_V = ...
        (this.data.Ao - pi/4*this.data.do^2)/this.data.Ao;
    % Surface area of source / total volume
    SourceA_V = pi*this.data.do/this.data.Ao;
    % Surface area of skin / total volume
    SkinA_V = pi*this.data.di/this.data.Ao;
    % Volume of skin / total volume
    SkinV_V = pi/4*...
        (this.data.do^2 - this.data.di^2)/this.data.Ao;

    % Define Source Node
    SourceNd = this.Nodes(ncount);
    SourceNd.Type = enumNType.SN;
    SourceNd.data = struct(...
        'matl',Material('Constant Temperature'),...
        'T',this.data.SourceTemperature,...
        'dT_dU',-1);
    [~,~,x1,x2] = this.Body.limits(enumOrient.Vertical);
    [~,~,y1,y2] = this.Body.limits(enumOrient.Horizontal);
    isHorizontal = this.Body.divides(1) > this.Body.divides(2);

```

```

SourceNd.xmin = x1;
SourceNd.ymin = y1;
if isHorizontal
    % Discretized along the x direction
    SourceNd.xmax = x2;
    front = offsety(SourceV_V,Node(enumNType.SN,x1,x2,y1,y2),y1);
    SourceNd.ymax = front;
else
    % Discretized along the y direction
    front = offsetx(SourceV_V,Node(enumNType.SN,x1,x2,y1,y2),x1);
    SourceNd.xmax = front;
    SourceNd.ymax = y2;
end

%{
% Remove Gas Contacts from overlapping solid neighbours
for iCon = this.Body.Connections
    i = 0;
    keep = true(size(iCon.NodeContacts));
    NodeContactsBackup = cell(length(iCon.NodeContacts),2);
    for iNC = iCon.NodeContacts
        i = i + 1;
        NodeContactsBackup{i,1} = iNC.Start;
        NodeContactsBackup{i,2} = iNC.End;
        if iNC.Node.Body == this.Body
            for iBody = iCon.Bodies
                if iBody ~= this.Body
                    if iBody.matl.Phase == enumMaterial.Solid
                        % Get the connections
                        switch iCon.Orient
                            case enumOrient.Vertical
                                S = iBody.get('Bottom Connection');
                                E = iBody.get('Top Connection');
                                Sx = S.x;
                                Ex = E.x;
                            case enumOrient.Horizontal
                                S = iBody.get('Inner Connection');
                                E = iBody.get('Outer Connection');
                                if ~isempty(S.RefFrame)
                                    Sx = S.x + S.RefFrame.Positions;
                                else
                                    Sx = S.x;
                                end
                                if ~isempty(E.RefFrame)
                                    Ex = E.x + E.RefFrame.Positions;
                                else
                                    Ex = E.x;
                                end
                            end
                        end
                        Mask = NodeContact(Node.empty,Sx,Ex,...
                            enumFType.Gas,this.Body.Connections);
                        keep(i) = Mask.AlignedMask(iNC,-inf,inf);
                        if ~keep(i); break; end
                    end
                end
            end
        end
    end
    if ~keep(i); break; end
end

% Replace them with References to the Source
newNCs = NodeContact.empty;
for i = 1:length(iCon.NodeContacts)
    oNC_Start = NodeContactsBackup{i,1};
    oNC_End = NodeContactsBackup{i,2};
    nNC = iCon.NodeContacts(i);
    if ~keep(i)
        nNC.Start = oNC_Start;
        nNC.End = oNC_End;
        nNC.Node = SourceNd;
        nNC.Type = enumFType.Solid;
    end
end

```



```

else
    d1 = nNC.Start - oNC_Start;
    d2 = oNC_End - nNC.End;
    if any(d1 > 0)
        newNCs(end+1) = NodeContact(SourceNd, ...
            oNC_Start,oNC_Start + d1, enumFType.Solid, ...
            this.Body.Connections);
    end
    if any(d2 > 0)
        newNCs(end+1) = NodeContact(SourceNd, ...
            oNC_End - d2,oNC_End, enumFType.Solid, ...
            this.Body.Connections);
    end
end
end
iCon.addNodeContacts(newNCs);
end
%}

ncount = ncount + 1;
j = 0;
backup = front;
for nd = pnd
    front = backup;
    if isHorizontal
        % Discretized along the x direction
        Lcond = (nd.xmax - nd.xmin);
    else
        % Discretized along the y direction
        Lcond = (nd.ymax - nd.ymin);
    end
    j = j + 1;
    Vp = nd.total_vol();
    % Define Skin Node
    SkinNd = this.Nodes(ncount);
    SkinNd.Type = enumNType.SN;
    SkinNd.data = struct(...
        'matl',this.matl,...
        'T',this.data.SourceTemperature,...
        'dT_dU',this.matl.dT_du);
    if isHorizontal
        SkinNd.xmin = nd.xmin;
        SkinNd.xmax = nd.xmax;
        SkinNd.ymin = front;
        front = offsety(SkinV_V,nd,front);
        SkinNd.ymax = front;
    else
        SkinNd.xmin = front;
        front = offsetx(SkinV_V,nd,front);
        SkinNd.xmax = front;
        SkinNd.ymin = nd.ymin(1);
        SkinNd.ymax = nd.ymax(1);
    end
    ncount = ncount + 1;

    % Define Conduction between Skin Node and Source
    if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
    newfc = this.Faces(fcount);
    newfc.Type = enumFType.Solid;
    newfc.Nodes = [SkinNd SourceNd];
    USkin2Source = ...
        (SourceA_V*Vp)*k*4/abs(this.data.do-this.data.di);
    newfc.data = struct('U',USkin2Source);
    newfc.isDynamic = false;
    fcount = fcount + 1;

    % Define Mixed Face to Skin Node
    th = (this.data.do - this.data.di)/2;
    if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
    newfc = this.Faces(fcount);
    newfc.Type = enumFType.Mix;
    newfc.Nodes = [SkinNd nd];

```

```

RxA = th/(k*2);
newfc.data = struct('R',RxA,'Area',SkinA_V*Vp);
fcount = fcount + 1;

if j > 1
    % Define Downstream Conduction
    if fcount > length(this.Faces); this.Faces(fcount) = Face(); end
    newfc = this.Faces(fcount);
    newfc.Type = enumFType.Solid;
    newfc.Nodes = [SkinNd oldnd];
    newfc.data = struct('U',k*this.data.Number*pi/4*...
        (this.data.do^2 - this.data.di^2)/Lcond);
    newfc.isDynamic = false;
    fcount = fcount + 1;
end
oldnd = SkinNd;
end
case 'Custom HX'
    this.data.ignore_canister = true;
    % Make the source node
    this.Nodes(1) = Node();
    for i = Np:-1:1; this.Faces(i) = Face(); end
    ncount = 1; fcount = 1;
    % Volume of source / total volume
    SourceV_V = 1-this.data.Porosity;
    % Surface area of source / total volume
    SourceA_V = this.data.SA_V/(1-this.data.Porosity);

    % Define Source Node
    SourceNd = this.Nodes(ncount);
    SourceNd.Type = enumNType.SN;
    SourceNd.data = struct(...
        'matl',Material('Constant Temperature'),...
        'T',this.data.SourceTemperature,...
        'dT_dU',-1);
    [~,~,x1,x2] = this.Body.limits(enumOrient.Vertical);
    [~,~,y1,y2] = this.Body.limits(enumOrient.Horizontal);
    isHorizontal = this.Body.divides(1) > this.Body.divides(2);
    SourceNd.xmin = x1;
    SourceNd.ymin = y1;
    if isHorizontal
        % Discretized along the x direction
        SourceNd.xmax = x2;
        front = offsety(SourceV_V,Node(enumNType.SN,x1,x2,y1,y2),y1);
        SourceNd.ymax = front;
    else
        % Discretized along the y direction
        front = offsetx(SourceV_V,Node(enumNType.SN,x1,x2,y1,y2),x1);
        SourceNd.xmax = front;
        SourceNd.ymax = y2;
    end
end

ncount = ncount + 1;
for nd = pnd
    Vp = nd.total_vol();

    % Define Mixed Face to Source Node
    if fcount > length(this.Faces); this.Faces(end+1) = Face(); end
    newfc = this.Faces(fcount);
    newfc.Type = enumFType.Mix;
    newfc.Nodes = [SourceNd nd];
    RxA = 0;
    newfc.data = struct('R',RxA,'Area',SourceA_V*Vp);
    fcount = fcount + 1;
end
end
end
end
% Remove extra elements
if ncount <= length(this.Nodes); this.Nodes(ncount:end) = []; end
if fcount <= length(this.Faces); this.Faces(fcount:end) = []; end
for i = 1:length(this.Nodes)

```

```

        this.Nodes(i).Body = this.Body;
    end
    nodes = this.Nodes;
    faces = this.Faces;
    for nd = nodes
        nd.Body = this.Body;
        nd.data.Porosity = this.data.Porosity;
    end
    % WSNM = this.WSNM;
end
% get Properties
function name = get.name(this)
if ~isempty(this.Geometry)
    for index = 1:length(this.GeometryEnum)
        if this.GeometryEnum(index) == this.Geometry
            break;
        end
    end
else
    name = 'Undefined Matrix';
    return;
end
name = [this.GeometrySource(index) 'Matrix with ' ...
        num2str(round(this.data.Porosity*100,1)) ...
        '% Porosity and Hydraulic Diameter: ' num2str(this.Dh)];
end
end

function newy = offsety(V_V, parent, y)
    newy = ...
        y + V_V*parent.total_vol()/(pi*(parent.xmax^2 - parent.xmin^2));
end

function newx = offsetx(V_V, parent, x)
    newx = ...
        sqrt(V_V*parent.total_vol()/(pi*(parent.ymax(1)-parent.ymin(1)) + x^2));
end

%{
function [xvals,j] = xvals_by_alpha_omega(alpha_omega,dw)
scale = 0.112167*Sqrt(alpha_omega);
% Element size = scale*e^(sqrt(omega/(2*alpha))*x)
% Elements are sized such that there are 10 elements within the
% oscillation penetration depth. With a growth rate cap at
% 1.5 times
% e^sqrt(omega/2*alpha)
expAlphaOmega = exp(sqrt(1/(2*alpha_omega)));
x = 0;
j = 1;
xvals = zeros(1,10);
dx = scale;
while x < dw/2
    % Move Inward
    dx = min([dx*1.5 scale*(expAlphaOmega^x)]);
    x = x + dx;
    j = j + 1;
    xvals(j) = x;
end
xvals(j) = min([xvals(j) dw/2]);
if xvals(j) - xvals(j-1) < 0.1 * (xvals(j-1) - xvals(j-2))
    j = j - 1;
    xvals(j) = dw/2;
    xvals(j+1:end) = [];
elseif j < 10
    xvals(j+1:end) = [];
end
xvals = dw/2 - xvals;
end
%}

```

```

function [U,x1,x2,y1,y2] = InternalNodesVertical(xmin,xmax,ymin,ymax,N,Perc,k,Dir)
x = xmin:(xmax-xmin)/N:xmax;
L = ymax-ymin;
U = zeros(1,N-1);
x1 = zeros(1,N); x2 = x1;
for i = 1:N-1
    U(i) = Perc*k*2*pi*L/log((x(i+2)+x(i+1))/(x(i)+x(i+1)));
end
Vi = Perc*pi*(xmax^2-xmin^2)*L/N;
switch Dir
    case 'In'; xstart = xmin;
    case 'Out'; xstart = sqrt(Vi*N/(pi*L)-xmax^2);
end
for i = 1:N
    xend = sqrt(Vi/(pi*L)+xstart^2);
    x1(i) = xstart;
    x2(i) = xend;
    xstart = xend;
end
y1 = ymin(ones(1,N));
y2 = ymax(ones(1,N));
end

function [U,x1,x2,y1,y2] = InternalNodesHorizontal(xmin,xmax,ymin,ymax,N,Perc,k,Dir)
y = ymin:(ymax-ymin)/N:ymax;
U = zeros(1,N-1);
y1 = zeros(1,N); y2 = y1;
for i = 1:N-1
    U(i) = Perc*k*pi*(xmax^2-xmin^2)*N/(ymax-ymin);
end
d = Perc*(ymax-ymin)/N;
switch Dir
    case 'Down'; ystart = ymin;
    case 'Up'; ystart = ymax - N*d;
end
for i = 1:N
    yend = ymin + d;
    y1(i) = ystart;
    y2(i) = yend;
    ystart = yend;
end
x1 = xmin(ones(1,N));
x2 = xmax(ones(1,N));
y1 = y(1:end-1);
y2 = y(2:end);
end

function GenNodeContact(Connection,Perc,NodeToReference,NodeToFind)
found = false;
for ncontact = Connection.NodeContacts
    if ncontact.Node == NodeToReference
        if ~isempty(ncontact.data) && isfield(ncontact.data,'Perc')
            ncontact.data.Perc = ncontact.data.Perc.*(1-Perc);
        else
            ncontact.data.Perc = this.data.Porostiy;
        end
        found = true;
        break;
    end
end
if found
    NewNodeContact = NodeContact(NodeToFind,ncontact.Start,...
        ncontact.End,enumFType.Solid,Connection);
    NewNodeContact.data.Perc = Perc;
    Connection.addNodeContacts(NewNodeContact);
end
end

function i = findmatching(pnd, nd)
i = 1;
for p = pnd

```

```
if p.xmin <= nd.xmin && p.xmax >= nd.xmax
    if p.ymin(1) <= nd.ymin && p.ymax(1) >= nd.ymax
        return;
    end
end
end
i = i + 1;
end
end
```

# Mesher

The mesher is a class that contains the following functionality:

A get / set interface.

A function for determining if a body is exposed to an air body on a particular side.

```
classdef Mesher < handle
    %UNTITLED2 Summary of this class goes here
    % Detailed explanation goes here

    properties
        % Solid Related
        oscillation_depth_N int16 = 6;
        maximum_thickness double = 0.02;
        maximum_growth double = 1.5;
        HeatExchangerFinDivisions int16 = 3;
        MinSolidTimeStep = 1e-4;

        % Gas Related
        Gas_Entrance_Exit_N int16 = 8;
        Gas_Maximum_Size double = 0.02;
        Gas_Minimum_Size double = 0.003;
        name = 'Universal Mesher';
    end

    methods
        function item = get(this,Property)
            switch Property
                case 'name'
                    item = this.name;
                case 'Nodes through Oscillation Depth'
                    item = this.oscillation_depth_N;
                case 'Maximum Node Thickness'
                    item = this.maximum_thickness;
                case 'Maximum Growth Rate'
                    item = this.maximum_growth;
                case 'Heat Exchanger Fin Divisions'
                    item = this.HeatExchangerFinDivisions;
                case 'Minimum Solid Time Step'
                    item = this.MinSolidTimeStep;
                case 'Gas Entrance Exit N'
                    item = this.Gas_Entrance_Exit_N;
                case 'Gas Maximum Size'
                    item = this.Gas_Maximum_Size;
                case 'Gas Minimum Size'
                    item = this.Gas_Minimum_Size;
                otherwise
                    fprintf(['XXX Mesher GET Inteface for ' Property ' is not found XXX\n']);
            end
        end

        function set(this,Property,item)
            switch Property
                case 'name'
                    this.name = item;
                case 'Nodes through Oscillation Depth'
                    this.oscillation_depth_N = item;
                case 'Maximum Node Thickness'
                    this.maximum_thickness = item;
                case 'Maximum Growth Rate'
                    this.maximum_growth = item;
                case 'Heat Exchanger Fin Divisions'
                    this.HeatExchangerFinDivisions = item;
                case 'Minimum Solid Time Step'
                    this.MinSolidTimeStep = item;
            end
        end
    end
end
```

```

        this.MinSolidTimeStep = item;
    case 'Gas Entrance Exit N'
        this.Gas_Entrance_Exit_N = item;
    case 'Gas Maximum Size'
        this.Gas_Maximum_Size = item;
    case 'Gas Minimum Size'
        this.Gas_Minimum_Size = item;
    otherwise
        fprintf(['XXX Mesher SET Inteface for ' Property ' is not found XXX\n']);
    end
end
function doesit = isInsideRadiiExposed(~,Body)
    [~,~,xmin,~] = Body.limits(enumOrient.Vertical);
    [ymin,ymax,~,~] = Body.limits(enumOrient.Horizontal);
    xdepth = 3*sqrt(2*Body.matl.thermaldiffusivity/...
        Body.Group.Model.engineSpeed) * 1.5;
    for iBody = Body.Group.Bodies
        if iBody ~= Body && iBody.matl.Phase == enumMaterial.Gas
            % Get x limits and see if they could touch
            [~,~,~,xmaxi] = iBody.limits(enumOrient.Vertical);
            if abs(xmin - xmaxi) < xdepth
                % Get y limits and see if they infact overlap at any time
                [ymini,ymaxi,~,~] = iBody.limits(enumOrient.Horizontal);
                if any(~((ymin >= ymaxi) + (ymini >= ymax)))
                    doesit = true;
                    return;
                end
            end
        end
    end
    doesit = false;
end
function doesit = isOutsideRadiiExposed(~,Body)
    [~,~,~,xmax] = Body.limits(enumOrient.Vertical);
    [ymin,ymax,~,~] = Body.limits(enumOrient.Horizontal);
    xdepth = 3*sqrt(2*Body.matl.thermaldiffusivity/...
        Body.Group.Model.engineSpeed) * 1.5;
    for iBody = Body.Group.Bodies
        if iBody ~= Body && iBody.matl.Phase == enumMaterial.Gas
            % Get x limits and see if they could touch
            [~,~,xmini,~] = iBody.limits(enumOrient.Vertical);
            if abs(xmax - xmini) < xdepth
                % Get y limits and see if they infact overlap at any time
                [ymini,ymaxi,~,~] = iBody.limits(enumOrient.Horizontal);
                if any(~((ymin >= ymaxi) + (ymini >= ymax)))
                    doesit = true;
                    return;
                end
            end
        end
    end
    doesit = false;
end
function doesit = isBottomExposed(~,Body)
    [~,~,xmin,xmax] = Body.limits(enumOrient.Vertical);
    xdepth = 3*sqrt(2*Body.matl.thermaldiffusivity/...
        Body.Group.Model.engineSpeed) * 1.5;
    for iBody = Body.Group.Bodies
        if iBody ~= Body && iBody.matl.Phase == enumMaterial.Gas
            % Get x limits and see if they could touch
            [~,~,xmini,xmaxi] = iBody.limits(enumOrient.Vertical);
            if ~(xmax <= xmini) && ~(xmin >= xmaxi)
                % See if they get close to each other
                [ymin,~,~,~] = Body.limits(enumOrient.Horizontal);
                [~,ymaxi,~,~] = iBody.limits(enumOrient.Horizontal);
                if min(abs(ymin-ymaxi)) < xdepth
                    doesit = true;
                    return;
                end
            end
        end
    end
end

```

```

    end
    doesit = false;
end
function doesit = isTopExposed(~,Body)
    [~,~,xmin,xmax] = Body.limits(enumOrient.Vertical);
    xdepth = 3*sqrt(2*Body.matl.thermaldiffusivity/...
        Body.Group.Model.engineSpeed) * 1.5;
    for iBody = Body.Group.Bodies
        if iBody ~= Body && iBody.matl.Phase == enumMaterial.Gas
            % Get x limits and see if they could touch
            [~,~,xmini,xmaxi] = iBody.limits(enumOrient.Vertical);
            if ~(xmax <= xmini) && ~(xmin >= xmaxi)
                % See if they get close to each other
                [~,ymax,~,~] = Body.limits(enumOrient.Horizontal);
                [ymini,~,~,~] = iBody.limits(enumOrient.Horizontal);
                if min(abs(ymax-ymini)) < xdepth
                    doesit = true;
                    return;
                end
            end
        end
    end
    doesit = false;
end
end
end
end
end
end

```



# Model

The model is a class that contains the following functionality:

A constructor and destructor.

A get / set interface.

A set of internal list managers.

A set of functions for triggering updates and checking things in the object.

A discretize function.

A run function with various modes.

A function for assigning snapshots to nodes.

A set of functions which support the interface: the nearest group, nearest body, set of nearest objects, find reference frames, determining if something is in the display window, formatting the display options, getting the active group, model extents in the GUI.

A set of functions managing selection.

A set of functions for showing the model and animating it.

```
classdef Model < handle
%MODEL Summary of this class goes here
% Detailed explanation goes here
properties (Constant)
    ProportionTolerance = 0.02; % 2% error is pretty reasonable
    dt = 0.01; % Seconds ???
    NTheta = 400; % Number of divisions 400 intervals
    dOmega2 = pi^2/2; % (Radians/second)^2 32 intervals between 0->2 Hz
    dAppliedForce = 1; % Newtons ???
    AnimationLength_s = 30;
    AnimationSpeed_rads = pi;
    MaxFourierNumber = 0.25;
end

properties (Dependent)
    ActiveGroup;
    isDefaultModel;
    isDiscretized;
end

properties
    isChanged logical = true;
    Selection = cell(0); % Various Objects
    name = '';
    Groups Group; % A container of Group
    Mesher Mesher; % A container for meshing options
    Bridges Bridge;
```

```

LeakConnections LeakConnection;
RefFrames Frame;
Sensors Sensor;
PVoutputs PVoutput;
SnapShots cell;
NonConnections NonConnection;
CustomMinorLosses CustomMinorLoss;
OptimizationSchemes OptimizationScheme;
CurrentSim Simulation; % Simulations are stored in a named file folder
Converters LinRotMechanism;
AxisReference;
setConditions Environment;
MechanicalSystem MechanicalSystem;
initConditions Environment;
surroundings Environment;
roughness double = 0.000045; % 0.045 mm - Commercial or welded steel

Faces Face;
Nodes Node;

Simulations Simulation;

PressureContacts PressureContact;
ShearContacts ShearContact;

Results Result;
engineTemperature double = 298;
enginePressure double = 101325;
engineSpeed double = 1;

RelationOn = true;
end

properties (Hidden)
StaticGUIObjects = [];
DynamicGUIObjects = [];
GhostGUIObjects = [];
BodyIDIndex = 1;
ConIDIndex = 1;
OptIDIndex = 1;
LRMIDIndex = 1;
% GUI Options
showGroups = true;
showBodies = true;
showBodyGhosts = true;
showConnections = true;
showLeaks = true;
showBridges = true;
showSensors = true;
showInterConnections = false;
showEnvironmentConnections = false;
showNodes = false;
showRelations = false;

% Simulation Options
showLivePV = true;
showPressureAnimation = true;
recordPressure = true;
showTemperatureAnimation = true;
recordTemperature = true;
showVelocityAnimation = true;
recordVelocity = true;
showTurbulenceAnimation = true;
recordTurbulence = true;
showConductionAnimation = true;
recordConductionFlux = true;
showPressureDropAnimation = true;
recordPressureDrop = true;
recordOnlyLastCycle = true;
recordStatistics = true;
outputPath = '';

```

```

warmUpPhaseLength = 0;
animationFrameTime = 0.05;
deRefinementFactorInput = 1;

MaxCourantFinal = 0.025;
MaxFourierFinal = 0.025;
MaxCourantConverging = 0.025;
MaxFourierConverging = 0.025;

% RunTime Options
stopSimulation = false;

isStateDiscretized logical;
isAnimating logical;
end

methods
%% Creating, Reseting, Debugging
function this = Model(AxisReference)
    this.initConditions = Environment();
    this.surroundings = Environment();
    this.Mesher = Mesher();
    this.MechanicalSystem =
MechanicalSystem(this, LinRotMechanism.empty, [], 1, function_handle.empty);
    switch nargin
        case 0
            this.Groups = Group(this, Position(0,0,pi/2)); % The first Group
        case 1
            this.Groups = Group(this, Position(0,0,pi/2)); % The first Group
            this.AxisReference = AxisReference;
        end
    this.isChanged = true;
end
function ID = getBodyID(this)
    % Creates a unique id when called
    this.BodyIDIndex = this.BodyIDIndex + 1;
    ID = this.BodyIDIndex;
end
function ID = getConID(this)
    % Creates a unique id when called
    this.ConIDIndex = this.ConIDIndex + 1;
    ID = this.ConIDIndex;
end
function ID = getOptimizationStudyID(this)
    % Creates a unique id when called
    this.OptIDIndex = this.OptIDIndex + 1;
    ID = this.OptIDIndex;
end
function ID = getLRMID(this)
    % Creates a unique id when called
    this.LRMIDIndex = this.LRMIDIndex + 1;
    ID = this.LRMIDIndex;
end
function Bodies = BodyList(this)
    % Makes a list of all bodies in the Model, spanning multiple groups
    n = 0;
    for iGroup = this.Groups
        n = n + length(iGroup.Bodies);
    end
    Bodies(n) = Body();
    n = 0;
    for iGroup = this.Groups
        Bodies(n+1:n+length(iGroup.Bodies)) = iGroup.Bodies;
        n = n + length(iGroup.Bodies);
    end
end
function resetDiscretization(this)
    % Reset the discretization of the entire model, removing all faces
    % ... and nodes
    for iLRM = this.Converters
        iLRM.Model = this;
    end
end

```

```

        if isempty(iLRM.ID)
            iLRM.ID = this.getLRMID();
        end
    end
    for iGroup = this.Groups
        iGroup.resetDiscretization();
    end
    for iBridge = this.Bridges
        iBridge.resetDiscretization();
    end
    for iLeak = this.LeakConnections
        iLeak.resetDiscretization();
    end
    this.Nodes(:) = [];
    this.Faces(:) = [];
    this.PressureContacts(:) = [];
    this.ShearContacts(:) = [];
    this.CurrentSim(:) = [];
    this.surroundings.resetDiscretization();
    this.change();
end
function dispNodeIndexes(this)
    % Prints to screen the index associated with a node its display
    % ... position
    for iNd = this.Nodes
        pnt = iNd.minCenterCoords;
        text(pnt.x,pnt.y,num2str(iNd.index));
    end
end

%% GET/SET Interface
function Item = get(this,PropertyName)
    switch PropertyName
        case 'Name'
            Item = this.name;
        case 'Groups'
            Item = this.Groups;
        case 'Bridges'
            Item = this.Bridges;
        case 'Leaks'
            Item = this.LeakConnections;
        case 'Sensors'
            Item = this.Sensors;
        case 'PVoutputs'
            Item = this.PVoutputs;
        case 'Lin. to Rot. Mechanisms'
            Item = this.Converters;
        case 'Optimization Studies'
            Item = this.OptimizationSchemes;
        case 'Initial Internal Conditions'
            Item = this.initConditions;
        case 'External Conditions'
            Item = this.surroundings;
        case 'Engine Temperature'
            Item = this.engineTemperature;
        case 'Engine Pressure'
            Item = this.enginePressure;
        case 'Minimum Speed'
            Item = this.engineSpeed;
        case 'SnapShots'
            Item = cell(length(this.SnapShots),1);
            for i = 1:length(this.SnapShots)
                Item{i} = this.SnapShots{i}.Name;
            end
        case 'NonConnections'
            Item = cell(length(this.NonConnections),1);
            for i = 1:length(this.NonConnections)
                Item{i} = this.NonConnections(i).name;
            end
        case 'Custom Minor Losses'
            Item = cell(length(this.CustomMinorLosses),1);

```

```

        for i = 1:length(this.CustomMinorLosses)
            Item{i} = this.CustomMinorLosses(i).name;
        end
    case 'Mesher'
        Item = this.Mesher;
    case 'Mechanical System'
        if isempty(this.MechanicalSystem)
            this.MechanicalSystem = MechanicalSystem(this,...
                LinRotMechanism.empty,[],1,function_handle.empty);
        end
        Item = this.MechanicalSystem;
    case 'Max Courant Final'
        Item = this.MaxCourantFinal;
    case 'Max Fourier Final'
        Item = this.MaxFourierFinal;
    case 'Max Courant Converging'
        Item = this.MaxCourantConverging;
    case 'Max Fourier Converging'
        Item = this.MaxFourierConverging;
    otherwise
        fprintf(['XXX Model GET Inteface for ' PropertyName ' is not found XXX\n']);
        return;
    end
    this.change();
end
function set(this,PropertyName,Item)
    switch PropertyName
        case 'Name'
            this.name = Item;
        case 'Engine Temperature'
            this.engineTemperature = Item;
        case 'Engine Pressure'
            this.enginePressure = Item;
        case 'Minimum Speed'
            this.engineSpeed = Item;
        case 'SnapShots'
            for i = length(Item):-1:1
                if Item(i); this.SnapShots(i) = []; end
            end
        case 'NonConnections'
            for i = length(Item):-1:1
                if Item(i); this.NonConnections(i) = []; end
            end
        case 'Custom Minor Losses'
            for i = length(Item):-1:1
                if Item(i); this.CustomMinorLosses(i) = []; end
            end
        case 'Max Courant Final'
            this.MaxCourantFinal = Item;
        case 'Max Fourier Final'
            this.MaxFourierFinal = Item;
        case 'Max Courant Converging'
            this.MaxCourantConverging = Item;
        case 'Max Fourier Converging'
            this.MaxFourierConverging = Item;
        otherwise
            fprintf(['XXX Model SET Inteface for ' PropertyName ' is not found XXX\n']);
        end
    end
end

%% Adding Elements
function addGroup(this,GroupToAdd)
    if isrow(GroupToAdd)
        this.Groups = [this.Groups GroupToAdd];
    else
        this.Groups = [this.Groups GroupToAdd'];
    end
end
function addBridge(this,BridgeToAdd)
    if isrow(BridgeToAdd)
        this.Bridges = [this.Bridges BridgeToAdd];
    end
end

```

```

else
    this.Bridges = [this.Bridges BridgeToAdd'];
end
end
function addLeakConnection(this,LeakToAdd)
if isrow(LeakToAdd)
    this.LeakConnections = [this.LeakConnections LeakToAdd];
else
    this.LeakConnections = [this.LeakConnections LeakToAdd'];
end
end
function addConverter(this,ConverterToAdd)
LEN = length(this.Converters);
for i = length(ConverterToAdd):-1:1
    this.Converters(LEN+i) = ConverterToAdd(i);
    this.addFrame(ConverterToAdd(i).Frames);
end
end
function addFrame(this,FrameToAdd)
LEN = length(this.RefFrames);
for i = length(FrameToAdd):-1:1
    this.RefFrames(LEN+i) = FrameToAdd(i);
end
end
function addSensor(this,SensorToAdd)
LEN = length(this.Sensors);
for i = length(SensorToAdd):-1:1
    this.Sensors(LEN+i) = SensorToAdd(i);
    if this.isDiscretized
        this.Sensors(LEN+1).update();
    end
end
this.Sensors = unique(this.Sensors);
end
function addPVoutput(this,PVoutputToAdd)
LEN = length(this.PVoutputs);
for i = length(PVoutputToAdd):-1:1
    this.PVoutputs(LEN+i) = PVoutputToAdd(i);
end
this.PVoutputs = unique(this.PVoutputs);
end
function addSnapShot(this,SnapShotToAdd)
this.SnapShots{end+1} = SnapShotToAdd;
end
function addNonConnection(this,NonConnectionToAdd)
LEN = length(this.NonConnections);
for i = length(NonConnectionToAdd):-1:1
    this.NonConnections(LEN+i) = NonConnectionToAdd(i);
    this.resetDiscretization();
end
this.NonConnections = unique(this.NonConnections);
end
function addCustomMinorLoss(this,CustomMinorLossToAdd)
LEN = length(this.CustomMinorLosses);
for i = length(CustomMinorLossToAdd):-1:1
    this.CustomMinorLosses(LEN+i) = CustomMinorLossToAdd(i);
    this.resetDiscretization();
end
this.CustomMinorLosses = unique(this.CustomMinorLosses);
end

%% Update on Demand
function update(this)
this.isStateDiscretized = true;
if any(~isvalid(this.Bridges))
    this.Bridges = this.Bridges(isvalid(this.Bridges));
end
keep = true(size(this.Bridges));
for i = 1:length(this.Bridges)
    for j = i+1:length(this.Bridges)
        if (this.Bridges(i).Body1 == this.Bridges(j).Body1 && ...

```

```

                this.Bridges(i).Body2 == this.Bridges(j).Body2) ...
                || (this.Bridges(i).Body2 == this.Bridges(j).Body1 && ...
                this.Bridges(i).Body1 == this.Bridges(j).Body2)
                keep(j) = false;
            end
        end
    end
end
for i = length(keep):-1:1
    if ~keep(i)
        this.Bridges(i).deReference();
    end
end
end
if any(~isvalid(this.Groups))
    this.Groups = this.Groups(isvalid(this.Groups));
end
if any(~isvalid(this.LeakConnections))
    this.LeakConnections = this.LeakConnections(isvalid(this.LeakConnections));
end
if any(~isvalid(this.Converters))
    this.Converters = this.Converters(isvalid(this.Converters));
end
for iConverter = this.Converters
    if isempty(iConverter.Model); iConverter.Model = this; end
end
if any(~isvalid(this.RefFrames))
    this.RefFrames = this.RefFrames(isvalid(this.RefFrames));
end
if any(~isvalid(this.Sensors))
    this.Sensors = this.Sensors(isvalid(this.Sensors));
end
if any(~isvalid(this.PVoutputs))
    this.PVoutputs = this.PVoutputs(isvalid(this.PVoutputs));
end
for i = length(this.Selection):-1:1
    if ~isvalid(this.Selection{i})
        this.Selection(i) = [];
    end
end
for iGroup = this.Groups
    if ~iGroup.isDiscretized
        this.isStateDiscretized = false;
        break;
    end
end
this.isChanged = false;
end
function change(this)
    % Records that the model is changed and should be updated when
    % ... required
    this.isChanged = true;
    this.Faces(:) = [];
    this.Nodes(:) = [];
    this.CurrentSim(:) = [];
    this.isStateDiscretized = false;
end

%% Process nodes and faces
function isit = get.isDiscretized(this)
    if this.isChanged; this.update(); end
    isit = this.isStateDiscretized;
end
function discretize(this, run)
    this.resetDiscretization();
    for iLinRot = this.Converters
        iLinRot.Populate(iLinRot.Type, iLinRot.originalInput);
    end
    if this.isChanged; this.update(); end

    %% Initializing Meshing
    progressbar('Calculating Surroundings');

```

```

% Test if everything is discretized
this.surroundings.discretize();
if nargin > 1 && isfield('rpm',run)
    this.engineSpeed = run.rpm;
end

if nargin > 1 && isfield('NodeFactor',run) && run.NodeFactor ~= 1
    backup_ODN = this.Mesher.oscillation_depth_N;
    backup_MXT = this.Mesher.maximum_thickness;
    backup_HEFD = this.Mesher.HeatExchangerFinDivisions;
    backup_gas_entrance = this.Mesher.Gas_Entrance_Exit_N;
    backup_gas_maximum_size = this.Mesher.Gas_Maximum_Size;
    backup_gas_minimum_size = this.Mesher.Gas_Minimum_Size;
    this.Mesher.oscillation_depth_N = ...
        ceil(sqrt(double(derefinement_factor))*double(backup_ODN));
    this.Mesher.maximum_thickness = ...
        backup_MXT/sqrt(double(derefinement_factor));
    this.Mesher.Gas_Entrance_Exit_N = ...
        double(run.NodeFactor)*double(backup_gas_entrance);
    this.Mesher.Gas_Maximum_Size = ...
        double(backup_gas_maximum_size)/double(run.NodeFactor);
    this.Mesher.Gas_Minimum_Size = ...
        double(backup_gas_minimum_size)/double(run.NodeFactor);
end

progressbar('Discretizing Bridges');
% Test and Discretize Bridges
for iBridge = this.Bridges
    if ~iBridge.isDiscretized
        iBridge.discretize();
        if ~iBridge.isDiscretized
            fprintf(['XXX Exited Discretization at Bridge Connection: ' ...
                iBridge.name '. XXX\n']);
            if nargin > 1 && isfield('NodeFactor',run) && run.NodeFactor ~= 1
                this.Mesher.oscillation_depth_N = backup_ODN;
                this.Mesher.maximum_thickness = backup_MXT;
                this.Mesher.HeatExchangerFinDivisions = backup_HEFD;
                this.Mesher.maximum_growth = backup_growth;
                this.Mesher.Gas_Entrance_Exit_N = backup_gas_entrance;
                this.Mesher.Gas_Maximum_Size = backup_gas_maximum_size;
                this.Mesher.Gas_Minimum_Size = backup_gas_minimum_size;
                clear backup_ODN;
                clear backup_MXT;
                clear backup_HEFD;
                clear backup_growth;
            end
            return;
        end
    end
end
progressbar('Discretizing Groups');
% Test and Discretize Groups
for iGroup = this.Groups
    if ~iGroup.isDiscretized
        if nargin > 1
            iGroup.discretize(run.NodeFactor);
        end
        if ~iGroup.isDiscretized
            fprintf(['XXX Exited Discretization at Group: ' iGroup.name '. XXX\n']);
            if nargin > 1 && isfield('NodeFactor',run) && run.NodeFactor ~= 1
                this.Mesher.oscillation_depth_N = backup_ODN;
                this.Mesher.maximum_thickness = backup_MXT;
                this.Mesher.HeatExchangerFinDivisions = backup_HEFD;
                this.Mesher.Gas_Entrance_Exit_N = backup_gas_entrance;
                this.Mesher.Gas_Maximum_Size = backup_gas_maximum_size;
                this.Mesher.Gas_Minimum_Size = backup_gas_minimum_size;
                clear backup_ODN;
                clear backup_MXT;
                clear backup_HEFD;
                clear backup_growth;
            end
        end
    end
end

```



```

        return;
    end
end
end
progressbar('Discretizing Leaks');
% Test and Discretize LeakConnections
for iLeak = this.LeakConnections
    if ~iLeak.isDiscretized
        iLeak.getFaces();
        if ~iLeak.isDiscretized
            fprintf(['XXX Exited Discretization at Leak Connection: ' ...
                iLeak.name '. XXX\n']);
            if nargin > 1 && isfield('NodeFactor',run) && run.NodeFactor ~= 1
                this.Mesher.oscillation_depth_N = backup_ODN;
                this.Mesher.maximum_thickness = backup_MXT;
                this.Mesher.HeatExchangerFinDivisions = backup_HEFD;
                this.Mesher.Gas_Entrance_Exit_N = backup_gas_entrance;
                this.Mesher.Gas_Maximum_Size = backup_gas_maximum_size;
                this.Mesher.Gas_Minimum_Size = backup_gas_minimum_size;
                clear backup_ODN;
                clear backup_MXT;
                clear backup_HEFD;
                clear backup_growth;
            end
            return;
        end
    end
end

if nargin > 1 && isfield('NodeFactor',run) && run.NodeFactor ~= 1
    this.Mesher.oscillation_depth_N = backup_ODN;
    this.Mesher.maximum_thickness = backup_MXT;
    this.Mesher.HeatExchangerFinDivisions = backup_HEFD;
    this.Mesher.Gas_Entrance_Exit_N = backup_gas_entrance;
    this.Mesher.Gas_Maximum_Size = backup_gas_maximum_size;
    this.Mesher.Gas_Minimum_Size = backup_gas_minimum_size;
    clear backup_ODN;
    clear backup_MXT;
    clear backup_HEFD;
    clear backup_growth;
end
progressbar('Counting Elements');
% Count the Nodes and Faces
ndequ = 1; % <-- Environment Node
fcequ = 0;
for iLeak = this.LeakConnections
    fcequ = fcequ + length(iLeak.Faces);
end
for iBridge = this.Bridges
    fcequ = fcequ + length(iBridge.Faces);
end
for iGroup = this.Groups
    fcequ = fcequ + length(iGroup.Faces);
    ndequ = ndequ + length(iGroup.Nodes);
end

% Start Simulation Definition
this.CurrentSim = Simulation();
Sim = this.CurrentSim;
Sim.Model = this;

%% Acquiring Nodes and Faces
progressbar('Acquiring Nodes and Faces');

% Collect Nodes and Faces
% Environment
this.Nodes(ndequ) = this.surroundings.Node;
ndequ = ndequ + 1;
% LeakConnections
for iLeak = this.LeakConnections
    len = length(iLeak.Faces);

```

```

    this.Faces(fcequ - len + 1:fcequ) = iLeak.Faces;
    fcequ = fcequ - len;
end
% Bridges
for iBridge = this.Bridges
    len = length(iBridge.Faces);
    this.Faces(fcequ - len + 1:fcequ) = iBridge.Faces;
    fcequ = fcequ - len;
end
% Groups
for iGroup = this.Groups
    this.Faces(fcequ - length(iGroup.Faces) + 1:fcequ) = iGroup.Faces;
    fcequ = fcequ - length(iGroup.Faces);
    this.Nodes(ndequ - length(iGroup.Nodes) + 1:ndequ) = iGroup.Nodes;
    ndequ = ndequ - length(iGroup.Nodes);
end

% Exclude invalid nodes
keep = true(size(this.Nodes));
for i = 1:length(this.Nodes)
    if ~isvalid(this.Nodes(i))
        keep(i) = false;
    end
end
this.Nodes = this.Nodes(keep);

% Exclude Faces with invalid nodes
keep = true(size(this.Faces));
for i = 1:length(this.Faces)
    Fc = this.Faces(i);
    if ~isvalid(Fc) || ...
        ~isvalid(Fc.Nodes(1)) || ...
        ~isvalid(Fc.Nodes(2))
        keep(i) = false;
    end
end
this.Faces = this.Faces(keep);

% Assign Faces/Node Connections to Nodes
for Nd = this.Nodes
    Nd.Faces = Face.empty;
    Nd.Nodes = Node.empty;
end
for Fc = this.Faces
    % Add to each Node
    Fc.Nodes(1).addFace(Fc);
    Fc.Nodes(2).addFace(Fc);
end

% Remove faces that are not allowed
keep2 = true(size(this.NonConnections));
i = 1;
for nonCon = this.NonConnections
    iBody = nonCon.Body1;
    if ~isvalid(iBody)
        keep2(i) = false;
    else
        if isa(nonCon.Body2, 'Environment')
            for nd = iBody.Nodes
                keep = true(size(nd.Faces));
                i = 1;
                for fc = nd.Faces
                    if (fc.Nodes(1).Body == iBody && ...
                        (isa(fc.Nodes(2).Body, 'Environment') && ...
                         fc.Nodes(2).Body == nonCon.Body2)) || ...
                       ((isa(fc.Nodes(1).Body, 'Environment') && ...
                        fc.Nodes(1).Body == nonCon.Body2) && ...
                        fc.Nodes(2).Body == iBody)
                        keep(i) = false;
                    end
                end
                i = i + 1;
            end
        end
    end
end

```

```

        end

        if any(~keep)
            for i = 1:length(keep)
                if ~keep(i)
                    this.Faces(this.Faces==nd.Faces(i)) = [];
                end
            end
            nd.Faces = nd.Faces(keep);
        end
    end
else
    for nd = iBody.Nodes
        keep = true(size(nd.Faces));
        i = 1;
        for fc = nd.Faces
            if (fc.Nodes(1).Body == iBody && ...
                fc.Nodes(2).Body == nonCon.Body2) || ...
                (fc.Nodes(1).Body == nonCon.Body2 && ...
                 fc.Nodes(2).Body == iBody)
                keep(i) = false;
            end
            i = i + 1;
        end

        if any(~keep)
            for i = 1:length(keep)
                if ~keep(i)
                    this.Faces(this.Faces==nd.Faces(i)) = [];
                end
            end
            nd.Faces = nd.Faces(keep);
        end
    end
end
end
i = i + 1;
end
if any(~keep2)
    this.NonConnections = this.NonConnections(keep2);
end

%% Cleaning up solid connections that are too small
progressbar('Cleaning up solid connections that are too small');

% Clean up small nodes near bigger nodes
keep = true(size(this.Faces));
nds2del = Node.empty;
count = 0;
if nargin > 1 && isfield('NodeFactor',run) && run.NodeFactor ~= 1
    for i = 1:length(this.Faces)
        fc = this.Faces(i);
        [should_remove, nd2del, ~] = fc.Nodes(1).combineSolid(fc.Nodes(2),run.NodeFactor);
        if should_remove
            count = count + 1;
            keep(i) = false;
            nds2del(end+1) = nd2del;
        end
    end
else
    for i = 1:length(this.Faces)
        fc = this.Faces(i);
        [should_remove, nd2del, ~] = fc.Nodes(1).combineSolid(fc.Nodes(2),1);
        if should_remove
            count = count + 1;
            keep(i) = false;
            nds2del(end+1) = nd2del;
        end
    end
end
fprintf([num2str(count) ' Node pairs collapsed\n']);

```

```

this.Faces = this.Faces(keep);

% Remove these nodes from where they came
for nd = nds2del
    if ~isempty(nd.Body) && isa(nd.Body, 'Body')
        keep2 = true(size(nd.Body.Nodes));
        for i = 1:length(nd.Body.Nodes)
            if nd.Body.Nodes(i) == nd
                keep2(i) = false;
            end
        end
        nd.Body.Nodes = nd.Body.Nodes(keep2);
    end
end

% Remove the nodes from the bulk list
keep = true(size(this.Nodes));
for nd = nds2del
    keep(this.Nodes==nd) = false;
end
this.Nodes = this.Nodes(keep);
clear nds2del;

%% Assigning Node and Face Indexes
progressbar('Assigning Node and Face Indexes');

% Assign Face/Node indexes to Faces and Nodes
% Determine the amount of Solid, Wall, Environment and Gas Nodes
S_count = 0;
E_count = 0;
for Nd = this.Nodes
    if Nd.Type == enumNType.SN
        S_count = S_count + 1;
    elseif Nd.Type == enumNType.EN
        E_count = E_count + 1;
    end
end
% Arrange, GN, EN, SN
G_count = length(this.Nodes) - E_count - S_count;
fprintf(['Found: ' num2str(G_count) ' Gas Nodes, ' ...
        num2str(E_count) ' Environment Nodes, ' ...
        num2str(S_count) ' Solid Nodes\n']);
E_count = G_count + E_count;
S_count = length(this.Nodes);

S_count_backup = S_count;
E_count_backup = E_count;
G_count_backup = G_count;

for Nd = this.Nodes
    if Nd.Type == enumNType.SN
        Nd.index = S_count;
        S_count = S_count - 1;
    elseif Nd.Type == enumNType.EN
        Nd.index = E_count;
        E_count = E_count - 1;
    else
        Nd.index = G_count;
        G_count = G_count - 1;
    end
end

% Exclude faces with nodes with no index
keep = true(size(this.Faces));
for i = 1:length(this.Faces)
    Fc = this.Faces(i);
    if isempty(Fc.Nodes(1).index) || isempty(Fc.Nodes(2).index) || ...
        Fc.Nodes(1).index < 1 || Fc.Nodes(2).index < 1
        keep(i) = false;
    end
end
end

```

```

this.Faces = this.Faces(keep);

%% Assessing Connections
progressbar('Assessing Connections');

% Orient them such that the node closer to 0,0 is
% ... listed first
for Fc = this.Faces
    if Fc.Type == enumFType.Gas || Fc.Type == enumFType.MatrixTransition
        if isempty(Fc.Connection)
            O = Fc.Orient;
        else
            O = Fc.Connection.Orient;
        end
        switch O
            case enumOrient.Vertical
                if Fc.Nodes(1).xmin(1) > Fc.Nodes(2).xmin(1)
                    % Swap Nodes
                    TempNode = Fc.Nodes(1);
                    Fc.Nodes(1) = Fc.Nodes(2);
                    Fc.Nodes(2) = TempNode;
                end
            case enumOrient.Horizontal
                if Fc.Nodes(1).ymin(1) > Fc.Nodes(2).ymin(1)
                    % Swap Nodes
                    TempNode = Fc.Nodes(1);
                    Fc.Nodes(1) = Fc.Nodes(2);
                    Fc.Nodes(2) = TempNode;
                end
            end
        end
    end
end

% For Gas-Gas Faces that have a connection (from node contacts), determine K
% Determine if applicable
isSubject = false(length(this.Faces),1);
for fcequ = 1:length(this.Faces)
    % Gather all Gas-Gas faces that are on possible discontinuities
    isSubject(fcequ) = ...
        ((this.Faces(fcequ).Type == enumFType.Gas || ...
        this.Faces(fcequ).Type == enumFType.MatrixTransition) && ...
        ~isempty(this.Faces(fcequ).Connection)) && ...
        ~isfield(this.Faces(fcequ).data,'K12');
end

% Group based on common connection & body
subSet = this.Faces(isSubject);
isExcluded = false(length(subSet),1);
n = 1;
for i = 1:length(subSet)
    if ~isExcluded(i)
        isSubject = false(length(subSet),1);
        isSubject(i) = true;
        for j = 1:length(subSet)
            if ~isExcluded(j) && ...
                subSet(i).Connection == subSet(j).Connection
                % The two Faces are very likely somehow adjacent
                isSubject(j) = true;
            end
        end
    end
end
% should have acquired a subSet with a common connection
% Mark off Exclusion
isExcluded(isSubject) = true;
subsubSet = subSet(isSubject);

% So we have grabbed a subset of the select nodes that share a
% ... connection with element i
index = zeros(length(subsubSet),1);

% Determine if they are part of some adjacent chain by going

```

```

% ... through each combination and passing a index between
% ... connected elements.
for k = 1:length(subsubSet)
    for x = k+1:length(subsubSet)
        % Test if they link to the same nodes or the linked nodes are
        % touching, for both sides of the face.
        if ((subsubSet(k).Nodes(1) == subsubSet(x).Nodes(1) || ...
            subsubSet(k).Nodes(1).isTouching(subsubSet(x).Nodes(1))) && ...
            (subsubSet(k).Nodes(2) == subsubSet(x).Nodes(2) || ...
            subsubSet(k).Nodes(2).isTouching(subsubSet(x).Nodes(2))))
            if index(k) == 0
                if index(x) == 0
                    index(k) = n;
                    index(x) = n;
                else
                    index(k) = index(x);
                end
            else
                if index(x) == 0
                    index(x) = index(k);
                else
                    index(index==index(x)) = index(k);
                end
            end
        else
            if index(k) == 0
                index(k) = n;
                n = n + 1;
            end
        end
    end
end

% Pick out groups that have the same index (i.e. part of the same
% ... chain)
issubExcluded = false(length(subsubSet),1);
for k = 1:length(subsubSet)
    if ~issubExcluded(k)
        issubExcluded(index==index(k)) = true;
        neighbourhood = subsubSet(index==index(k));
        isDynamic = false;
        for Fc = neighbourhood
            if Fc.isDynamic
                isDynamic = true;
                break;
            end
        end
        if isDynamic
            % For each moment in time, get the total area as a vector
            Area1 = zeros(1,Frame.NTheta);
            Area2 = zeros(1,Frame.NTheta);
            for x = 1:length(neighbourhood)
                for ind = 0:Frame.NTheta-1
                    Area1(ind+1) = Area1(ind+1) + ...
                        neighbourhood(x).Nodes(1).getArea(ind,neighbourhood(x).Connection);
                    Area2(ind+1) = Area2(ind+1) + ...
                        neighbourhood(x).Nodes(2).getArea(ind,neighbourhood(x).Connection);
                end
            end
            Area1 = CollapseVector(Area1);
            Area2 = CollapseVector(Area2);
            ratio = Area1./Area2;
            if ~all(ratio == 1)
                ratio(ratio>1)=1./ratio(ratio>1);
                firstformula = ratio>0.76;
                K12 = zeros(size(firstformula));
                K12(firstformula) = (1-ratio(firstformula)).^2.^2;
                K12(~firstformula) = 0.42*(1-ratio(~firstformula)).^2;
                K12 = CollapseVector(K12);
                K21 = K12;
                Entrance = Area1 > Area2;
            end
        end
    end
end

```

```

        if length(Entrance) > 1
            for b = 1:length(Entrance)
                x12 = min(b,length(K12));
                x21 = min(b,length(K21));
                if Entrance(b)
                    if K12(x12) > 0.5; K12(x12) = 0.5; end
                else
                    if K21(x21) > 0.5; K21(x21) = 0.5; end
                end
            end
        elseif Entrance
            K12(K12>0.5) = 0.5;
        elseif ~Entrance
            K21(K21>0.5) = 0.5;
        end
    else
        K12 = 0;
        K21 = 0;
    end
else
    % Get the area as a static scalar
    Area1 = 0; Area2 = 0;
    for x = 1:length(neighbourhood)
        Area1 = Area1 +
neighbourhood(x).Nodes(1).getArea(0,neighbourhood(x).Connection);
        Area2 = Area2 +
neighbourhood(x).Nodes(2).getArea(0,neighbourhood(x).Connection);
    end
    ratio = Area1/Area2;
    if ratio ~= 1
        if ratio > 1; ratio = 1/ratio; end
        if ratio > 0.76; K12 = (1-ratio.^2).^2;
        else; K12 = 0.42*(1-ratio.^2);
        end
        K21 = K12;
        if Area1 > Area2
            if K12 > 0.5
                K12 = 0.5;
            end
        else
            if K21 > 0.5
                K21 = 0.5;
            end
        end
    end
else
    K12 = 0;
    K21 = 0;
end
end
if all(K12 == 0)
    % It is straight, this is simply a pipe
    for Fc = neighbourhood
        if Fc.Orient == enumOrient.Vertical
            if Fc.Nodes(1).xmin == 0
                % Cylindrical
                C = 64;
            else
                % Annular
                C = 96;
            end
        else % Horizontal
            C = 96;
        end
        Fc.data.fFunc_l = @(Re) C./Re;
        Fc.data.fFunc_t = @(Re) 0.11*(this.roughness/Fc.data.Dh+68./Re).^0.25;

        % Streamwise conduction enhancement
        Fc.data.NkFunc_l = @(Re) 1;
        Fc.data.NkFunc_t = @(Re,Pr) 0.022*(Re.^0.75).*(Pr);
    end
else

```

```

        for Fc = neighbourhood
            Fc.data.K12 = K12;
            Fc.data.K21 = K21;
        end
    end
end
end
end
end

% Overwrite K of Custom Minor Losses
for CustomK = this.CustomMinorLosses
    if this.CustomMinorLosses.isValid()
        for Fc = this.Faces
            if isa(Fc.Nodes(1).Body, 'Body') && ...
                isa(Fc.Nodes(2).Body, 'Body')
                if (Fc.Nodes(1).Body == CustomK.Body1 && ...
                    Fc.Nodes(2).Body == CustomK.Body2)
                    Fc.data.K12 = CustomK.K12;
                    Fc.data.K21 = CustomK.K21;
                elseif (Fc.Nodes(2).Body == CustomK.Body1 && ...
                    Fc.Nodes(1).Body == CustomK.Body2)
                    Fc.data.K12 = CustomK.K21;
                    Fc.data.K21 = CustomK.K12;
                end
            end
        end
    end
end
end
end

%% Decimating Loops
progressbar('Decimating Extra Loops');
% debug_loopPlot(this, false);
% Decimate extra loops
Triads = cell(0,0);
for Nd = this.Nodes
    if Nd.Type ~= enumNType.SN && Nd.Type ~= enumNType.EN
        visited = GetTriad(Nd);
        for set = visited
            fcs = set{1};
            % Prevent duplicate loops from showing up
            found = false;
            for i = 1:length(Triads)
                if any(Triads{i}(1) == fcs && any(Triads{i}(2) == fcs) && ...
                    any(Triads{i}(3) == fcs))
                    found = true; break;
                end
            end
            if ~found; Triads{end+1} = fcs; end
        end
    end
end

% Look at Triads
Scores = cell(size(Triads));
Tri_Nodes = Scores;
for i = 1:length(Scores)
    Scores{i} = zeros(1,3);
    Tri_Nodes{i} = Node.empty;
end
% Score All the Triads
for k = 1:length(Triads)
    Tri = Triads{k};
    Tri_Node_i = 3;
    backup = [0 0];
    % Get nodes for the Tri
    for fc = Tri
        for nd = fc.Nodes
            if isempty(Tri_Nodes{k}) || ...
                all(Tri_Nodes{k} ~= nd)
                Tri_Nodes{k}(Tri_Node_i) = nd; Tri_Node_i = Tri_Node_i - 1;
            end
        end
    end
end

```



```

        end
    end
end
% Assign a score based on the area that enters than node
for fc = Tri
    score = mean(fc.data.Area);
    for nd = fc.Nodes
        index = find(Tri_Nodes{k}==nd);
        Scores{k}(index) = Scores{k}(index) + score;
    end
end
% Normalize the Scores According to the other Options
backup(1) = Scores{k}(1); backup(2) = Scores{k}(2);
Scores{k}(1) = backup(1) / (backup(2) + Scores{k}(3));
Scores{k}(2) = backup(2) / (backup(1) + Scores{k}(3));
Scores{k}(3) = Scores{k}(3) / (backup(1) + backup(2));
Scores{k}(isnan(Scores{k})) = 0;
% Ensure that faces that can't be closed will not be looked at
% ... As the number of faces that can't will only increase
Backup_Tri = Tri;
for i = 1:3
    for j = 1:length(Backup_Tri)
        fc = Backup_Tri(j);
        if ~any(fc.Nodes == Tri_Nodes{k}(i))
            Tri(i) = fc;
            if ~canClose(fc)
                Scores{k}(i) = 0;
            end
            break;
        end
    end
end
end
% Rearrange the Tri so that the faces correspond to the correct
% ... score
end

while ~isempty(Triads)
    Best_Tri = 0;
    Open_Triads = true(length(Triads),1);
    Best_Score = 0;
    Best_Index = 0;

    % Find Best Possible Face to close
    finding_best = true;
    while finding_best
        for k = 1:length(Triads)
            % Get the best
            finding_best = true;
            for i = 1:3
                if Scores{k}(i) > Best_Score
                    Best_Score = Scores{k}(i);
                    Best_Index = i;
                    Best_Tri = k;
                end
            end
        end
        if Best_Score == 0; break; end
        closing_face = Triads{Best_Tri}(Best_Index);
        if canClose(closing_face)
            finding_best = false;
        else
            Scores{Best_Tri}(Best_Index) = 0;
            Best_Score = 0;
        end
    end

    % Collapse the face, closing the triad
    if Best_Score > 0

        % Adjust the area and minor loss coefficients of the other two faces

```

```

for fc = Triads{Best_Tri}
    if fc ~= closing_face
        if isfield(fc.data,'K12')
            if isfield(closing_face.data,'K12')
                fc.data.K12 = (fc.data.K12.*fc.data.Area + ...
                    closing_face.data.K12.*closing_face.data.Area)./ ...
                    (fc.data.Area + closing_face.data.Area);
                fc.data.K21 = (fc.data.K21.*fc.data.Area + ...
                    closing_face.data.K21.*closing_face.data.Area)./ ...
                    (fc.data.Area + closing_face.data.Area);
            end
        end
        fc.data.Area = fc.data.Area + closing_face.data.Area;
    end
end

% Delete the face from the list
closing_face.data.Area = 0;
for nd = closing_face.Nodes
    nd.Faces(nd.Faces == closing_face) = [];
end
this.Faces(this.Faces == closing_face) = [];

for k = 1:length(Triads)
    if any(Triads{k} == closing_face)
        Open_Triads(k) = false;
    end
end
Open_Triads(Best_Tri) = false;
Triads = Triads(Open_Triads);
Scores = Scores(Open_Triads);
fprintf(['Decimated a Triad with ' num2str(length(Open_Triads) - sum(Open_Triads) - 1)
' others.\n']);
else
    Triads = cell(0);
end
end
end
%{
for Tri = Triads
    Tri_Nodes = Node.empty;
    Scores = {0,0,0};
    for fc = Tri{1}
        for nd = fc.Nodes
            if isempty(Tri_Nodes) || all(Tri_Nodes ~= nd)
                Tri_Nodes = [Tri_Nodes nd];
                index = length(Tri_Nodes);
            else
                index = find(Tri_Nodes==nd);
            end
            Scores{index} = Scores{index} + fc.data.Area;
        end
    end
end
%}

% Starting at the node with maximum area, test if the opposite face
% can be closed
%{
bestrecord = 0;
bestindex = 0;
for i = 1:3
    if mean(Scores{i}) > bestrecord
        for fc = Tri{1}
            if ~any(fc.Nodes == Tri_Nodes(i))
                if canClose(fc)
                    bestindex = i;
                    bestrecord = mean(Scores{i});
                end
            end
        end
    end
end
end
end
end

```

```

end

% Collapse the triad
if bestindex > 0 && bestrecord > 0
    fprintf('Decimated a Triad\n');
    % Find closing face
    for fc = Tri{1}
        if ~any(fc.Nodes == Tri_Nodes(i))
            closing_face = fc;
            break;
        end
    end
end

% Adjust the area and minor loss coefficients of the other two faces
for fc = Tri{1}
    if fc ~= closing_face
        if isfield(fc.data, 'K12')
            if isfield(closing_face.data, 'K12')
                fc.data.K12 = (fc.data.K12.*fc.data.Area + ...
                    closing_face.data.K12.*closing_face.data.Area) ./ ...
                    (fc.data.Area + closing_face.data.Area);
                fc.data.K21 = (fc.data.K21.*fc.data.Area + ...
                    closing_face.data.K21.*closing_face.data.Area) ./ ...
                    (fc.data.Area + closing_face.data.Area);
            end
        end
        fc.data.Area = fc.data.Area + closing_face.data.Area;
    end
end

% Delete the face from the list
closing_face.data.Area = 0;
for nd = closing_face.Nodes
    nd.Faces(nd.Faces == closing_face) = [];
end
this.Faces(this.Faces == closing_face) = [];
end
%}

% Faces
% Determine the amount of Solid, Environment, Mix and Gas Faces
S_count = 0;
E_count = 0;
M_count = 0;
for Fc = this.Faces
    switch Fc.Type
        case enumFType.Solid
            S_count = S_count + 1;
        case enumFType.Mix
            M_count = M_count + 1;
        case enumFType.Environment
            E_count = E_count + 1;
    end
end
G_count = length(this.Faces) - S_count - E_count - M_count;
fprintf(['Found: ' num2str(G_count) ' Gas Faces, ' ...
    num2str(E_count) ' Environment Faces, ' ...
    num2str(M_count) ' Mixed Faces, ' ...
    num2str(S_count) ' Solid Faces\n']);
M_count = G_count + M_count;
E_count = M_count + E_count;
S_count = length(this.Faces);
G_count_backup_faces = G_count;
E_count_backup_faces = E_count;
M_count_backup_faces = M_count;
for Fc = this.Faces
    switch Fc.Type
        case enumFType.Solid
            Fc.index = S_count;
            S_count = S_count - 1;
        case enumFType.Mix

```

```

        Fc.index = M_count;
        M_count = M_count - 1;
    case enumFType.Environment
        Fc.index = E_count;
        E_count = E_count - 1;
    case enumFType.Leak

        otherwise % Gas
            Fc.index = G_count;
            G_count = G_count - 1;
    end
end

% Remove Nodal Faces that have been deleted
for iNd = this.Nodes
    keep = true(size(iNd.Faces));
    j = 1;
    for Fc = iNd.Faces
        if isempty(Fc.index) || Fc.index < 1
            keep(j) = false;
        end
        j = j + 1;
    end
    iNd.Faces = iNd.Faces(keep);
end

% Deal with input options
% 1. NodeFactor -> Already handled in initial discretization
% 2. HX Convection ->
if isfield(run,'HX_Convection') && run.HX_Convection ~= 1
    % Find all bodies which are gases, but contain source nodes
    for iGroup = this.Groups
        for iBody = iGroup.Bodies
            if iBody.matl.Phase == enumMaterial.Gas
                if ~isempty(iBody.Matrix) && ...
                    isfield(iBody.Matrix.data,'SourceTemperature')
                    for nd = iBody.Nodes
                        if nd.Type ~= enumNType.SN
                            if isfield(nd.data,'NuFunc_l')
                                func = nd.data.NuFunc_l;
                                if nargin(func) == 2
                                    nd.data.NuFunc_l = @(Re,Pr) run.HX_Convection.*func(Re,Pr);
                                else
                                    nd.data.NuFunc_l = @(Re) run.HX_Convection.*func(Re);
                                end
                            end
                            if isfield(nd.data,'NuFunc_t')
                                func = nd.data.NuFunc_t;
                                if nargin(func) == 2
                                    nd.data.NuFunc_t = @(Re,Pr) run.HX_Convection.*func(Re,Pr);
                                else
                                    nd.data.NuFunc_t = @(Re) run.HX_Convection.*func(Re);
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end

% 3. Regen_Convection ->
if isfield(run,'Regen_Convection') && run.Regen_Convection ~= 1
    % Find all bodies which are gases but contain solids nodes without
    % ... source nodes
    for iGroup = this.Groups
        for iBody = iGroup.Bodies
            if iBody.matl.Phase == enumMaterial.Gas
                if ~isempty(iBody.Matrix) && ...
                    ~isfield(iBody.Matrix.data,'SourceTemperature')
                    for nd = iBody.Nodes

```





```

        end
    end
    if ~isempty(iBody.Matrix) && ...
        iBody.Matrix.Geometry == enumMatrix.CustomRegen
        if isfield(run,'Regen_C1')
            iBody.Matrix.data.C1 = run.Regen_C1;
        end
        if isfield(run,'Regen_C2')
            iBody.Matrix.data.C2 = run.Regen_C2;
        end
        if isfield(run,'Regen_C3')
            iBody.Matrix.data.C3 = run.Regen_C3;
        end
        if isfield(run,'Regen_C4')
            iBody.Matrix.data.C4 = run.Regen_C4;
        end
        if isfield(run,'Regen_Porosity')
            iBody.Matrix.data.Porosity = run.Regen_Porosity;
        end
        if isfield(run,'Regen_SA_V')
            iBody.Matrix.data.SA_V = run.Regen_SA_V;
        end
    end
end
end
end

%% Vectorizing Nodes
progressbar('Vectorizing Nodes');
% Generic Properties
Sim.dT_dU = zeros(S_count_backup,1);
Sim.u = Sim.dT_dU;
Sim.T = Sim.dT_dU;
Sim.CondFlux = Sim.dT_dU;

% Environment Additional Properties
Sim.P = zeros(E_count_backup,1);
Sim.dP = Sim.P;
Sim.dh_dT = Sim.P;
Sim.rho = Sim.P;
Sim.m = Sim.dT_dU;
Sim.vol = Sim.T;
Sim.dV_dt = Sim.P;

% Gas Node Additional Properties
Sim.k = Sim.P;
Sim.mu = Sim.P;
Sim.Dh = zeros(G_count_backup,1);
Sim.Nu = Sim.Dh;
Sim.NuFunc_l = cell(G_count_backup,1);
Sim.NuFunc_t = Sim.NuFunc_l;
Sim.isDynVol = Sim.P;
Sim.DynVol = zeros(6,0);
% Interpolated from Faces
Sim.RE = Sim.Dh;
Sim.U = Sim.Dh;
Sim.f = Sim.Dh;
% Turbulence
Sim.useTurbulenceNd = false(length(Sim.P)-1,1);
Sim.turb = Sim.P;
Sim.dturb = Sim.P;
Sim.Area = zeros(2,length(Sim.Dh));
Sim.Va = Sim.Dh;
Sim.to = Sim.Dh;

% Gas Regions
% Growth algorithm propegating through gas faces that are always open
region = zeros(length(Sim.P),1);
region_count = 0;
for Nd = this.Nodes
    if Nd.index <= length(region) && region(Nd.index) == 0
        region_count = region_count + 1;
    end
end

```

```

        region = PropegateRegion(Nd,region,region_count);
    if all(region > 0); break; end
end
end

% Define Functions
DynVol_n = 1;
DynDh_n = 1;
Rs = Sim.P;
for Nd = this.Nodes
    if isfield(Nd.data,'matl')
        if Nd.data.matl.Phase ~= enumMaterial.Solid
            matl = Material(Nd.data.matl.name);
        else
            matl = Nd.data.matl;
        end
    else
        if Nd.Body.matl.Phase ~= enumMaterial.Solid
            matl = Material(Nd.Body.matl.name);
        else
            matl = Nd.Body.matl;
        end
    end
end
switch Nd.Type
case enumNType.SN
    % Sim.dU(Nd.index) = 0; - Needs to be zeroed
    Sim.m(Nd.index) = Nd.vol()*matl.Density;
    if matl.dT_du <= 0
        Sim.dT_dU(Nd.index) = 0;
    else
        Sim.dT_dU(Nd.index) = matl.dT_du;
    end
    Sim.T(Nd.index) = Nd.data.T;
    Sim.u(Nd.index) = matl.initialInternalEnergy(Nd.data.T);
    Sim.vol(Nd.index) = Nd.vol();

    % Static Volume Gas Node %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
case {enumNType.SVGN, enumNType.VVGN, enumNType.SAGN}
    V = Nd.vol();
    Nd.recalc_Dh();
    if isempty(Nd.Body.Matrix)
        if ~isscalar(V)
            Sim.useTurbulenceNd(Nd.index) = true;
        else
            Sim.useTurbulenceNd(Nd.index) = false;
        end
    else
        Sim.useTurbulenceNd(Nd.index) = true;
    end

    if isscalar(V)
        Sim.vol(Nd.index) = V;
    else
        Sim.vol(Nd.index) = V(1);
        Sim.isDynVol(Nd.index) = DynVol_n;
        Sim.DynVol(1,DynVol_n) = Nd.index;
        Sim.DynVol(2,DynVol_n) = Sim.Dyn;
        DynVol_n = DynVol_n + 1;
        Sim.Dynamic(Sim.Dyn,:) = V;
        Sim.Dyn = Sim.Dyn + 1;
    end
    if isscalar(Nd.data.Dh)
        Sim.Dh(Nd.index) = Nd.data.Dh;
    else
        Sim.DynDh(1,DynDh_n) = Nd.index;
        Sim.DynDh(2,DynDh_n) = Sim.Dyn;
        DynDh_n = DynDh_n + 1;
        Sim.Dynamic(Sim.Dyn,:) = Nd.data.Dh;
        Sim.Dyn = Sim.Dyn + 1;
    end
end
Sim.dT_dU(Nd.index) = 1;

```



```

Sim.dT_duFunc{region(Nd.index)} = matl.dT_duFunc;
try
    Sim.dh_dTFunc{region(Nd.index)} = matl.dh_dTFunc; % Additions
catch
    matl.Configure(matl.name);
    Sim.dh_dTFunc{region(Nd.index)} = matl.dh_dTFunc; % Additions
end
Sim.u2T{region(Nd.index)} = matl.u2T;
Sim.T(Nd.index) = Nd.data.T;

Sim.P(Nd.index) = Nd.data.P;
Sim.rho(Nd.index) = Nd.data.P/(matl.R*Nd.data.T);
Sim.u(Nd.index) = matl.initialInternalEnergy(Nd.data.T);% +
Nd.data.P/Sim.rho(Nd.index);
Sim.m(Nd.index) = Sim.rho(Nd.index)*Sim.vol(Nd.index);

Sim.kFunc{region(Nd.index)} = matl.kFunc;
Sim.muFunc{region(Nd.index)} = matl.muFunc;
Rs(Nd.index) = matl.R;

if isfield(Nd.data,'NuoFunc_1')
    Sim.NuFunc_1{Nd.index} = Nd.data.NuoFunc_1;
else
    Sim.NuFunc_1{Nd.index} = Nd.data.NuFunc_1;
end
Sim.NuFunc_t{Nd.index} = Nd.data.NuFunc_t;
% Environment Node (Static Properties Node) %%%%%%%%%%%%%%%
case enumNType.EN
    Sim.dT_dU(Nd.index) = 0;
    Sim.T(Nd.index) = Nd.Body.Temperature;
    Sim.u2T{region(Nd.index)} = matl.u2T;
    Sim.kFunc{region(Nd.index)} = matl.kFunc;
    Sim.muFunc{region(Nd.index)} = matl.muFunc;
    Sim.dT_duFunc{region(Nd.index)} = matl.dT_duFunc;
    try
        Sim.dh_dTFunc{region(Nd.index)} = matl.dh_dTFunc; % Additions
    catch
        matl.Configure(matl.name);
        Sim.dh_dTFunc{region(Nd.index)} = matl.dh_dTFunc; % Additions
    end
    Rs(Nd.index) = matl.R;
    Sim.k(Nd.index) = matl.kFunc(Nd.data.T);
    Sim.mu(Nd.index) = matl.muFunc(Nd.data.T);
    Sim.u(Nd.index) = matl.initialInternalEnergy(Nd.data.T);% + Nd.data.P/Nd.data.rho;
    Sim.P(Nd.index) = Nd.data.P;
    Sim.rho(Nd.index) = Nd.data.rho;
    Sim.vol(Nd.index) = Inf;
    Sim.m(Nd.index) = Inf;
    Sim.turb(Nd.index) = 0;
end
end

%% Vectorizing Faces
progressbar('Vectorizing Faces');

Sim.Fc_Nd = zeros(length(this.Faces),2);
Sim.Fc_U = zeros(G_count_backup_faces,1);
Sim.Fc_PR = Sim.Fc_U;
Sim.Fc_dx = Sim.Fc_U;
Sim.Fc_RE = Sim.Fc_U;
Sim.Fc_f = Sim.Fc_U;
Sim.Fc_R = zeros(1,M_count_backup_faces);
Sim.Fc_fFunc_1 = cell(G_count_backup_faces,1);
Sim.Fc_fFunc_t = Sim.Fc_fFunc_1;
Sim.Fc_NkFunc_1 = Sim.Fc_fFunc_1;
Sim.Fc_NkFunc_t = Sim.Fc_fFunc_1;
Sim.Fc_Dist = Sim.Fc_U;
Sim.Fc_Cond_Dist = Sim.Fc_U;
Sim.Fc_K12 = Sim.Fc_U;
Sim.Fc_K21 = Sim.Fc_U;
Sim.Fc_u = Sim.Fc_U;

```

```

% Sim.dL_dt = Sim.Fc_U;
% Sim.dD_dt = Sim.Fc_U;
Sim.KpU_2A = Sim.Fc_U;
Sim.Fc_V = Sim.Fc_U;
Sim.Fc_dP = Sim.Fc_U;
Sim.Fc_V_backup = Sim.Fc_U;
Sim.Fc_W = Sim.Fc_U;

% For gas-gas, mix and environment faces
Sim.Fc_Area = zeros(E_count_backup_faces,1);
Sim.Fc_Dh = Sim.Fc_U;
Sim.Fc_Cond = Sim.Fc_Area;
Sim.Fc_T = Sim.Fc_U;
Sim.Fc_k = Sim.Fc_U;
Sim.Fc_mu = Sim.Fc_U;
Sim.Fc_rho = Sim.Fc_U;
Sim.Fc_Vel_Factor = Sim.Fc_U;
Sim.Fc_Shear_Factor = Sim.Fc_U;

% Turbulence
Sim.Fc_turb = Sim.Fc_U;
Sim.Fc_to = Sim.Fc_U;
Sim.useTurbulenceFc = true(G_count_backup_faces,1);

% Flux Limiters
Sim.Fc_Nd03 = Sim.Fc_Nd;
Sim.Fc_A = Sim.Fc_U;
Sim.Fc_B = Sim.Fc_U;
Sim.Fc_C = Sim.Fc_U;
Sim.Fc_D = Sim.Fc_U;

% Find V and S for the faces
for Fc = this.Faces
    [V, S, SContact] = FaceMotion(Fc);
    if ~isempty(V); Fc.data.V = V; end
    if ~isempty(S)
        Fc.data.S = S;
        if ~isempty(SContact)
            this.SheerContacts = [this.SheerContacts SContact];
        end
    end
end

%% Creating Shear/Pressure Contacts
progressbar('Creating Shear/Pressure Contacts');

% For all Faces, attempt to make a PressureContact
for Fc = this.Faces
    if Fc.Type == enumFType.Mix
        PContact = Fc.getPressureContact();
        if ~isempty(PContact)
            this.PressureContacts = [this.PressureContacts PContact];
        end
    elseif (Fc.Nodes(1).Type == enumNType.SN && ...
            Fc.Nodes(2).Type == enumNType.EN) || ...
            (Fc.Nodes(1).Type == enumNType.EN && ...
            Fc.Nodes(2).Type == enumNType.SN)
        PContact = Fc.getPressureContact();
        if ~isempty(PContact)
            this.PressureContacts = [this.PressureContacts PContact];
        end
    end
end

% For all Faces, distribute the friction length to neighbours if
% ... K enabled.
for Fc = this.Faces
    if isfield(Fc.data, 'Dist')
        Fc.data.Cond_Dist = Fc.data.Dist;
        if isfield(Fc.data, 'dx') && isfield(Fc.data, 'K12')
            if any(Fc.data.K12 > 0) || any(Fc.data.K21 > 0)

```

```

% Will only run this code if:
% ... It is a gas face
% ... It has a minor loss coefficient

% This face will not utilize the value of Dist
Fc.data.Dist = 0;

% Get the location of the center of this face
x = getCenterOfOverlapRegion(...
    Fc.Nodes(1).xmin, Fc.Nodes(2).xmin,...
    Fc.Nodes(1).xmax, Fc.Nodes(2).xmax);
y = getCenterOfOverlapRegion(...
    Fc.Nodes(1).ymin, Fc.Nodes(2).ymin,...
    Fc.Nodes(1).ymax, Fc.Nodes(2).ymax);

% Find all neighbours
for nd = Fc.Nodes
    ndx = (nd.xmin + nd.xmax)/2;
    ndy = (nd.ymin + nd.ymax)/2;
    count = 0;
    for fci = nd.Faces
        if fci ~= Fc
            if isfield(fci.data,'Dist') && (...
                (isfield(Fc.data, 'K12') && all(Fc.data.K12 == 0)) || ...
                ~isfield(Fc.data, 'K12'))
                count = count + 1;
            end
        end
    end
end
if count < 2
    % Will only run if this node has only one other gas face
    for fci = nd.Faces
        if fci ~= Fc
            if isfield(fci.data,'Dist') && (...
                (isfield(Fc.data, 'K12') && all(Fc.data.K12 == 0)) || ...
                ~isfield(Fc.data, 'K12'))
                % Will only run if this face can use it
                % Determine orientation of fci
                if fci.Nodes(1).xmin == fci.Nodes(2).xmax
                    dDist = abs(ndx - x);
                    % If the connection is actually closer than
                    % ... assumed then the distance is negative
                    if abs(fci.Nodes(1).xmin - x) < ...
                        abs(fci.Nodes(1).xmin - ndx)
                        dDist = -dDist;
                    end
                elseif fci.Nodes(1).xmax == fci.Nodes(2).xmin
                    dDist = abs(ndx - x);
                    % If the connection is actually closer than
                    % ... assumed then the distance is negative
                    if abs(fci.Nodes(2).xmin - x) < ...
                        abs(fci.Nodes(2).xmin - ndx)
                        dDist = -dDist;
                    end
                elseif all(fci.Nodes(1).ymin == fci.Nodes(2).ymax)
                    dDist = abs(ndy - y);
                    % If the connection is actually closer than
                    % ... assumed then the distance is negative
                    if abs(fci.Nodes(1).ymin - y) < ...
                        abs(fci.Nodes(1).ymin - ndy)
                        dDist = -dDist;
                    end
                else
                    dDist = abs(ndy - y);
                    % If the connection is actually closer than
                    % ... assumed then the distance is negative
                    if abs(fci.Nodes(2).ymin - y) < ...
                        abs(fci.Nodes(2).ymin - ndy)
                        dDist = -dDist;
                    end
                end
            end
        end
    end
end
end

```



```

    Sim.Fc_DynArea(2, Fc_DynArea_n) = Sim.Dyn;
    Fc_DynArea_n = Fc_DynArea_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
else
    Fc.data.R = Fc.data.R(~isnan(Fc.data.R));
    Fc.data.R = Fc.data.R(1);
    Sim.Fc_R(Fc.index) = Fc.data.R;
    Sim.Fc_Area(Fc.index) = Fc.data.Area;
end
% Look at the mixed face and determine if it can be
% ... approximated
%{
if isfield(N2.data, 'matl')
    if N2.data.matl.Phase ~= enumMaterial.Solid
        matl = Material(N2.data.matl.name);
    else
        matl = N2.data.matl;
    end
else
    if N2.Body.matl.Phase ~= enumMaterial.Solid
        matl = Material(N2.Body.matl.name);
    else
        matl = N2.Body.matl;
    end
end
if matl.dT_du < 0; matl.dT_du = 1e-8; end
if length(N2.Faces) == 1
    % True for most regenerators and heat exchangers
    if any((Fc.data.Area./Fc.data.R)./(N2.vol()*matl.Density./matl.dT_du)*1.e-4 >
0.25)
        IsApprox(Fc.index) = true;
        Sim.FcApprox(ApproxCount) = Fc.index;
        ApproxCount = ApproxCount + 1;
    end
else
    CU = 0;
    Cother = 0;
    for fc = N2.Faces
        if fc == Fc
            CU = fc.data.Area./fc.data.R;
            if double(sum(CU < Cother))/...
                double(max(length(CU), length(Cother))) > 0.5
                CU = 0;
                break;
            end
        else
            if fc.Type == enumFType.Mix
                Cother = Cother + fc.data.Area./fc.data.R;
                if any(CU ~= 0) && ...
                    double(sum(CU < Cother))/...
                        double(max(length(CU), length(Cother))) > 0.5
                    CU = 0;
                    break;
                end
            end
        end
    end
end
if CU ~= 0
    if any((Fc.data.Area./Fc.data.R)./(N2.vol()*matl.Density./matl.dT_du)*1.e-4 >
0.25)
        IsApprox(Fc.index) = true;
        Sim.FcApprox(ApproxCount) = Fc.index;
        ApproxCount = ApproxCount + 1;
    end
end
end
%}
MF_n = MF_n + 1;
case {enumFType.Gas, enumFType.MatrixTransition} % Gas-Gas, Gas-Environment Faces
Fc.recalc_Area_Dh();

```

```

% Create Fc_Fcs array
[A,B,C,D] = populate_Fc_ABCD(Sim, Fc);

% Create Fc_A array
if isscalar(A)
    Sim.Fc_A(Fc.index) = A;
else
    Sim.Dynamic(Sim.Dyn,:) = A;
    Sim.Fc_DynA(1,Fc_DynA_n) = Fc.index;
    Sim.Fc_DynA(2,Fc_DynA_n) = Sim.Dyn;
    Fc_DynA_n = Fc_DynA_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
% Create Fc_B array
if isscalar(B)
    Sim.Fc_B(Fc.index) = B;
else
    Sim.Dynamic(Sim.Dyn,:) = B;
    Sim.Fc_DynB(1,Fc_DynB_n) = Fc.index;
    Sim.Fc_DynB(2,Fc_DynB_n) = Sim.Dyn;
    Fc_DynB_n = Fc_DynB_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
% Create Fc_C array
if isscalar(C)
    Sim.Fc_C(Fc.index) = C;
else
    Sim.Dynamic(Sim.Dyn,:) = C;
    Sim.Fc_DynC(1,Fc_DynC_n) = Fc.index;
    Sim.Fc_DynC(2,Fc_DynC_n) = Sim.Dyn;
    Fc_DynC_n = Fc_DynC_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
% Create Fc_D array
if isscalar(D)
    Sim.Fc_D(Fc.index) = D;
else
    Sim.Dynamic(Sim.Dyn,:) = D;
    Sim.Fc_DynD(1,Fc_DynD_n) = Fc.index;
    Sim.Fc_DynD(2,Fc_DynD_n) = Sim.Dyn;
    Fc_DynD_n = Fc_DynD_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end

% Area
if isscalar(Fc.data.Area)
    Sim.Fc_Area(Fc.index) = Fc.data.Area;
else
    Sim.Dynamic(Sim.Dyn,:) = Fc.data.Area;
    Sim.Fc_DynArea(1,Fc_DynArea_n) = Fc.index;
    Sim.Fc_DynArea(2,Fc_DynArea_n) = Sim.Dyn;
    Fc_DynArea_n = Fc_DynArea_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
% Length / Friction Length
if isscalar(Fc.data.Dist)
    Sim.Fc_Dist(Fc.index) = Fc.data.Dist;
else
    Sim.Dynamic(Sim.Dyn,:) = Fc.data.Dist;
    Sim.Fc_DynDist(1,Fc_DynDist_n) = Fc.index;
    Sim.Fc_DynDist(2,Fc_DynDist_n) = Sim.Dyn;
    Fc_DynDist_n = Fc_DynDist_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
if isscalar(Fc.data.Cond_Dist)
    Sim.Fc_Cond_Dist(Fc.index) = Fc.data.Cond_Dist;
else
    Sim.Dynamic(Sim.Dyn,:) = Fc.data.Cond_Dist;
    Sim.Fc_DynCond_Dist(1,Fc_DynCond_Dist_n) = Fc.index;
    Sim.Fc_DynCond_Dist(2,Fc_DynCond_Dist_n) = Sim.Dyn;
    Fc_DynCond_Dist_n = Fc_DynCond_Dist_n + 1;
end

```

```

    Sim.Dyn = Sim.Dyn + 1;
end
if isscalar(Fc.data.dx)
    Sim.Fc_dx(Fc.index) = Fc.data.dx;
else
    Sim.Dynamic(Sim.Dyn,:) = Fc.data.dx;
    Sim.Fc_Dyndx(1,Fc_Dyndx_n) = Fc.index;
    Sim.Fc_Dyndx(2,Fc_Dyndx_n) = Sim.Dyn;
    Fc_Dyndx_n = Fc_Dyndx_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
% Hydraulic Diameter
if isscalar(Fc.data.Dh)
    Sim.Fc_Dh(Fc.index) = Fc.data.Dh;
else
    Sim.Dynamic(Sim.Dyn,:) = Fc.data.Dh;
    Sim.Fc_DynDh(1,Fc_DynDh_n) = Fc.index;
    Sim.Fc_DynDh(2,Fc_DynDh_n) = Sim.Dyn;
    Fc_DynDh_n = Fc_DynDh_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
% Friction Function
if isfield(Fc.data,'K12') && any(Fc.data.K12 > 0) && any(Fc.data.K21 > 0)
    if isscalar(Fc.data.K12)
        Sim.Fc_K12(Fc.index) = Fc.data.K12;
    else
        Sim.Dynamic(Sim.Dyn,:) = Fc.data.K12;
        Sim.Fc_DynK12(1,Fc_DynK12_n) = Fc.index;
        Sim.Fc_DynK12(2,Fc_DynK12_n) = Sim.Dyn;
        Fc_DynK12_n = Fc_DynK12_n + 1;
        Sim.Dyn = Sim.Dyn + 1;
    end
    if isscalar(Fc.data.K21)
        Sim.Fc_K21(Fc.index) = Fc.data.K21;
    else
        Sim.Dynamic(Sim.Dyn,:) = Fc.data.K21;
        Sim.Fc_DynK21(1,Fc_DynK21_n) = Fc.index;
        Sim.Fc_DynK21(2,Fc_DynK21_n) = Sim.Dyn;
        Fc_DynK21_n = Fc_DynK21_n + 1;
        Sim.Dyn = Sim.Dyn + 1;
    end
else
    Sim.Fc_fFunc_l{Fc.index} = Fc.data.fFunc_l;
    Sim.Fc_fFunc_t{Fc.index} = Fc.data.fFunc_t;
end
% Mixing Function
Sim.Fc_NkFunc_l{Fc.index} = Fc.data.NkFunc_l;
Sim.Fc_NkFunc_t{Fc.index} = Fc.data.NkFunc_t;
% Shear Speed Factor
if isfield(Fc.data,'S')
    Sim.Dynamic(Sim.Dyn,:) = Fc.data.S;
    Sim.Fc_DynShear_Factor(1,Fc_DynShear_Factor_n) = Fc.index;
    Sim.Fc_DynShear_Factor(2,Fc_DynShear_Factor_n) = Sim.Dyn;
    Fc_DynShear_Factor_n = Fc_DynShear_Factor_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
% Face Speed Factor
if isfield(Fc.data,'V')
    Sim.Dynamic(Sim.Dyn,:) = Fc.data.V;
    Sim.Fc_DynVel_Factor(1,Fc_DynVel_Factor_n) = Fc.index;
    Sim.Fc_DynVel_Factor(2,Fc_DynVel_Factor_n) = Sim.Dyn;
    Fc_DynVel_Factor_n = Fc_DynVel_Factor_n + 1;
    Sim.Dyn = Sim.Dyn + 1;
end
case enumFType.Leak % Gas-Gas Leaks
    % Do nothing, this is handled elsewhere
end
end
Sim.Dynamic = Sim.Dynamic';

```

```

%% Creating Face - Use Turbulence
progressbar('Creating Face - Use Turbulence');

L = G_count_backup_faces + 1;
for FC = this.Faces
    if FC.index < L
        % Don't use turbulence if
        % Matrix exist in either connected node
        % Either Node is variable volume
        for Nd = Fc.Nodes
            if ~Sim.Fc_K12(Fc.index)
                if Sim.isDynVol(Nd.index)
                    Sim.useTurbulenceFc(Fc.index) = false;
                    break;
                end
                if isa(Nd.Body, 'Body')
                    if ~isempty(Nd.Body.Matrix)
                        Sim.useTurbulenceFc(Fc.index) = false;
                        break;
                    end
                else
                    Sim.useTurbulenceFc(Fc.index) = false;
                    break;
                end
            end
        end
    end
end

%% Defining Max Time Step
progressbar('Defining Max Time Step');

BoundaryNodes = [];
NGas = length(Sim.P);
NGas = NGas - 1;
NSolid = length(Sim.T) - NGas;
MixFcs = cell(NSolid,1);
ACond = zeros(NSolid,NSolid);
bCond = zeros(NSolid,1);

for nd = this.Nodes
    if nd.Type == enumNType.SN
        if (isfield(nd.data, 'matl') && nd.data.matl.dT_du < 0) || ...
            nd.Body.matl.dT_du < 0
            % Do not give these any sort of special privileges as they are
            % ... no different than an environment node.
            continue;
        end
        added = false;
        for fc = nd.Faces
            if fc.Type == enumFType.Mix
                if ~added
                    added = true;
                    BoundaryNodes(length(BoundaryNodes)+1,1) = nd.index - NGas;
                end
                MixFcs{nd.index - NGas} = [MixFcs{nd.index - NGas} fc.index];
            end
        end
    end
end

fprintf(['Number of Solid Nodes + 1 Env node: ' num2str(NSolid) '\n']);

for nd = this.Nodes
    if nd.index > NGas
        row = nd.index-NGas;
        if nd.Type == enumNType.SN
            if (isfield(nd.data, 'matl') && nd.data.matl.dT_du < 0) || ...
                nd.Body.matl.dT_du < 0
                ACond(row,row) = 1;
                bCond(row) = nd.data.T;
            end
        end
    end
end

```



```

else
    for fc = nd.Faces
        if fc.Type ~= enumFType.Mix
            avgCond = mean(fc.data.U);
            if fc.Nodes(1) == nd
                col = fc.Nodes(2).index - NGas;
            else
                col = fc.Nodes(1).index - NGas;
            end
            ACond(row, row) = ACond(row, row) + avgCond;
            ACond(row, col) = ACond(row, col) - avgCond;
        end
    end
end
else
    if nd.Type == enumNType.EN
        ACond(row, row) = 1;
        bCond(row) = nd.Body.Temperature;
    else
        fprintf(['XXX Undefined node type during ACond and bCond ' ...
            'pre-calculation XXX/n']);
    end
end
end
end
end

Sim.ACond = ACond;
Sim.bCond = bCond;
Sim.CondEff = zeros(M_count_backup_faces,1);
Sim.CondTempEff = zeros(M_count_backup_faces,1);
Sim.BoundaryNodes = BoundaryNodes';
Sim.MixFcs = MixFcs;
Sim.CycleTime = 0;

a = 1000;
Sim.Solid_dt_max = a(ones(1,Frame.NTheta));
Sim.Nd_Solid_dt = a(ones(size(Sim.T)));
for fc = this.Faces
    if ~IsApprox(fc.index)
        if fc.Type == enumFType.Solid
            for nd = fc.Nodes
                if isfield(nd.data,'matl'); matl = nd.data.matl;
                else; matl = nd.Body.matl;
                end
                if isa(nd.Body,'Body')
                    if matl.dT_du > 0
                        timesteps = (this.MaxFourierNumber*nd.vol()*matl.Density/...
                            matl.dT_du)./fc.data.U;
                        if length(timesteps) == 1
                            Sim.Solid_dt_max(Sim.Solid_dt_max > timesteps) = timesteps;
                        else
                            if iscolumn(timesteps)
                                timesteps = timesteps';
                            end
                            Sim.Solid_dt_max = min([Sim.Solid_dt_max; timesteps]);
                        end
                        Sim.Nd_Solid_dt(nd.index) = min(Sim.Nd_Solid_dt(nd.index), min(timesteps));
                    end
                end
            end
        end
    end
end
end
end

%% Defining Conduction/Transport Network
progressbar('Defining Conduction/Transport Network');

% SolidNds
% Sim.Cond_Nds = all nodes except the environment, which is
% ... automatically excluded
Sim.Cond_Nds = [1:G_count_backup E_count_backup+1:S_count_backup];

```

```

%Sim.Cond_Fcs = Sim.Solid_Fc;
Sim.Cond_Fcs = 1:length(this.Faces);
Sim.Cond_Fcs(IsApprox) = []; % Remove faces that are approximated by a different method.
Nds1 = Sim.Fc_Nd(Sim.Cond_Fcs,1);
Nds2 = Sim.Fc_Nd(Sim.Cond_Fcs,2);
Sim.Cond_Nds1 = Nds1;
Sim.Cond_Nds2 = Nds2;
Nds1(Sim.dT_dU(Nds1)<0) = 0;
Nds2(Sim.dT_dU(Nds2)<0) = 0;
% Determine set of element sets with sign
i = 1;
% Node 1's
if any(Nds1~=0); N1 = mode(Nds1(Nds1~=0)); N1 = sum(Nds1(:)==N1);
else; N1 = 0; end
if any(Nds2~=0); N2 = mode(Nds2(Nds2~=0)); N2 = sum(Nds2(:)==N2);
else; N2 = 0; end
Temp = cell(1,3*(N1+N2));
if N1 ~= 0
    % First element = sign
    % Second element = nodes
    % Third element = faces
    for k = 1:N1
        % Q is flow into Node 1
        N = length(unique(Nds1(Nds1>0)));
        Temp{i} = -1;
        Temp{i+1} = zeros(1,N);
        Temp{i+2} = zeros(1,N);
        el = 1;
        for x = 1:length(Nds1)
            if Nds1(x) > 0
                if ~any(Temp{i+1}(1:el-1) == Nds1(x))
                    Temp{i+1}(el) = Nds1(x);
                    Temp{i+2}(el) = x; % index with respect to Fc_Cond
                    Nds1(x) = 0;
                    el = el + 1;
                end
            end
        end
        i = i + 3;
    end
end
% Node 2's
if N2 ~= 0
    % First element = sign
    % Second element = nodes
    % Third element = faces
    for k = 1:N2
        % Q is flow into Node 1
        N = length(unique(Nds2(Nds2>0)));
        Temp{i} = 1;
        Temp{i+1} = zeros(1,N);
        Temp{i+2} = zeros(1,N);
        el = 1;
        for x = 1:length(Nds2)
            if Nds2(x) > 0
                if ~any(Temp{i+1}(1:el-1) == Nds2(x))
                    Temp{i+1}(el) = Nds2(x);
                    Temp{i+2}(el) = x; % index with respect to Fc_Cond
                    Nds2(x) = 0;
                    el = el + 1;
                end
            end
        end
        i = i + 3;
    end
end
end
Sim.CondNet = Temp;

% GasNds
Sim.Trans_Fcs = 1:length(Sim.Fc_U);
Nds1 = Sim.Fc_Nd(Sim.Trans_Fcs,1);

```

```

Nds2 = Sim.Fc_Nd(Sim.Trans_Fcs,2);
Nds1(Sim.dT_dU(Nds1)==0) = 0;
Nds2(Sim.dT_dU(Nds2)==0) = 0;
% Determine the set of element sets with sign
i = 1;
% Node 1's
if any(Nds1~=0); N1 = mode(Nds1(Nds1~=0)); N1 = sum(Nds1(:)==N1);
else; N1 = 0; end
if any(Nds2~=0); N2 = mode(Nds2(Nds2~=0)); N2 = sum(Nds2(:)==N2);
else; N2 = 0; end
Temp = cell(1,3*(N1+N2));
if N1 ~= 0
    Excluded = false(1,length(Nds1)); Excluded(Nds1==0) = true;
    % First element = sign
    % Second element = nodes
    % Third element = faces
    for k = 1:N1
        % Q is flow into Node 1
        N = length(unique(Nds1(~Excluded)));
        Temp{i} = -1;
        Temp{i+1} = zeros(1,N);
        Temp{i+2} = zeros(1,N);
        el = 1;
        for x = 1:length(Nds1)
            if ~Excluded(x) && ~any(Temp{i+1}(1:el-1) == Nds1(x))
                Temp{i+1}(el) = Nds1(x); % Target Node
                Temp{i+2}(el) = x; % index with respect to Fc_Cond
                Excluded(x) = true;
                el = el + 1;
            end
        end
        i = i + 3;
    end
end
% Node 2's
if N2 ~= 0
    Excluded = false(1,length(Nds2)); Excluded(Nds2==0) = true;
    % First element = sign
    % Second element = nodes
    % Third element = faces
    for k = 1:N2
        % Q is flow into Node 1
        N = length(unique(Nds2(~Excluded)));
        Temp{i} = 1;
        Temp{i+1} = zeros(1,N);
        Temp{i+2} = zeros(1,N);
        el = 1;
        for x = 1:length(Nds2)
            if ~Excluded(x) && ~any(Temp{i+1}(1:el-1) == Nds2(x))
                Temp{i+1}(el) = Nds2(x);
                Temp{i+2}(el) = x; % index with respect to Fc_Cond
                Excluded(x) = true;
                el = el + 1;
            end
        end
        i = i + 3;
    end
end
end
Sim.TransNet = Temp;

%% Creating Lookup Tables, Regions and Loops For Solver
progressbar('Creating Regions and Loops For Solver');

%% Group functions
% Laminar Nusselt
novel = true(size(Sim.P));
novel(end) = [];
nodes = zeros(size(Sim.P));
Sim.NuFunc_1_el = cell(0);
x = 1;
for i = 1:length(Sim.P)-2

```

```

if novel(i)
    func = Sim.NuFunc_l{i}; k = 1; nodes(k) = i;
    for j = i+1:length(Sim.P)-1
        if novel(j)
            if nargin(func) == nargin(Sim.NuFunc_l{j})
                if nargin(func) == 1
                    x1 = rand(1);
                    if func(x1) == Sim.NuFunc_l{j}(x1)
                        k = k + 1; nodes(k) = j; novel(j) = false;
                    end
                else
                    x1 = rand(1); x2 = rand(1);
                    if func(x1,x2) == Sim.NuFunc_l{j}(x1,x2)
                        k = k + 1; nodes(k) = j; novel(j) = false;
                    end
                end
            end
        end
    end
    Sim.NuFunc_l_el{x} = nodes(1:k);
    x = x + 1;
end
end
if novel(end); Sim.NuFunc_l_el{x} = length(novel); end
Sim.NuFunc_l(~novel) = [];

% Turbulent Nusselt
novel(:) = true;
nodes = zeros(size(Sim.P));
Sim.NuFunc_t_el = cell(0);
x = 1;
for i = 1:length(Sim.P)-2
    if novel(i)
        func = Sim.NuFunc_t{i}; k = 1; nodes(k) = i;
        for j = i+1:length(Sim.P)-1
            if novel(j)
                if nargin(func) == nargin(Sim.NuFunc_t{j})
                    if nargin(func) == 1
                        x1 = rand(1);
                        if func(x1) == Sim.NuFunc_t{j}(x1)
                            k = k + 1; nodes(k) = j; novel(j) = false;
                        end
                    else
                        x1 = rand(1); x2 = rand(1);
                        if func(x1,x2) == Sim.NuFunc_t{j}(x1,x2)
                            k = k + 1; nodes(k) = j; novel(j) = false;
                        end
                    end
                end
            end
        end
        Sim.NuFunc_t_el{x} = nodes(1:k);
        x = x + 1;
    end
end
if novel(end); Sim.NuFunc_t_el{x} = length(novel); end
Sim.NuFunc_t(~novel) = [];

% Laminar Conduction
novel = true(size(Sim.Fc_U));
nodes = zeros(size(Sim.Fc_U));
Sim.Fc_NkFunc_l_el = cell(0);
x = 1;
if ~isempty(novel)
    for i = 1:length(Sim.Fc_U)-1
        if novel(i)
            func = Sim.Fc_NkFunc_l{i}; k = 1; nodes(k) = i;
            for j = i+1:length(Sim.Fc_U)
                if novel(j)
                    if nargin(func) == nargin(Sim.Fc_NkFunc_l{j})
                        if nargin(func) == 1

```

```

        x1 = rand(1);
        if func(x1) == Sim.Fc_NkFunc_1{j}(x1)
            k = k + 1; nodes(k) = j; novel(j) = false;
        end
    else
        x1 = rand(1); x2 = rand(1);
        if func(x1,x2) == Sim.Fc_NkFunc_1{j}(x1,x2)
            k = k + 1; nodes(k) = j; novel(j) = false;
        end
    end
end
end
end
end
end
Sim.Fc_NkFunc_1_el{x} = nodes(1:k);
x = x + 1;
end
end
if novel(end); Sim.Fc_NkFunc_1_el{x} = length(novel); end
Sim.Fc_NkFunc_1(~novel) = [];
end

% Turbulent Conduction
novel(:) = true;
nodes = zeros(size(Sim.Fc_U));
Sim.Fc_NkFunc_t_el = cell(0);
x = 1;
if ~isempty(novel)
    for i = 1:length(Sim.Fc_U)-1
        if novel(i)
            func = Sim.Fc_NkFunc_t{i}; k = 1; nodes(k) = i;
            for j = i+1:length(Sim.Fc_U)
                if novel(j)
                    if nargin(func) == nargin(Sim.Fc_NkFunc_t{j})
                        if nargin(func) == 1
                            x1 = rand(1);
                            if func(x1) == Sim.Fc_NkFunc_t{j}(x1)
                                k = k + 1; nodes(k) = j; novel(j) = false;
                            end
                        else
                            x1 = rand(1); x2 = rand(1);
                            if func(x1,x2) == Sim.Fc_NkFunc_t{j}(x1,x2)
                                k = k + 1; nodes(k) = j; novel(j) = false;
                            end
                        end
                    end
                end
            end
        end
    end
end
end
Sim.Fc_NkFunc_t_el{x} = nodes(1:k);
x = x + 1;
end
end
if novel(end); Sim.Fc_NkFunc_t_el{x} = length(novel); end
Sim.Fc_NkFunc_t(~novel) = [];
end

% region now represents how the engine is divided up
regions = cell(region_count,1);
isEnvironmentRegion = false(region_count,1);
isEnvironmentRegion(region(end)) = true;
loop_ind_cell = cell(region_count,1);
loops_cell = cell(region_count,1);
regionFcCount = zeros(region_count,1);
regionFcs = cell(region_count,1);
Sim.ActiveRegionFcs = cell(region_count,1);
Sim.A_Press = cell(region_count,1);

for i = 1:region_count
    % loop_ind = [ start end condition ]
    %           [ ... .. ind or 0 ]
    loop_ind_cell{i} = zeros(3,0);
    % loops = [ndl, fcl2, sign]

```

```

loops_cell{i} = zeros(3,0);
end
% Make a list of nodes that are under this region
for i = 1:region_count
    c = 1;
    regions{i} = zeros(length(region),1);
    for k = 1:length(region)-1
        if region(k) == i; regions{i}(c) = k; c = c + 1; end
    end
    if isEnvironmentRegion(i)
        regions{i}(c) = length(region);
        c = c + 1;
    end
    if c <= length(region); regions{i}(c:end) = []; end
end

% Take a count of faces that are under this region
% ... This will tell us how many loops we need to define
% ... May not be necessary

%     extfc = cell(n,2);
for Fc = this.Faces
    if isfield(Fc.data,'dx')
        if region(Fc.Nodes(1).index) == region(Fc.Nodes(2).index)
            r = region(Fc.Nodes(1).index);
            regionFcCount(r) = regionFcCount(r) + 1;
            regionFcs{r}(end+1) = Fc.index;
        end
    end
end
%     Sim.extfc = extfc;

% Find Loops in each region
LEN = length(Sim.Fc_U);
% Make visited/closed any string who goes to a dead end
Closed_Edge = TrimFaces(this,region,false(LEN,1));

% What is left is all the nodes that could possibly be a part of a
% ... loop.
% Create Loops using the available faces and nodes
open = LoopNode.empty;
for i = 1:region_count
    lequ = 1;
    lcount = 0;
    %     TimesVisited = zeros(length(Sim.Fc_dx),1);
    % Find the edges that close during the cycle. These are classified
    % ... as holes.
    holes = Face.empty;
    for Fc = this.Faces
        if Fc.index <= LEN && ...
            region(Fc.Nodes(1).index) == i && ...
            region(Fc.Nodes(2).index) == i && ...
            any(Fc.data.Area == 0)
            % This is a node that is transient, used to trim loops
            holes(end+1) = Fc;
            Closed_Edge(Fc.index) = true;
        end
    end
end

% We have defined all the "holes", the first n loops will be
% ... dedicated to covering those holes.
% NIndependent_Equations = # of Nodes - 1
% N loops = Unknowns - NIndependent_Equations - Environment_Node
% Nloops = (regionFcCount(i)-length(regions{i}))+1-isEnvironmentRegion(i));
for k = 1:(regionFcCount(i)-length(regions{i}))+1)
    Vis_Edge = Closed_Edge;
    % Get a starting Point
    if k <= length(holes)
        % Find loops that cover these holes
        Fc = holes(k);
    else

```

```

% Find open edges, and find loops that cover them
found = false;
for Fc = this.Faces
    if Fc.index <= LEN && ~Closed_Edge(Fc.index) && ...
        region(Fc.Nodes(1).index) == i && ...
        region(Fc.Nodes(2).index) == i
        found = true; break;
    end
end
if ~found
    fprintf('XXX No valid Loop Starting Point! XXX\n'); return;
end
Closed_Edge(Fc.index) = true; Vis_Edge(Fc.index) = true;
end
closed = LoopNode(LoopNode.empty, Face.empty, Fc.Nodes(1));
target = Fc.Nodes(1);
open = LoopNode(closed(1), Fc, Fc.Nodes(2));
% EdgeClosed = Fc;

% Use open as a starting point and path to the closed
% ... "target" is the end node
% ... Do not path though Closed_Edge's
done = false;
while ~(isempty(open) || done)
    len = length(open);
    for x = len:-1:1
        % Expand it
        LpNd = open(x);
        for Fc = LpNd.Nd.Faces
            if Fc.index <= length(Vis_Edge) && ...
                region(Fc.Nodes(1).index) == i && ...
                region(Fc.Nodes(2).index) == i && ...
                ~Vis_Edge(Fc.index)
                Vis_Edge(Fc.index) = true;
                % Add it to the open list
                if Fc.Nodes(1) == LpNd.Nd; newNd = Fc.Nodes(2);
                else; newNd = Fc.Nodes(1); end
                if newNd == target
                    done = true;
                    closed(end+1) = LoopNode(LpNd, Fc, newNd);
                    break;
                else
                    open(end+1) = LoopNode(LpNd, Fc, newNd);
                end
            end
        end
    end
    if done; break; end
end
if ~done; open = open(len+1:end); end
end

% The loop should of reached its target.
if done
    % Backtrace the loop
    current = closed(end);
    lcount = lcount + 1;
    loop_ind_cell{i}(1,lcount) = lequ;
    while ~isempty(current.parent)
        loops_cell{i}(1,lequ) = current.Nd.index;
        loops_cell{i}(2,lequ) = current.parentFc.index;
        if current.parentFc.Nodes(1) == current.Nd
            loops_cell{i}(3,lequ) = 1;
        else
            loops_cell{i}(3,lequ) = -1;
        end
        lequ = lequ + 1;
        current = current.parent;
    end
    loop_ind_cell{i}(2,lcount) = lequ - 1;
    % closed(end) is the start node
    % Determine if the loop has a condition

```

```

    if k <= length(holes)
        % This one is connected to the area state of holes(k)
        loop_ind_cell{i}(3,lcoun) = holes(k).index;
    else
        % This one is unconnected to a hole
        loop_ind_cell{i}(3,lcoun) = 0;
    end

    % Close all edges that run into the "EdgeClosed"
    if k >= length(holes)
        Closed_Edge = TrimFaces(this, region, Closed_Edge);
    end

    else
        fprintf(['XXX Failed to complete a loop. Loop: ' num2str(k) ' XXX\n']);
    end
end

end

% Find ActiveFaces
Vis_Node = false(length(region),1);
for i = 1:region_count
    k = 0;
    Temp = zeros(regionFcCount(i),1);
    for Nd = this.Nodes
        if Nd.index <= length(region) && ...
            ~Vis_Node(Nd.index) && ...
            region(Nd.index) == i
            [k,Temp,Vis_Node] = PropegateActiveFaces(Nd,Vis_Node,k,Temp);
            break;
        end
    end
    Temp(k+1:end) = [];
    Sim.ActiveRegionFcs{i} = Temp;
end

% Find A-PressureLoss
for i = 1:region_count
    Sim.A_Press{i} = zeros(length(regions{i}));
    for x = 1:length(Sim.ActiveRegionFcs{i})
        Fc = Sim.ActiveRegionFcs{i}(x);
        % n1 = +ve;
        % n2 = -ve;
        temp = Sim.Fc_Nd(Fc,1);
        for k = 1:length(regions{i})
            if temp == regions{i}(k)
                Sim.A_Press{i}(x,k) = 1;
                break;
            end
        end
        temp = Sim.Fc_Nd(Fc,2);
        for k = 1:length(regions{i})
            if temp == regions{i}(k)
                Sim.A_Press{i}(x,k) = -1;
                break;
            end
        end
    end
end

% Calculate what the gas constant would be
% We have Rs
for i = 1:region_count
    if isEnvironmentRegion(i)
        Sim.R(i) = Rs(end);
    else
        % Pick the most common
        Sim.R(i) = mode(Rs(regions{i}));
    end
end
for j = regions{i}

```



```

        if Rs(j) ~= Sim.R(i)
            fprintf(['XXX Node in region ' num2str(i) ...
                ' found that had a different gas than the bulk. XXX\n']);
            fprintf(['XXX ... Region is of size: ' ...
                num2str(length(regions{i})) ' . XXX\n']);
        end
    end
end
Sim.Rs = Rs;

% loops_cell
% loop_ind_cell
% regions (cell array containing all nodes separated by a region)
Sim.Regions = regions;
Sim.isEnvironmentRegion = isEnvironmentRegion;

Sim.RegionFcs = regionFcs;
for i = 1:length(regionFcs)
    Sim.Fc2Col(regionFcs{i}(:)) = 1:length(regionFcs{i});
end
Sim.RegionFcCount = regionFcCount;
Sim.RegionLoops = loops_cell;
Sim.RegionLoops_Ind = loop_ind_cell;
for list = loop_ind_cell
    count = count + size(list{1},2);
end
fprintf(['Found ' num2str(count) ' loops. \n']);

% Collapse F2C for the limited set
% Sim.isLoopRegionFcs = cell(size(Sim.RegionFcs));
% Sim.Fc2Col_loop = zeros(size(Sim.Fc_V));
% for i = 1:region_count
%     if ~isempty(Sim.RegionLoops{i})
%         Sim.isLoopRegionFcs{i} = false(size(Sim.RegionFcs{i}));
%         for x = 1:size(Sim.RegionLoops{i},2)
%             Sim.isLoopRegionFcs{i}(Sim.RegionLoops{i}(2,x)) = true;
%         end
%     end
% end
end

Sim.Faces = cell(length(Sim.Dh),1);

% Define "Sim.Faces"
for Nd = this.Nodes
    if Nd.index <= length(Sim.Dh)
        Fcs = Face.empty;
        % It is a gas node
        % ... Get list of gas faces for this node
        for Fc = Nd.Faces
            if isfield(Fc.data,'dx')
                Fcs(end+1) = Fc;
            end
        end
        if ~isempty(Fcs)
            % This node has many gas faces
            Sim.Faces{Nd.index} = zeros(length(Fcs),3);
            for i = 1:length(Fcs)
                if Fcs(i).Nodes(1) == Nd
                    dir = -1;
                else
                    dir = 1;
                end
                Sim.Faces{Nd.index}(i,:) = [Fcs(i).index dir...
                    region(Fcs(i).Nodes(1).index) ~= region(Fcs(i).Nodes(2).index)];
            end
        else
            Sim.Faces{Nd.index} = zeros(0,3);
        end
    end
end
end
end

```

```

% So we have faces, which has an entry for each node
% ... List of Face Indexes
% ... List of BValues
% ... List of signs (-1 for outlet, 1 for inlet)
% ... List of 0 = implicit, 1 = explicit

% Need to make a list of implicit velocities that need to be
% ... calculated, use region pressure
Sim.ExplicitLeak = zeros(0,5);
Sim.ExplicitNorm = zeros(0,3);
for Fc = this.Faces
    if isfield(Fc.data,'dx')
        if region(Fc.Nodes(1).index) ~= region(Fc.Nodes(2).index)
            % Add to list
            if Fc.Type == enumFType.Leak
                % FAVAD equation
                % ...  $V = C * (dP)^N$ 
                Sim.ExplicitLeak = [Sim.ExplicitLeak; [Fc.index Fc.data.C Fc.data.N
region(Fc.Nodes(1).index) region(Fc.Nodes(2).index)]];
            else
                Sim.ExplicitNorm = [Sim.ExplicitNorm; [Fc.index region(Fc.Nodes(1).index)
region(Fc.Nodes(2).index)]];
            end
        end
    end
end

% Flow Network
% NEED EXTERNAL FACES TO BE LABELLED
% ... explicit faces
% ... ... Between regions, sources, sinks, leaks.. etc
% ... ... Used on region scale for mass change
% ... ... Used on local scale
% ... implicit faces
% ... ... Internal to region,
% NEED FACES AND NODES ORGANIZED BY REGION

%% Pressure/Shear Contacts, Sensors, PVoutputs
progressbar('Pressure/Shear Contacts, Sensors, PVoutputs');

% Pressure Contacts
PC_n = 1;
for PC = this.PressureContacts
    addto = true;
    for i = 1:PC_n-1
        if PC.GasNode == Sim.Press_Contact(3,i) && ...
            PC.Area == Sim.Press_Contact(2,i) && ...
            PC.MechanismIndex == Sim.Press_Contact(1,i)
            addto = false;
        end
    end
    if addto
        Sim.Press_Contact(1,PC_n) = PC.ConverterIndex;
        Sim.Press_Contact(2,PC_n) = PC.MechanismIndex;
        Sim.Press_Contact(3,PC_n) = PC.Area;
        Sim.Press_Contact(4,PC_n) = PC.GasNode.index;
        PC_n = PC_n + 1;
    end
end
this.PressureContacts = PressureContact.empty;

% Shear Contacts
SC_n = 1;
SC_Active_n = 1;
for SC = this.SheerContacts
    addto = true;
    for i = 1:SC_n-1
        if SC.UpperNode.index == Sim.Sheer_Contact(4,i) && ...
            SC.LowerNode.index == Sim.Sheer_Contact(3,i) && ...
            SC.Area == Sim.Sheer_Contact(2,i) && ...
            SC.MechanismIndex == Sim.Sheer_Contact(1,i)

```

```

        addto = false;
    end
end
if addto
    if any(SC.ActiveTimes)
        Sim.Shear_Contact(1,SC_n) = SC.ConverterIndex;
        Sim.Shear_Contact(2,SC_n) = SC.MechanismIndex;
        Sim.Shear_Contact(3,SC_n) = SC.Area;
        Sim.Shear_Contact(4,SC_n) = SC.LowerNode.index;
        Sim.Shear_Contact(5,SC_n) = SC.UpperNode.index;
        Sim.Shear_Contact(6,SC_n) = 1;
        if ~all(SC.ActiveTimes)
            if size(SC.ActiveTimes,2) ~= 1
                SC.ActiveTimes = SC.ActiveTimes';
            end
            Sim.Dynamic(:,Sim.Dyn) = SC.ActiveTimes;
            Sim.SC_Active(1,SC_Active_n) = SC_n;
            Sim.SC_Active(2,SC_Active_n) = Sim.Dyn;
            SC_Active_n = SC_Active_n + 1;
            Sim.Dyn = Sim.Dyn + 1;
        end
        SC_n = SC_n + 1;
    end
end
end
this.ShearContacts = ShearContact.empty;

% Sensors
for i = length(this.Sensors):-1:1
    if ~isValid(this.Sensors(i))
        len = length(this.Sensors);
        this.Sensors(i).deReference()
        if len == length(this.Sensors)
            this.Sensors(i) = [];
        end
    end
end
end
if ~isempty(this.Sensors)
    for iSense = this.Sensors
        iSense.update();
    end
end
end
if ~isempty(this.PVoutputs)
    for iPVoutput = this.PVoutputs; iPVoutput.update(region); end
end
Sim.PRegion = zeros(length(Sim.Regions),1);
Sim.PRegionTime = 0;

progressbar(12/13);
progressbar('Defining Area for Turbulence');

% Node Faces For Turbulent Decay and Generation
Len = 1 + size(Sim.Area,2);
for Nd = this.Nodes
    if Nd.index < Len
        if Nd.Body.divides(1) > Nd.Body.divides(2)
            % It is divides along by cylindrical shells
            % Look for two dynamic Dh faces that have the same motion
            % ... Pattern

            % 1 and 2 dynamic face pairs within body
            if (~isscalar(Nd.ymax) || ~isscalar(Nd.ymin)) && ...
                ~all(Nd.ymax-Nd.ymin == Nd.ymax(1)-Nd.ymin(1))
                startindex = 1;
                count = 0;
                while startindex ~= 0 && startindex <= length(Nd.Faces)
                    Pattern = 0;
                    oldstartindex = startindex;
                    startindex = 1;
                    i = 0;
                    for Fc = Nd.Faces(startindex:end)

```

```

i = i + 1;
if Fc.Type ~= enumFType.Mix
    if ~isscalar(Fc.data.Dh) && isempty(Fc.Connection)
        count = count + 1;
        if ~any(Pattern)
            Pattern = Fc.data.Dh;
            startindex = i;
        else
            temp = Pattern./Fc.data.Dh;
            if all(temp == temp(1))
                % Simply take the average of the faces
                Sim.Area(1,Nd.index) = Fc.index;
                Sim.Area(2,Nd.index) = (temp(1)+1)/2;
                startindex = 0;
                break;
            end
        end
    end
end
end
end
if oldstartindex == startindex
    startindex = 0;
elseif startindex ~= 0
    startindex = startindex + 1;
end
end
if count == 1
    % There is only one non-Connection Face
    for Fc = Nd.Faces
        if Fc.Type ~= enumFType.Mix && ~isscalar(Fc.data.Dh) && isempty(Fc.Connection)
            if all(temp == temp(1))
                % Simply take the average of the faces
                Sim.Area(1,Nd.index) = Fc.index;
                Sim.Area(2,Nd.index) = 1;
                break;
            end
        end
    end
end
end
end
end

% 1 and 2 static face pairs within body
if ~Sim.Area(2,Nd.index)
    % No two faces were found
    % Find a face that is static and not a connection
    count = 0;
    for Fc = Nd.Faces
        if Fc.Type ~= enumFType.Mix
            if isscalar(Fc.data.Dh) && isempty(Fc.Connection)
                count = count + 1;
            end
        end
    end
end
if count == 2
    for Fc = Nd.Faces
        if Fc.Type ~= enumFType.Mix
            if isscalar(Fc.data.Area) && isempty(Fc.Connection)
                Sim.Area(2,Nd.index) = Sim.Area(2,Nd.index) + 0.5*Fc.data.Area;
            end
        end
    end
end
if count == 1
    for Fc = Nd.Faces
        if Fc.Type ~= enumFType.Mix
            if isscalar(Fc.data.Area) && isempty(Fc.Connection)
                Sim.Area(2,Nd.index) = Sim.Area(2,Nd.index) + Fc.data.Area;
            end
        end
    end
end
end
end
end

```

```

        end

        %
        if ~Sim.Area(2,Nd.index)
            fprintf('XXX Deficiency in Node Face Calculation in Model XXX');
        end
    else
        % It is divided by horizontal planes or not divided,
        % ... simply take the radius of the shape
        Sim.Area(1,Nd.index) = 0;
        Sim.Area(2,Nd.index) = pi*Nd.xmax^2;
    end
end
end

if isempty(this.MechanicalSystem)
    Sim.MechanicalSystem = ...
        MechanicalSystem(this,this.Converters,[],...
            1,function_handle.empty);
else
    Sim.MechanicalSystem = ...
        MechanicalSystem(this,this.Converters,[],...
            this.MechanicalSystem.Inertia,this.MechanicalSystem.LoadFunction);
end

%% Defining Energy Statistics Handlers
progressbar('Defining Energy Statistics Handlers');

% Statistics
% Find all Solid Faces that go to the Environment
Sim.ToEnvironmentSolid = zeros(2,length(this.surroundings.Node.Faces));
Sim.ToEnvironmentGas = zeros(2,length(this.surroundings.Node.Faces));
nS = 1; nG = 1;
for Fc = this.surroundings.Node.Faces
    if Fc.Nodes(1).Type == enumNType.SN
        % It is a solid-environment face
        Sim.ToEnvironmentSolid(1,nS) = Fc.index;
        Sim.ToEnvironmentSolid(2,nS) = -1;
        nS = nS + 1;
    elseif Fc.Nodes(2).Type == enumNType.SN
        % It is a solid-environment face
        Sim.ToEnvironmentSolid(1,nS) = Fc.index;
        Sim.ToEnvironmentSolid(2,nS) = 1;
        nS = nS + 1;
    elseif Fc.Nodes(1).Type == enumNType.EN
        % It is a gas-environment face
        Sim.ToEnvironmentGas(1,nG) = Fc.index;
        Sim.ToEnvironmentGas(2,nG) = 1;
        nG = nG + 1;
    else
        % It is a gas-environment face
        Sim.ToEnvironmentGas(1,nG) = Fc.index;
        Sim.ToEnvironmentGas(2,nG) = -1;
        nG = nG + 1;
    end
end
Sim.ToEnvironmentSolid = Sim.ToEnvironmentSolid(:,1:nS-1);
Sim.ToEnvironmentGas = Sim.ToEnvironmentGas(:,1:nG-1);

% Find all Faces that go to a Source
isToSourceOrSink = false(1, length(this.Faces));
isSourceOrSink = false(1, length(this.Nodes));
for Nd = this.Nodes
    if Nd.Type == enumNType.SN && ...
        (strcmp(Nd.Body.matl.name, 'Constant Temperature') || (...
            isfield(Nd.data,'matl') && ...
            strcmp(Nd.data.matl.name, 'Constant Temperature')) ...
        )
        isSourceOrSink(Nd.index) = true;
    for Fc = Nd.Faces
        isToSourceOrSink(Fc.index) = ~isToSourceOrSink(Fc.index);
    end
end

```

```

        end
    end
end

temp = 1:length(this.Faces);
Subject_Faces = temp(isToSourceOrSink);
temp = 1:length(this.Nodes);
Subject_Nodes = temp(isSourceOrSink);

% Get Average Temperatures
T = mean(Sim.T(isSourceOrSink));
Sim.ToSource = zeros(2,length(this.Faces));
nSr = 1;
Sim.ToSink = zeros(2,length(this.Faces));
nSi = 1;
IsSource = false(1, length(this.Nodes));
if T > this.surroundings.Temperature
    IsSource(Subject_Nodes) = Sim.T(Subject_Nodes) >= T;
else
    IsSource(Subject_Nodes) = Sim.T(Subject_Nodes) > T;
end
for findex = Subject_Faces
    if isSourceOrSink(Sim.Fc_Nd(findex,1))
        if IsSource(Sim.Fc_Nd(findex,1))
            Sim.ToSource(1,nSr) = findex;
            Sim.ToSource(2,nSr) = -1;
            nSr = nSr + 1;
        else
            Sim.ToSink(1,nSi) = findex;
            Sim.ToSink(2,nSi) = -1;
            nSi = nSi + 1;
        end
    else
        if IsSource(Sim.Fc_Nd(findex,2))
            Sim.ToSource(1,nSr) = findex;
            Sim.ToSource(2,nSr) = 1;
            nSr = nSr + 1;
        else
            Sim.ToSink(1,nSi) = findex;
            Sim.ToSink(2,nSi) = 1;
            nSi = nSi + 1;
        end
    end
end
end
Sim.ToSource = Sim.ToSource(:,1:nSr-1);
Sim.ToSink = Sim.ToSink(:,1:nSi-1);
Sim.Sources = temp(IsSource);
Sim.Sinks = temp(and(isSourceOrSink, ~IsSource));

% identify shearing faces
% All Mixed Faces, All Solid Faces
if ~isempty(Sim.Fc_DynArea)
    ShuttleFaces = Sim.Fc_DynArea(1,:); % [Sim.Fc_DynArea(1,:) Sim.Fc_DynCond(1,:)];
else
    ShuttleFaces = zeros(1,0);
end
if ~isempty(Sim.Fc_DynCond)
    ShuttleFaces = [ShuttleFaces Sim.Fc_DynCond(1,:)];
end
% Exclude Gas Faces
Sim.ShuttleFaces = ShuttleFaces(ShuttleFaces>length(Sim.Fc_U));

% identify static faces
% All Mixed Faces, All Solid Faces
% Exclude Gas Faces
Sim.StaticFaces = 1:length(Sim.Fc_Cond);
Sim.StaticFaces(Sim.ShuttleFaces) = [];

Sim.ExergyLossShuttle = 0;
Sim.ExergyLossStatic = 0;

```

```

this.Simulations = Sim;

this.isStateDiscretized = true;

Sim.Fc_K12(isnan(Sim.Fc_K12)) = 1;
Sim.Fc_K21(isnan(Sim.Fc_K21)) = 1;
Sim.Dynamic(isnan(Sim.Dynamic)) = 0;

progressbar(1);
end

function [success] = Run(ME, runs)
success = false;
backup_path = ME.outputPath;
if nargin > 1
    tests = length(runs);
    ME.showLivePV = true;
    ME.showPressureAnimation = true;
    ME.recordPressure = true;
    ME.showTemperatureAnimation = true;
    ME.recordTemperature = true;
    ME.showVelocityAnimation = true;
    ME.recordVelocity = true;
    ME.showTurbulenceAnimation = true;
    ME.recordTurbulence = true;
    ME.showConductionAnimation = true;
    ME.recordConductionFlux = true;
    ME.showPressureDropAnimation = true;
    ME.recordPressureDrop = true;
    ME.recordOnlyLastCycle = true;
    ME.recordStatistics = true;
    for i = 1:length(runs)
        runs(i).isManual = false;
    end
else
    tests = 1;
    crun = struct(...
        'isManual',true,...
        'Model',ME.name,...
        'title',[ME.name ' Test- ' date], ...
        'NodeFactor',ME.deRefinementFactorInput);
end
for Nt = 1:tests
    if nargin > 1
        crun = runs(Nt);
    end
    % If it has a steady state end condition and only the last cycle is
    % ... important then use the Multi-Grid Formulation.
    useTrials = nargin > 1 && crun.SS == true && ME.recordOnlyLastCycle;
    ntrials = 1;

    [status] = mkdir('./Runs',crun.title);
    if status
        ME.outputPath = ['./Runs/' crun.title];
    else
        msgbox(['Please create file: ../Runs/' crun.title])
    end
    for trial = 1:ntrials
        % Only do warmup when starting from scratch
        do_warmup = (Nt == 1 && trial == 1);

        ME.resetDiscretization();

        %% Apply Geometry Modifications
        % Uniform Scale Modification
        if nargin > 1
            if isfield(crun,'Uniform_Scale')
                % Scale the connections
                for iGroup = ME.Groups
                    for iCon = iGroup.Connections
                        iCon.x = iCon.x*crun.Uniform_Scale;
                    end
                end
            end
        end
    end
end

```

```

        end
        % Scale the positions
        iGroup.Position.x = iGroup.Position.x*crun.Uniform_Scale;
        iGroup.Position.y = iGroup.Position.y*crun.Uniform_Scale;
    end

    % Scale the bridge offsets
    for iBridge = ME.Bridges
        iBridge.x = iBridge.x*crun.Uniform_Scale;
    end

    % Scale the mechanisms
    for iLRM = ME.Converters
        iLRM.Uniform_Scale(crun.Uniform_Scale);
    end

    % Scale the view window
    XL = get(ME.AxisReference, 'XLim');
    YL = get(ME.AxisReference, 'YLim');
    set(ME.AxisReference,'XLim', XL*crun.Uniform_Scale);
    set(ME.AxisReference,'YLim', YL*crun.Uniform_Scale);
end
end

%% Run
ME.update();

% Discretize according to the Multigrid Optimization
if useTrials
    if isfield(crun,'NodeFactor') && crun.NodeFactor ~= 1
        islast = true;
        ss_tolerance = 0.01;
    else
        if trial == ntrials % 3/3 or 2/2 or 1/1
            islast = true;
            ss_tolerance = 0.01;
        else
            islast = false;
            if ntrials - trial == 1 % 2/3 or 1/2
                crun.NodeFactor = crun.NodeFactor*0.1;
                ss_tolerance = 0.01;
            else % 1/3
                crun.NodeFactor = crun.NodeFactor*0.001;
                ss_tolerance = 0.025;
            end
        end
    end
end
else
    islast = true;
    ss_tolerance = 0.01;
end

ME.discretize(crun);

% If discretization was successful
if ME.isStateDiscretized
    % Apply Snapshot
    % ... Which Snapshot would the user like to use?
    if ~isempty(ME.SnapShots)
        names = cell(length(ME.SnapShots)+1,1);
        if nargin < 2
            % Have the user pick a starting Snap-Shot
            for i = 1:length(ME.SnapShots)
                names{i} = ME.SnapShots{i}.Name;
            end
            names{end} = '... From Scratch';
            [answer, selectionMade] = listdlg(...
                'PromptString','Select a SnapShot',...
                'ListString',names,...
                'SelectionMode','single');
        else

```



```

    % Try to find a snapshot with matching name
    selectionMade = true;
    found = false;
    for i = 1:length(ME.SnapShots)
        if strcmp(ME.SnapShots{i}.Name, crun.title)
            found = true;
            answer = i;
            break;
        end
    end

    % If it did not find a match then take the last one listed
    if ~found
        answer = length(ME.SnapShots);
    end

end

% Apply the snapshot if it is selected
if selectionMade && answer ~= length(names)
    SS = ME.SnapShots{answer};
    ME.assignSnapShot(SS);
end

end

if nargin > 1 && ...
    crun.movement_option == 'V' && ...
    crun.SS && ...
    ME.recordStatistics && ...
    ME.recordOnlyLastCycle
    dynamic = true;
    record_P_backup = ME.recordPressure;
    record_T_backup = ME.recordTemperature;
    record_t_backup = ME.recordTurbulence;
    ME.recordPressure = true;
    ME.recordTemperature = true;
    ME.recordTurbulence = true;
    crun.movement_option = 'C';
    ss_tolerance = 0.01;
else
    dynamic = false;
end

tic;
[ME.Results, success] = ME.Simulations(1).Run(...
    islast, do_warmup, ss_tolerance, crun);
if isempty(ME.Results)
    ME.CurrentSim(:) = [];
    ME.Results(:) = [];
    ME.resetDiscretization();
    return;
end

if dynamic
    % Reset Settings
    crun.movement_option = 'V';
    crun.set_Load = mean(ME.Results.Data.Power)/mean(ME.Results.Data.dA);
    % Save and Reload Snapshot
    ME.Results.getSnapShot(ME, 'Temp');
    ME.assignSnapShot(ME.SnapShots{end});
    ME.SnapShots(end) = [];
    ME.recordPressure = record_P_backup;
    ME.recordTemperature = record_T_backup;
    ME.recordTurbulence = record_t_backup;
    % Run
    [ME.Results, success] = ME.Simulations(1).Run(...
        islast, do_warmup, ss_tolerance, crun);
end
toc;

if ~success || isempty(ME.Results); return; end

```

```

% If it is a recording set then ready the display matrices and
% record everything
if islast
    % Calculate Node Locations
    if ME.showPressureAnimation || ...
        ME.showTemperatureAnimation || ...
        ME.showTurbulenceAnimation || ...
        ME.showConductionAnimation || ...
        ME.showPressureDropAnimation
    cpnts = cell(1,length(ME.Nodes));
    nodesleft = 0;
    for Nd = ME.Nodes
        if nodesleft == 0
            if isa(Nd.Body,'Body')
                iGroup = Nd.Body.Group;
                Rot = RotMatrix(iGroup.Position.Rot - pi/2);
                Trans = [iGroup.Position.x; iGroup.Position.y];
                AxisAligned = (iGroup.Position.Rot == pi/2);
                nodesleft = length(iGroup.Nodes)-1;
            else
                Rot = [1 0; 0 1];
                Trans = [0; 0];
                AxisAligned = true;
                % nodesleft = 0;
            end
        else
            nodesleft = nodesleft - 1;
        end
        % Corner points

        % Type 1 (static) ,4 (dynamic): Translation Only
        % ... [Type d1x d2x d3x cx ... ]
        % ... [ -- d1y d2y d3y cy ... ]
        % ... c is the center of the node
        % ... d1 is the diagonal between the center and top right corner
        % ... d2 is the diagonal between the center and the bottom right corner
        % ... d3 is the vector between centers of the ring, (0,0) if node is centered

        % Type 2: Stretching in One Direction
        % ... [Type cx d1x d2x d3x ... ]
        % ... [ -- cy d1y d2y d3y ... ]
        % ... c is the bottom left corner
        % ... d1 is the vector to the bottom right corner from c
        % ... d2 is the vector to the bottom right corner of the other side
        % ... ... (0,0) if the node is centered
        % ... d3 is the vector to the top left corner

        % Type 3: Movement of both ybounds
        % ... [Type d1x cx ... ]
        % ... [ -- d1y cy ... ]
        % ... [ -- d2x d3x ... ]
        % ... [ -- d2y d3y ... ]
        % ... c is the bottom left corner
        % ... d1 is the vector to the bottom right corner from c
        % ... d2 is the vector to the bottom right corner of the other side
        % ... ... (0,0) if the node is centered
        % ... d3 is the vector to the top left corner

        if isscalar(Nd.ymin)
            if isscalar(Nd.ymax)
                Type = 1;
                if Nd.xmin == 0
                    pnts = [Type Nd.xmax                Nd.xmax                0 0
; ...
                        0 (Nd.ymax-Nd.ymin)/2 -(Nd.ymax-Nd.ymin)/2 0 (Nd.ymax+Nd.ymin)/2];
                else
                    pnts = [Type (Nd.xmax-Nd.xmin)/2 (Nd.xmax-Nd.xmin)/2 -(Nd.xmin+Nd.xmax)
(Nd.xmax+Nd.xmin)/2; ...
                        0 (Nd.ymax-Nd.ymin)/2 -(Nd.ymax-Nd.ymin)/2 0
(Nd.ymax+Nd.ymin)/2];

```

```

end
% Rotate
if ~AxisAligned; pnts(:,2:5) = Rot*pnts(:,2:5); end
pnts(:,5) = pnts(:,5) + Trans;
else
Type = 2;
pnts = zeros(2,4+length(Nd.ymax));
% Nd.ymax is dynamic
if Nd.xmin == 0
pnts(:,1:4) = [Type -Nd.xmax 2*Nd.xmax 0; ...
0 Nd.ymin 0 0];
else
pnts(:,1:4) = [Type Nd.xmin Nd.xmax-Nd.xmin -Nd.xmax-Nd.xmin; ...
0 Nd.ymin 0 0];
end
pnts(1,5:end) = 0;
pnts(2,5:end) = Nd.ymax - pnts(2,2);
% Rotate
if ~AxisAligned; pnts(:,2:end) = Rot*pnts(:,2:end); end
pnts(:,2) = pnts(:,2) + Trans;
end
else
if isscalar(Nd.ymin)
Type = 2;
pnts = zeros(2,4+length(Nd.ymin));
% Nd.ymin is dynamic
if Nd.xmin == 0
pnts(:,1:4) = [Type -Nd.xmax 2*Nd.xmax 0; ...
0 Nd.ymin 0 0];
else
pnts(:,1:4) = [Type Nd.xmin Nd.xmax-Nd.xmin -Nd.xmax-Nd.xmin; ...
0 Nd.ymin 0 0];
end
pnts(1,5:end) = 0;
pnts(2,5:end) = Nd.ymin - pnts(2,2);
% Rotate
if ~AxisAligned; pnts(:,2:end) = Rot*pnts(:,2:end); end
pnts(:,2) = pnts(:,2) + Trans;
else
if isfield(Nd.data,'matl'); matl = Nd.data.matl;
else; matl = Nd.Body.matl;
end
if matl.Phase == enumMaterial.Solid
% ... [Type dlx d2x d3x cx ... ]
% ... [ -- dly d2y d3y cy ... ]
Type = 4; % Stretching is impossible
pnts = zeros(2,4+length(Nd.ymin));
if Nd.xmin == 0
pnts(:,1:4) = [Type Nd.xmax Nd.xmax-Nd.xmin Nd.xmax-Nd.xmin; ...
0 (Nd.ymin(1)-Nd.ymin(1))/2 - (Nd.ymin(1)-Nd.ymin(1))/2 0];
x = 0;
else
pnts(:,1:4) = [Type (Nd.xmax-Nd.xmin)/2 (Nd.xmax-Nd.xmin)/2
-(Nd.xmin+Nd.xmax); ... (Nd.xmax+Nd.xmin)/2 ; ...
0 (Nd.ymin(1)-Nd.ymin(1))/2 - (Nd.ymin(1)-Nd.ymin(1))/2 0];
x = (Nd.xmax+Nd.xmin)/2;
end
pnts(1,5:end) = x;
pnts(2,5:end) = (Nd.ymin(1)+Nd.ymin(1))/2;
% Rotate
if ~AxisAligned; pnts(:,2:end) = Rot*pnts(:,2:end); end
pnts(:,5:end) = pnts(:,5:end) + Trans;
else
Type = 3; % Stretching is very probable
pnts = zeros(4,2+length(Nd.ymin));
if Nd.xmin == 0
pnts(:,1:2) = [Type 2*Nd.xmax; ...
0 0 ; ...
0 0 ; ...
];
end
end
end

```

```

        0    0    ];
        x = -Nd.xmax;
    else
        pnts(:,1:2) = [Type Nd.xmax-Nd.xmin ; ...
            0    0 ; ...
            0    -(Nd.xmax+Nd.xmin); ...
            0    0 ];
        x = Nd.xmin;
    end
    pnts(1,3:end) = x;
    pnts(2,3:end) = Nd.ymin;
    pnts(3,3:end) = 0;
    pnts(4,3:end) = (Nd.ymax-Nd.ymin);
    % Rotate
    if ~AxisAligned
        pnts(1:2,2:end) = Rot*pnts(1:2,2:end);
        pnts(3:4,2:end) = Rot*pnts(3:4,2:end);
    end
    % Translate
    pnts(1:2,3:end) = pnts(1:2,3:end) + Trans;
end
end
end
cpnts{Nd.index} = pnts;
end
end
end

% Calculate Face Locations and directions
if isfield(ME.Results.Data,'U') && ME.showPressureDropAnimation
    if isfield(ME.Results.Data,'U')
        fpnts = cell(1,size(ME.Results.Data.U,1));
    end
    % Define X,Y,Nx,Ny
    maxIndex = length(ME.Simulations.Fc_U);
    if ~isempty(ME.Faces)
        if isa(ME.Faces(1).Nodes(1).Body,'Body')
            iGroup = ME.Faces(1).Nodes(1).Body.Group;
        else
            iGroup = ME.Faces(1).Nodes(2).Body.Group;
        end
        Rot = RotMatrix(iGroup.Position.Rot - pi/2);
        Trans = [iGroup.Position.x; iGroup.Position.y];
        for Fc = ME.Faces
            if Fc.index <= maxIndex
                if isa(Fc.Nodes(1).Body,'Environment')
                    Rot = RotMatrix(0);
                    Trans = [0; 0];
                else
                    if iGroup ~= Fc.Nodes(1).Body.Group
                        iGroup = Fc.Nodes(1).Body.Group;
                        Rot = RotMatrix(iGroup.Position.Rot - pi/2);
                        Trans = [iGroup.Position.x; iGroup.Position.y];
                    end
                end
            end
            i = Fc.index;
            if Fc.Nodes(1).Body == Fc.Nodes(2).Body
                if Fc.Nodes(1).xmax == Fc.Nodes(2).xmin
                    % Aligned horizontally
                    x = Fc.Nodes(1).xmax;
                    y = (Fc.Nodes(1).ymin + Fc.Nodes(1).ymax)/2;
                    Nx = 1;
                    Ny = 0;
                elseif Fc.Nodes(1).xmin == Fc.Nodes(2).xmax
                    % Aligned horizontally
                    x = Fc.Nodes(1).xmin;
                    y = (Fc.Nodes(1).ymin + Fc.Nodes(1).ymax)/2;
                    Nx = -1;
                    Ny = 0;
                elseif abs(Fc.Nodes(1).ymax(1) - Fc.Nodes(2).ymin(1)) < ...
                    abs(Fc.Nodes(1).ymin(1) - Fc.Nodes(2).ymax(1))
                    % Aligned Vertically

```

```

        x = (Fc.Nodes(1).xmin + Fc.Nodes(1).xmax)/2;
        y = Fc.Nodes.yymax;
        Nx = 0;
        Ny = 1;
    else
        % Aligned Vertically
        x = (Fc.Nodes(1).xmin + Fc.Nodes(1).xmax)/2;
        y = Fc.Nodes.ymin;
        Nx = 0;
        Ny = -1;
    end
else
    if Fc.Nodes(1).xmax == Fc.Nodes(2).xmin
        % Aligned horizontally
        x = Fc.Nodes(1).xmax; % Done
        y = getCenterOfOverlapRegion(...
            Fc.Nodes(1).ymin,...
            Fc.Nodes(2).ymin,...
            Fc.Nodes(1).ymax,...
            Fc.Nodes(2).ymax);
        Nx = 1; % Done
        Ny = 0; % Done
    elseif Fc.Nodes(1).xmin == Fc.Nodes(2).xmax
        % Aligned horizontally
        x = Fc.Nodes(1).xmin; % Done
        y = getCenterOfOverlapRegion(...
            Fc.Nodes(1).ymin,...
            Fc.Nodes(2).ymin,...
            Fc.Nodes(1).ymax,...
            Fc.Nodes(2).ymax);
        Nx = -1; % Done
        Ny = 0; % Done
    elseif abs(Fc.Nodes(1).ymax(1) - Fc.Nodes(2).ymin(1)) < ...
        abs(Fc.Nodes(1).ymin(1) - Fc.Nodes(2).ymax(1))
        % Aligned Vertically
        x = getCenterOfOverlapRegion(...
            Fc.Nodes(1).xmin,...
            Fc.Nodes(2).xmin,...
            Fc.Nodes(1).xmax,...
            Fc.Nodes(2).xmax);
        y = Fc.Nodes(1).ymax; % Done
        Nx = 0; % Done
        Ny = 1; % Done
    else
        % Aligned Vertically
        x = getCenterOfOverlapRegion(...
            Fc.Nodes(1).xmin,...
            Fc.Nodes(2).xmin,...
            Fc.Nodes(1).xmax,...
            Fc.Nodes(2).xmax);
        y = Fc.Nodes(1).ymin; % Done
        Nx = 0; % Done
        Ny = -1; % Done
    end
end
    if isscalar(y)
        fpnts{i} = Rot*[Nx x; Ny y] + [[0;0] Trans];
    else
        fpnts{i} = [Rot*[Nx; Ny] Rot*[x(ones(1,length(y))); y]+Trans];
    end
end
end
end
end

% Calculate Solid Body Boundaries
n = 0;
for iGroup = ME.Groups
    for iBody = iGroup.Bodies
        if iBody.matl.Phase == enumMaterial.Solid
            n = n + 1;

```

```

        end
    end
end
bpnts = cell(1,n);
n = 1;
for iGroup = ME.Groups
    Rot = RotMatrix(iGroup.Position.Rot - pi/2);
    Trans = [iGroup.Position.x; iGroup.Position.y];
    for iBody = iGroup.Bodies
        if iBody.matl.Phase == enumMaterial.Solid
            [~,~,x1,x2] = iBody.limits(enumOrient.Vertical);
            [y1,y2,~,~] = iBody.limits(enumOrient.Horizontal);
            if isscalar(y1) %&& isscalar(y2)
                if x1 == 0
                    bpnts{n} = Rot*[-x2 x2 x2 -x2; y1 y1 y2 y2] + Trans;
                else
                    bpnts{n} = Rot*[x1 x2 x2 x1; y1 y1 y2 y2] + Trans;
                    n = n + 1;
                    bpnts{n} = Rot*[-x1 -x2 -x2 -x1; y1 y1 y2 y2] + Trans;
                end
            else
                bpnts{n} = zeros(2,4,length(y1));
                if x1 == 0
                    for i = 1:length(y1)
                        bpnts{n}(:, :, i) = ...
                            Rot*[-x2 x2 x2 -x2; y1(i) y1(i) y2(i) y2(i)] + Trans;
                    end
                else
                    for i = 1:length(y1)
                        bpnts{n}(:, :, i) = ...
                            Rot*[x1 x2 x2 x1; y1(i) y1(i) y2(i) y2(i)] + Trans;
                    end
                    n = n + 1;
                    bpnts{n} = zeros(2,4,length(y1));
                    for i = 1:length(y1)
                        bpnts{n}(:, :, i) = ...
                            Rot*[-x1 -x2 -x2 -x1; y1(i) y1(i) y2(i) y2(i)] + Trans;
                    end
                end
            end
            n = n + 1;
        end
    end
end
end

% Animate
frate = ME.animationFrameTime;
if isfield(ME.Results.Data,'P') %&& ME.showPressureAnimation
    ME.Results.animateNode('P',cpnts,bpnts,frate,[],[],crun.title);
end
if isfield(ME.Results.Data,'T') %&& ME.showTemperatureAnimation
    ME.Results.animateNode('T',cpnts,bpnts,frate,[],[],crun.title);
end
if isfield(ME.Results.Data,'U') %&& ME.showVelocityAnimation
    ME.Results.animateFace('U',fpnts,bpnts,frate,[],[],crun.title);
end
if isfield(ME.Results.Data,'turb') %&& ME.showTurbulenceAnimation
    ME.Results.animateNode('turb',cpnts,bpnts,frate,[],[],crun.title);
end
if isfield(ME.Results.Data,'cond') %&& ME.showConductionAnimation
    ME.Results.animateNode('cond',cpnts,bpnts,frate,[],[],crun.title);
end
if isfield(ME.Results.Data,'dP') %&& ME.showPressureDropAnimation
    ME.Results.animateNode('dP',cpnts,bpnts,frate,[],[],crun.title);
end
end

% Ask if the user would like to save a snapshot
if nargin > 1
    if ~ME.Simulations(1).stop
        % Remove the snapshot as it will be replaced now
    end
end

```

```

        for i = 1:length(ME.SnapShots)
            if strcmp(ME.SnapShots{i}.Name, crun.title)
                ME.SnapShots(i) = [];
                break;
            end
        end
        ME.Results.getSnapshot(ME, crun.title);
        saveME(ME);
    end
else
    response = questdlg('Would you like to save a Snapshot?', ...
        'Save Snapshot', 'Yes', 'No', 'Yes');
    if strcmp(response, 'Yes')
        ME.Results.getSnapshot(ME, getProperName('Snapshot'));
    end
end
end

%% Undo Geometry Modifications
% Uniform Scale Modification
if nargin > 1
    if isfield(crun, 'Uniform_Scale')
        % Scale the connections
        for iGroup = ME.Groups
            for iCon = iGroup.Connections
                iCon.x = iCon.x/crun.Uniform_Scale;
            end
            % Scale the positions
            iGroup.Position.x = iGroup.Position.x/crun.Uniform_Scale;
            iGroup.Position.y = iGroup.Position.y/crun.Uniform_Scale;
        end

        % Scale the bridge offsets
        for iBridge = ME.Bridges
            iBridge.x = iBridge.x/crun.Uniform_Scale;
        end

        % Scale the mechanisms
        for iLRM = ME.Converters
            iLRM.Uniform_Scale(1/crun.Uniform_Scale);
        end

        % Scale the view window
        set(ME.AxisReference, ...
            'XLim', get(ME.AxisReference, 'XLim')/crun.Uniform_Scale);
        set(ME.AxisReference, ...
            'YLim', get(ME.AxisReference, 'YLim')/crun.Uniform_Scale);
    end

    % To save the modified geometry
    saveME(ME);
end
end
end
ME.outputPath = backup_path;
ME.CurrentSim(:) = [];
ME.Results(:) = [];
ME.resetDiscretization();
end

function assignSnapshot(ME, SS)
    Sim = ME.Simulations;
    for iGroup = ME.Groups
        for iBody = iGroup.Bodies
            for BData = SS.Data
                if iBody.ID == BData.ID
                    if applyBody(BData, iBody)
                        if iBody.matl.Phase == enumMaterial.Solid
                            for Nd = iBody.Nodes
                                i = Nd.index;
                                if isnan(Nd.data.T)

```

```

        fprintf('err detected');
    else
        Sim.T(i) = Nd.data.T;
    end
end
else
    for Nd = iBody.Nodes
        i = Nd.index;
        if isfield(Nd.data,'matl')
            matl = Material(Nd.data.matl.name);
        else
            matl = Material(Nd.Body.matl.name);
        end
        if i <= length(Sim.P)
            if isfield(Nd.data,'P') && ~isnan(Nd.data.P)
                P = Nd.data.P;
            else
                P = ME.enginePressure;
                for j = 1:length(Sim.Regions)
                    if any(Sim.Regions{j} == i)
                        if Sim.isEnvironmentRegion(j)
                            P = Sim.P(end);
                        end
                        break;
                    end
                end
            end
            if isfield(Nd.data,'T')
                if isnan(Nd.data.T)
                    fprintf('err detected');
                else
                    Sim.T(i) = Nd.data.T;
                end
            end
            if isfield(Nd.data,'Turb')
                if isnan(Nd.data.Turb)
                    fprintf('err detected');
                else
                    Sim.turb(i) = Nd.data.Turb;
                end
            end
            if ~isnan(P)
                vol = Nd.vol();
                % Need to figure out what gas constant to
                % ... use
                Rgas = matl.R;
                Sim.m(i) = P*vol/(Rgas*Sim.T(i));
            end
        else
            if isnan(Nd.data.T)
                fprintf('err detected');
            else
                Sim.T(i) = Nd.data.T;
            end
        end
        if matl.Phase == enumMaterial.Gas
            Sim.u(i) = matl.initialInternalEnergy(Sim.T(i));
        end
    end
end
end
end
break;
end
end
end
end
end
end

function saveME(Model)
    Model.Faces(:) = [];
    Model.Nodes(:) = [];
end

```



```

Model.Simulations(:) = [];
Model.CurrentSim(:) = [];
Model.Results(:) = [];
Model.PressureContacts(:) = [];
Model.ShearContacts(:) = [];
for iPv = Model.PVoutputs
    iPv.reset();
end
for iSense = Model.Sensors
    if ~isempty(iSense)
        iSense.reset();
        iSense.GUIObjects(:) = [];
    end
end
backupAxis = Model.AxisReference;
Model.AxisReference(:) = [];
for iGroup = Model.Groups
    for iBody = iGroup.Bodies
        iBody.GUIObjects(:) = [];
        iBody.Nodes(:) = [];
        iBody.Faces(:) = [];
        if ~isempty(iBody.Matrix)
            iBody.Matrix.Nodes(:) = [];
            iBody.Matrix.Faces(:) = [];
        end
    end
    for iCon = iGroup.Connections
        iCon.GUIObjects(:) = [];
        iCon.Faces(:) = [];
        iCon.NodeContacts(:) = [];
    end
    iGroup.GUIObjects(:) = [];
end
for iBridge = Model.Bridges
    iBridge.GUIObjects(:) = [];
    iBridge.Faces(:) = [];
end
save(['Saved Files\' Model.name '.mat'],'Model');
Model.AxisReference = backupAxis;
fprintf('Model Saved.\n');
end

%% Interface / Find stuff
function FindGroup(this,Pos)
    TheGroup = this.findNearestGroup(Pos,inf);
    this.HighLight(TheGroup);
end
function distributeGroup(this, GroupSpacing)
    % Take existing horizontal order and distribute
    for i = 1:length(this.Groups)
        for j = i+1:length(this.Groups)
            if this.Groups(j).Position.x < this.Groups(i).Position.x
                tempGroup = this.Groups(i);
                this.Groups(i) = this.Groups(j);
                this.Groups(j) = tempGroup;
            end
        end
    end
    x = 0;
    for iGroup = this.Groups
        iGroup.Position.x = x;
        dim1 = RotMatrix(iGroup.Position.Rot-pi/2)*[iGroup.Width*2; iGroup.Height];
        dim2 = RotMatrix(iGroup.Position.Rot-pi/2)*[iGroup.Width*2; -iGroup.Height];
        x = x + max([dim1(1) dim2(1)])+GroupSpacing;
    end
end
function [names, objects] = findNearest(this,Pnt,Tolerance)
    objects = cell(0);
    names = cell(0);
    % Find, within a radius of confidence, the nearest...
    % Body, Group, Connection, Bridge and Leak Connection

```

```

Tolerance = Tolerance^2;
index = 1;
%% Group
if isempty(this.ActiveGroup)
    obj = this.findNearestGroup(Pnt,Tolerance);
    if ~isempty(obj)
        objects{index} = obj;
        names{index} = obj.name;
        index = index + 1;
    end
    TheGroup = obj;
else
    TheGroup = this.ActiveGroup;
    objects{index} = TheGroup;
    names{index} = TheGroup.name;
    index = index + 1;
end

%% Body
mindist = Tolerance;
Pntmod = (RotMatrix(pi/2 - TheGroup.Position.Rot)*Pnt') - ...
    [TheGroup.Position.x; TheGroup.Position.y];
for iBody = TheGroup.Bodies
    % Establish Rectangle of iBody
    [~,~,x1,x2] = iBody.limits(enumOrient.Vertical);
    [~,~,y1,y2] = iBody.limits(enumOrient.Horizontal);

    R.Width = x2-x1;
    R.Height = y2-y1;
    R.Cx = (x1+x2)/2;
    R.Cy = (y1+y2)/2;
    dist = Dist2Rect(Pntmod(1), Pntmod(2), R.Cx, R.Cy, R.Width, R.Height);
    if dist < mindist
        mindist = dist;
        TheBody = iBody;
    else
        R.Cx = -R.Cx;
        dist = Dist2Rect(...
            Pntmod(1), Pntmod(2), R.Cx, R.Cy, R.Width, R.Height);
        if dist < mindist
            mindist = dist;
            TheBody = iBody;
        end
    end
end
if mindist < Tolerance
    objects{index} = TheBody;
    names{index} = TheBody.name;
    index = index + 1;
end

%% Connection
mindist = Tolerance;
Pntmod = (RotMatrix(pi/2 - TheGroup.Position.Rot)*Pnt') - ...
    [TheGroup.Position.x; TheGroup.Position.y];
for iConnection = TheGroup.Connections
    % Find nearest Connection
    switch iConnection.Orient
        case enumOrient.Vertical
            % Two lines to test
            if abs(Pntmod(1) - iConnection.x) < mindist
                mindist = abs(Pntmod(1) - iConnection.x);
                TheConnection = iConnection;
            end
            if abs(Pntmod(1) + iConnection.x) < mindist
                mindist = abs(Pntmod(1) + iConnection.x);
                TheConnection = iConnection;
            end
        case enumOrient.Horizontal
            % One line to test
            if abs(Pntmod(2) - iConnection.x) < mindist

```

```

        mindist = abs(Pntmod(2)-iConnection.x);
        TheConnection = iConnection;
    end
end
end
if mindist < Tolerance
    objects{index} = TheConnection;
    names{index} = TheConnection.name;
    index = index + 1;
end

%% Bridge
mindist = Tolerance;
for iBridge = this.Bridges

end

%% Leak Connection
mindist = Tolerance;
for iLeakCon = this.LeakConnections

end
end
function [TheGroup] = findNearestGroup(this,Pos,Tolerance)
    try Pnt = Pos(1,1:2);
    catch
        try Pnt = Pos(1:2,1)';
        catch; msgbox('Group Not Found due to improper input coordinates');
        end
    end
    [iBody, mindist] = this.findNearestBody(Pnt,Tolerance);
    for iGroup = this.Groups
        if isempty(iGroup.Bodies)
            if mindist > Dist2Rect(Pnt(1),Pnt(2),iGroup.Position.x,iGroup.Position.y,0,0)
                TheGroup = iGroup;
                return;
            end
        end
    end
    if ~isempty(iBody)
        TheGroup = iBody.Group;
    else
        TheGroup = this.Groups(1);
    end
end
function [TheBody, mindist] = findNearestBody(this,Pnt,Tolerance)
    mindist = Tolerance;
    TheBody = Body.empty;
    for iGroup = this.Groups
        Pntmod = (RotMatrix(pi/2 - iGroup.Position.Rot)*Pnt') - ...
            [iGroup.Position.x; iGroup.Position.y];
        for iBody = iGroup.Bodies
            % Establish Rectangle of iBody
            [~,~,x1,x2] = iBody.limits(enumOrient.Vertical);
            [~,~,y1,y2] = iBody.limits(enumOrient.Horizontal);

            R.Width = x2-x1;
            R.Height = y2-y1;
            R.Cx = (x1+x2)/2;
            R.Cy = (y1+y2)/2;
            dist = Dist2Rect(Pntmod(1),Pntmod(2),R.Cx,R.Cy,R.Width,R.Height);
            if dist < mindist
                mindist = dist;
                TheBody = iBody;
            else
                R.Cx = -R.Cx;
                dist = Dist2Rect(...
                    Pntmod(1),Pntmod(2),R.Cx,R.Cy,R.Width,R.Height);
                if dist < mindist
                    mindist = dist;
                    TheBody = iBody;
                end
            end
        end
    end
end

```

```

        end
    end
end
end
end
function [names, objects] = findFrames(this)
    for i = length(this.RefFrames):-1:1
        names{i} = this.RefFrames(i).name;
        objects{i} = this.RefFrames(i);
    end
end
end

%% Graphics
% Tests
function isInWindow = inWindow(this,pnt1,pnt2)
    if nargin == 2
        if isempty(this.AxisReference)
            this.AxisReference = gca;
            axes = this.AxisReference;
        else
            axes = this.AxisReference;
        end
        xlim = axes.XLim;
        ylim = axes.YLim;
        isInWindow = pnt1.x < xlim(2) && pnt1.x > xlim(1) && ...
            pnt1.y < ylim(2) && pnt1.y > ylim(1);
    elseif nargin == 3
        xlim = this.AxisReference.XLim;
        ylim = this.AxisReference.YLim;
        isInWindow = pnt1.x < xlim(2) && pnt1.x > xlim(1) && ...
            pnt1.y < ylim(2) && pnt1.y > ylim(1) && ...
            pnt2.x < xlim(2) && pnt2.x > xlim(1) && ...
            pnt2.y < ylim(2) && pnt2.y > ylim(1);
    end
end
end
function showOptions = produceShowOptions(this,showOptions)
    if nargin > 1 && length(showOptions) == 9
        this.showGroups = showOptions(1); % Groups
        if ~showOptions(2) && showOptions(2) ~= this.showBodies
            for iGroup = this.Groups
                for iBody = iGroup.Bodies
                    iBody.removeFromFigure(this.AxisReference);
                end
            end
        end
        this.showBodies = showOptions(2); % Bodies
        if ~showOptions(3) && showOptions(3) ~= this.showConnections
            for iGroup = this.Groups
                for iCon = iGroup.Connections
                    iCon.removeFromFigure(this.AxisReference);
                end
            end
        end
        this.showConnections = showOptions(3); % Connections
        if ~showOptions(4) && showOptions(4) ~= this.showLeaks
            for iLeak = this.LeakConnections
                iLeak.removeFromFigure(this.AxisReference);
            end
        end
        this.showLeaks = showOptions(4); % Leaks
        if ~showOptions(5) && showOptions(5) ~= this.showBridges
            for iBridge = this.Bridges
                iBridge.removeFromFigure(this.AxisReference);
            end
        end
        this.showBridges = showOptions(5); % Bridges
        % Already deleted
        this.showInterConnections = showOptions(6); % Node Connections
        % Already deleted
        this.showEnvironmentConnections = showOptions(7); % Environment Surround
        % Already deleted
    end
end

```

```

        this.showBodyGhosts = showOptions(8); % Motion Ghosts
        % ?????
        this.showNodes = showOptions(9); % Node Outlines
    elseif nargin > 1 && ~isempty(showOptions)
        fprintf('XXX showOptions in "Model.show" should be a vector of length 9 containing
logical show conditions XXX\n');
        return;
    else
        % Define showOptions;
        showOptions = zeros(8,1);
        showOptions(1) = this.showGroups;
        showOptions(2) = this.showBodies;
        showOptions(3) = this.showConnections;
        showOptions(4) = this.showLeaks;
        showOptions(5) = this.showBridges;
        showOptions(6) = this.showInterConnections;
        showOptions(7) = this.showEnvironmentConnections;
        showOptions(8) = this.showBodyGhosts;
        showOptions(9) = this.showNodes;
    end
end

% Highlighting and Selecting
function ActiveGroup = get.ActiveGroup(this)
    ActiveGroup = [];
    for obj = this.Selection
        if isa(obj{1},'Group')
            ActiveGroup = obj{1};
            return;
        end
    end
end

function switchHighLightedGroup(this,otherGroup)
    update(this);
    if ~isempty(otherGroup) && isvalid(otherGroup)
        for i = 1:length(this.Selection)
            if isa(this.Selection{i},'Group')
                this.Selection{i}.isActive = false;
                this.Selection{i} = otherGroup;
                otherGroup.isActive = true;
            end
        end
    end
end

function switchHighLighting(this,NewHighlightedObjects)
    update(this);
    this.clearHighLighting();
    for iObj = NewHighlightedObjects
        iObj.isActive = true;
        this.Selection{end+1} = iObj;
    end
end

function HighLight(this,HighlightedObjects)
    update(this);
    for iObj = HighlightedObjects
        iObj.isActive = true;
        this.Selection{end+1} = iObj;
    end
end

function clearHighLighting(this)
    update(this);
    i = 1; j = 0;
    for iObj = this.Selection
        if ~isa(iObj,'Group')
            iObj{1}.isActive = false; %#ok<FXSET>
        else; j = i;
        end
        i = i + 1;
    end
    if j > 0; this.Selection = {this.Selection{j}};
    else; this.Selection = cell(0);
end

```

```

end
end

% Bulk Display
function XLim = getXLim(this)
    XLim = [inf -inf];
    for iGroup = this.Groups
        w = iGroup.Width;
        h = iGroup.Height;
        dx = w/2*sin(iGroup.Position.Rot);
        dy = h*cos(iGroup.Position.Rot);
        lim = iGroup.Position.x + [dx dx+dy -dx -dx+dy];
        limmx = max(lim);
        limmn = min(lim);
        if limmx > XLim(2); XLim(2) = limmx; end
        if limmn < XLim(1); XLim(1) = limmn; end
    end
end

function YLim = getYLim(this)
    YLim = [inf -inf];
    for iGroup = this.Groups
        w = iGroup.Width;
        h = iGroup.Height;
        dx = w/2*cos(iGroup.Position.Rot);
        dy = h*sin(iGroup.Position.Rot);
        lim = iGroup.Position.y + [dx dx+dy -dx -dx+dy];
        limmx = max(lim);
        limmn = min(lim);
        if limmx > YLim(2); YLim(2) = limmx; end
        if limmn < YLim(1); YLim(1) = limmn; end
    end
end

function removeStaticFromFigure(this)
    if ~isempty(this.StaticGUIObjects)
        children = get(this.AxisReference, 'Children');
        for j = 1:length(this.StaticGUIObjects)
            if isgraphics(this.StaticGUIObjects(j))
                for i = length(children):-1:1
                    if isgraphics(children(i)) && children(i) == this.StaticGUIObjects(j)
                        children(i).delete;
                        break;
                    end
                end
            end
        end
        this.StaticGUIObjects = [];
    end
end

function removeDynamicFromFigure(this)
    if ~isempty(this.DynamicGUIObjects)
        children = get(this.AxisReference, 'Children');
        for j = 1:length(this.DynamicGUIObjects)
            if isgraphics(this.DynamicGUIObjects(j))
                for i = length(children):-1:1
                    if isgraphics(children(i)) && children(i) == this.DynamicGUIObjects(j)
                        children(i).delete;
                        break;
                    end
                end
            end
        end
        this.DynamicGUIObjects = [];
    end
end

function removeGhostFromFigure(this)
    if ~isempty(this.GhostGUIObjects)
        children = get(this.AxisReference, 'Children');
        for obj = this.GhostGUIObjects
            if isgraphics(obj)
                for i = length(children):-1:1
                    if isgraphics(children(i)) && children(i) == obj

```

```

        children(i).delete;
        break;
    end
end
end
end
this.DynamicGUIObjects = [];
end
end
function bringGhostToFront(this)
    if ~isempty(this.GhostGUIObjects)
        children = get(this.AxisReference,'Children');
        END = length(children);
        for obj = this.GhostGUIObjects
            if isgraphics(obj)
                for i = END:-1:1
                    if isgraphics(children(i)) && children(i) == obj
                        uistack(obj,'top');
                        break;
                    end
                end
            end
        end
    end
end
end
function show(this,showOptions)
    if this.isChanged; this.update(); end
    this.removeStaticFromFigure();
    this.removeGhostFromFigure();
    if nargin > 1
        showOptions = this.produceShowOptions(showOptions);
    else
        showOptions = this.produceShowOptions();
    end
    %showOptions =
[inputshowGroup,inputshowBodies,inputshowConnections,ishowLeaks,ishowBridges,ishowIntCon,ishowEnv
irCon]
    % Fig = get(this.AxisReference,'parent');
    % hP = pan(Fig);
    % Go down through the hierarchy
    for iGroup = this.Groups
        %
show(this,CODE,AxisReference,Inc,showGroups,showBodies,showConnections,showLeaks,showInterConnect
ions,showEnvironmentConnections)
        iGroup.show('all',this.AxisReference,0,showOptions);
        % showGroups showBodies showConnections showLeaks showBridges showInterConnections
showEnvironmentConnections]
    end
    if this.showInterConnections || this.showNodes
        if ~this.isDiscretized()
            crun = struct('Model',this.name,...
                'title',[this.name ' test: ' date],...
                'rpm',this.engineSpeed,...
                'NodeFactor',this.deRefinementFactorInput);
            this.discretize(crun);
            if ~this.isDiscretized()
                fprintf('XXX No Nodes generated. XXX\n');
            end
        end
        n = length(this.Nodes);
        if n ~= 0
            nodeCenter(n) = Pnt2D(0,0);
            for iNode = this.Nodes
                nodeCenter(iNode.index) = iNode.minCenterCoords;
                isVis(iNode.index) = this.inWindow(nodeCenter(iNode.index));
            end
        end
    end
end
end
if this.showInterConnections % Show Inter-Node Connections
    if this.isDiscretized()

```

```

% Make array of Node Centers
% Count nodes, stored in Groups
n = length(this.Nodes);
if n ~= 0
    % Make array of face coords
    % Count faces, stored in Model
    n = length(this.Faces);
    faceCoord = zeros(4,n);
    n = 1;
    % Take each face, assess whether it is active, then record
    for iFace = this.Faces
        if ~(iFace.Nodes(1).Type == enumNType.EN || ...
            iFace.Nodes(2).Type == enumNType.EN)
            if iFace.Nodes(2).index < 1
                fprintf('XXX error XXX');
            end
            if isVis(iFace.Nodes(1).index) && ...
                isVis(iFace.Nodes(2).index)
                c1 = nodeCenter(iFace.Nodes(1).index);
                c2 = nodeCenter(iFace.Nodes(2).index);
                faceCoord(:,n) = [c1.x,c2.x,c1.y,c2.y];
                n = n + 1;
            end
        end
    end
    n = n - 1;

    % Plot
    nT = 3*n;
    xData = NaN(nT,1);
    yData = NaN(nT,1);
    ind = 1;
    for i = 1:3:nT-2
        xData(i:i+1) = faceCoord(1:2,ind);
        yData(i:i+1) = faceCoord(3:4,ind);
        ind = ind + 1;
    end
    if isempty(this.StaticGUIObjects)
        this.StaticGUIObjects = line(xData,yData,'Color',[0 1 0]);
    else
        this.StaticGUIObjects(end+1:end+length(this.Faces)) = line(xData,yData,'Color',[0 1
0]);
    end
end
end
end
if this.showNodes
    if this.isDiscretized()
        % Make array of Node Centers
        % Count nodes, stored in Groups
        n = length(this.Nodes);
        if n ~= 0
            % Plot
            xData = NaN(n,1);
            yData = NaN(n,1);
            j = 1;
            for nd = this.Nodes
                if isVis(nd.index)
                    c1 = nodeCenter(nd.index);
                    xData(j) = c1.x;
                    yData(j) = c1.y;
                    j = j + 1;
                end
            end
            if isempty(this.StaticGUIObjects)
                this.StaticGUIObjects = plot(xData,yData,'o',...
                    'MarkerSize',2,...
                    'MarkerEdgeColor',[0 0 1]);
            else
                this.StaticGUIObjects(end+1) = ...
                    plot(xData,yData,'o',...

```



```

        'MarkerSize',2,...
        'MarkerEdgeColor',[0 0 1]);
    end
end
end
if this.showBridges
    for iBridge = this.Bridges
        iBridge.show(this.AxisReference);
    end
else
    for iBridge = this.Bridges
        iBridge.removeFromFigure(this.AxisReference);
    end
end
if this.showLeaks
    for iLeak = this.LeakConnections
        iLeak.show(this.AxisReference);
    end
else
    for iLeak = this.LeakConnections
        iLeak.removeFromFigure(this.AxisReference);
    end
end
if this.showSensors
    if ~isempty(this.Sensors)
        for iSensor = this.Sensors
            iSensor.show(this.AxisReference);
        end
    end
else
    if ~isempty(this.Sensors)
        for iSensor = this.Sensors
            iSensor.removeFromFigure(this.AxisReference);
        end
    end
end
if this.showBodyGhosts
    this.bringGhostToFront();
end
end
function Animate(this,showOptions)
    if this.isChanged; this.update(); end
    this.removeStaticFromFigure();
    this.removeGhostFromFigure();
    %showOptions =
[inputshowGroup,inputshowBodies,inputshowConnections,ishowLeaks,ishowBridges,ishowIntCon,ishowEnv
irCon]
    if nargin > 1
        showOptions = this.produceShowOptions(showOptions);
    else
        showOptions = this.produceShowOptions();
    end

    % Don't show connections in animation
    showOptions(3) = false;

    cla;
    % Set Screen to be constant dimensions
    ReferenceAxis = gca;
    mode = get(ReferenceAxis,'XLimMode');
    set(ReferenceAxis,'XLimMode','manual');
    set(ReferenceAxis,'YLimMode','manual');
    set(ReferenceAxis,'ZLimMode','manual');

    % Initialize
    t = cputime;
    FrameTime = ((2*pi)/(this.AnimationSpeed_rads*Frame.NTheta));
    figure(gcf);
    axes(this.AxisReference);
    Inc = 1;

```

```

% Make all static Bodies visible
for iGroup = this.Groups
    iGroup.show('Static',this.AxisReference,0,showOptions);
end

k = 1;
while this.isAnimating && cputime-t < this.AnimationLength_s
    nexttime = cputime + FrameTime;

    % Go down through the hierarchy
    for iGroup = this.Groups
        iGroup.show('Dynamic',this.AxisReference,Inc,showOptions);
    end
    if showOptions(4) % Leak Connections
        for iLeak = this.LeakConnections
            if iLeak.isDynamic
                iLeak.show(this.AxisReference);
            end
        end
    end
    if showOptions(5) % Bridge Connections
        for iBridge = this.Bridges
            iBridge.show(this.AxisReference);
        end
    end
    if showOptions(6) % Inter Node Connections, dynamic
        if this.isDiscretized()
            this.removeDynamicFromFigure();
            % Make array of Node Centers
            % Count nodes, stored in Groups
            n = length(this.Nodes);
            if n ~= 0
                % Calculate the node center, at bottom dead center for all
                for iNode = this.Nodes
                    nodeCenter(iNode.index) = iNode.CenterCoords(Inc);
                end

                % Make array of face coords
                % Count faces, stored in Model
                n = 1;
                xData = NaN(3*length(this.Faces),1);
                yData = NaN(3*length(this.Faces),1);

                % Take each face, assess whether it is active, then record
                for iFace = this.Faces
                    %if iFace.isDynamic && ...
                    if ~(iFace.Nodes(1).Type == enumNType.EN || ...
                        iFace.Nodes(2).Type == enumNType.EN) && ...
                        iFace.isActive(Inc)
                        c1 = nodeCenter(iFace.Nodes(1).index);
                        c2 = nodeCenter(iFace.Nodes(2).index);
                        if this.inWindow(c1,c2)
                            xData(n) = c1.x;
                            xData(n+1) = c2.x;
                            yData(n) = c1.y;
                            yData(n+1) = c2.y;
                            n = n + 3;
                        end
                    end
                end
                n = n-1;
                xData = xData(1:n);
                yData = yData(1:n);

                if isempty(this.StaticGUIObjects)
                    this.DynamicGUIObjects = line(xData,yData,'Color',[0 1 0]);
                else
                    this.DynamicGUIObjects(end+1) = line(xData,yData,'Color',[0 1 0]);
                end
            end
        end
    end
end

```

```

        end
    end
    % Iterate the counter
    Inc = Inc + 1;
    if Inc > Frame.NTheta
        Inc = 1;
    end
    % Wait
    pause(10*max([0 nexttime-cputime]));
end

% Reset Screen to previous settings
set(ReferenceAxis, 'XLimMode', mode);
set(ReferenceAxis, 'YLimMode', mode);
set(ReferenceAxis, 'ZLimMode', mode);
end

end

end

function [Closed_Edge] = TrimFaces(this, region, Closed_Edge)
LEN = length(Closed_Edge);
for Nd = this.Nodes
    if Nd.index <= length(region)
        from = Nd;
        c = 1;
        while c == 1
            c = 0;
            % Determine the number of access points for the node
            for Fc = from.Faces
                if Fc.index <= LEN && ~Closed_Edge(Fc.index) && ...
                    region(Fc.Nodes(1).index) == region(Fc.Nodes(2).index)
                    c = c + 1; if c > 1; break; end; edge = Fc;
                end
            end
            if c == 1
                % This Node has only one access point (withing a region),
                % ... therefore it cannot be a part of a loop.
                % Any nodes that are chain to this node with only two total
                % ... access points must also not be part of a loop.
                Closed_Edge(edge.index) = true;
                if from == edge.Nodes(1)
                    from = edge.Nodes(2);
                else
                    from = edge.Nodes(1);
                end
            end
        end
    end
end
end
end
end

function [A,B,C,D] = populate_Fc_ABCD(Sim, Fc)
fcb = Face.empty;
fcf = Face.empty;
count = 0;
for fc = Fc.Nodes(1).Faces
    if fc.Type == enumFType.Gas && fc ~= Fc
        if count == 0
            fcb = fc;
            count = 1;
        else
            % pick the larger face
            if mean(fcb.data.Area) < mean(fc.data.Area)
                fcb = fc;
            end
        end
    end
end
end
end
if count == 1

```

```

% Reference that backwards face
if fcb.Nodes(2) == Fc.Nodes(1)
    Sim.Fc_Nd03(Fc.index,1) = fcb.Nodes(1).index;
else
    Sim.Fc_Nd03(Fc.index,1) = fcb.Nodes(2).index;
end
else
% Reference itself
    Sim.Fc_Nd03(Fc.index,1) = Fc.Nodes(1).index;
    fcb = Fc;
end
count = 0;
for fc = Fc.Nodes(2).Faces
    if fc.Type == enumFType.Gas && ...
        fc ~= Fc
        if count == 0
            fcf = fc;
            count = 1;
        else
            % pick the larger face
            if mean(fcf.data.Area) < mean(fc.data.Area)
                fcf = fc;
            end
        end
    end
end
if count == 1
% Reference that backwards face
    if fcb.Nodes(1) == Fc.Nodes(2)
        Sim.Fc_Nd03(Fc.index,2) = fcb.Nodes(2).index;
    else
        Sim.Fc_Nd03(Fc.index,2) = fcb.Nodes(1).index;
    end
else
% Reference itself
    Sim.Fc_Nd03(Fc.index,2) = Fc.Nodes(2).index;
    fcf = Fc;
end

x1 = -0.5*Fc.data.Dist;
x0 = x1 - fcb.data.Dist;
x2 = -x1;
x3 = x2 + fcf.data.Dist;
if fcb ~= Fc && all(fcb.data.Area > 0) % Fc i - 1 exists
    if fcf ~= Fc && all(fcf.data.Area > 0) % Fc i + 1 exists
        % can fill xi, xi-1, xi+1 and xi+2, as well as A, B, C and D
        A = -((x1.*x2.*x3)./(x0-x1)./(x0-x2)./(x0-x3));
        B = -((x0.*x2.*x3)./(x1-x0)./(x1-x2)./(x1-x3));
        C = -((x0.*x1.*x3)./(x2-x0)./(x2-x1)./(x2-x3));
        D = -((x0.*x1.*x2)./(x3-x0)./(x3-x1)./(x3-x2));
    else
        A = -((x1.*x2)./(x0-x1)./(x0-x2));
        B = -((x0.*x2)./(x1-x0)./(x1-x2));
        C = -((x0.*x1)./(x2-x0)./(x2-x1));
        D = 0;
    end
else
    if fcf ~= Fc && all(fcf.data.Area > 0) % Fc i + 1 exists
        A = 0;
        B = -((x2.*x3)./(x1-x2)./(x1-x3));
        C = -((x1.*x3)./(x2-x1)./(x2-x3));
        D = -((x1.*x2)./(x3-x1)./(x3-x2));
    else
        A = 0.0;
        B = 0.5;
        C = 0.5;
        D = 0.0;
    end
end
sum = A+B+C+D;
for i = 1:length(sum)

```

```
roundsum = round(sum);
if roundsum == 1
    % No change
elseif roundsum == -1
    A(min(i,length(A))) = -A(min(i,length(A)));
    B(min(i,length(B))) = -B(min(i,length(B)));
    C(min(i,length(C))) = -C(min(i,length(C)));
    D(min(i,length(D))) = -D(min(i,length(D)));
else
    A(min(i,length(A))) = 0;
    B(min(i,length(B))) = 0.5;
    C(min(i,length(C))) = 0.5;
    D(min(i,length(D))) = 0;
end
end
A = CollapseVector(A);
B = CollapseVector(B);
C = CollapseVector(C);
D = CollapseVector(D);
end
```

### **G.3. Minor Elements**

## Custom Minor Loss

The minor loss is a class that includes the following functionality:

A constructor.

A validity test.

```
classdef CustomMinorLoss < handle
    %UNTITLED Summary of this class goes here
    % Detailed explanation goes here

    properties
        name string;
        Body1 Body;
        Body2 Body;
        K12 double;
        K21 double;
    end

    methods
        function this = CustomMinorLoss(B1,B2)
            answers = {'untitled', '1','1'};
            firstround = true;
            while firstround || ~isStrNumeric(answers{2}) || ~isStrNumeric(answers{3})
                if firstround; firstround = false;
                else; msgbox('Numeric Values only'); end
                answers = inputdlg( ...
                    {'Descriptive Name', ...
                     'Loss Coefficient 1 - 2', ...
                     'Loss Coefficient 2 - 1'}, ...
                    ['Define a minor loss from: ' B1.name ...
                     ' to ' B2.name '.'],[1 35], ...
                    answers);
                if isempty(answers)
                    this.K12 = 0;
                    this.K21 = 0;
                    return;
                end
            end
            this.Body1 = B1;
            this.Body2 = B2;
            this.name = answers{1};
            this.K12 = str2double(answers{2});
            this.K21 = str2double(answers{3});
        end

        function isit = isValid(this)
            isit = ~isempty(this.Body1) && ...
                ~isempty(this.Body2) && ...
                isa(this.Body1,'Body') && ...
                isa(this.Body2,'Body') && (this.K12 > 0 || this.K21 > 0);
        end
    end
end
```

## Face

The face is a class that includes the following functionality:

A constructor that assigns a variety of properties based on the nodes involved.

A name calculator.

An function for defining generic properties for faces defined inside of bodies.

A function for checking if the face is a pressure contact.

A function for checking which of the nodes is the largest, in the event this face is decimated and the remaining faces moved onto the larger.

Functions for calculating the total area, and setting the area of the face. Used in area weighted functions or for determining the minor loss coefficient.

```
classdef CustomMinorLoss < handle
    %UNTITLED Summary of this class goes here
    % Detailed explanation goes here

    properties
        name string;
        Body1 Body;
        Body2 Body;
        K12 double;
        K21 double;
    end

    methods
        function this = CustomMinorLoss(B1,B2)
            answers = {'untitled', '1','1'};
            firstround = true;
            while firstround || ~isStrNumeric(answers{2}) || ~isStrNumeric(answers{3})
                if firstround; firstround = false;
                else; msgbox('Numeric Values only'); end
                answers = inputdlg( ...
                    {'Descriptive Name', ...
                     'Loss Coefficient 1 - 2', ...
                     'Loss Coefficient 2 - 1'}, ...
                    ['Define a minor loss from: ' B1.name ...
                     ' to ' B2.name '.'],[1 35], ...
                    answers);
                if isempty(answers)
                    this.K12 = 0;
                    this.K21 = 0;
                    return;
                end
            end
            this.Body1 = B1;
            this.Body2 = B2;
            this.name = answers{1};
            this.K12 = str2double(answers{2});
            this.K21 = str2double(answers{3});
        end

        function isit = isValid(this)
```



```
        isit = ~isempty(this.Body1) && ...  
            ~isempty(this.Body2) && ...  
            isa(this.Body1,'Body') && ...  
            isa(this.Body2,'Body') && (this.K12 > 0 || this.K21 > 0);  
    end  
end  
end
```

## Material

The material is a class containing hardcoded properties of various materials. It contains the following functions:

A creation functions that takes a material name and calls configure.

A modify function that opens a user form and calls configure.

Functions for calculating the thermal diffusivity and internal energy from the temperature.

```
classdef Material < handle
    %MATERIAL Summary of this class goes here
    % Detailed explanation goes here

    properties (Constant)
        Source = {...
            'Carbon Steel';
            'Forged Carbon Steel (Medium Carbon Steel)';
            '304 Stainless Steel';
            '6061 Aluminum';
            'Pure Copper';
            'Plastic, ABS';
            'Plastic, Acrylic';
            'Plastic, Polycarbonate (High Viscosity)';
            'Plastic, Poly-Ethylene (High Density)';
            'Rubber, Polychloroprene (Neoprene)';
            'Rubber, Acrylonitrile-Butadiene (Nitrile)';
            'Rubber, Silicone';
            'Foam, Expanded Polystyrene';
            'Foam, Extruded Polystyrene';
            'Foam, Rigid Polyurethane';
            'AIR';
            'N2 Gas';
            'H2 Gas';
            'Helium Gas';
            'Perfect Insulator';
            'Constant Temperature'};
    end

    properties
        % General Properties
        name;
        Color double;
        Phase enumMaterial;
        ThermalConductivity double;
        dT_du double;
        dh_dT double;
        u2T function_handle;
        Density double;

        % Gas Properties
        R double;
        dT_duFunc function_handle;
        dh_dTFunc function_handle;
        kFunc function_handle;
        muFunc function_handle;
        gammaFunc function_handle;
    end

    methods
        function this = Material(MaterialName)
```

```

if nargin == 0
    return;
end
this.Configure(MaterialName)
end
function Modify(this)
    for index = 1:length(this.Source)
        if strcmp(this.Source{index},this.name)
            break;
        end
    end
    index = listdlg('ListString',this.Source,...
        'SelectionMode','single',...
        'InitialValue',index);
    this.Configure(this.Source{index});
end
function Configure(this,MaterialName)
    this.name = MaterialName;
    % Thermal Conductivity
    % https://www.engineeringtoolbox.com/thermal-conductivity-d_429.html
    switch MaterialName
        case 'Carbon Steel' % Carbon Steel
            this.Color = [0.400 0.384 0.384];% [102 98 98];
            this.Phase = enumMaterial.Solid;
            this.ThermalConductivity = 43; % W/(m*K)
            this.dT_du = 1/502.416; % J/(kg*K)
            this.Density = 7850; % kg/(m^3)
        case 'Forged Carbon Steel (Medium Carbon Steel)'
            % See Medium Carbon Steel - MatWeb.pdf
            this.Color = [0.380 0.365 0.365];
            this.Phase = enumMaterial.Solid;
            this.ThermalConductivity = 47.7; % W/(m*K)
            this.dT_du = 1/477; % J/(kg*K)
            this.Density = 7850; % kg/(m^3)
        case '304 Stainless Steel' % 304 Stainless Steel
            this.Color = [0.510 0.526 0.537];% [130 134 137];
            this.Phase = enumMaterial.Solid;
            this.ThermalConductivity = 14.4; % W/(m*K)
            this.dT_du = 1/500; % J/(kg*K)
            this.Density = 8000; % kg/(m^3)
        case '6061 Aluminum' % 6061 Aluminum
            this.Color = [0.628 0.628 0.628];% [160 160 160];
            this.Phase = enumMaterial.Solid;
            this.ThermalConductivity = 176.5; % 151-202 W/(m*K)
            this.dT_du = 1/897; % J/(kg*K)
            this.Density = 2700; % kg/(m^3)
        case 'Pure Copper' % Pure Copper
            this.Color = [0.628 0.416 0.259];% [160 106 66];
            this.Phase = enumMaterial.Solid;
            this.ThermalConductivity = 401; % W/(m*K) % At 0 C
            this.dT_du = 1/385; % kg*K/J
            this.Density = 8960; % kg/(m^3)
        case 'Plastic, ABS' % Acrylonitrile Butadiene Styrene
            % http://www.substech.com/dokuwiki/doku.php?id=thermoplastic_acrylonitrile-butadiene-
            styrene_abs
            % https://www.sciencedirect.com/topics/materials-science/acrylonitrile-butadiene-
            styrene
            this.Color = [0.1 0.1 0.8]; % Blue
            this.Phase = enumMaterial.Solid;
            this.ThermalConductivity = 0.25; % W/(m*K)
            this.dT_du = 1/1690; % Specific heat capacity    1,390    -    1,920    J/kg·K
            this.Density = 1050;
        case 'Plastic, Acrylic'
            % http://www.matweb.com/search/datasheet.aspx?bassnum=01303&ckck=1
            this.Color = [0.909 0.941 1]; %
            this.Phase = enumMaterial.Solid;
            this.ThermalConductivity = 0.198; % W/(m*K)
            this.dT_du = 1/1810; % Specific heat capacity    1,460    -    2,160    J/kg·K
            this.Density = 1185; % kg/(m^3)
        case 'Plastic, Polycarbonate (High Viscosity)'
            % MatWeb

```

```

    this.Color = [0.909 0.941 1]; %
    this.Phase = enumMaterial.Solid;
    this.ThermalConductivity = 0.198; % W/(m*K)
    this.dT_du = 1/1810; % Specific heat capacity    1,460    -    2,160    J/kg·K
    this.Density = 1200; % kg/(m^3)
case 'Plastic, Poly-Ethylene (High Density)'
    % MatWeb
    this.Color = [0.909 0.941 1]; %
    this.Phase = enumMaterial.Solid;
    this.ThermalConductivity = 0.196; % W/(m*K)
    this.dT_du = 1/1540; % Specific heat capacity    1,460    -    2,160    J/kg·K
    this.Density = 946; % kg/(m^3)
case 'Rubber, Polychloroprene (Neoprene)'
    % https://thermtest.com/materials-database#NEOPRENE
    this.Color = [0.1 0.1 0.1];
    this.Phase = enumMaterial.Solid;
    this.ThermalConductivity = 0.192; % W/(m*K)
    this.dT_du = 1/1029; % Specific heat capacity    1,460    -    2,160    J/kg·K
    this.Density = 1250; % kg/(m^3)
case 'Rubber, Acrylonitrile-Butadiene (Nitrile)'
    % https://thermtest.com/materials-database#NITRILE
    this.Color = [33/255 16/255 0];
    this.Phase = enumMaterial.Solid;
    this.ThermalConductivity = 0.243; % W/(m*K)
    this.dT_du = 1/1966; % Specific heat capacity    1,460    -    2,160    J/kg·K
    this.Density = 1000; % kg/(m^3)
case 'Rubber, Silicone'
    % https://thermtest.com/materials-database#SILICONE-RUBBER-(MEDIU
    this.Color = [22/255 25/255 37/255];
    this.Phase = enumMaterial.Solid;
    this.ThermalConductivity = 0.335; % W/(m*K)
    this.dT_du = 1/1255; % Specific heat capacity    1,460    -    2,160    J/kg·K
    this.Density = 1300; % kg/(m^3)
case 'Foam, Expanded Polystyrene'
    % http://www.eyoungindustry.com/uploadfile/file/20160612/20160612155656_94768.pdf
    % Implement subtype interface
    index = 0;
    while index == 0
        index = 1;
        index = listdlg('ListString',{...
            'L - 11 kg/m3','SL - 13.5 kg/m3',...
            'S - 16 kg/m3','M - 19 kg/m3',...
            'H - 24 kg/m3','VH - 28 kg/m3',...
            'Custom Density'},...
            'SelectionMode','single',...
            'InitialValue',index);
    end
    if index < 7
        Temp = [11 13.5 16 19 24 28];
        this.Density = Temp(index);
    else
        this.Density = str2double(inputdlg('Enter EPS density',...
            'Custom Expanded Polystyrene',[1 35],'16'));
    end
    this.Color = [1 0.83 0.83];
    this.Phase = enumMaterial.Solid;
    this.ThermalConductivity = 0.1142*(this.Density)^(-0.371); % W/(m*K)
    % https://www.engineeringtoolbox.com/specific-heat-capacity-d_391.html
    this.dT_du = 1/(1400); % Specific heat capacity    1,300    -    1,500    J/kg·K
case 'Foam, Extruded Polystyrene'
    % http://www.eyoungindustry.com/uploadfile/file/20160612/20160612155656_94768.pdf
    % Implement subtype interface
    index = 0;
    while index == 0
        index = 1;
        index = listdlg('ListString',{...
            'Custom Density'},...
            'SelectionMode','single',...
            'InitialValue',index);
    end
    if index < 1

```

```

Temp = [];
this.Density = Temp(index);
else
    this.Density = str2double(inputdlg('Enter XPS density',...
        'Custom Expanded Polystyrene',[1 35],{'30'}));
end
this.Color = [1 0.83 0.83];
this.Phase = enumMaterial.Solid;
this.ThermalConductivity = 0.036; % W/(m*K)
% https://www.engineeringtoolbox.com/specific-heat-capacity-d_391.html
this.dT_du = 1/(1400); % Specific heat capacity 1,300 - 1,500 J/kg·K
case 'Foam, Rigid Polyurethane'
% http://www.react-ite.eu/uploads/tx_mddownloadbox/PP02_Thermal_insulation_materials_
_PP02_20130715.pdf
% Implement subtype interface
index = 0;
while index == 0
    index = 1;
    index = listdlg('ListString',{...
        '30 kg/m3',...
        '40 kg/m3',...
        '80 kg/m3',...
        'Custom Density'},...
        'SelectionMode','single',...
        'InitialValue',index);
end
if index < 4
    Temp = [30 40 80];
    this.Density = Temp(index);
else
    this.Density = str2double(inputdlg('Enter PUR density',...
        'Custom Rigid Polyurethane',[1 35],{'16'}));
end
this.Color = [0.957 0.937 0.745];
this.Phase = enumMaterial.Solid;
this.ThermalConductivity = 0.0371*(this.Density)^(-0.098); % W/(m*K)
this.dT_du = 1/1500; % Specific heat capacity 1,300 - 1,500 J/kg·K
case 'AIR' % Standard Air
this.Color = [0.906 0.906 0.906];% [231 231 231];
this.Phase = enumMaterial.Gas;
this.ThermalConductivity = 0.0262; % W/(m*K)
this.R = 287;
this.dT_duFunc = @(u) (-2.88367e-10)*u + (1.42462651e-3); % Verified
this.dT_du = this.dT_duFunc(298);
this.dh_dTFunc = @(T) 1013.5 - 0.15709*T + 0.00049079*T.^2 - 0.00000020552*T.^3;
this.dh_dT = this.dh_dTFunc(298);
this.u2T = @(u) (-1.44183718e-10)*u.^2 + (1.42462651e-3)*u; % Verified
this.kFunc = @(T) (-1.3974e-11)*T.^3 + (-4.5769e-8)*T.^2 + ...
    (9.8961e-5)*T + 3.4920e-4; % Verified / Updated
this.muFunc = @(T) (1.6834E-14)*T.^3 - (4.7591E-11)*T.^2 + ...
    7.1598E-08*T + 7.5908E-07; % Verified / Updated
case 'N2 Gas' % Nitrogen
fprintf('XXX Need cv and u2T for nitrogen XXX\n');
this.Color = [0.906 0.906 0.906];
this.Phase = enumMaterial.Gas;
this.R = 296.8;
this.dT_duFunc = @(u) -3.7744e-10*u + 1.447e-3; % Verified
this.dT_du = this.dT_duFunc(298);
% http://www.colby.edu/chemistry/PChem/notes/Ch7Tables.pdf
this.dh_dTFunc = @(T) (28.882 - 0.00157*T + 0.00000808*T.^2 -
0.000000002871*T.^3)/0.0280134;
this.dh_dT = this.dh_dTFunc(298);
this.u2T = @(u) -1.8872e-10*(u.^2) + 1.447e-3*u - 1.2188e1; % Verified
this.kFunc = @(T) 3.3552E-11*(T.^3) - 7.3741E-08*(T.^2) + 1.0792E-04*T - 6.5862E-04; %
Verified / Update
this.muFunc = @(T) 1.9072E-14*(T.^3) - 4.9344E-11*(T.^2) + 7.1568E-08*T + 3.4160E-07; %
Verified / Updated
case 'H2 Gas' % Hydrogen
this.Color = [0.906 0.906 0.906];% [231 231 231];
this.Phase = enumMaterial.Gas;
this.R = 4124.2;

```



# Node

A node is a class that includes the following functionality:

A constructor.

A function use to merge two nodes into one.

A group of functions designed to calculate specific properties.

```
classdef Node < handle
%NODE Summary of this class goes here
% Detailed explanation goes here

properties
    isDynamic logical;
    Type enumNType;
    data;
    iPressure double;
    iTemperature double;
    xmin double;
    xmax double;
    ymin double;
    ymax double;
    Faces Face;
    Nodes Node;
    Body;

    isEnd logical; % Allows for the required distance calculation

    index int32; % For translation to array for solving
end

properties (Hidden)
    stateLocation Pnt2D;
    isminCenterCoordsCalcd logical = false;
    stateminCenterCoords Pnt2D;
    useStoredVolume logical;
    StoredVolume double;
end

properties (Dependent)
    minCenterCoords Pnt2D;
end

methods
    function this = Node(Type,xmin,xmax,ymin,ymax,Faces,Nodes,theBody,index)
        if nargin == 0; return; end
        % Node(Type, data, iPressure, iTemperature, xmin, xmax, ymin, ymax, Faces, Nodes, matl)
        this.Type = Type;

        this.xmin = xmin;
        this.xmax = xmax;
        this.ymin = ymin;
        this.ymax = ymax;
        if nargin == 5; return; end
        this.Faces = Faces;
        this.Nodes = Nodes;

        if this.vol() < 0
            fprintf('err');
        end

        this.index = index;
    end
end
```

```

this.Body = theBody;
if isempty(this.data); this.calcData(); end

this.updateisDynamic();
end
function [success, nd2del, fc2del] = combineSolid(this,other,refinementfactor)
% Set outputs to default values
nd2del = Node.empty;
fc2del = Face.empty;
success = false;

% It must be a Face between two solid nodes
if this.Type == enumNType.SN && other.Type == enumNType.SN

% Find the face between the two nodes
for fc = this.Faces
if (fc.Nodes(1) == this && fc.Nodes(2) == other) || (...
    fc.Nodes(2) == this && fc.Nodes(1) == other)

% Get the two materials
if isfield(this.data,'matl'); matl1 = this.data.matl;
else; matl1 = this.Body.matl;
end
if isfield(other.data,'matl'); matl2 = other.data.matl;
else; matl2 = other.Body.matl;
end

% Calculate the timestep based on a Fourier Number of 0.25
if matl1.dT_du == -1
dT_du1 = 1e-30; else; dT_du1 = matl1.dT_du; end
if matl2.dT_du == -1
dT_du2 = 1e-30; else; dT_du2 = matl2.dT_du; end

timestep1 = (0.25*matl1.Density/dT_du1)*this.vol()./fc.data.U;
timestep2 = (0.25*matl2.Density/dT_du2)*other.vol()./fc.data.U;

% If the timestep for this interaction would be too small
if all(timestep1 < 1e-3) || all(timestep2 < 1e-3)
%
if sum(timestep1 < timestep2) > length(timestep1)/2
collector = other;
nd2del = this;
else
collector = this;
nd2del = other;
end
end
fc2del = fc;
success = true;
break;
end

% Exit because only one face will be between these nodes
break;
end
end

if success
% Calculate the updated properties
collector.StoredVolume = this.vol() + other.vol();
vol1 = this.vol();
vol2 = other.vol();
collector.useStoredVolume = true;
if matl1.dT_du == -1
collector.data.matl = matl1;
elseif matl2.dT_du == -1
collector.data.matl = matl2;
elseif matl1 ~= matl2
if isfield(collector.data,'matl'); matl = collector.data.matl;
else; matl = collector.Body.matl;
end
mass = vol1*matl1.Density + vol2*matl2.Density;

```



```

    collector.data.matl = Material();
    collector.data.matl.Color = matl1.Color;
    collector.data.matl.Phase = enumMaterial.Solid;
    collector.data.matl.ThermalConductivity = ...
        (vol1*matl1.Density*matl1.ThermalConductivity + ...
         vol2*matl2.Density*matl2.ThermalConductivity)/...
        (vol1*matl1.Density + vol2*matl2.Density);
    collector.data.matl.Density = (vol1*matl1.Density + ...
        vol2*matl.Density)/collector.StoredVolume;
    collector.data.matl.dT_du = (vol1*matl1.Density*dT_du1 + ...
        vol2*matl2.Density*dT_du2)/mass;
end

% Modify the faces in the weaker node so that they reference the
% ... collector instead.
keep = true(size(nd2del.Faces));
i = 1;
for fc = nd2del.Faces
    if fc ~= fc2del
        if fc.Nodes(1) == nd2del
            fc.Nodes(1) = collector;
        elseif fc.Nodes(2) == nd2del
            fc.Nodes(2) = collector;
        end
    else
        keep(i) = false;
    end
    i = i + 1;
end
nd2del.Faces = nd2del.Faces(keep);

% Remove the face that goes between the two nodes
keep = true(size(collector.Faces));
for i = 1:length(collector.Faces)
    if collector.Faces(i) == fc2del
        keep(i) = false;
    end
end
collector.Faces = collector.Faces(keep);
collector.Faces(end+1:end+length(nd2del.Faces)) = nd2del.Faces;
end
end
end

function calcData(this)
    if this.Type ~= enumNType.EN
        matl = this.Body.matl;
        switch matl.Phase
            case enumMaterial.Solid
                %% Solids
                this.data.T = this.Body.Temperature();
                this.data.dT_dU = matl.dT_du;

            case {enumMaterial.Gas, enumMaterial.Liquid}
                %% Fluids
                this.data.P = this.Body.Pressure();
                this.data.T = this.Body.Temperature();
                if ~isempty(this.Body.Matrix) && ~isempty(this.Body.Matrix.Dh)
                    %% Not an empty Volume
                    % Scale the volume
                    this.data.vol = this.vol()*this.Body.Matrix.data.Porosity;
                    this.data.Dh = this.Body.Matrix.Dh;
                    % Assign the Nusselt number function (Re,Pr)
                    this.data.NuFunc_l = this.Body.Matrix.NuFunc_l;
                    if ~this.Body.Matrix.isFullyLaminar
                        this.data.NuFunc_t = this.Body.Matrix.NuFunc_t;
                    else
                        this.data.NuFunc_t = this.data.NuFunc_l;
                    end
                end
                dir = getBodyDirection(this.Body);
                if dir == 1

```

```

        % Horizontal
        this.data.Area = (this.ymax-this.ymin)*pi*(this.xmax + this.xmin)*...
        this.Body.Matrix.data.Porosity;
    else
        % Vertical
        this.data.Area = pi*(this.xmax^2 - this.xmin^2)*...
        this.Body.Matrix.data.Porosity;
    end
else
    %% An empty channel
    this.data.vol = this.vol();
    dir = getBodyDirection(this.Body);
    if dir == 1
        % Horizontal
        this.data.Orient = enumOrient.Horizontal;
        this.data.Dh = 2.*(this.ymax-this.ymin);
        this.data.Area = (this.ymax-this.ymin)*pi*(this.xmax + this.xmin);
        % Assign default Nusselt Number Correlation
        this.data.NuFunc_1 = @(Re) 3.66; % Fully Developed, Uniform Surface Temperature
    else
        this.data.Orient = enumOrient.Vertical;
        this.data.Dh = 2.*(this.xmax-this.xmin);
        this.data.Area = pi*(this.xmax^2 - this.xmin^2);
        % Assign default Nusselt Number Correlation
        ri_ro = this.xmin/this.xmax;
        if ri_ro == 0
            Nuo = 3.66;
            Nui = 0;
        else
            Nuo = 4.6961*(ri_ro)^(0.0548);
            Nui = 4.4438*(ri_ro)^(-0.43);
        end
        this.data.NuiFunc_1 = @(Re) Nui;
        this.data.NuoFunc_1 = @(Re) Nuo;
    end
    this.data.NuFunc_t = @(Re,Pr) 0.035*(Re.^0.75).*(Pr.^0.33);
end
if ~isscalar(this.data.Dh); this.data.Dh = CollapseVector(this.data.Dh); end
if ~isscalar(this.data.Area); this.data.Area = CollapseVector(this.data.Area); end
end
else
    % Body is actually an environment
    this.data.T = this.Body.Temperature;
    this.data.P = this.Body.Pressure;
    this.data.h = this.Body.h;
    if isempty(this.Body.matl)
        this.Body.matl = Material('AIR');
    end
    this.data.rho = this.data.P/(this.data.T*this.Body.matl.R);
end
end
function addFace(this,Face)
    this.Faces(end+length(Face):-1:end+1) = Face;
end
function updateisDynamic(this)
    if length(this.xmin) > 1 || length(this.xmax) > 1 || length(this.ymin) > 1 ||
length(this.ymax) > 1 % this.Type ~= enumNType.SN
        this.isDynamic = true;
    else
        for Face = this.Faces
            if Face.isDynamic
                this.isDynamic = true;
                return;
            end
        end
        this.isDynamic = false;
    end
end
function var = total_vol(this)
    var = (pi*(this.xmax^2 - this.xmin^2)).*(this.ymax - this.ymin);
end
end

```

```

function var = vol(this)
    if this.Type == enumNType.SN
        if this.useStoredVolume
            var = this.StoredVolume;
        else
            var = pi.*(this.xmax.^2-this.xmin.^2)*...
                (this.ymax(1)-this.ymin(1));
        end
    else
        if ~isa(this.Body, 'Body')
            P = 1;
        elseif isempty(this.Body.Matrix)
            P = 1;
        else
            if isfield(this.Body.Matrix.data, 'Porosity')
                P = this.Body.Matrix.data.Porosity;
            else
                P = 1;
            end
        end
        var = P*pi.*(this.xmax.^2-this.xmin.^2)*...
            (this.ymax-this.ymin);
        var = CollapseVector(var);
    end
end
function recalcd_Dh(this)
    if this.Type ~= enumNType.SN && this.Type ~= enumNType.EN
        % Dh = 4 * Volume / Surface Area

        V = this.vol();
        for fc = this.Faces
            if fc.Type == enumFType.Mix
                S_total = S_total + fc.data.Area;
            end
        end
        S_total = 2*pi*(this.xmax^2 - this.xmin^2) + ... Top & Bottom
            2*pi*(this.xmax + this.xmin)*(this.ymax - this.ymin); % Sides
        S_total = 2*pi*(this.xmin + this.xmax)*(this.ymax-this.ymin) + ...
            2*pi*(this.xmax^2-this.xmin^2);
        if isempty(this.Body.Matrix) || ...
            strcmp(this.Body.Matrix.name, 'Undefined Matrix') == 1
            for fc = this.Faces
                if fc.Type == enumFType.Gas
                    S_total = S_total - fc.data.Area;
                end
            end
        else
            if isfield(this.Body.Matrix.data, 'ignore_canister') && ...
                this.Body.Matrix.data.ignore_canister
                S_total = 0;
                includeGas = false;
            else
                includeGas = true;
            end
            for fc = this.Faces
                if fc.Type == enumFType.Mix
                    % Include only matrix faces when it is a matrix, as the heat
                    % ... exchange equations assume that it is just the heat
                    % ... exchanger geometry.
                    if fc.Nodes(1).Body == fc.Nodes(2).Body
                        S_total = S_total + fc.data.Area;
                    end
                end
            end
            if includeGas
                for fc = this.Faces
                    if fc.Type == enumFType.Gas
                        S_total = S_total - fc.data.Area;
                    end
                end
            end
        end
    end
end
end
end

```

```

        Dh = 4*V./S_total; % 4*A*L/(P*L)
        this.data.Dh = CollapseVector(Dh);
    end
end
function center = get.minCenterCoords(this)
    if ~this.isminCenterCoordsCalcd || isempty(this.stateminCenterCoords)
        if this.isDynamic
            this.stateminCenterCoords = ...
                Pnt2D(0.5*(this.xmin+this.xmax),...
                    0.5*(min(this.ymin)+min(this.ymax)));
        else
            this.stateminCenterCoords = ...
                Pnt2D(0.5*(this.xmin+this.xmax),...
                    0.5*(this.ymin+this.ymax));
        end
    end
    if isa(this.Body,'Body')
        center = this.Body.Group.TranslatePnt2D(this.stateminCenterCoords);
    else
        center = this.stateminCenterCoords;
    end
end
function center = CenterCoords(this,Inc)
    if this.isDynamic
        if isscalar(this.ymin)
            if isscalar(this.ymax)
                center = this.minCenterCoords;
            else
                center = ...
                    Pnt2D(0.5*(this.xmin+this.xmax), ...
                        0.5*(this.ymin+this.ymax(Inc)));
            end
        else
            if isscalar(this.ymax)
                center = ...
                    Pnt2D(0.5*(this.xmin+this.xmax), ...
                        0.5*(this.ymin(Inc)+this.ymax));
            else
                center = ...
                    Pnt2D(0.5*(this.xmin+this.xmax), ...
                        0.5*(this.ymin(Inc)+this.ymax(Inc)));
            end
        end
    else
        center = this.stateminCenterCoords;
    end
    if isa(this.Body,'Body')
        center = this.Body.Group.TranslatePnt2D(center);
    else
        center = this.stateminCenterCoords;
    end
end
function Struct = getGrouping(this,Struct,n,sourceFace)
    % if this node is a transition, stop this recursion
    if nargin == 4
        for Fc = this.Faces
            if isfield(Fc.data,'K12') || ...
                (isfield(Fc.data,'dx') && ...
                    abs(Fc.data.dx - sourceFace.data.dx)/sourceFace.data.dx > 0.1)
                return;
            end
        end
        this.data.Group = n;
        Struct.Nds = [Struct.Nds this];
    elseif nargin == 3
        val = 0;
        for Fc = this.Faces
            if isfield(Fc.data,'dx')

                if isscalar(val)
                    if isscalar(Fc.data.dx)

```

```

        if Fc.data.dx > val
            val = Fc.data.dx;
        end
    else
        val = max([val(ones(size(Fc.data.dx))); Fc.data.dx]);
    end
else
    if isscalar(Fc.data.dx)
        val(val<Fc.data.dx) = Fc.data.dx;
    else
        val = max([val; Fc.data.dx]);
    end
end
end

end
end
if isscalar(val) && val == 0
    return;
end
for Fc = this.Faces
    if isfield(Fc.data,'K12') || ...
        (isfield(Fc.data,'dx') && ...
            any(abs(Fc.data.dx - val)./val > 0.1))
        return;
    end
end
end
for Fc = this.Faces
    if ~isfield(Fc.data,'Group') && isfield(Fc.data,'dx')
        Struct.Fcs = [Struct.Fcs Fc];
    end
end
for i = 1:length(this.Nodes)
    if isfield(this.Nodes(i).data,'P') && isfield(this.Faces(i).data,'dx')
        Struct = getGrouping(this.Nodes(i),Struct,n,this.Faces(i));
    end
end
end
function value = getArea(this,ind,Connection)
    if ~isa(this.Body,'Body')
        value = 1e8;
        return;
    end
    if this.Body.divides(1) ~= this.Body.divides(2) || nargin < 3
        if isfield(this.data,'Area')
            if isscalar(this.data.Area)
                value = this.data.Area;
            else
                if ind == 0
                    value = this.data.Area(end);
                else
                    value = this.data.Area(ind);
                end
            end
        end
    else
        value = 0;
    end
else
    switch Connection.Orient
    case enumOrient.Vertical
        if ind == 0
            imin = 1;
            imax = 1;
        else
            imin = min(length(this.ymin),ind);
            imax = min(length(this.ymax),ind);
        end
        value = 2*pi*Connection.x*(this.ymax(imax)-this.ymin(imin));
    case enumOrient.Horizontal
        value = pi*(this.xmax^2-this.xmin^2);
    end
end
end

```



## Node Contact

The node contact is a class that includes the following functionality:

A constructor.

A function used for calculating the periods of time where the two included nodes are touching.

A set of functions of calculating the face properties.

A set of functions that allow for masking or subtracting of two node contacts.

```
classdef NodeContact < handle
    %NODECONTACT Summary of this class goes here
    % Detailed explanation goes here

    properties
        Node Node;
        Start double = 0;
        End double = 0;
        Type enumFType;
        Connection Connection;
        data struct;
    end

    methods
        function this = NodeContact(Node,Start,End,Type,Connection)
            if nargin == 0; return; end
            %NODECONTACT Construct an instance of this class
            % Detailed explanation goes here
            this.Node = Node;
            this.Start = Start;
            this.End = End;
            this.Type = Type;
            this.Connection = Connection;
        end
        function ActiveTimes = activeTimes(NC1,NC2)
            ActiveTimes = ~(NC1.Start >= NC2.End) + (NC2.Start >= NC1.End);
            if ~any(ActiveTimes)
                ActiveTimes = logical([]);
                return;
            end
            if all(ActiveTimes)
                ActiveTimes = true;
                return;
            end
        end
        function [Area] = getArea(NC1,NC2,ActiveTimes)
            if nargin < 3
                ActiveTimes = NC1.activeTimes(NC2);
            end
            ActiveTimes = logical(ActiveTimes);
            if isscalar(ActiveTimes)
                % scalar: s1,e1,s2,e2
                if NC1.Connection.Orient == enumOrient.Vertical
                    if isscalar(NC1.Start)
                        if isscalar(NC2.Start)
                            TheStart = max([NC1.Start NC2.Start]);
                        else
                            TheStart = NC2.Start;
                            TheStart(TheStart<NC1.Start) = NC1.Start;
                        end
                    end
                else
                    TheStart = NC1.Start;
                end
            end
        end
    end
end
```

```

        if isscalar(NC2.Start)
            TheStart = NC1.Start;
            TheStart(TheStart<NC2.Start) = NC2.Start;
        else
            TheStart = max([NC1.Start; NC2.Start]);
        end
    end
    if isscalar(NC1.End)
        if isscalar(NC2.End)
            TheEnd = min([NC1.End NC2.End]);
        else
            TheEnd = NC2.End;
            TheEnd(TheEnd>NC1.End) = NC1.End;
        end
    else
        if isscalar(NC2.End)
            TheEnd = NC1.End;
            TheEnd(TheEnd>NC2.End) = NC2.End;
        else
            TheEnd = min([NC1.End; NC2.End]);
        end
    end
    end
    Area = 2*pi*NC1.Connection.x*(TheEnd-TheStart);
else
    Area = pi*(min([NC1.End NC2.End])^2-max([NC1.Start NC2.Start])^2);
end
else % This case will only include Vertical because Horizontal never changes activation
% Vertical
    if isscalar(NC1.Start)
        if isscalar(NC2.Start)
            TheStart = max([NC1.Start NC2.Start]);
        else
            TheStart = NC2.Start;
            TheStart(TheStart<NC1.Start) = NC1.Start;
        end
    else
        if isscalar(NC2.Start)
            TheStart = NC1.Start;
            TheStart(TheStart<NC2.Start) = NC2.Start;
        else
            TheStart = max([NC1.Start; NC2.Start]);
        end
    end
    end
    if isscalar(NC1.End)
        if isscalar(NC2.End)
            TheEnd = min([NC1.End NC2.End]);
        else
            TheEnd = NC2.End;
            TheEnd(TheEnd>NC1.End) = NC1.End;
        end
    else
        if isscalar(NC2.End)
            TheEnd = NC1.End;
            TheEnd(TheEnd>NC2.End) = NC2.End;
        else
            TheEnd = min([NC1.End; NC2.End]);
        end
    end
    end
    Area = 2*pi*NC1.Connection.x*(TheEnd-TheStart);
    Area(~ActiveTimes) = 0;
end
if NC1.Node.Type ~= enumNType.SN && NC2.Node.Type ~= enumNType.SN
    P = 1;
    for NC = [NC1 NC2]
        if isa(NC.Node.Body,'Body')
            if ~isempty(NC.Node.Body.Matrix)
                Mat = NC.Node.Body.Matrix;
                if ~strcmp(Mat.name,'Undefined Matrix') && ...
                    isfield(Mat.data,'Porosity')
                    P = min(P,Mat.data.Porosity);
                end
            end
        end
    end
end

```



```

        end
    end
end
if P ~= 1
    Area = Area*P;
end
end
end
function [U] = getConductance(NC1,NC2,ActiveTimes)
    U = 0;
    ActiveTimes = logical(ActiveTimes);
    if isfield(NC1.Node.data,'matl'); matl1 = NC1.Node.data.matl;
    else; matl1 = NC1.Node.Body.matl;
    end
    if isfield(NC2.Node.data,'matl'); matl2 = NC2.Node.data.matl;
    else; matl2 = NC2.Node.Body.matl;
    end
    if isscalar(ActiveTimes)
        % Static
        % scalar: s1,e1,s2,e2
        L = abs(min([NC1.End NC2.End])-max([NC1.Start NC2.Start]));
        if NC1.Connection.Orient == enumOrient.Vertical
            if NC1.Node.Type == enumNType.SN % Solid Node
                U = AnnularConduction(...
                    NC1.Node, NC1.Connection.x,...
                    L, matl1);
                if U == 0
                    return;
                end
            elseif NC1.Node.Type == enumNType.EN
                U = 2*pi*NC1.Connection.x*L*NC1.Node.Body.h;
            end
            if NC2.Node.Type == enumNType.SN % Solid Node
                if U ~= 0
                    U = 1/(1/U + 1/...
                        AnnularConduction(...
                            NC2.Node, NC2.Connection.x,...
                            L, matl2));
                else
                    U = AnnularConduction(...
                        NC2.Node, NC2.Connection.x,...
                        L, matl2);
                end
            elseif NC2.Node.Type == enumNType.EN
                if U ~= 0
                    U = 1/(1/U + 1/(2*pi*NC2.Connection.x*L*NC2.Node.Body.h));
                else
                    U = 2*pi*NC2.Connection.x*L*NC2.Node.Body.h;
                end
            end
        end
    else
        r1 = max([NC1.Start NC2.Start]);
        r2 = r1 + L;
        if NC1.Node.Type == enumNType.SN % Solid Node
            U = LinearConduction(NC1.Node, r1, r2, matl1);
            if U == 0; return; end
        elseif NC1.Node.Type == enumNType.EN
            U = 2*pi*(r2^2-r1^2)*NC1.Node.Body.h;
        end
        if NC2.Node.Type == enumNType.SN % Solid Node
            if U ~= 0
                U = 1/(1/U + 1/LinearConduction(NC2.Node, r1, r2, matl2));
            else
                U = LinearConduction(NC2.Node, r1, r2, matl2);
            end
        elseif NC2.Node.Type == enumNType.EN
            if U ~= 0
                U = 1/(1/U + 1/(2*pi*(r2^2-r1^2)*NC2.Node.Body.h));
            else
                U = 2*pi*(r2^2-r1^2)*NC2.Node.Body.h;
            end
        end
    end
end
end

```

```

        end
    end
    return;
else
    % Dynamic - Vertical Only
    TheStart = zeros(1,Frame.NTheta);
    TheEnd = zeros(1,Frame.NTheta);
    if isscalar(NC1.Start)
        if isscalar(NC2.Start)
            TheStart = max([NC1.Start NC2.Start]);
        else
            TheStart = NC2.Start;
            TheStart(ActiveTimes & TheStart<NC1.Start) = NC1.Start;
        end
    else
        if isscalar(NC2.Start)
            TheStart = NC1.Start;
            TheStart(ActiveTimes & TheStart<NC2.Start) = NC2.Start;
        else
            TheStart(ActiveTimes) = max([NC1.Start(ActiveTimes); NC2.Start(ActiveTimes)]);
        end
    end
    if isscalar(NC1.End)
        if isscalar(NC2.End)
            TheEnd = min([NC1.End NC2.End]);
        else
            TheEnd = NC2.End;
            TheEnd(ActiveTimes & TheEnd>NC1.End) = NC1.End;
        end
    else
        if isscalar(NC2.End)
            TheEnd = NC1.End;
            TheEnd(ActiveTimes & TheEnd>NC2.End) = NC2.End;
        else
            TheEnd(ActiveTimes) = min([NC1.End(ActiveTimes); NC2.End(ActiveTimes)]);
        end
    end
    end
    U = 2*pi*NC1.Connection.x*(TheEnd-TheStart);
    U(~ActiveTimes) = 0;
    % Actual Conduction Modifier
    r = NC1.Connection.x;
    L = U./(2*pi*r);
    if NC1.Node.Type == enumNType.SN
        if NC2.Node.Type == enumNType.SN
            % Both are solid
            U = 1./(1./AnnularConduction(NC1.Node,r,L,matl1) + ...
                1./AnnularConduction(NC2.Node,r,L,matl2));
        else
            % NC1 is the solid
            U = AnnularConduction(NC1.Node,r,L,matl1);
        end
    else
        % NC2 must be solid
        U = AnnularConduction(NC2.Node,r,L,matl2);
    end
end
end
end
function [Dist] = getDistance(NC1,NC2,ActiveTimes)
% Get distance to center of face
c = getCenterOfOverlapRegion(NC1.Start,NC2.Start,NC1.End,NC2.End);
switch NC1.Connection.Orient
case enumOrient.Vertical
    Dist = 0;
    for NC = [NC1 NC2]
        if NC.Node.Type ~= enumNType.EN
            cx = (NC.Node.xmin + NC.Node.xmax)./2;
            cy = (NC.Node.ymin + NC.Node.ymax)./2;
            Dist = Dist + sqrt(...
                (cx - NC.Connection.x).^2 + ...
                (cy - c).^2);
        end
    end
end
end

```

```

end
case enumOrient.Horizontal
    Dist = 0.0;
    for NC = [NC1 NC2]
        if NC.Node.Type ~= enumNType.EN
            cx = (NC.Node.xmin + NC.Node.xmax) ./ 2;
            cy = (NC.Node.ymin + NC.Node.ymax) ./ 2;
            if ~isempty(NC.Connection.RefFrame)
                Dist = Dist + sqrt(...
                    (cx - c).^2 + ...
                    (cy - NC.Connection.x - ...
                    NC.Connection.RefFrame.Positions).^2);
            else
                Dist = Dist + sqrt(...
                    (cx - c).^2 + ...
                    (cy - NC.Connection.x).^2);
            end
        end
    end
end
end
Dist(~ActiveTimes) = 1e8;
Dist = CollapseVector(Dist);
end
function [Dist] = getStabilityDistance(NC1,NC2,ActiveTimes)
    Dist = getDistance(NC1,NC2,ActiveTimes);
end
function [Dh] = getDh(NC1,NC2,ActiveTimes)
    % Determine if it is a transition or not (if not, define Dh)
    if NC1.Connection.Orient == NC2.Connection.Orient
        if isscalar(ActiveTimes)
            switch NC1.Connection.Orient
                case enumOrient.Vertical
                    if all(NC1.Node.ymin == NC2.Node.ymin) && ...
                        all(NC1.Node.ymax == NC2.Node.ymax)
                        Dh = 2*(NC1.End - NC1.Start);
                    else
                        if isscalar(NC1.End)
                            if isscalar(NC2.End)
                                D2 = min([NC1.End NC2.End]);
                            else
                                D2 = NC2.End;
                                D2(D2 > NC1.End) = NC1.End;
                            end
                        else
                            if isscalar(NC2.End)
                                D2 = NC1.End;
                                D2(D2 > NC2.End) = NC2.End;
                            else
                                D2 = min([NC1.End; NC2.End]);
                            end
                        end
                    end
                case enumOrient.Horizontal
                    if isscalar(NC1.Start)
                        if isscalar(NC2.Start)
                            D1 = max([NC1.Start NC2.Start]);
                        else
                            D1 = NC2.Start;
                            D1(D1 < NC1.Start) = NC1.Start;
                        end
                    else
                        if isscalar(NC2.Start)
                            D1 = NC1.Start;
                            D1(D1 < NC2.Start) = NC2.Start;
                        else
                            D1 = max([NC1.Start; NC2.Start]);
                        end
                    end
                end
            end
            Dh = 2*(D2 - D1);
        end
    end
end
function [Dh] = getDh(NC1,NC2,ActiveTimes)
    % Determine if it is a transition or not (if not, define Dh)
    if NC1.Connection.Orient == NC2.Connection.Orient
        if isscalar(ActiveTimes)
            switch NC1.Connection.Orient
                case enumOrient.Vertical
                    if all(NC1.Node.ymin == NC2.Node.ymin) && ...
                        all(NC1.Node.ymax == NC2.Node.ymax)
                        Dh = 2*(NC1.End - NC1.Start);
                    else
                        if isscalar(NC1.End)
                            if isscalar(NC2.End)
                                D2 = min([NC1.End NC2.End]);
                            else
                                D2 = NC2.End;
                                D2(D2 > NC1.End) = NC1.End;
                            end
                        else
                            if isscalar(NC2.End)
                                D2 = NC1.End;
                                D2(D2 > NC2.End) = NC2.End;
                            else
                                D2 = min([NC1.End; NC2.End]);
                            end
                        end
                    end
                case enumOrient.Horizontal
                    if isscalar(NC1.Start)
                        if isscalar(NC2.Start)
                            D1 = max([NC1.Start NC2.Start]);
                        else
                            D1 = NC2.Start;
                            D1(D1 < NC1.Start) = NC1.Start;
                        end
                    else
                        if isscalar(NC2.Start)
                            D1 = NC1.Start;
                            D1(D1 < NC2.Start) = NC2.Start;
                        else
                            D1 = max([NC1.Start; NC2.Start]);
                        end
                    end
                end
            end
            Dh = 2*(D2 - D1);
        end
    end
end
function [Dist] = getStabilityDistance(NC1,NC2,ActiveTimes)
    Dist = getDistance(NC1,NC2,ActiveTimes);
end
function [Dh] = getDh(NC1,NC2,ActiveTimes)
    % Determine if it is a transition or not (if not, define Dh)
    if NC1.Connection.Orient == NC2.Connection.Orient
        if isscalar(ActiveTimes)
            switch NC1.Connection.Orient
                case enumOrient.Vertical
                    if all(NC1.Node.ymin == NC2.Node.ymin) && ...
                        all(NC1.Node.ymax == NC2.Node.ymax)
                        Dh = 2*(NC1.End - NC1.Start);
                    else
                        if isscalar(NC1.End)
                            if isscalar(NC2.End)
                                D2 = min([NC1.End NC2.End]);
                            else
                                D2 = NC2.End;
                                D2(D2 > NC1.End) = NC1.End;
                            end
                        else
                            if isscalar(NC2.End)
                                D2 = NC1.End;
                                D2(D2 > NC2.End) = NC2.End;
                            else
                                D2 = min([NC1.End; NC2.End]);
                            end
                        end
                    end
                case enumOrient.Horizontal
                    if all(NC1.Node.xmin == NC2.Node.xmin) && ...
                        all(NC1.Node.xmax == NC2.Node.xmax)

```

```

    Dh = 2*(NC1.End - NC2.Start);
else
    if isscalar(NC1.End)
        if isscalar(NC2.End)
            D2 = min([NC1.End NC2.End]);
        else
            D2 = NC2.End;
            D2(D2 > NC1.End) = NC1.End;
        end
    else
        if isscalar(NC2.End)
            D2 = NC1.End;
            D2(D2 > NC2.End) = NC2.End;
        else
            D2 = min([NC1.End; NC2.End]);
        end
    end
    if isscalar(NC1.Start)
        if isscalar(NC2.Start)
            D1 = max([NC1.Start NC2.Start]);
        else
            D1 = NC2.Start;
            D1(D1 < NC1.Start) = NC1.Start;
        end
    else
        if isscalar(NC2.Start)
            D1 = NC1.Start;
            D1(D1 < NC2.Start) = NC2.Start;
        else
            D1 = max([NC1.Start; NC2.Start]);
        end
    end
    Dh = 2*(D2 - D1);
end
end
else
    switch NC1.Connection.Orient
    case enumOrient.Vertical
        if isscalar(NC1.End)
            if isscalar(NC2.End)
                D2 = min([NC1.End NC2.End]);
            else
                D2 = NC2.End;
                D2(D2 > NC1.End) = NC1.End;
            end
        else
            if isscalar(NC2.End)
                D2 = NC1.End;
                D2(D2 > NC2.End) = NC2.End;
            else
                D2 = min([NC1.End; NC2.End]);
            end
        end
        if isscalar(NC1.Start)
            if isscalar(NC2.Start)
                D1 = max([NC1.Start NC2.Start]);
            else
                D1 = NC2.Start;
                D1(D1 < NC1.Start) = NC1.Start;
            end
        else
            if isscalar(NC2.Start)
                D1 = NC1.Start;
                D1(D1 < NC2.Start) = NC2.Start;
            else
                D1 = max([NC1.Start; NC2.Start]);
            end
        end
        Dh = 2*(D2 - D1);
        Dh(Dh<0) = 0;
    case enumOrient.Horizontal

```

```

        if isscalar(NC1.End)
            if isscalar(NC2.End)
                D2 = min([NC1.End NC2.End]);
            else
                D2 = NC2.End;
                D2(D2 > NC1.End) = NC1.End;
            end
        else
            if isscalar(NC2.End)
                D2 = NC1.End;
                D2(D2 > NC2.End) = NC2.End;
            else
                D2 = min([NC1.End; NC2.End]);
            end
        end
        if isscalar(NC1.Start)
            if isscalar(NC2.Start)
                D1 = max([NC1.Start NC2.Start]);
            else
                D1 = NC2.Start;
                D1(D1 < NC1.Start) = NC1.Start;
            end
        else
            if isscalar(NC2.Start)
                D1 = NC1.Start;
                D1(D1 < NC2.Start) = NC2.Start;
            else
                D1 = max([NC1.Start; NC2.Start]);
            end
        end
        Dh = 2*(D2 - D1);
        Dh(Dh<0) = 0;
    end
end
else
    % This should never happen
    fprintf('XXX Perpendicular NodeContacts in Hydraulic Diameter Calc XXX\n');
end
end
function [keep] = AlignedMask(M,T,b1,b2)
Ms = M.Start;
Me = M.End;
N = max([length(Ms) length(Me) length(b1) length(b2)]);
if nargin > 2
    % Test lower bounds of Mask
    for i = 1:N
        msi = min(length(Ms),i);
        mei = min(length(Me),i);
        bli = min(length(b1),i);
        b2i = min(length(b2),i);
        if Ms(msi) >= b2(b2i)
            Ms(msi) = inf;
            Me(mei) = inf;
        else
            if Ms(msi) < b1(bli)
                Ms(msi) = b1(bli);
            end
        end
    end
end
% Test upper bounds of Mask
for i = 1:N
    msi = min(length(Ms),i);
    mei = min(length(Me),i);
    bli = min(length(b1),i);
    b2i = min(length(b2),i);
    if Me(mei) ~= inf
        if Me(mei) <= b1(bli)
            Me(mei) = -inf;
            Ms(msi) = -inf;
        else
            if Me(mei) > b2(b2i)

```

```

        Me(me_i) = b2(b2_i);
    end
end
end
end
end

keep = true;
ActiveTimes = ~(Ms >= T.End) + (T.Start >= Me));
if any(ActiveTimes)
    if isscalar(ActiveTimes)
        if Ms <= T.Start
            T.Start = Me;
        elseif Me >= T.End
            T.End = Ms;
        else
            temp = T.End;
            T.End = Ms;
            NewNC = NodeContact(...
                T.Node, Me, temp, T.Type, T.Connection);
            if NewNC.Start < NewNC.End
                T.Connection.addNodeContacts(NewNC);
            end
        end
        if T.Start >= T.End
            keep = false;
            return;
        end
    else
        for i = 1:length(ActiveTimes)
            ms = min(length(Ms),i);
            me = min(length(Me),i);
            ts = min(length(T.Start),i);
            te = min(length(T.End),i);
            if Ms(ms) <= T.Start(ts)
                T.Start(ts) = Me(me);
            elseif Me(me) >= T.End(te)
                T.End(te) = Ms(ms);
            else
                temp = T.End(te);
                T.End(te) = Ms(ms);
                NewNC = NodeContact(...
                    T.Node, Me(me), temp, T.Type, T.Connection);
                if NewNC.Start < NewNC.End
                    T.Connection.addNodeContacts(NewNC);
                end
            end
            if T.Start(ts) >= T.End(te)
                T.Start(ts) = T.End(te);
            end
        end
        if all(T.Start == T.End)
            keep = false;
            return;
        end
    end
end
end
function [keep1, keep2] = MutualMask(M1,M2)
    Mask1 = CopyClass(M1);
    Mask2 = CopyClass(M2);
    keep1 = Mask1.AlignedMask(M2,-inf,inf);
    keep2 = Mask2.AlignedMask(M1,-inf,inf);
end
end
end

```

## NonConnection

The non-connection is a class that includes a constructor and a name generation function.

```
classdef NonConnection
    %NONCONNECTION Summary of this class goes here
    % Detailed explanation goes here

    properties
        Body1;
        Body2;
    end

    properties (Dependent)
        name;
    end

    methods
        function this = NonConnection(B1,B2)
            if nargin == 0
                return;
            end
            this.Body1 = B1;
            this.Body2 = B2;
        end

        function name = get.name(this)
            name = [this.Body1.name ' XXX ' this.Body2.name];
        end
    end
end
```

# Position

The position is class that contains a constructor, plus operator and get/set interface.

```
classdef Position < handle
    %POSITION Summary of this class goes here
    % Detailed explanation goes here

    properties
        x double = 0;
        y double = 0;
        Rot double = pi/2;
    end

    properties (Dependent)
        name;
    end

    methods
        function this = Position(x,y,Rot)
            switch nargin
                case 1
                    this.x = x;
                case 2
                    this.x = x;
                    this.y = y;
                case 3
                    this.x = x;
                    this.y = y;
                    this.Rot = Rot;
            end
        end

        function newPosition = plus(base,offset)
            newPosition.x = base.x + offset.x;
            newPosition.y = base.y + offset.y;
            newPosition.Rot = base.Rot;
            newPosition.Model = base.Model;
        end

        function name = get.name(this)
            name = sprintf('x: %f.0 y: %f.0 Rot: %f.00',this.x,this.y,this.Rot);
        end

        function Item = get(this,PropertyName)
            switch PropertyName
                case 'x'
                    Item = this.x;
                case 'y'
                    Item = this.y;
                case 'Theta'
                    Item = this.Rot;
                otherwise
                    fprintf(['XXX Position GET Inteface for ' PropertyName ' is not found XXX\n']);
            end
        end

        function set(this,PropertyName,Item)
            switch PropertyName
                case 'x'
                    this.x = Item;
                case 'y'
                    this.y = Item;
                case 'Theta'
                    this.Rot = Item;
                otherwise
                    fprintf(['XXX Position SET Inteface for ' PropertyName ' is not found XXX\n']);
            end
        end
    end
end
```



end

## Pressure Contact

The pressure contact is a class that contains a constructor and a equals operator.

```
classdef PressureContact
    %FORCECONTACT Summary of this class goes here
    % Detailed explanation goes here

    properties
        ConverterIndex;
        MechanismIndex;
        Area;
        GasNode;
    end

    methods
        function this = PressureContact(ConverterIndex,MechanismIndex,Area,Node)
            %FORCECONTACT Construct an instance of this class
            % Detailed explanation goes here
            this.ConverterIndex = ConverterIndex;
            this.MechanismIndex = MechanismIndex;
            this.Area = Area;
            this.GasNode = Node;
        end

        function iseq = equal(this,other)
            if this.MechanismIndex == other.MechanismIndex && ...
                this.Area == other.Area && this.GasNode == other.GasNode
                iseq = true;
            else
                iseq = false;
            end
        end
    end
end
end
```

## PVoutput

The PVoutput is a class that contains the following functionality:

A constructor.

A get / set interface.

A set of property update functions called during discretization.

A function called during destruction.

A set of functions for getting data during recording.

A plotting function for output.

```
classdef PVoutput < handle
    %UNTITLED Summary of this class goes here
    % Detailed explanation goes here

    properties
        Body Body;
        name;
        % A series of node indexes over
        % ... which the pressure and volume is calculated
        Nodes cell;
        RegionNodes;
        Region;
        Model Model;
        P double;
        V double;
        Power double;
        Fig = [];
    end

    methods
        function this = PVoutput(Body)
            if nargin == 1
                Body.addPVoutput(this);
                this.Body = Body;
                this.Model = Body.Group.Model;
                this.Nodes = [];
                this.P = zeros(1,Frame.NTheta-1);
                this.V = this.P;
                % Define the name
                this.name = getProperName( 'PV output' );
            end
        end

        function Item = get(this,PropertyName)
            switch PropertyName
            case 'name'
                Item = this.name;
            case 'Source Body/Region'
                Item = this.Body;
            otherwise
                fprintf(['XXX PVoutput GET Inteface for ' PropertyName ' is not found XXX\n']);
            end
        end

        function set(this,PropertyName,Item)
```

```

switch PropertyName
case 'name'
    this.name = Item;
case 'Source Body/Nodes'
    if ~isempty(Item)
        this.Body = Item;
        this.Body.change();
    end
    otherwise
        fprintf(['XXX Connection SET Inteface for ' PropertyName ' is not found XXX\n']);
        return;
    end
end
end

function update(this,Region)
% Grab a node from the body
this.RegionNodes = [];
this.Nodes = cell(0);
if this.Body.isDiscretized()
    i = 1;
    this.Region = Region(this.Body.Nodes(1).index);
    consideredbodies = Body.empty;
    for iGroup = this.Model.Groups
        for iBody = iGroup.Bodies
            if ~any(consideredbodies == iBody) && ...
                iBody.matl.Phase == enumMaterial.Gas && ...
                iBody.isDiscretized() && ...
                Region(iBody.Nodes(1).index) == this.Region
                % get this.Nodes{i}(j)
                j = 1;
                otherbodies = iBody;
                k = 1;
                while k <= length(otherbodies)
                    for nd = otherbodies(k).Nodes % Search this body & any other bodies
                        if nd.Type ~= enumNType.SN && ~isscalar(nd.vol())
                            this.Nodes{i}(j) = nd.index; j = j + 1;
                            for fc = nd.Faces
                                if fc.Type == enumFType.Gas && ...
                                    fc.Nodes(1).Body ~= fc.Nodes(2).Body
                                    if fc.Nodes(1) == nd
                                        if ~isscalar(fc.Nodes(2).vol()) && ...
                                            ~any(otherbodies == fc.Nodes(2).Body)
                                            otherbodies(end+1) = fc.Nodes(2).Body;
                                        end
                                    else
                                        if ~isscalar(fc.Nodes(1).vol()) && ...
                                            ~any(otherbodies == fc.Nodes(1).Body)
                                            otherbodies(end+1) = fc.Nodes(1).Body;
                                        end
                                    end
                                end
                            end
                        end
                    end
                    k = k + 1;
                end
                consideredbodies(end+1:end+length(otherbodies)) = otherbodies(:);
                % get this.RegionNodes(this.Nodes{i}(:));
                if j > 1
                    i = i + 1;
                end
            end
        end
    end
    indexes = 1:length(Region);

    this.RegionNodes = indexes(Region(end) ~= this.Region);
else
    this.Nodes = cell(0);
    return;
end
end

```

```

end

function reset(this)
    this.P = [];
    this.V = [];
end

function deReference(this)
    iModel = this.Model;
    for i = length(iModel.PVoutputs):-1:1
        if iModel.PVoutputs(i) == this
            iModel.PVoutputs(i) = [];
            break;
        end
    end
    if ~isempty(this.Body) && isValid(this.Body)
        for i = length(this.Body.PVoutputs):-1:1
            if this.Body.PVoutputs(i) == this
                this.Body.PVoutputs(i) = [];
                break;
            end
        end
        this.Body.change();
    end
    this.delete();
end

function isequal = equal(this,other)
    isequal = (this.Body == other.Body);
end

function getData(this,Sim)
    if isempty(this.P)
        this.P = zeros(Frame.NTheta-1,length(this.Nodes));
    end
    index = Sim.Inc;
    for i = 1:length(this.Nodes)
        indV = (Sim.vol(this.Nodes{i})' + Sim.old_vol(this.Nodes{i})')/2;
        sumV = sum(sum(indV));
        this.P(index,i) = sum(indV.*Sim.P(this.Nodes{i})')/sumV;
        this.V(index,i) = sumV;
    end
end

function updatePlot(this)
    if isempty(this.Fig) || ~isValid(this.Fig) || this.Fig < 1
        this.Fig = figure();
    end
    figure(this.Fig);
    title('Pressure vs Volume Diagram');
    xlabel('Volume (m^3)');
    ylabel('Pressure (Pa)');
    set(gcf,'color','w');
    a = this.Fig.CurrentAxes;
    WTotal = 0;
    Text = '';
    for i = 1:length(this.Nodes)
        pV = zeros(size(this.V,1)+1,1); pP = pV;
        pV(1:end-1) = this.V(:,i); pV(end) = this.V(1,i);
        pP(1:end-1) = this.P(:,i); pP(end) = this.P(1,i);
        W = PowerFromPV(pP,pV);
        WTotal = WTotal + W;
        if W > 0; Color = 'b'; % Color is Blue
        else; Color = 'r'; % Color is Red
        end
        plot(pV,pP,'Color',Color,'LineStyle','-');
        hold on;
        plot(pV(1),pP(1),'Color','k','Marker','o');
    end
    this.Power = WTotal;
    Text = [Text 'Total = ' num2str(WTotal,4) 'Joules/Cycle'];
end

```

```

    %fprintf([num2str(WTotal) '\n']);
    text(a.XLim(1)+0.01*(a.XLim(2)-a.XLim(1)),...
         a.YLim(2)-0.05*(a.YLim(2)-a.YLim(1)), Text);
    drawnow;
    hold off;
end

function plotData(this,is_saved,ModelName)
    if ~(isempty(this.Fig) || ~isvalid(this.Fig) || this.Fig < 1)
        close(this.Fig);
    end
    oldfigure =(gcf);
    oldaxes = gca;
    a = gca;
    updatePlot(this);
    h =(gcf);
    xlabel('Volume (m^3)');
    ylabel('Pressure (Pa)');

    if is_saved
        frame = getframe(h);
        im = frame2im(frame);
        [imind,cm] = rgb2ind(im,256);
        data = struct(...
            'Name',this.name,...
            'IndependentVariable',this.V,...
            'DependentVariable',this.P);
        if isempty(this.Body.Group.Model.outputPath)
            str = [this.name '_' ModelName];
        else
            str = [this.Body.Group.Model.outputPath '\' ...
                this.name '_' ModelName];
        end
        str = replace(str,':',' -');
        save([str '.mat'],'data');
        imwrite(imind,cm,[str '.jpg']);
    end

    close(h);
    figure(oldfigure);
    axes(oldaxes);
end
end
end

```

## Sensor

A sensor is a class that contains the following functionality:

A constructor.

A dereference function (called prior to deletion).

A get / set interface.

An update function called during discretization.

A set of functions for getting data during recording.

A plotting function for output.

A display function for displaying on the GUI.

```
classdef Sensor < handle
    %UNTITLED Summary of this class goes here
    % Detailed explanation goes here

    properties
        name;
        Body;
        Model;
        LocationStyle;
        averaging; % 1 for phase based, 2 for time based
        IndependentVariable; % May be time or angle
        DataType; % Cell Array
        Data; % Cell Array
        dimensions; % 1 for point, 2 for line
        PntCount; % Number of elements along a line
        Nodes; % Vector
        Interp; % Matrix

        % Derived Components
        PlotCoordinates; % Vector 1xN
        LocalCoordinates;

        index = 0; % adding to the data set
        GUIObjects;
    end

    methods
        function this = Sensor(Modelobj,Body)
            if nargin == 2
                this.Body = Body;
                this.Model = Modelobj;
                this.index = 0;

                % Define the name
                this.name = getProperName( 'Sensor' );

                % Define the location
                notdone = true;
                source = {...
                    'Body Center','Body xmax',...
                    'Body xmin','Body ymax',...
                }
            end
        end
    end
end
```

```

        'Body ymin','Body xaxis',...
        'Body yaxis'};
while notdone
    index = listdlg('PromptString','What is the position of this Senso?','...
        'SelectionMode','single',...
        'ListString',source);
    if index ~= 0
        notdone = false;
    end
end
this.LocationStyle = source{index};

% Define the independent variable
notdone = true;
source = {'Phase','Time'};
while notdone
    index = listdlg('PromptString','What is the independent variable?','...
        'SelectionMode','single',...
        'ListString',source);
    if index ~= 0
        notdone = false;
    end
end
this.averaging = index;

% Define the dependent variable
notdone = true;
source = {'T','P','turb'};
while notdone
    index = listdlg('PromptString','What is the dependent variable?','...
        'SelectionMode','single',...
        'ListString',source);
    if index ~= 0
        notdone = false;
    end
end
this.DataType = source{index};

% Define the point count
switch this.LocationStyle
case {'Body xaxis','Body yaxis'}
    notdone = true;
    while notdone
        answer = inputdlg('How many sample points along the line?','...
            'Integer only',[1 50]);
        test = str2double(answer);
        if ~isnan(test)
            if floor(test) == test
                this.PntCount = test-1;
                notdone = false;
            end
        end
    end
end
Body.addSensor(this);
this.update();
end
end
function deReference(this)
if ~isempty(this.Model) && isvalid(this.Model)
for i = length(this.Model.Sensors):-1:1
    if this.Model.Sensors(i) == this
        this.Model.Sensors(i) = [];
        break;
    end
end
end
end
if ~isempty(this.Body) && isvalid(this.Body)
for i = length(this.Body.Sensors):-1:1
    if this.Body.Sensors(i) == this
        this.Body.Sensors(i) = [];
    end
end
end

```



```

        break;
    end
    end
    this.Body.change();
end
this.removeFromFigure(gca);
this.delete();
end
function isit = isValid(this)
    isit = invalid(this) && isempty(this.Body) && ...
        ~isempty(this.Model) && invalid(this.Body) && invalid(this.Model);
end
function item = get(this,PropertyName)
    switch PropertyName
    case 'Name'
        item = this.name;
    case 'Samples'
        item = this.PntCount + 1;
    otherwise
        fprintf(['XXX Sensor GET Inteface for ' PropertyName ...
            ' is not found XXX\n']);
    end
end
function set(this,PropertyName,Item)
    switch PropertyName
    case 'Name'
        this.name = Item;
    case 'Samples'
        this.PntCount = Item - 1;
        this.update();
    otherwise
        fprintf(['XXX Sensor SET Inteface for ' PropertyName ...
            ' is not found XXX\n']);
    end
end
function update(this)
    if ~this.Body.isDiscretized()
        this.Body.discretize();
    if ~this.Body.isDiscretized()
        fprintf(['XXX Sensor: ' this.name ...
            ' Update failed XXX\n']);
        return;
    end
end
switch this.LocationStyle
case 'Body Center'
    % place the point right in the middle of the body volume and use
    % the nodes as the interpolation points
    this.dimensions = 1; % Point
    [~,~,xmin,xmax] = this.Body.limits(enumOrient.Vertical);
    [~,~,ymin,ymax] = this.Body.limits(enumOrient.Horizontal);
    x = (xmin+xmax)/2; y = (ymin+ymax)/2;
    this.LocalCoordinates = [x; y];
case 'Body xmax'
    % place the point right at the middle of the xmax edge.
    this.dimensions = 1; % Point
    [~,~,~,xmax] = this.Body.limits(enumOrient.Vertical);
    [~,~,ymin,ymax] = this.Body.limits(enumOrient.Horizontal);
    x = xmax; y = (ymin+ymax)/2;
    this.LocalCoordinates = [x; y];
case 'Body xmin'
    % place the point right at the middle of the xmin edge.
    this.dimensions = 1; % Point
    [~,~,xmin,~] = this.Body.limits(enumOrient.Vertical);
    [~,~,ymin,ymax] = this.Body.limits(enumOrient.Horizontal);
    x = xmin; y = (ymin+ymax)/2;
    this.LocalCoordinates = [x; y];
case 'Body ymax'
    % place the point right at the middle of the ymax edge.
    this.dimensions = 1; % Point
    [~,~,xmin,xmax] = this.Body.limits(enumOrient.Vertical);

```

```

    [~,~,~,ymax] = this.Body.limits(enumOrient.Horizontal);
    x = (xmin+xmax)/2; y = ymax;
    this.LocalCoordinates = [x; y];
case 'Body ymin'
    % place the point right at the middle of the ymin edge.
    this.dimensions = 1; % Point
    [~,~,xmin,xmax] = this.Body.limits(enumOrient.Vertical);
    [~,~,ymin,~] = this.Body.limits(enumOrient.Horizontal);
    x = (xmin+xmax)/2; y = ymin;
    this.LocalCoordinates = [x; y];
case 'Body yaxis'
    % Place the point along the xaxis align volume center
    this.dimensions = 2; % Line
    [~,~,xmin,xmax] = this.Body.limits(enumOrient.Vertical);
    [~,~,ymin,ymax] = this.Body.limits(enumOrient.Horizontal);
    x1 = (xmin+xmax)/2; y1 = ymin;
    x2 = (xmin+xmax)/2; y2 = ymax;
    this.PlotCoordinates = linspace(y1,y2,this.PntCount+1);
    this.LocalCoordinates = [x1 y1; x2 y2];
case 'Body xaxis'
    % Place the point along the yaxis align volume center
    this.dimensions = 2; % Line
    [~,~,xmin,xmax] = this.Body.limits(enumOrient.Vertical);
    [~,~,ymin,ymax] = this.Body.limits(enumOrient.Horizontal);
    x1 = xmin; y1 = (ymin+ymax)/2;
    x2 = xmax; y2 = (ymin+ymax)/2;
    this.PlotCoordinates = linspace(x1,x2,this.PntCount+1);
    this.LocalCoordinates = [x1 y1; x2 y2];
end
if this.dimensions == 1
    % Point
    loc = Pnt2D(x,y);
    [nds,intrp] = findClosest4(loc,this.Body);
    this.Nodes = nds;
    this.Interp = intrp;
else
    % Line
    this.Nodes = zeros(this.PntCount+1,4);
    this.Interp = zeros(this.PntCount+1,4);
    loc(this.PntCount+1) = Pnt2D();
    for i = 1:this.PntCount+1
        loc(i) = Pnt2D(...
            x1*(this.PntCount-i+1)/(this.PntCount) + x2*(i-1)/this.PntCount,...
            y1*(this.PntCount-i+1)/(this.PntCount) + y2*(i-1)/this.PntCount);
        [nds, intrp] = findClosest4(loc(i),this.Body);
        this.Nodes(i,1:length(nds)) = nds;
        this.Interp(i,1:length(intrp)) = intrp;
    end
    % Simplify
    % Form the nodes array
    tempNodes = zeros(numel(this.Nodes),1);
    start = 1;
    len = 3;
    for i = 1:this.PntCount+1
        tempNodes(start:start+len) = this.Nodes(i,:);
        start = start + len + 1;
    end
    tempNodes = unique(tempNodes(tempNodes>0));
    tempNodes = sort(tempNodes);
    tempInterp = zeros(length(tempNodes),this.PntCount+1);
    for i = 1:this.PntCount+1
        for j = 1:length(tempNodes)
            where = find(this.Nodes(i,:) == tempNodes(j));
            if isempty(where)
                tempInterp(j,i) = 0;
            else
                tempInterp(j,i) = this.Interp(i,where(1));
            end
        end
    end
end
this.Interp = tempInterp;

```

```

        this.Nodes = tempNodes;
    end
    if this.Model.showSensors
        this.show(gca);
    end
end
function reset(this)
    this.IndependentVariable = [];
    this.index = 0;
    this.Data = [];
end

function getData(this,Simulation)
    property = this.DataType;
    switch property
        case 'T'
            SourceData = Simulation.T(this.Nodes);
        case 'P'
            SourceData = Simulation.P(this.Nodes);
        case 'Turb'
            SourceData = Simulation.turb(this.Nodes);
        otherwise
            fprintf(['XXX Property: ' property ...
                ' not supported in the Sensor Class XXX\n']);
            return;
        end
    % Get the data by interpolating the cells
    switch this.averaging
        case 1 % Angular Recording with overwrite
            this.index = Simulation.Inc;
            if isempty(this.IndependentVariable)
                LEN = Frame.NTheta-1;
                AInc = 2*pi/(Frame.NTheta-1);
                this.IndependentVariable = linspace(0,AInc*LEN,LEN);
            end
        case 2 % Temporal Recording
            this.index = this.index + 1;
            this.IndependentVariable(this.index) = Simulation.curTime;
    end
    switch this.dimensions
        case 1
            % Grab a single point
            this.Data(this.index) = sum(this.Interp(:).*SourceData(:));
        case 2
            % Grab a vector of points
            if isempty(this.Data)
                this.Data(this.PntCount+1,1) = 0;
            end
            for j = 1:this.PntCount+1
                this.Data(j,this.index) = sum(this.Interp(:,j).*SourceData(:));
            end
        end
    end
end

function plotData(this,is_saving,ModelName)
    oldfigure = gcf;
    oldaxes = gca;
    h = figure();
    set(h,'color','w');
    a = gca;
    switch this.DataType
        case 'T'
            titleStr = [this.name ': Temperature vs '];
            label2 = 'Temperature (K)';
            switch this.dimensions
                case 1
                    a.YAxis.TickLabelFormat = '%.1f';
                case 2
                    a.YAxis.TickLabelFormat = '%.2f';
            end
        case 'P'

```

```

titleStr = [this.name ': Pressure vs '];
label2 = 'Pressure (Pa)';
switch this.dimensions
    case 1
        a.YAxis.TickLabelFormat = '%.0f';
    case 2
        a.YAxis.TickLabelFormat = '%.2f';
    end
case 'turb'
    titleStr = [this.name ': Turbulent Weight vs '];
    label2 = 'Turbulence Weight (0-1)';
    switch this.dimensions
        case 1
            a.YAxis.TickLabelFormat = '%.2f';
        case 2
            a.YAxis.TickLabelFormat = '%.2f';
        end
    end
end
switch this.dimensions
    case 1
        % Make a line plot
        if length(this.Data) == length(this.IndependentVariable)
            plot(this.IndependentVariable,this.Data);
        else
            plot(this.Data);
            fprintf('XXX Sensor could not plot due to unequal lengthed vectors. XXX\n');
        end
        switch this.averaging
            case 1
                % Make a plot in relation to angle
                titleStr = [titleStr 'angle'];
                title(titleStr);
                xlabel('angle (rad)');
                ylabel(label2);
                a.XAxis.TickLabelFormat = '%.2f';
            case 2
                % Make a plot in relation to time
                titleStr = [titleStr 'time'];
                title(titleStr);
                xlabel('time (s)');
                ylabel(label2);
                a.XAxis.TickLabelFormat = '%.0f';
            end
        end
    case 2
        % Make a surface plot
        [X,Y] = meshgrid(this.IndependentVariable,this.PlotCoordinates);
        Z = this.Data;
        s = surf(X,Y,Z);
        s.EdgeColor = 'none';
        view(0,90);
        xlim([0, max(this.IndependentVariable)]);
        ylim([min(this.PlotCoordinates), max(this.PlotCoordinates)]);
        colormap jet;
        hcb = colorbar;
        switch this.DataType
            case 'T'
                yt=get(hcb,'Ticks');
                set(hcb,'XTickLabel',strtrim(cellstr(num2str(yt,'%1f'))));
            case 'P'
                yt=get(hcb,'Ticks');
                set(hcb,'XTickLabel',strtrim(cellstr(num2str(yt,'%2e'))));
            case 'turb'
                yt=get(hcb,'Ticks');
                set(hcb,'XTickLabel',strtrim(cellstr(num2str(yt,'%2f'))));
        end
    end
    switch this.averaging
        case 1
            % Make a plot in relation to angle
            titleStr = [titleStr 'angle'];
            t = title(titleStr);
            xlabel('angle (rad)');

```

```

        ylabel('position (m)');
        xlabel(label2);
    case 2
        % Make a plot in relation to time
        titleStr = [titleStr 'time'];
        t = title(titleStr);
        xlabel('time (s)');
        ylabel('position (m)');
        xlabel(label2);
    end
    ylabel(hcb, titleStr);
end

if is saving
    frame = getframe(h);
    im = frame2im(frame);
    [imind,cm] = rgb2ind(im,256);
    data = struct(...
        'Name',this.name,...
        'IndependentVariable',this.IndependentVariable,...
        'DependentVariable',this.Data);
    if isempty(this.Body.Group.Model.outputPath)
        str = [ModelName '_' titleStr];
    else
        str = [this.Body.Group.Model.outputPath '\' ...
            ModelName '_' titleStr];
    end
    str = replace(str,':',' -');
    save([str '.mat'],'data');
    imwrite(imind,cm,[str '.jpg']);
end

close(h);
figure(oldfigure);
axes(oldaxes);
end
function removeFromFigure(this,AxisReference)
    if ~isempty(this.GUIObjects)
        children = get(AxisReference,'Children');
        for obj = this.GUIObjects
            if isgraphics(obj)
                for i = length(children):-1:1
                    if isgraphics(children(i)) && children(i) == obj
                        children(i).delete;
                        break;
                    end
                end
            end
        end
        this.GUIObjects = [];
    end
end
function show(this,AxisReference)
    this.removeFromFigure(AxisReference);
    color = [1 0 1]; % magenta
    % Plot a yellow symbol where the sensor is
    if size(this.LocalCoordinates,2) == 2
        % It is a line
        pos1 = RotMatrix(this.Body.Group.Position.Rot-pi/2)*this.LocalCoordinates(1,:);
        pos2 = RotMatrix(this.Body.Group.Position.Rot-pi/2)*this.LocalCoordinates(2,:);
        this.GUIObjects = line([pos1(1) pos2(1)],[pos1(2) pos2(2)],...
            'Color',color,'Marker','o','MarkerSize',8);
    else
        % It is a point
        pos = RotMatrix(this.Body.Group.Position.Rot-pi/2)*this.LocalCoordinates(:);
        this.GUIObjects = line(pos(1),pos(2),...
            'Color',color,'Marker','o','MarkerSize',8);
    end
end
end
end

```

end

## Shear Contact

A shear contact is a class that includes a constructor and an equals operator.

```
classdef ShearContact
    %FORCECONTACT Summary of this class goes here
    % Detailed explanation goes here

    properties
        ConverterIndex;
        MechanismIndex;
        Area;
        LowerNode;
        UpperNode;
        ActiveTimes;
    end

    methods
        function this = ShearContact(...)
            ConverterIndex,MechanismIndex,Area,Node1,Node2,ActiveTimes)
            if nargin == 6
                %FORCECONTACT Construct an instance of this class
                % Detailed explanation goes here
                this.ConverterIndex = ConverterIndex;
                this.MechanismIndex = MechanismIndex;
                this.Area = Area;
                this.LowerNode = Node1;
                this.UpperNode = Node2;
                this.ActiveTimes = ActiveTimes;
            end
        end

        function iseq = equal(this,other)
            if this.ConverterIndex == other.ConverterIndex && ...
                this.MechanismIndex == other.MechanismIndex && ...
                this.LowerNode == other.LowerNode && ...
                this.UpperNode == other.UpperNode
                iseq = true;
            else
                iseq = false;
            end
        end
    end
end
```

## **G.4. Mechanical**



## Linear to Rotational Mechanism

The linear to rotational mechanism is a class that includes the following functionality:

A static function that assembles the property table.

A constructor and destruction function.

A function that takes the user inputted properties and turns them into coefficients for load calculations.

A function that opens up the mechanism interface.

A set of functions that may be referenced in the form of anonymous functions during simulation.

As they apply to different mechanism types.

```
classdef LinRotMechanism < handle
    %MECHANISM Summary of this class goes here
    % Detailed explanation goes here

    properties (Constant)
        g = 9.81;
        Source = {...
            'Ideal Sinusoid';
            'Custom Profile Mechanism';
            'Slider Crank'};
        %     'Scotch Yoke';
        %     'Ross Yoke';
        %     'Rhombic Drive';
        StrokeText = 'Stroke (m)';
        PhaseText = 'Phase (rad)';
        PistonMassText = 'Piston Mass (kg)';
        TiltAngleText = 'Tilt from Hor. (rad)';
        OrientationText = 'Orientation: "u" aligned with positive y, "d" opposite';
        EfficiencyText = 'Mechanical Efficiency (0.##)';
        %     isPropertyEditable ...
        %     = {'Stroke (m)', 'Phase (rad)', 'Weight (kg)', 'Tilt from Hor. (rad)', 'Aspect
Ratio', 'Custom Profile Fcn';
        %         true           , true           , true           , true           , false
, true           ;
        %         true           , true           , true           , true           , true
, false          ;
        %         true           , true           , true           , true           , false
, false          };

    end

    properties
        ID;
        Model Model;

        isValid logical = true;
        Type char;
        Stroke double = [];
        Phase double = [];
        Frames Frame;

        STilt double;
        CTilt double;
```

```

%     lengths double = [];
%     masses double = [];
%     descriptions char = [];

originalInput = [];

outputFcn function_handle;

Data;

dont_propegate logical = false;

end

properties (Dependent)
    name;
end

methods (Static)
function [Source, Instructions, Widths] = GetPropertyTableSource(Type,originalSource)
    switch Type
        case 'Ideal Sinusoid'
            Source = {...
                LinRotMechanism.StrokeText, ...
                LinRotMechanism.PhaseText, ...
                LinRotMechanism.PistonMassText, ...
                'Other Mass (kg)', ...
                LinRotMechanism.TiltAngleText, ...
                LinRotMechanism.OrientationText, ...
                LinRotMechanism.EfficiencyText};
            Source = AddRow(Source,1);
            Instructions = [...
                'The ideal Sinusoid produces N frictionless osscilating ' ...
                'mass mechanism that follows a perfect sinusoidal motion. ' ...
                'This is best used when the mechanism is unknown or seal ' ...
                'and pumping losses are much greater than mechanism friction.'];
        case 'Custom Profile Mechanism'
            Source = {...
                LinRotMechanism.StrokeText, ...
                LinRotMechanism.PhaseText, ...
                LinRotMechanism.PistonMassText, ...
                'Other Mass (kg)', ...
                'Mech. Mom. Inert. (kg m^2)', ...
                LinRotMechanism.TiltAngleText, ...
                'Custom Profile Fcn', ...
                LinRotMechanism.OrientationText, ...
                LinRotMechanism.EfficiencyText};
            Source = AddRow(Source,1);
            Instructions = [...
                'The Custom Profile Mechanism produces N simulated osscilating ' ...
                'mass mechanism that follow a custom motion profile. ' ...
                'Piston Mass includes the mass of any attached components. ' ...
                'The user has the option to simulate the mass effects of ' ...
                'non-circular gear pairs or cam drives.'];
        case 'Slider Crank'
            Source = {...
                LinRotMechanism.StrokeText, ...
                LinRotMechanism.PhaseText, ...
                'Crank Mass (kg)', ...
                'Crank C.O.M Radius (m)', ...
                'Crank C.O.M Angle (rad)', ...
                'Crank Rot. Inertia (kgn^2)',...
                'Crank-Con Fric. Fcn', ...
                'Con Length (m)', ...
                'Con. Mass (kg)', ...
                'Con. C.O.M Radius from CR-CN pin (m)', ...
                'Con. Rot. Inertia (kgn^2)',...
                'Con-Piston Fric. Fcn', ...
                LinRotMechanism.PistonMassText, ...
                'Piston Fric. Fcn', ...
                LinRotMechanism.TiltAngleText, ...

```

```

'Slider-Offset (m)', ...
LinRotMechanism.OrientationText);
Source = AddRow(Source,1);
Instructions = [...
'The slider crank produces N psuedo sinusoidal motions. ' ...
'It auto-defines mechanism lengths based on ' ...
'Stroke and Aspect Ratio. Piston, Crank and Connecting Arm ' ...
'Masses are by default placed in the center of their bars; ' ...
'the user can modify this location or choose the ' ...
'AutoBalance option. Friction Fcns take an input of a normal '...
'force and provide back a opposing force.'];
%
case 'Scotch Yoke'
%
Source = {...
%
LinRotMechanism.StrokeText, ...
%
LinRotMechanism.PhaseText, ...
%
LinRotMechanism.PistonMassText, ...
%
'Crank Mass (kg)', ...
%
'Crank Length (m)', ...
%
'Crank C.O.M Radius (m)', ...
%
'Crank C.O.M Angle (rad)', ...
%
'Roller Fric. Fcn', ...
%
'Linear Bearing Fric. Fcn', ...
%
'Mech. Mom. Inter. (kg m^2)', ...
%
LinRotMechanism.TiltAngleText
%
LinRotMechanism.OrientationText);
Source = AddRow(Source,1);
Instructions = [...
%
'The scotch yoke produces N perfect sinusoidal motions. ' ...
%
'It auto-defines mechanism lengths based on ' ...
%
'Stroke. Crank structural masses are placed in the center ' ...
%
'of its bar, the user can modify this location or choose ' ...
%
'the AutoBalance option. Piston Mass includes the mass of ' ...
%
'attached components. Friction Fcns take an input of a ' ...
%
'normal force and provide back an opposing force.'];
case 'Rhombic Drive'
%
Source = {...
%
LinRotMechanism.StrokeText, ...
%
LinRotMechanism.PhaseText, ...
%
'Crank Mass (kg)', ...
%
'Crank C.O.M Radius (m)', ...
%
'Crank C.O.M Angle (rad)', ...
%
'Crank Rot. Inertia (kgn^2)',...
%
'Crank-Con Fric. Fcn', ...
%
'Con Length (m)', ...
%
'Con. Mass (kg)', ...
%
'Con. C.O.M Radius from CR-CN pin (m)', ...
%
'Con. Rot. Inertia (kgn^2)',...
%
'Con-Piston Fric. Fcn', ...
%
LinRotMechanism.PistonMassText, ...
%
'Piston Fric. Fcn', ...
%
LinRotMechanism.TiltAngleText, ...
%
'Slider-Offset (m)', ...
%
LinRotMechanism.OrientationText);
Source = AddRow(Source,1);
Instructions = [...
%
'The rhombic drive is a type of slider crank mechanism ' ...
%
'that uses 2 sets of cranks that are mirrored in such a ' ...
%
'way that they have 0 side loads. The Type of Rhombic ' ...
%
'drive used in Beta Type Stirling engines contains two ' ...
%
'sets one of which is phased at 180 degrees relative and ' ...
%
'of opposite orientation.'];
end
if nargin > 1
% Then try to prefill it as best as you can
Source = MergeTables(Source,originalSource);
end
for col = size(Source,2):-1:1
Widths{col} = length(Source{1,col})*6;
end
end
end
end
end

```

```

methods
function this = LinRotMechanism(Model,Type,PropertyTable)
    if nargin > 1
        this.Model = Model;
        this.ID = Model.getLRMID();
        this.Populate(Type,PropertyTable);
    end
end
function deReference(this)
    % Get index of this LinRotMechanism
    ind = 1;
    for iLinRotMech = this.Model.Converters
        if iLinRotMech == this; break;
        else; ind = ind + 1;
        end
    end

    for iGroup = this.Model.Groups
        for iCon = this.Model.Connections
            if ~isempty(iCon.RefFrame)
                if iCon.RefFrame.Mechanism == ind
                    iCon.RefFrame = [];
                    iCon.change();
                end
            end
        end
    end

    for i = length(this.Model.RefFrames):-1:1
        if this.Model.RefFrames(i).Mechanism == ind
            this.Model.RefFrames(i) = [];
        elseif this.Model.RefFrames(i).Mechanism > ind
            this.Model.RefFrames(i).Mechanism = ...
                this.Model.RefFrames(i).Mechanism - 1;
        end
    end

    this.Model.Converters(ind) = [];
    this.Model.change();
    this.delete();
end
function Populate(this,Type,PropertyTable)
    if isempty(this.ID)
        this.ID = this.Model.getLRMID();
    end
    this.Data = struct.empty;
    this.Type = Type;
    this.originalInput = PropertyTable;
    LEN = size(PropertyTable,1)-1;
    if isempty(this.Frames)
        this.Frames(LEN) = Frame();
    elseif length(this.Frames) < LEN
        % Chop off
        for i = length(this.Frames):-1:LEN+1
            this.Frames(i).deReference();
            this.Frames(i) = [];
        end
    elseif length(this.Frames) > LEN
        % Top up
        this.Frames(LEN) = Frame();
    end
    this.populateTilt();
    for i = 1:LEN
        newValue = str2double(FindInTable(this,this.StrokeText,i+1));
        if length(this.Stroke) >= i
            shift = newValue - this.Stroke(i);
            if shift ~= 0
                % Stroke
                if ~this.dont_propegate && this.Model.RelationOn
                    for iGroup = this.Model.Groups

```

```

for RMan = iGroup.RelationManagers
    if RMan.Orient == enumOrient.Horizontal
        maxconPiston = Connection.empty;
        maxconStroke = Connection.empty;
        maxxPiston = -inf;
        maxxStroke = -inf;
        for rel = RMan.Relations
            if ~isempty(rel.frame) && ...
                rel.frame == this.Frames(i)
                switch rel.mode
                    case enumRelation.Piston
                        if rel.con1.x > maxxPiston
                            maxxPiston = rel.con1.x;
                            maxconPiston = rel.con1;
                        end
                        if rel.con2.x > maxxPiston
                            maxxPiston = rel.con2.x;
                            maxconPiston = rel.con2;
                        end
                    case enumRelation.Stroke
                        if rel.con1.x > maxxStroke
                            maxxStroke = rel.con1.x;
                            maxconStroke = rel.con1;
                        end
                        if rel.con2.x > maxxStroke
                            maxxStroke = rel.con2.x;
                            maxconStroke = rel.con2;
                        end
                end
            end
        end
        success = true;
        if ~isempty(maxconStroke)
            [success, ~] = ...
                RMan.Edit(maxconStroke, shift);
        end
        if success && ...
            isempty(maxconStroke) && ~isempty(maxconPiston)
            [success, ~] = ...
                RMan.Edit(maxconPiston, -shift);
        end
        if ~success
            fprintf(['XXX Could not edit mechanism ' ...
                'due to geometric conflict XXX\n']);
        end
    end
end
end
end
end
end
end
end
this.Stroke(i) = newValue;
end
this.dont_propegate = false;
switch Type
    case {'Ideal Sinusoid', 'Custom Profile Mechanism'}
        for i = 1:LEN
            Im1 = 0;
            % Find Properties
            % Phase
            this.Phase(i) = str2double(FindInTable(this,this.PhaseText,i+1));
            % Piston Mass
            mp = str2double(FindInTable(this,this.PistonMassText,i+1));
            % Other Mass
            m1 = str2double(FindInTable(this,'Other Mass (kg)',i+1));
            % Tilt is already covered
            % Orientation
            orientation = FindInTable(this,LinRotMechanism.OrientationText,i+1);
            switch orientation
                case 'u'; orient = 1;
                case 'd'; orient = -1;
            end
        end
    end
end

```

```

        otherwise; orient = nan();
    end
    % Mechanical Efficiency
    eff = str2double(FindInTable(this, LinRotMechanism.EfficiencyText, i+1));

    if strcmp(Type, 'Custom Profile Mechanism')
        CustomFcn = str2func(FindInTable(this, 'Custom Profile Fcn', i+1));
        fh = functions(CustomFcn);
        isFcnValid = ~isempty(fh.file);
    else
        isFcnValid = true;
    end
    if isnan(this.Stroke(i)) || isnan(this.Phase(i)) || ...
        isnan(mp) || isnan(m1) || isnan(orient) || ...
        isnan(eff) || ~isFcnValid
        fprintf(...
            ['XXX ' Type ' is invalid, Frames not created. Trouble Components below.
            XXX\n']);
    end
    if isnan(this.Stroke(i))
        fprintf(...
            ['Stroke = ' FindInTable(this, this.StrokeText, i+1) '\n']);
    end
    if isnan(this.Phase(i))
        fprintf(...
            ['Phase = ' FindInTable(this, this.PhaseText, i+1) '\n']);
    end
    if isnan(mp)
        fprintf(...
            ['mp = ' FindInTable(this, this.PistonMassText, i+1) '\n']);
    end
    if isnan(m1)
        fprintf(...
            ['m1 = ' FindInTable(this, 'Other Mass (kg)', i+1) '\n']);
    end
    if isnan(orient)
        fprintf(...
            ['Orient = ' orientation '\n']);
    end
    if isnan(eff)
        fprintf(...
            ['Eff. = ' FindInTable(this, this.EfficiencyText, i+1) '\n']);
    end
    if isFcnValid
        fprintf(...
            ['Fcn. = ' FindInTable(this, 'Custom Profile Fcn', i+1) '\n']);
    end
    this.isValid = false;
    return;
end
%% Define motion of Frames
if strcmp(Type, 'Custom Profile Mechanism')
    CustomProfile = CustomFcn(Frame.NTheta, this.Phase(i));
else
    Ang = (0:Frame.NTheta-1)/(Frame.NTheta-1)*2*pi + this.Phase(i);
    CustomProfile = this.Stroke(i)/2 + this.Stroke(i)*cos(Ang)/2;
end
if orient == 1
    xmin = min(CustomProfile);
    xmax = max(CustomProfile);
else
    xmin = max(CustomProfile);
    xmax = min(CustomProfile);
end
this.Frames(i).Positions = (CustomProfile-xmin).*...
    (this.Stroke/(xmax-xmin));
this.Frames(i).MechanismIndex = i;
this.Frames(i).Mechanism = this;
defineDataFromMotionProfile(Iml, mp, m1, eff, orient, this, i); % (Iml, mp, m1, eff, this, ind)
end
case 'Slider Crank'
    Ang = zeros(LEN, Frame.NTheta);

```

```

for i = 1:LEN
    % Phase
    this.Phase(i) = str2double(FindInTable(this,this.PhaseText,i+1));
    Ang(i,:) = (0:Frame.NTheta-1)/(Frame.NTheta-1)*2*pi + this.Phase(i);
end
for i = 1:LEN
    d1 = this.Stroke(i);
    % Crank Mass (kg)
    m1 = str2double(FindInTable(this,'Crank Mass (kg)',i+1));
    % Crank C.O.M Radius (m)
    r1 = str2double(FindInTable(this,'Crank C.O.M Radius (m)',i+1));
    % Crank C.O.M Angle (rad)
    Ang_g1 = str2double(FindInTable(this,'Crank C.O.M Angle (rad)',i+1));
    % Crank Rot. Inertia (kgn^2)
    Im1 = str2double(FindInTable(this,'Crank Rot. Inertia (kgn^2)',i+1));
    % Crank-Con Fric. Fcn
    this.Data.F12{i} = str2func(FindInTable(this,'Crank-Con Fric. Fcn',i+1));
    fh = functions(this.Data.F12{i});
    isF12Valid = ~isempty(fh.file);
    % Con Length (m)
    d2 = str2double(FindInTable(this,'Con Length (m)',i+1));
    % Con. Mass (kg)
    m2 = str2double(FindInTable(this,'Con. Mass (kg)',i+1));
    % Con. C.O.M Radius from CR-CN pin (m)
    r2 = str2double(FindInTable(this,'Con. C.O.M Radius from CR-CN pin (m)',i+1));
    % Con. Rot. Inertia (kgn^2)
    Im2 = str2double(FindInTable(this,'Con. Rot. Inertia (kgn^2)',i+1));
    % Con-Piston Fric. Fcn
    this.Data.F23{i} = str2func(FindInTable(this,'Con-Piston Fric. Fcn',i+1));
    fh = functions(this.Data.F23{i});
    isF23Valid = ~isempty(fh.file);
    % Piston Mass (kg)
    m3 = str2double(FindInTable(this,'Piston Mass (kg)',i+1));
    % Piston Fric. Fcn
    this.Data.F3{i} = str2func(FindInTable(this,'Piston Fric. Fcn',i+1));
    fh = functions(this.Data.F3{i});
    isF3Valid = ~isempty(fh.file);
    % Tilt Angle
    Tilt = str2double(FindInTable(this,LinRotMechanism.TiltAngleText,i+1));
    % Slider-Offset (m)
    d3 = str2double(FindInTable(this,'Slider-Offset (m)',i+1));
    % Orientation: "u" aligned with positive y, "d" opposite
    orientation = str2double(FindInTable(this,'Orientation: "u" aligned with positive y,
"d" opposite',i+1));
    switch orientation
        case 'u'; orient = 1;
        case 'd'; orient = -1;
        otherwise; orient = nan();
    end

    if isnan(d1) || isnan(this.Phase(i)) || isnan(m1) || ...
        isnan(r1) || isnan(Ang_g1) || isnan(Im1) || ...
        ~isF12Valid || isnan(d2) || isnan(m2) || ...
        isnan(r2) || isnan(Im2) || ~isF23Valid || ...
        isnan(m3) || ~isF3Valid || isnan(Tilt) || ...
        isnan(d3) || isnan(orient)
        fprintf(...
            ['XXX ' Type ' is invalid, Frames not created. Trouble Components below.
XXX\n']);
    end

    if isnan(d1)
        fprintf(...
            ['Stroke = ' FindInTable(this,this.StrokeText,i+1) '\n']);
    end
    if isnan(this.Phase(i))
        fprintf(...
            ['Phase = ' FindInTable(this,this.PhaseText,i+1) '\n']);
    end
    if isnan(m1)
        fprintf(...
            ['Crank Mass (kg) = ' FindInTable(this,'Crank Mass (kg)',i+1) '\n']);
    end
end

```

```

        if isnan(r1)
            fprintf(...
                ['Crank C.O.M Radius (m) = ' FindInTable(this,'Crank C.O.M Radius (m)',i+1)
'\n']);
        end
        if isnan(Ang_g1)
            fprintf(...
                ['Crank C.O.M Angle (rad) = ' FindInTable(this,'Crank C.O.M Angle
(rad)',i+1)]);
        end
        if isnan(Im1)
            fprintf(...
                ['Crank Rot. Inertia (kgn^2) = ' FindInTable(this,'Crank Rot. Inertia
(kgn^2)',i+1) '\n']);
        end
        if isF12Valid
            fprintf(...
                ['Crank-Con Fric. Fcn = ' FindInTable(this,'Crank-Con Fric. Fcn',i+1) '\n']);
        end
        if isnan(d2)
            fprintf(...
                ['Con Length (m) = ' FindInTable(this,'Con Length (m)',i+1) '\n']);
        end
        if isnan(m2)
            fprintf(...
                ['Con. Mass (kg) = ' FindInTable(this,'Con. Mass (kg)',i+1) '\n']);
        end
        if isnan(r2)
            fprintf(...
                ['Con. C.O.M Radius from CR-CN pin (m) = ' FindInTable(this,'Con. C.O.M Radius
from CR-CN pin (m)',i+1) '\n']);
        end
        if isnan(Im2)
            fprintf(...
                ['Con. Rot. Inertia (kgn^2) = ' FindInTable(this,'Con. Rot. Inertia
(kgn^2)',i+1) '\n']);
        end
        if isF23Valid
            fprintf(...
                ['Con-Piston Fric. Fcn = ' FindInTable(this,'Con-Piston Fric. Fcn',i+1) '\n']);
        end
        if isnan(m3)
            fprintf(...
                ['Piston Mass (kg) = ' FindInTable(this,'Piston Mass (kg)',i+1) '\n']);
        end
        if isF3Valid
            fprintf(...
                ['Piston Fric. Fcn = ' FindInTable(this,'Piston Fric. Fcn',i+1) '\n']);
        end
        if isnan(Tilt)
            fprintf(...
                [LinRotMechanism.TiltAngleText ' = '
FindInTable(this,LinRotMechanism.TiltAngleText,i+1) '\n']);
        end
        if isnan(d3)
            fprintf(...
                ['Slider-Offset (m) = ' FindInTable(this,'Slider-Offset (m)',i+1) '\n']);
        end
        if isnan(orient)
            fprintf(...
                ['Orientation: "u" aligned with positive y, "d" opposite = ' orientation
'\n']);
        end
        this.isValid = false;
        return;
    end

    % Theta_SC is defined as Ang(i,:)
    Ang_sc = Ang(i,:);
    C1 = cos(Ang_sc);
    S1 = sin(Ang_sc);

```



```

C2 = cos(Beta_sc);
S2 = sin(Beta_sc);
Beta_sc = asin((d3 - d1*S1)/d2);
Beta_g = Beta_sc + Tilt;
Ang_g = Ang_sc + Tilt + Ang_g1;
this.Data.T2(:,i) = tan(Beta_sc);

% Coefficients on Alpha 2
C_Omega2 = (-d1/d2).*(C1./C2);
this.Data.Omega2(:,i) = C_Omega2;
B_Alpha2 = ((d1*S1+d2*S2.*C_Omega2.^2)./(d2*C2));

% Coefficients on Acceleration 1 x
B_alx = -r1*cos(Ang_sc + Ang_g1);
C_alx = -r1*sin(Ang_sc + Ang_g1);

% Coefficients on Acceleration 1 y
B_aly = -r1*sin(Ang_sc + Ang_g1);
C_aly = r1*cos(Ang_sc + Ang_g1);

% Coefficients on Acceleration 2 x
B_a2x = (-d1*C1 - r2*C2.*C_Omega2.^2 - r2*S2.*B_Alpha2);
C_a2x = (-d1*S1 - r2*S2.*C_Omega2);

% Coefficients on Acceleration 2 y
B_a2y = (-d1*S1 - r2*S2.*C_Omega2.^2 + r2*C2.*B_Alpha2);
C_a2y = (d1*C1 + r2*C2.*C_Omega2);

% Coefficients on Acceleration 3 x
B_a3x = (-d1*C1 - d2*C2.*C_Omega2.^2 - d2*S2.*B_Alpha2);
C_a3x = (-d1*S1 - d2*S2.*C_Omega2);

% Piston Position & Velocity Coefficient on Omega
this.Data.x_p(:,i) = d1*C1 + d2*C2 - sqrt((d2-d1)^2 - d3^2);
this.Data.v_p(:,i) = -d1*S1 - d2*S2*C_Omega2;

this.Data.A1(:,i) = m3*C_a3x;
this.Data.A2(:,i) = m2*C_a2x + this.Data.A1(:,i);
this.Data.A3(:,i) = (-Im2*C_Omega2./(d2*C2) + this.Data.T2(:,i).*this.Data.A1(:,i));
this.Data.A4(:,i) = m2*C_a2y + this.Data.A3(:,i);
this.Data.A5(:,i) = m1*C_alx + this.Data.A2(:,i);
this.Data.A6(:,i) = m1*C_aly + this.Data.A4(:,i);

this.Data.B1(:,i) = m3*B_a3x;
this.Data.B2(:,i) = m2*B_a2x + this.Data.B1(:,i);
this.Data.B3(:,i) = (-Im2*B_Alpha2./(d2*C2) + this.Data.T2(:,i).*this.Data.B1(:,i));
this.Data.B4(:,i) = m2*B_a2y + this.Data.B3(:,i);
this.Data.B5(:,i) = m1*B_alx + this.Data.B2(:,i);
this.Data.B6(:,i) = m1*B_aly + this.Data.B4(:,i);

this.Data.G1(:,i) = this.g*m3*this.STilt(i);
this.Data.G2(:,i) = this.g*m2*this.STilt(i) + this.Data.G1(:,i);
this.Data.G3(:,i) = (-this.g*r2*cos(Beta_g)./(d2*C2) +
this.Data.T2(:,i).*this.Data.G1(:,i));
this.Data.G4(:,i) = this.g*m2*cos(Beta_g) + this.Data.G3(:,i);
this.Data.G5(:,i) = this.g_m1*this.STilt(i) + this.Data.G2(:,i);
this.Data.G6(:,i) = this.g*m1*this.CTilt(i) + this.Data.G4(:,i);

temp_5 = -this.CTilt(i)*this.Data.A5(:,i) + this.STilt(i)*this.Data.A6(:,i);
this.Data.A6(:,i) = -this.STilt(i)*this.Data.A5(:,i) -
this.CTilt(i)*this.Data.A6(:,i);
this.Data.A5(:,i) = temp_5;

temp_5 = -this.CTilt(i)*this.Data.B5(:,i) + this.STilt(i)*this.Data.B6(:,i);
this.Data.B6(:,i) = -this.STilt(i)*this.Data.B5(:,i) -
this.CTilt(i)*this.Data.B6(:,i);
this.Data.B5(:,i) = temp_5;

temp_5 = -this.CTilt(i)*this.Data.G5(:,i) + this.STilt(i)*this.Data.G6(:,i);
this.Data.G6(:,i) = -this.STilt(i)*this.Data.G5(:,i) -
this.CTilt(i)*this.Data.G6(:,i);

```

```

this.Data.G5(:,i) = temp_5;

this.Data.E5(:,i) = orient*(-this.CTilt(i) + this.STilt(i)*this.Data.T2(Inc,i));
this.Data.E6(:,i) = orient*(-this.STilt(i) - this.CTilt(i)*this.Data.T2(Inc,i));

this.Data.AM(:,i) = -(Im1 - d1*S1.*this.Data.A2(:,i) + d1*C1.*this.Data.A4(:,i));
this.Data.BM(:,i) = -(-d1*S1.*this.Data.B2(:,i) + d1*C1.*this.Data.B4(:,i));
this.Data.GM(:,i) = -(this.g*m1*r1*cos(Ang_g) - ...
    d1*S1.*this.Data.A2(:,i) + d1*C1.*this.Data.A4(:,i));
this.Data.EM(:,i) = orient*-1*(-d1*S1 + d1*C1*this.Data.T2(:,i));
this.outputFcn = @this.SliderCrank;

if strcmp(Type,'Custom Profile Mechanism')
    CustomProfile = CustomFcn(Frame.NTheta,this.Phase(i));
else
    Ang = (0:Frame.NTheta-1)/(Frame.NTheta-1)*2*pi + this.Phase(i);
    CustomProfile = this.Stroke(i)/2 + this.Stroke(i)*cos(Ang)/2;
end
if orient == 1
    xmin = min(CustomProfile);
    xmax = max(CustomProfile);
else
    xmin = max(CustomProfile);
    xmax = min(CustomProfile);
end
this.Frames(i).Positions = (CustomProfile-xmin).*...
    (this.Stroke/(xmax-xmin));
this.Frames(i).MechanismIndex = i;
this.Frames(i).Mechanism = this;
end
%
%     case 'Scotch Yoke'
%         Ang = zeros(LEN,Frame.NTheta);
%         for i = 1:LEN
%             Ang(i,:) = (0:Frame.NTheta-1)/(Frame.NTheta-1)*2*pi + this.Phase(i);
%         end
%         for i = 1:LEN
%
%
%         end
%     case 'Rhombic Drive'
%
end
end
function Modify(this)
persistentData = Holder({this.Type,this.originalInput});
h = CreateMechanismInterface(persistentData);
uiwait(h);
if isempty(persistentData.vars)
    this.deReference();
else
    this.Populate(persistentData.vars{1},persistentData.vars{2});
end

% Find all Connections that have frames that reference this index
for iGroup = this.Model.Groups
    for iCon = iGroup.Connections
        if ~isempty(iCon.RefFrame)
            if iCon.RefFrame.Mechanism == this
                iCon.change();
            end
        end
    end
end
end
end
end

%% Get/Set Interface
function Item = get(this,PropertyName)
switch PropertyName
case 'Name'
    Item = this.name;
case 'Stroke'
    if size(this.originalInput,1)-1 == 1

```

```

        for c = 1:size(this.originalInput,2)
            if contains(this.originalInput{1,c},'Stroke')
                if isStrNumeric(this.originalInput{2,c})
                    Item = str2double(this.originalInput{2,c});
                    return;
                end
            end
        end
    else
        fprintf(['XXX Gradient Descent does ' ...
            'not support mechanisms with ' ...
            'multiple strokes XXX\n']);
        return;
    end
    otherwise
        fprintf(['XXX Lin Rot Mechanism GET Inteface for ' PropertyName ...
            ' is not found XXX\n']);
    end
end
function set(this,PropertyName,Item)
    switch PropertyName
        case 'Stroke'
            if size(this.originalInput,1)-1 == 1
                for c = 1:size(this.originalInput,2)
                    if contains(this.originalInput{1,c},'Stroke')
                        if isStrNumeric(this.originalInput{2,c})
                            this.originalInput{2,c} = num2str(Item);
                            return;
                        end
                    end
                end
            end
            this.Populate(this.Type,this.originalInput);
        else
            fprintf(['XXX Gradient Descent does ' ...
                'not support mechanisms with ' ...
                'multiple strokes XXX\n']);
            return;
        end
    otherwise
        fprintf(['XXX Lin Rot Mechanism SET Inteface for ' PropertyName ...
            ' is not found XXX\n']);
    end
end

function Uniform_Scale(this, Uniform_Scale)
    for c = 1:size(this.originalInput,2)
        if contains(this.originalInput{1,c},' (m) ')
            factor = Uniform_Scale;
        elseif contains(this.originalInput{1,c},' (kg) ')
            factor = Uniform_Scale ^ 3;
        elseif contains(this.originalInput{1,c},' (kg m^2) ')
            factor = Uniform_Scale ^ 5;
        else
            continue;
        end
        for r = 2:size(this.originalInput,1)
            if isStrNumeric(this.originalInput{r,c})
                this.originalInput{r,c} = ...
                    num2str(...
                        factor * str2double(this.originalInput{r,c}));
            end
        end
    end
    this.dont_propegate = true;
    this.Populate(this.Type,this.originalInput);
end

%% Dependent
function var = get.name(this)
    var = this.Type;
    for i = 1:length(this.Stroke)

```

```

        var = [var ' ' num2str(i) ':(L=' num2str(this.Stroke(i)) ', P=' num2str(this.Phase(i))
    ')'];
    end
end

%% Internal Helpers
function populateTilt(this)
    for i = 2:size(this.originalInput,1)
        Tilt = str2double(FindInTable(this,this.TiltAngleText,i));
        if isnan(Tilt)
            fprintf('XXX Invalid value for Tilt from Horizontal, perfectly horizontal assumed.
XXX\n');
            ReplaceInTable(this,'0',this.TiltAngleText,i)
            this.STilt(i-1) = 0;
            this.CTilt(i-1) = 1;
        else
            this.STilt(i-1) = sin(Tilt);
            this.CTilt(i-1) = cos(Tilt);
        end
    end
end

% Ideal Motions
function defineDataFromMotionProfile(Im1,mp,m1,eff,~,this,ind)
    %% Define Loads
    dx_dtheta = getFirstDer(this.Frames(ind).Positions);
    d2x_dtheta2 = getSecondDer(this.Frames(ind).Positions);

    Ax = -this.CTilt(ind)*mp*dx_dtheta;
    Bx = -this.CTilt(ind)*mp*d2x_dtheta2;
    Gx = -this.g*(mp + m1)*this.STilt(ind);
    Ex = this.CTilt(ind); % * Fp

    Ay = -this.STilt(ind)*mp*dx_dtheta;
    By = -this.STilt(ind)*mp*d2x_dtheta2;
    Gy = -this.g*(mp + m1)*this.CTilt(ind);
    Ey = this.STilt(ind); % * Fp

    Am = -Im1 - mp*(dx_dtheta).^2;
    Bm = -mp*dx_dtheta.*d2x_dtheta2;
    Gm = -this.g*this.STilt(ind)*mp*dx_dtheta;
    Em = dx_dtheta; % * Fp

    if isfield(this.Data,'Ax')
        appendData(this,'Ax',Ax,ind);
        appendData(this,'Bx',Bx,ind);
        appendData(this,'Gx',Gx,ind);
        appendData(this,'Ex',Ex,ind);

        appendData(this,'Ay',Ay,ind);
        appendData(this,'By',By,ind);
        appendData(this,'Gy',Gy,ind);
        appendData(this,'Ey',Ey,ind);

        appendData(this,'Am',Am,ind);
        appendData(this,'Bm',Bm,ind);
        appendData(this,'Gm',Gm,ind);
        appendData(this,'Em',Em,ind);
        this.Data.Eff(ind) = eff;
    else
        this.Data = struct(...
            'Ax',Ax,'Bx',Bx,'Gx',Gx,'Ex',Ex,...
            'Ay',Ay,'By',By,'Gy',Gy,'Ey',Ey,...
            'Am',Am,'Bm',Bm,'Gm',Gm,'Em',Em,...
            'Eff',eff);
    end

    this.outputFcn = @this.MotionWithEfficiency;
end

function output = MotionWithEfficiency(this,input)

```

```

% input = [dA, ddA, Fp, Inc, mechindex]
% output = [Fx, Fy, M]
output = zeros(3,1);
inc = input(4);
i = input(5);
if inc > 1
    incp = inc - 1;
else
    incp = size(this.Data.Ax,1) - 1;
end

% Horizontal Load, as felt by the drive shaft
if size(this.Data.Gx,1) == 1
    output(1) = (this.Data.Ax(incp,i) + this.Data.Ax(inc,i))*input(2) + ...
        (this.Data.Bx(incp,i) + this.Data.Bx(inc,i))*input(1)^2 + ...
        2*this.Data.Gx(1,i) + ...
        2*this.Data.Ex(i)*input(3);
else
    output(1) = (this.Data.Ax(incp,i) + this.Data.Ax(inc,i))*input(2) + ...
        (this.Data.Bx(incp,i) + this.Data.Bx(inc,i))*input(1)^2 + ...
        (this.Data.Gx(incp,i) + this.Data.Gx(inc,i)) + ...
        2*this.Data.Ex(i)*input(3);
end

% Vertical Load, as felt by the drive shaft\
if size(this.Data.Gy,1) == 1
    output(2) = (this.Data.Ay(incp,i) + this.Data.Ay(inc,i))*input(2) + ...
        (this.Data.By(incp,i) + this.Data.By(inc,i))*input(1)^2 + ...
        2*this.Data.Gy(i) + ...
        2*this.Data.Ey(i)*input(3);
else
    output(2) = (this.Data.Ay(incp,i) + this.Data.Ay(inc,i))*input(2) + ...
        (this.Data.By(incp,i) + this.Data.By(inc,i))*input(1)^2 + ...
        (this.Data.Gy(incp,i) + this.Data.Gy(inc,i)) + ...
        2*this.Data.Ey(i)*input(3);
end

% Moment as felt by shaft
if (this.Data.Em(incp,i) + this.Data.Em(inc,i))*input(3) < 0
    % Power is leaving the flywheel
    output(3) = (this.Data.Am(incp,i) + this.Data.Am(inc,i))*input(2) + ...
        (this.Data.Bm(incp,i) + this.Data.Bm(inc,i))*input(1)^2 + ...
        (this.Data.Gm(incp,i) + this.Data.Gm(inc,i)) + ...
        (this.Data.Em(incp,i) + this.Data.Em(inc,i))*input(3)/double(this.Data.Eff(i));
else
    % Power is entering the flywheel
    output(3) = (this.Data.Am(incp,i) + this.Data.Am(inc,i))*input(2) + ...
        (this.Data.Bm(incp,i) + this.Data.Bm(inc,i))*input(1)^2 + ...
        (this.Data.Gm(incp,i) + this.Data.Gm(inc,i)) + ...
        (this.Data.Em(incp,i) + this.Data.Em(inc,i))*input(3)*double(this.Data.Eff(i));
end
output = output / 2;
end

%{
% Simplified Arbitrary Motions
function defineDataForComplexCustomProfile(L,Mp,Ip,Mr,Iconst,Eff,this,ind)
dx_dtheta = getFirstDer(this.Frames(ind).Positions);
d2x_dtheta2 = getSecondDer(this.Frames(ind).Positions);
if Ip ~= 0
    % Gears
    A2 = asin_omni(this.Frames(ind).Positions/L - 1);
    dA2_dtheta = getFirstDer(A2);
    d2A2_dtheta2 = getSecondDer(A2);
    % Coefficient on omegal^2
    BddA2 = Ip.*d2A2_dtheta2;
    % Coefficient on alpha1
    AddA2 = -Ip.*dA2_dtheta;
else
    % Cam Drive
    BddA2 = 0; % no pulsing rotational elements
end
}

```

```

    AddA2 = 0; % " " "
end
% C1 - Coeff on ddA for Fx as felt by drive shaft
this.Data(1,:,ind) = -dx_dtheta;
% C2 - Coeff on dA^2 for Fx as felt by drive shaft
this.Data(2,:,ind) = -d2x_dtheta2;
% C3 - Gravity contribution for Fx as felt by drive shaft
this.Data(3,:,ind) = -this.STilt*this.g*(Mp + Mr);
% C4 - Coeff on ddA for Fy
this.Data(4,:,ind) = -this.CTilt*this.g*(Mr);
% Maybe consider a pressure angle, but center distance also
% required

% C - Coeff on Fp for M
this.Data(5,:,ind) = -dx_dtheta;
% C - Coeff on ddA for M
this.Data(6,:,ind) = (Mp.*(dx_dtheta.^2) - AddA2)*Eff;
% C - Coeff on dA^2 for M
this.Data(7,:,ind) = ...
    (-Mp.*(dx_dtheta).*(d2x_dtheta2) - BddA2)*Eff - Iconst;

this.outputFcn = @this.CustomProfileMechanism;
end

function output = CustomProfileMechanism(this,input)
% input = [dA, ddA, Fp, Inc, mechindex]
% output = [Fx, Fy, M]
output = zeros(3,1);
output(1) ...
    = input(3)... = Fp
    + this.Data(1,input(4))*input(2)... + C1*ddA
    + this.Data(2,input(4))*input(1)^2 ... + C2*dA^2
    + this.Data(3,input(4)); % + C3
output(2) ... Vertical Load, as felt by the drive shaft
    = this.STilt*output(1);
output(1) ... Horizontal Load, as felt by the drive shaft
    = this.CTilt*output(1);
% M =
output(3) ... Moment as felt by shaft
    = this.Data(4,input(4))*input(3)... % C4*Fp
    + this.Data(5,input(4))*input(2)... % C5*ddA
    + this.Data(6,input(4))*input(1)^2; % C6*dA^2
end
%}

% Slider Crank
function output = SliderCrank(this,input)
% input = [dA, ddA, Fp, Inc, mechindex]
% output = [Fx, Fy, M]
dA = input(1);
ddA = input(2);
Fp = input(3);
Inc = input(4);
i = input(5);
F12 = this.Data.F12{i}(sqrt(...
    this.Data.A2(Inc,i)*ddA + ...
    this.Data.B2(Inc,i)*dA^2 + ...
    this.Data.G2(Inc,i) + Fp)^2 + ...
    (this.Data.A4(Inc,i)*ddA + ...
    this.Data.B4(Inc,i)*dA^2 + ...
    this.Data.G4(Inc,i) - ...
    this.Data.T2(Inc,i)*Fp)^2));
F3y = abs(this.Data.A3(Inc,i)*ddA + ...
    this.Data.B3(Inc,i)*dA^2 + ...
    this.Data.G3(Inc,i) - ...
    this.Data.T2(Inc,i)*Fp);
F23 = this.Data.F23{i}(...
    sqrt((this.Data.A1(Inc,i)*ddA + ...
    this.Data.B1(Inc,i)*dA^2 + ...
    this.Data.G1(Inc,i) + Fp)^2 + (F3y)^2));
F3y = this.Data.F3{i}(F3y);

```

```

LostTorque = -dA*(...
    abs(F12*(this.Data.Omega2(Inc,i)-1)) + ...
    abs(F23*this.Data.Omega2(Inc,i)) + ...
    abs(F3y*this.Data.v_p(Inc,i)));

output = zeros(3,1);
output(1) = ... % Horizontal Load
    this.Data.A5(Inc,i)*ddA + this.Data.B5(Inc,i)*dA^2 + ...
    this.Data.G5(Inc,i) + ...
    this.Data.E5(Inc,i)*Fp;
output(2) = ... % Vertical Load
    this.Data.A6(Inc,i)*ddA + this.Data.B6(Inc,i)*dA^2 + ...
    this.Data.G6(Inc,i) + ...
    this.Data.E6(Inc,i)*Fp;
output(3) = ... % Moment as felt by shaft
    this.Data.AM(Inc,i)*ddA + this.Data.BM(Inc,i)*dA^2 + ...
    this.Data.GM(Inc,i) + this.Data.EM(Inc,i)*Fp + ...
    LostTorque;
end

function output = ScotchYoke(this,input)
    % input = [dA, ddA, Fp, Inc, mechindex]
    % output = [Fx, Fy, M]

end

function output = RhombicDrive(this,input)
    % input = [dA, ddA, Fp, Inc, mechindex]
    % output = [Fx, Fy, M]
    dA = input(1);
    ddA = input(2);
    Fp = input(3);
    Inc = input(4);
    i = input(5);
    F12 = this.Data.F12{i}(sqrt((...
        this.Data.A2(Inc,i)*ddA + ...
        this.Data.B2(Inc,i)*dA^2 + ...
        this.Data.G2(Inc,i) + Fp)^2 + ...
        (this.Data.A4(Inc,i)*ddA + ...
        this.Data.B4(Inc,i)*dA^2 + ...
        this.Data.G4(Inc,i) - ...
        this.Data.T2(Inc,i)*Fp)^2));
    F3y = abs(this.Data.A3(Inc,i)*ddA + ...
        this.Data.B3(Inc,i)*dA^2 + ...
        this.Data.G3(Inc,i) - ...
        this.Data.T2(Inc,i)*Fp);
    F23 = this.Data.F23{i}(...
        sqrt((this.Data.A1(Inc,i)*ddA + ...
        this.Data.B1(Inc,i)*dA^2 + ...
        this.Data.G1(Inc,i) + Fp)^2 + (F3y)^2));
    F3y = this.Data.F3{i}(F3y);

    % TANalpha_2L4 = tan(pressure angle)/(2*14)

    output = zeros(3,1);
    output(3) = ... % Moment as felt by shaft
        this.Data.AM(Inc,i)*ddA + this.Data.BM(Inc,i)*dA^2 + ...
        this.Data.GM(Inc,i) + this.Data.EM(Inc,i)*Fp;
    output(1) = 0.5*(... % Horizontal Load
        this.Data.A5(Inc,i)*ddA + this.Data.B5(Inc,i)*dA^2 + ...
        this.Data.G5(Inc,i) + ...
        this.Data.E5(Inc,i)*Fp + ...
        output(3)*this.Data.TANalpha_2L4(i)*this.STilt(i));
    output(2) = 0.5*(... % Vertical Load
        this.Data.A6(Inc,i)*ddA + this.Data.B6(Inc,i)*dA^2 + ...
        this.Data.G6(Inc,i) + ...
        this.Data.E6(Inc,i)*Fp - ...
        output(3)*this.Data.TANalpha_2L4(i)*this.CTilt(i));

    LostTorque = -dA*(...
        abs(F12*(this.Data.Omega2(Inc,i)-1)) + ...

```

```

        abs(F23*this.Data.Omega2(Inc,i)) + ...
        abs(F3y*this.Data.v_p(Inc,i)) + ...
        abs(this.Data.Faux{i} (sqrt(output(1)^2+output(2)^2)));

        output(3) = output(3) + LostTorque;
    end
end

end

function PropertyValue = FindInTable(LinRotMech,Item,row)
for col = 1:size(LinRotMech.originalInput,2)
    if strcmp(LinRotMech.originalInput{1,col},Item)
        PropertyValue = LinRotMech.originalInput{row,col};
        return;
    end
end
fprintf(['XXX no property called "' Item '" for LinRotMechanism/Type = "' ...
        LinRotMech.Type '". Value of "' applied. XXX\n']);
PropertyValue = '';
end

function ReplaceInTable(LinRotMech,PropertyValue,Item,row)
for col = 1:size(LinRotMech.originalInput,2)
    if strcmp(LinRotMech.originalInput{1,col},Item)
        LinRotMech.originalInput{row,col} = PropertyValue;
        return;
    end
end
fprintf(['XXX no property called "' Item '" for LinRotMechanism/Type = "' ...
        LinRotMech.Type '". Value of "' applied. XXX\n']);
end

function Template = MergeTables(Template,Data)
for Dcol = 1:size(Data,2)
    % For each column of Data
    % Find the representative column in Template
    for Tcol = 1:size(Template,2)
        if strcmp(Data{1,Dcol},Template{1,Tcol})
            for row = 2:size(Data,1)
                Template{row,Tcol} = Data{row,Dcol};
            end
            break;
        end
    end
end
end

function appendData(ME,field,Data,ind)
if isfield(ME.Data,field)
    if size(ME.Data.(field),1) == size(Data,1)
        ME.Data.(field) (:,ind) = Data;
    elseif size(ME.Data.(field),1) == 1
        temp = repmat(ME.Data.(field),size(Data,1),1);
        ME.Data.(field) = temp;
        ME.Data.(field) (:,ind) = Data;
    elseif size(Data,1) == 1
        ME.Data.(field) (:,ind) = Data;
    else
        fprintf('XXX Frame divisions have changed, please implement a fix XXX\n');
    end
else
    ME.Data.(field) (:,ind) = Data;
end
end

```



## Mechanical System

The mechanical system is a class that includes the following functionality:

A constructor.

A get / set interface.

**Solve:** Takes the motion state and piston force and translates it through the linear to rotational mechanisms to produce an acceleration.

A set of unused functions that were investigating approximating arbitrary mechanisms with N-dimensional interpolation.

```
classdef MechanicalSystem < handle
    %UNTITLED2 Summary of this class goes here
    % Detailed explanation goes here

    properties (Constant)
        SteadyStateRMS = 0.1;
        NDimModelCalcRadius = [10 10 3];
    end

    properties (Dependent)
        isConverged;
    end

    properties
        Model Model; %

        Converters LinRotMechanism; % Array Containing Linear-Rotational Converters
        Inertia double = 1; % Real Flywheel Inertia
        DriveTrainWeight double = 1;
        DriveTrainFricCoef double = 0;
        LoadFunction function_handle; % Function that takes current motion and provides a counter
load

        InertiaMod double = 1; % Modifier Used to allow the engine to get up to speed faster, or
stabilize it during slow times.
        KE double = 0; % Kinetic Energy
        Alpha double = 0; % The rotational acceleration
        Omega double = 0; % The rotational speed
        Theta double = 0; % The physical Angle

        LastCycle double; % (Theta, Omega)
        ThisCycle double; % Continously recorded and checked for convergence

        % Set in SetInitialConditions
        Points double; % Array indicating the points at which the model is calculated for each
variable
        Inc double; % Vector indicating the increment that is used to discretize each variable
    end

    properties (Dependent)
        name;
    end

    methods
        %% Constructor
```

```

function this = MechanicalSystem(Model,Converters,~,Inertia,LoadFunction)
this.Model = Model;
% Define other parameters
this.Converters = Converters;
this.Inertia = Inertia;
this.LoadFunction = LoadFunction;
end
function iname = get.name(~)
iname = 'Mechanical System';
end
function Item = get(this,PropertyName)
switch PropertyName
case 'name'
Item = this.name();
case 'Flywheel Inertia'
Item = this.Inertia;
case 'Drive Train Weight'
Item = this.DriveTrainWeight;
case 'Drive Train Normal Friction Coefficient'
Item = this.DriveTrainFricCoef;
case 'Load Function'
Item = this.LoadFunction;
otherwise
fprintf(['XXX MechanicalSystem GET Inteface for ' PropertyName ' is not found
XXX\n']);
return;
end
end
function set(this,PropertyName,Item)
switch PropertyName
case 'name'
% Do nothing
case 'Flywheel Inertia'
this.Inertia = Item;
case 'Drive Train Weight'
this.DriveTrainWeight = Item;
case 'Drive Train Normal Friction Coefficient'
this.DriveTrainFricCoef = Item;
case 'Load Function'
this.LoadFunction = Item;
otherwise
fprintf(['XXX MechanicalSystem SET Inteface for ' PropertyName ' is not found
XXX\n']);
return;
end
end

function Power = Solve(this,Inc,dA,ddA,Forces)
M = 0;
fric = abs(this.DriveTrainWeight*this.DriveTrainFricCoef);
for i = 1:length(this.Converters)
for j = 1:length(Forces{i})
F = this.Converters(i).outputFcn([dA ddA Forces{i}(j) Inc j]);
M = M + F(3);
fric = fric + sqrt(F(1)^2 + F(2)^2)*this.DriveTrainFricCoef;
end
%
% if i == length(this.Converters)
%     fprintf([' ' num2str(M*dA) '\n']);
%
% elseif i == 1
%     fprintf(num2str(M*dA));
%
% else
%     fprintf([' ' num2str(M*dA)]);
%
% end
end
Power = (M - fric)*dA;
end

function SetInitialConditions(this,iTheta,iOmega)
this.Theta = iTheta;
this.Omega = iOmega;
this.Alpha = 0;

```

```

end

%% Analysis Function
function RMS = compareCycles(this)
    % Calculate the RMS error between cycles
    Match = interp1(this.LastCycle(:).Theta,...
        this.LastCycle(:).Omega,...
        this.ThisCycle(:).Theta);
    RMS = rssq(Match - this.ThisCycle.Omega)/sqrt(length(Match));
end
function Converged = get.isConverged(this)
    Converged = this.compareCycles() < this.SteadyStateRMS;
end

%% Simulation Function
function move(this,InputForces)
    %% Do something with the Input Forces
    % http://matlab.izmiran.ru/help/techdoc/matlab\_prog/ch\_dat41.html
    % If you use the colon to index multiple cells in conjunction
    % with the curly brace notation, MATLAB treats the contents of
    % each cell as a separate variable. For example, assume you have
    % a cell array T where each cell contains a separate vector. The
    % expression T{1:5} is equivalent to a comma-separated list of
    % the vectors in the first five cells of T.
    if this.isKinematic
        % Collect the input parameters
        Parameters = cell(length(InputForces)+2,1);
        Parameters{1} = mod(this.Theta,2*pi);
        Parameters{2} = this.Omega*this.Omega;
        for r = 3:length(Parameters)
            Parameters{r} = InputForces(r-2);
        end
        % Interpolate the Mechanical System Results
        this.Alpha = interpn(Parameters{:},this.NDimModel,...
            this.Points{:},'linear',NaN);
        % ThetaInc, this.sqOmegaInc, this.ForceInc{:},...
        if isnan(this.Alpha)
            % Assess if the extrapolation array needs to be expanded to
            % the next parameter

            % Find the parameter that goes out
            % Points
            C = zeros(length(this.Points),1);
            for i = 1:length(this.Points)
                if Parameters(i) > this.Points{i}(end)
                    C(i) = ceil((Parameters(i)-this.Points{i}(end))/this.Inc(i));
                elseif Parameters(i) < this.Points{i}(1)
                    C(i) = -ceil(abs(Parameters(i)-this.Points{i}(end))/this.Inc(i));
                end
            end
            end

            this.UpdateMechanicalModel(C);

            this.Alpha = interpn(Parameters{:},this.NDimModel,...
                this.Points{:},'linear',NaN);
        end

        % Move Position ahead
        ChThetaeInVelocity = this.Model.dt*this.Alpha;
        this.Theta = this.Theta + this.Model.dt*(this.Omega+ChThetaeInVelocity/2);
        this.Omega = this.Omega + ChThetaeInVelocity;

        for iFrame = this.Model.RefFrames
            if iFrame.Mechanism == this
                iFrame.Position = getPosition(this.Theta);
            end
        end
    end
end
end
end
end
end
end
end
end

```

end

## **G.5. Motion**

# Motions

## Idealized Square Wave Motion

```
function [ pos ] = box_wave( Nang, Phase )
    if nargin == 0
        Nang = 200;
        Phase = 0;
    end
    Phase = Phase + pi/2;
    e = 0.8;
    n = 2;
    con_cr_ratio = 2;
    TransferPhase = pi/2;
    pos = Ellipsoidal(Nang,Phase,e,n,con_cr_ratio,TransferPhase);
end
```

## Idealized Saw Wave Motion

```
function [ pos ] = v_wave( Nang, Phase )
    Phase = Phase + pi;
    %r = 5;
    %L = sqrt((1-2*r)^2+(Nang-1)^2);
    %theta = r*sin(asin((Nang-1)/L)-asin(2*r/L));
    %d = r*sin(theta);
    %offset = r*cos(theta);
    pos = zeros(1,Nang);
    for i = 1:Nang
        x = mod(i-1 + (-Phase)/(2*pi)*(Nang-1),Nang-1);
        if x < 0.5*(Nang-1)
            % Within First Top Circle
            pos(i) = 1 - 2*x/(Nang-1);
        else
            pos(i) = 2*(x-(Nang-1))/(Nang-1) + 1;
        end
    end
    ends = zeros(1,4);
    for i = 1:100
        ends(1:2) = pos(1:2);
        ends(3:4) = pos(end-1:end);
        pos(2:end-1) = (pos(1:end-2) + pos(3:end))/2;
        pos(1) = (ends(2) + ends(4))/2;
        pos(end) = (ends(1) + ends(3))/2;
    end
    pos = pos - min(pos);
    pos = pos / max(pos);
end
```

## Slider Crank Motion with connecting rod 2 times larger than crank rod

```
function [ pos ] = e0_n2_r2( Nang, Phase )
    Phase = Phase + pi;
    e = 0;
    n = 2;
    con_cr_ratio = 2;
    TransferPhase = 0;
    pos = Ellipsoidal(Nang,Phase,e,n,con_cr_ratio,TransferPhase);
end
```

### Slider Crank Motion with connecting rod 6 times larger than crank rod

```
function [ pos ] = e0_n2_r6( Nang, Phase )
    Phase = Phase + pi;
    e = 0;
    n = 2;
    con_cr_ratio = 6;
    TransferPhase = 0;
    pos = Ellipsoidal(Nang,Phase,e,n,con_cr_ratio,TransferPhase);
end
```

Slider Crank Motion driven by 1/5 elliptical gears with connecting rod 2 times larger than crank rod, arranged to produce dwelling motion.

```
function [ pos ] = e15_n2_r2_Box( Nang, Phase )
    if nargin == 0
        Nang = 200;
        Phase = 0;
    end
    Phase = Phase + pi/2;
    e = 1/4;
    n = 2;
    con_cr_ratio = 2;
    TransferPhase = pi/2;
    pos = Ellipsoidal(Nang,Phase,e,n,con_cr_ratio,TransferPhase);
end
```

Slider Crank Motion driven by 1/5 elliptical gears with connecting rod 2 times larger than crank rod, arranged to produce saw wave motion.

```
function [ pos ] = e15_n2_r2_Sharp( Nang, Phase )
    Phase = Phase + pi;
    e = 1/5;
    n = 2;
    con_cr_ratio = 2;
    TransferPhase = 0;
    pos = Ellipsoidal(Nang,Phase,e,n,con_cr_ratio,TransferPhase);
end
```

Slider Crank Motion driven by 1/5 elliptical gears with connecting rod 6 times larger than crank rod, arranged to produce dwelling motion.

```
function [ pos ] = e15_n2_r6_Box( Nang, Phase )
    if nargin == 0
        Nang = 200;
        Phase = 0;
    end
    Phase = Phase + pi/2;
    e = 1/5;
    n = 2;
    con_cr_ratio = 6;
    TransferPhase = pi/2;
    pos = Ellipsoidal(Nang,Phase,e,n,con_cr_ratio,TransferPhase);
```

end

Slider Crank Motion driven by 1/5 elliptical gears with connecting rod 6 tiems larger than crank rod, arranged to produce saw wave motion.

```
function [ pos ] = e15_n2_r6_Sharp( Nang, Phase )
    Phase = Phase + pi;
    e = 1/5;
    n = 2;
    con_cr_ratio = 6;
    TransferPhase = 0;
    pos = Ellipsoidal(Nang,Phase,e,n,con_cr_ratio,TransferPhase);
end
```

### Basic Elliptical motion

```
function [pos] = Ellipsoidal(Nang,Phase,e,n,con_cr_ratio,TransferPhase)
    ang = mod(linspace(0,2*pi,Nang),2*pi);
    C = (sqrt(1+(n^2-1)*(1-e^2)) + e)/(n*(1-e));
    % Pass ang through the transfer function
    ang = atanSmooth(C*tan(ang)) + TransferPhase;
    l_cr = 0.5;
    l_con = l_cr*con_cr_ratio;
    ang2 = asin((-l_cr*sin(ang))/l_con);
    pos = l_con*(cos(ang2) - 1) + l_cr*(cos(ang) + 1);
    pos(1:end-1) = shiftVector(pos(1:end-1),Phase);
    pos(end) = pos(1);
end
```



## **G.6. Simulation**

## Simulation

Simulation is a class that performs all conduction and gas relations operations. It includes the following functionality:

A run function that sets up the simulation, performs warmup and during simulation it calls the time integration function (running over increments) and calls the mechanism solve function (between increments). This function also tracks steady-state, adjusts pressure, solves for steady-state temperatures and performs recording operations.

The time integration function.

The loop solving sub-function.

A function that calculates the cost, in relation to the Newton-Raphson algorithm.

A function that calculates the average of the dynamic properties.

A function that interpolates properties given an increment value.

A function that updates the engine pressure.

A set of functions for calculating turbulence weighting, flow loss coefficients and temperature related properties.

```
classdef Simulation < handle
    % Simulation objects contain all the arrays of each particular node, and
    % each particular interface type along with all supporting information
    % tables

    properties (Constant)
        U_Tolerance = 1e-6;
        turb_Tolerance = 0.05;
        RE_Tolerance = 1;
        InertiaMod = 0.1;
    end

    properties

        MaxCourant = 0.25;
        MaxFourier = 0.25;

        Model Model;
        %% Face Properties
        % Face Types
        Mix_Fc;

        % Dynamic Data
        Dynamic;
        Dyn = 1;

        % Node Property Pointers to Dynamic Data
```

```

isDynVol double;%
DynVol int16;
DynDh int16;
Fc_DynArea int16;
Fc_DynDh int16;
Fc_DynK12 int16;
Fc_DynK21 int16;
Fc_DynVel_Factor int16;
Fc_DynShear_Factor int16;
Fc_DynDist int16;
Fc_DynCond_Dist int16;
Fc_DynCond int16;
Fc_Dyndx int16;
Fc_DynWCond int16;
SC_Active int16;
%dL_dt double;
%dD_dt double;

%% Solid Conduction
Cond_Nds1;
Cond_Nds2;
Cond_Nds;
Cond_Fcs;
CondNet;
% Energy Transport
Trans_Fcs;
TransNet;
% Faces to Nodes

%% MEchanical System
Press_Contact;
Shear_Contact;
MechanicalSystem MechanicalSystem;
dA = 0;
dA_old = 0;

%% Max Courant
dt_max double;

%% General Gas
dT_duFunc cell;
dh_dTFunc cell; % Additions
kFunc cell;
muFunc cell;
R double; % Defined by Region
Rs; % All Nodes

NuFunc_l_el cell;
NuFunc_t_el cell;
Fc_NkFunc_l_el cell;
Fc_NkFunc_t_el cell;

%% Node Properties
CondFlux double;%
NuFunc_l cell;%
NuFunc_t cell;%
dT_dU double;%
dh_dT double;% Gas nodes only, Additions
dV_dt double;%
u double;%
T double;%
P double;%
dP double;%
rho double;%
m double;%
vol double;%
old_vol double;%
k double;%
mu double;%
Dh double;%
Nu double;%

```

```

RE double;%
f double;%
P_backup double;%
U double;%
dU double;

% Turbulence
turb double;
Area double;
Va double;
to double;

%% Face Properties
Fc_Nd int16;%
Fc_Dist double;%
Fc_Cond_Dist double;%
Fc_dx double;%
Fc_K12 double;%
Fc_K21 double;%
Fc_U double;%
Fc_RE double;%
Fc_f double;%
Fc_R double;%
Fc_fFunc_l cell;%
Fc_fFunc_t cell;%
Fc_NkFunc_l cell;%
Fc_NkFunc_t cell;%
Fc_Area double;%
Fc_Dh double;%
Fc_Cond double;%
Fc_T double;
Fc_u double;
Fc_k double;
Fc_mu double;
Fc_rho double;
Fc_Vel_Factor double;
Fc_Shear_Factor double;
Fc_W double;

% Prandtl Number
PR double;%
Fc_PR;

%% Parameter inspection made it here:

% Turbulence
Fc_turb double;
Fc_to double;
useTurbulenceFc logical;
useTurbulenceNd logical;
dturb double;
REcrit double;

stop logical;
curTime double;
Solid_t double;
MAXdt double;
A_Inc;
Inc;
MoveCondition int16;

% Solver
Regions cell;
isEnvironmentRegion logical;
RegionFcs cell;
ActiveRegionFcs cell;
RegionFcCount int16;
RegionLoops cell;
RegionLoops_Ind cell;
Faces cell;
Fc2Col double;

```

```

ExplicitLeak double;
ExplicitNorm double;
KpU_2A double;
Fc_V double;
Fc_V_averager double;
Fc_turb_averager double;
Fc_dP double;
Fc_V_backup double;
A_Press cell;
u2T cell;
%
    extfc cell;
Solid_dt_max;
isLoopRegionFcs;
Fc2Col_loop;

Nd_Solid_dt;
doScale;
Acceleration_Coef = 1;
SteadyState_Factor = 1;
Acceleration_Factor = 1;

% Statistics Collection
ToEnvironmentSolid;
ToEnvironmentGas;
ToSource;
ToSink;
E_ToEnvironment;
E_ToSource;
E_ToSink;
E_Flow_Loss;
Sources;
Sinks;
VolMin;
VolMax;
ShuttleFaces;
StaticFaces;
ExergyLossShuttle;
ExergyLossStatic;

countFailed = 0.0;
countSuccess = 0.0;

ACond;
bCond;
BoundaryNodes;
MixFcs;
CondEff;
CondTempEff;
CycleTime;
ss_condition;
continuettoSS;

%
    Fric_l_current;
%
    Fric_t_current;
%
    Fric_l_index;
%
    Fric_t_index;
%
    Fric_tbl;
%
    Fric_tbt;

% Engine Pressure Assignment
PRegion = [];
PRegionTime = [];
CycledE = 0;

Fc_Nd03 int16;
Fc_A double;
Fc_DynA;
Fc_B double;
Fc_DynB;
Fc_C double;

```

```

Fc_DynC;
Fc_D double;
Fc_DynD;

EffRE double;

FcApprox;
end

methods

function [Results, success] = Run(ME, islast, do_warmup, ss_tolerance, options)
    success = true;
    ME.stop = false;

    if nargin > 4 && options.isManual == false
        simTime = options.simTime; % Maximum Simulation Time
        ME.ss_condition = options.SS; % True or False for steady state detection
        ME.continuetoSS = false;
        switch options.movement_option % C - Continuous, V - Variable
            case 'C'; ME.MoveCondition = 1;
            case 'V'; ME.MoveCondition = 2;
        end
        ME.dA = options.rpm*2*pi/60;
        ME.dA_old = ME.dA;
        ME.MAXdt = options.max_dt;
        ME.T(ME.Sources) = options.SourceTemp;
        ME.T(ME.Sinks) = options.SinkTemp;
        engine_Pressure = options.EnginePressure;
    else
        %% Get user input
        while isempty(ME.Model.name) && ~isempty(ME.Model.Sensors)
            answer = inputdlg(...
                ['Please name the model so that ' ...
                'the data can be saved properly'],'Name Model',[1 50],{''});
            if ~isempty(answer{1})
                ME.Model.name = answer{1};
            end
        end
        % inputdlg( prompt , dlg_title , num_lines , defAns );
        invalid = true;
        output = {'10','SS','C','60','0.1'};
        while invalid
            output = inputdlg({...
                'Maximum Simulation Time (seconds)',...
                'End Condition (SS - Steady State)',...
                'Motion Condition (C - Constant Velocity, V - Variable Velocity)',...
                'Initial Velocity (rpm)',...
                'maximum time step (s)'},...
                'Enter Simulation Parameters',...
                [1 20],...
                output);
            invalid = false;
            if isempty(output)
                Results = [];
                return;
            end
            if ~all(ismember(output{1}, '0123456789+-.eE')); invalid = true;
            elseif ~strcmp(output{2},'SS') && ~isempty(output{2}); invalid = true;
            elseif ~strcmp(output{3},'C') && ~strcmp(output{3},'V'); invalid = true;
            elseif ~all(ismember(output{4}, '0123456789+-.eE')); invalid = true;
            elseif ~all(ismember(output{5}, '0123456789+-.eE')); invalid = true;
        end
    end

    simTime = str2double(output{1});
    switch output{2}
        case 'SS'; ME.ss_condition = true;
        otherwise; ME.ss_condition = false;
    end
end

```

```

ME.continuetoSS = false;
switch output{3}
    case 'C'; ME.MoveCondition = 1;
    case 'V'; ME.MoveCondition = 2;
end
ME.dA = str2double(output{4})*2*pi/60;
ME.dA_old = ME.dA;
ME.MAXdt = str2double(output{5});
engine_Pressure = ME.Model.enginePressure;
end

Load_Function_is_Not_Given = false;

if ME.Model.recordOnlyLastCycle
    ME.MaxCourant = ME.Model.MaxCourantConverging;
    ME.MaxFourier = ME.Model.MaxFourierConverging;
else
    ME.MaxCourant = ME.Model.MaxCourantFinal;
    ME.MaxFourier = ME.Model.MaxFourierFinal;
end

%% Early Initialize
ME.A_Inc = 2*pi/(Frame.NTheta-1);
ME.Inc = 1;
ME.curTime = 0;
Results = Result();
Results.Model = ME.Model;
Results.Data = struct();
Results.Data.QEnv = 0;
Results.Data.QSource = 0;
Results.Data.QSink = 0;
Results.Data.Flow_Loss = 0;
Results.Data.Power = 0;
Results.Data.CR = 0;

indf = 1:length(ME.Fc_V);
record_data = islast || ME.ss_condition == false;

%% Set up data acquisition
grab_Pressure = ME.Model.recordPressure;
grab_Temperature = ME.Model.recordTemperature;
grab_Velocity = ME.Model.recordVelocity;
grab_PressureDrop = ME.Model.recordPressureDrop;
grab_Turbulence = ME.Model.recordTurbulence;
grab_ConductionFlux = ME.Model.recordConductionFlux;
if ME.MoveCondition == 1 || ME.Model.recordOnlyLastCycle
    dt = ME.A_Inc/ME.dA;
    if ME.Model.recordOnlyLastCycle; LEN = Frame.NTheta-1;
    else; LEN = ceil((simTime*ME.dA/ME.A_Inc));
    end
    Results.Data.A = linspace(0,ME.A_Inc*LEN,LEN);
    Results.Data.dA = ME.dA(ones(1, LEN+2));
    Results.Data.t = linspace(0,LEN*dt,LEN+2);
    if grab_Pressure
        Results.Data.P = zeros(length(ME.P),LEN);
        Results.Data.P(:,1) = ME.P;
    end
    if grab_Temperature
        Results.Data.T = zeros(length(ME.T),LEN);
        Results.Data.T(:,1) = ME.T;
    end
    if grab_Velocity
        Results.Data.U = zeros(length(ME.Fc_dx),LEN);
        Results.Data.U(:,1) = ME.Fc_V./ME.Fc_Area(indf);
    end
    if grab_PressureDrop
        Results.Data.dP = zeros(length(ME.P),LEN);
        Results.Data.dP(:,1) = ME.dP;
    end
    if grab_Turbulence
        Results.Data.turb = zeros(length(ME.P),LEN);

```

```

    Results.Data.turb(:,1) = ME.turb;
end
if grab_ConductionFlux
    Results.Data.cond = zeros(length(ME.T),LEN);
    Results.Data.cond(:,1) = ME.CondFlux;
end
else
    % Grab Pressure
    if grab_Pressure; Results.Data.P = ME.P; end
    % Grab Temperature
    if grab_Temperature; Results.Data.T = ME.T; end
    % Grab Velocity
    if grab_Velocity
        Results.Data.U = ME.Fc_V./ME.Fc_Area(indf);
    end
    if grab_PressureDrop
        Results.Data.dP = ME.dP;
    end
    % Grab Turbulence
    if grab_Turbulence; Results.Data.turb = ME.turb; end
    if grab_ConductionFlux; Results.Data.cond = ME.CondFlux; end
    Results.Data.A = 0;
    Results.Data.dA = ME.dA;
    Results.Data.t = 0;
end
end
if ~isempty(ME.Model.Sensors)
    for iSensor = ME.Model.Sensors; iSensor.reset(); end
end
if ~isempty(ME.Model.PVoutputs)
    for iPVoutput = ME.Model.PVoutputs; iPVoutput.reset(); end
end
n = 2;

AdjustTime = 0;
previousTime = 0;
ME.dt_max = 2*ME.A_Inc/(ME.dA_old + ME.dA);

%% Initialize All the sub functions
clear assignDynamic;
clear implicitSolve;
% clear dUFunc;
clear KValue;
clear solve_loops;
% dUFunc(ME);
% KValue(ME);
assignDynamic(ME, ME.Inc, false);
implicitSolve(ME, true);
sindn = length(ME.P)+1:length(ME.T);
ME.old_vol = ME.vol;

%% Warm Up Phase
if do_warmup
    progressbar('Warmup Phase');
    hmax = mean(ME.Solid_dt_max);
    t = 0;
    progressbar_update = 1;

    if ME.Model.warmUpPhaseLength > 0; assignAvgDynamic(ME); end

    % Record a backup of the gas properties, just so that we don't have
    % ... to bother.
    indn = 1:length(ME.P); indf = 1:length(ME.Fc_U); indmf = ME.Mix_Fc;
    Cfc = ME.Cond_Fcs; Cnd1 = ME.Cond_Nds1; Cnd2 = ME.Cond_Nds2;
    Gm = ME.m(indn); Gu = ME.u(indn); GT = ME.T(indn);
    Sm = ME.m(sindn); ST = ME.T(sindn);
    Q = zeros(length(ME.T),1);
    ME.Fc_Cond(indmf) = ME.Fc_Area(indmf)./(ME.Fc_R(indmf)' + 15);
    ME.Fc_Cond(indf) = 0.5*ME.Fc_Area(indf)./ME.Fc_Cond_Dist(indf);
    while (t < ME.Model.warmUpPhaseLength)
        h = min(hmax, ME.Model.warmUpPhaseLength - t);
    end
end

```



```

Ti = [GT; ME.T(sindn)];

Q(:) = 0;
Qfc = ME.Fc_Cond(Cfcs).*(Ti(Cnd1) - Ti(Cnd2));
for i = 1:3:length(ME.CondNet)-2
    Q(ME.CondNet{i+1}) = Q(ME.CondNet{i+1}) + ...
        ME.CondNet{i}.*Qfc(ME.CondNet{i+2});
end

% Internal Energy Change - Gas
Gu = Gu + h*Q(indn)./Gm;

% Temperature - Gas
GT = ME.u2T(Gu);

% Temperature - Solid
ST = ME.dT_dU(sindn).*Q(sindn)./Sm;

t = t + h;
if progressbar_update > 1/h
    progressbar(t/ME.Model.warmUpPhaseLength);
    progressbar_update = 1;
else
    progressbar_update = progressbar_update + 1;
end
end

ME.T(sindn) = ST;
end

sindn = length(ME.P):length(ME.T);

progressbar('Main Dynamic Loop');
%% Set up Steady State Detection Code
if ME.ss_condition
    ss_cycles = 5;
    Precord = zeros(1,ss_cycles);
    Load_Function_is_Not_Given = isempty(ME.MechanicalSystem.LoadFunction);
    if Load_Function_is_Not_Given && ME.MoveCondition == 2
        ME.MaxCourant = ME.Model.MaxCourantFinal;
        ME.MaxFourier = ME.Model.MaxFourierFinal;
        % Turn off pressure, temperature adjustments.
        % Set timestep to final value.
        ME.continuetoSS = true;
        if nargin > 4 && ...
            isfield(options,'set_Load') && options.set_Load ~= 0
            temp = options.set_Load;
        else
            temp = 0;
        end
        LoadRecord = temp;
        SetSpeed = ME.dA;
    else
        ME.continuetoSS = false;
    end
end

%% Main Loop

ME.curTime = 0;
ME.CycledE = 0;
cycle_count = 0;
MechCycleEnergy = 0;
Plot_Powers = zeros(1000,1);
Plot_Learning_Rate = zeros(1000,1);
since_inflection = 0;
Plot_Number = 0;
power_factor = 1;
%Convergence_Plot = figure();
%Factor_Plot = figure();
if ME.ss_condition

```

```

    Tavg = zeros(size(ME.T));
    Tavg_count = 0;
end
while (ME.curTime < simTime)
    %% Main Solve
    ME.dt_max = 2*ME.A_Inc/(ME.dA_old + ME.dA);
    Forces = ME.Iteration_Solve();
    for i = length(Forces)
        for j = length(Forces{i})
            if isnan(Forces{i}(j)) || ~isreal(Forces{i}(j))
                ME.stop = true;
                success = false;
            end
        end
    end
end
if ME.stop
%   fprintf('Simulation Finished Prematurely. (in Run)\n');
    clear assignDynamic;
    clear implicitSolve;
    % clear dUFunc;
    clear KValue;
    clear solve_loops;
    return;
end
ME.curTime = ME.curTime + ME.dt_max;
progressbar(ME.curTime/simTime);
Power = ME.MechanicalSystem.Solve(ME.Inc, (ME.dA_old + ME.dA)/2, Forces);
MechCycleEnergy = MechCycleEnergy + Power*ME.dt_max;
if ~isempty(ME.MechanicalSystem.LoadFunction)
    Power = Power - ME.MechanicalSystem.LoadFunction((ME.dA_old + ME.dA)/2)*(ME.dA_old +
ME.dA)/2;
end
switch ME.MoveCondition
    case 1 % For Constant Motion Systems
        % Do nothing
    case 2 % For variable systems
        ME.dA_old = ME.dA;
        % fprintf([num2str(Power*ME.dt_max) '\n']);
        new_ke = Power*ME.dt_max + 0.5*ME.MechanicalSystem.Inertia*ME.dA^2;
        if new_ke < 0 && ME.ss_condition && ...
            ME.Model.recordOnlyLastCycle && ...
            Load_Function_is_Not_Given
            ME.dA = 0.1*2*pi; % Minimum speed of 0.1 Hz
        else
            if new_ke < 0
                ME.stop = true;
            else
                ME.dA = sqrt(2*new_ke/ME.MechanicalSystem.Inertia);
            end
        end
        if ME.ss_condition && ...
            ME.Model.recordOnlyLastCycle && ...
            Load_Function_is_Not_Given
            if ME.dA < 0.1*2*pi
                ME.dA = 0.1*2*pi;
            end
        end
        end
        % fprintf([num2str(ME.dA) '\n']);
    end

if ME.stop
    fprintf('Simulation Finished Prematurely. (in Run)\n');
    clear assignDynamic;
    clear implicitSolve;
    % clear dUFunc;
    clear KValue;
    clear solve_loops;
    return;
end

%% Obtain Results

```

```

if ME.Model.recordOnlyLastCycle
    Results.Data.dA(ME.Inc) = ME.dA;
    Results.Data.t(ME.Inc) = ME.curTime - AdjustTime;
    if grab_Pressure; Results.Data.P(:,ME.Inc) = ME.P; end
    if grab_Temperature; Results.Data.T(:,ME.Inc) = ME.T; end
    if grab_Velocity
        Results.Data.U(:,ME.Inc) = ME.Fc_V./ME.Fc_Area(indf);
    end
    if grab_PressureDrop
        Results.Data.dP(:,ME.Inc) = ME.dP;
    end
    if grab_Turbulence; Results.Data.turb(:,ME.Inc) = ME.turb; end
    if grab_ConductionFlux; Results.Data.cond(:,ME.Inc) = ME.CondFlux; end
    if ME.Model.recordStatistics
        Results.Data.QEnv(ME.Inc) = ME.E_ToEnvironment;
        Results.Data.QSource(ME.Inc) = ME.E_ToSource;
        Results.Data.QSink(ME.Inc) = ME.E_ToSink;
        Results.Data.Flow_Loss(ME.Inc) = ME.E_Flow_Loss;
        Results.Data.Power(ME.Inc) = Power;
        Results.Data.CR = ME.VolMax(:)./ME.VolMin(:);
        % Reset them
        ME.E_ToEnvironment = 0;
        ME.E_ToSource = 0;
        ME.E_ToSink = 0;
        ME.E_Flow_Loss = 0;
    end
else
    if ME.MoveCondition == 2
        Results.Data.A(n) = Results.Data.A(n-1) + ME.A_Inc;
        Results.Data.dA(n) = ME.dA;
        Results.Data.t(n) = ME.curTime;
        if length(Results.Data.t) == n
            LEN = abs(min([100 ceil((simTime-ME.curTime)*ME.dA/ME.A_Inc)]));
            Results.Data.A(n:n+LEN) = linspace(Results.Data.A(n-1),Results.Data.A(n-1)+...
                LEN*ME.A_Inc,LEN+1);
            Results.Data.dA(n+LEN) = 0;
            Results.Data.t(n+LEN) = 0;
            if grab_Pressure; Results.Data.P(length(ME.P),n+LEN) = 0; end
            if grab_Temperature; Results.Data.T(length(ME.P),n+LEN) = 0; end
            if grab_Velocity
                Results.Data.U(length(ME.Fc_dx),n+LEN) = 0;
            end
            if grab_PressureDrop
                Results.Data.dP(length(ME.P),n+LEN) = 0;
            end
            if grab_Turbulence; Results.Data.turb(length(ME.turb),n+LEN) = 0; end
            if grab_ConductionFlux; Results.Data.cond(length(ME.CondFlux),n+LEN) = 0; end
            if ME.Model.recordStatistics
                Results.Data.QEnv(n+LEN) = 0;
                Results.Data.QSource(n+LEN) = 0;
                Results.Data.QSink(n+LEN) = 0;
                Results.Data.Power(n+LEN) = 0;
            end
        end
    end
    if grab_Pressure; Results.Data.P(:,n) = ME.P; end
    if grab_Temperature; Results.Data.T(:,n) = ME.T; end
    if grab_Velocity
        Results.Data.U(:,n) = ME.Fc_V./ME.Fc_Area(indf);
    end
    if grab_PressureDrop
        Results.Data.dP(:,n) = ME.dP;
    end
    if grab_Turbulence; Results.Data.turb(:,n) = ME.turb; end
    if grab_ConductionFlux; Results.Data.cond(:,n) = ME.CondFlux; end
    if ME.Model.recordStatistics
        Results.Data.QEnv(n) = ME.E_ToEnvironment;
        Results.Data.QSource(n) = ME.E_ToSource;
        Results.Data.QSink(n) = ME.E_ToSink;
        Results.Data.Flow_Loss(n) = ME.E_Flow_Loss;
        Results.Data.Power(n) = Power;
    end
end

```

```

    Results.Data.CR = ME.VolMax(:)./ME.VolMin(:);
    % Reset them
    ME.E_ToEnvironment = 0;
    ME.E_ToSource = 0;
    ME.E_ToSink = 0;
    ME.E_Flow_Loss = 0;
end
end
if ~isempty(ME.Model.Sensors)
    for iSensor = ME.Model.Sensors; iSensor.getData(ME); end
end
if ~isempty(ME.Model.PVoutputs)
    for iPVoutput = ME.Model.PVoutputs; iPVoutput.getData(ME); end
end
if ME.curTime > previousTime + ME.Model.animationFrameTime
    previousTime = ME.curTime;
end
ME.old_vol = ME.vol;
n = n + 1;

%% Test Conditions (Reverse, Steady State, etc...)
if ME.ss_condition && ME.Model.recordOnlyLastCycle
    if ME.MoveCondition == 2
        if Load_Function_is_Not_Given
            if ME.dA < 0
                ME.dA = 0.1*2*pi; % Minimum speed of 0.1 Hz
            end
        end
    end
end
if ME.dA < 0
    fprintf('XXX Engine Reversed Directions, solving exited XXX\n');
    return;
else
    if ME.ss_condition && ~ME.continuetoSs
        Tavg = Tavg + ME.T;
        Tavg_count = Tavg_count + 1;
        if Tavg_count == 1
            T_previous = Tavg;
        end
    end
    ME.Inc = ME.Inc + 1;
    if ME.Inc == Frame.NTheta
        cycle_count = cycle_count + 1;
        Plot_Number = Plot_Number + 1;
        Plot_Powers(Plot_Number) = MechCycleEnergy/(ME.curTime - AdjustTime);
        fprintf(['Speed: (rpm) ' num2str(60*ME.dA/(2*pi)) ...
            'Power: (W) ' num2str(Plot_Powers(Plot_Number)) '\n']);

        Results.Data.SnapShot_P = ME.Rs.*ME.T(1:length(ME.P)).*...
            ME.m(1:length(ME.P))./ME.vol(1:length(ME.P));

        if ME.Model.showLivePV
            for iPVoutput = ME.Model.PVoutputs; iPVoutput.updatePlot(); end
        end

        % Acquire an understanding of the solution plateauing
        Plot_Number = Plot_Number + 1;
        Plot_Powers(Plot_Number) = MechCycleEnergy/(ME.curTime - AdjustTime);
        MechCycleEnergy = 0;
        Plot_Learning_Rate(Plot_Number) = power_factor;
        if isempty(Convergence_Plot) || ...
            ~isvalid(Convergence_Plot) || ...
            Convergence_Plot < 1
            Convergence_Plot = figure();
        end
        figure(Convergence_Plot);
        plot(1:Plot_Number,Plot_Powers(1:Plot_Number));

        %cycle_count = cycle_count + 1;
        ME.Inc = 1;
    end
end

```



```

        b(i) = 298;
    end
end
end

ME.CondEff(:) = 0;
ME.CondTempEff(:) = 0;
ME.CycleTime = 0;

A = sparse(A);
var = A\b;

% Calculate shifted values based on current transient
ME.T(sindn) = var + T_delta(sindn);

% Reset Tavg for the next cycle
Tavg(:) = 0;
Tavg_count = 0;

end

% Detect if it is steady state
% Compare the average over 10 cycles to see if it appears to
% be converged
Precord(1:ss_cycles-1) = Precord(2:ss_cycles);
Precord(ss_cycles) = Plot_Powers(Plot_Number);
temp = ss_tolerance*max(Precord(end),1);
% ContinuettoSS is the a flag for the last cycle which is run
% ... at a finer timestep that the converging cycles.
if ME.continuettoSS; break; end
if CustomRSSQ(diff(Precord)) < temp
    ME.continuettoSS = true;
    ME.MaxCourant = ME.Model.MaxCourantFinal;
    ME.MaxFourier = ME.Model.MaxFourierFinal;
end
end

% Modify Gas Mass so that the engine is at the engine pressure.
if ME.Model.recordOnlyLastCycle
    %i = 1;
    for i = 1:length(ME.Regions)
        if ~ME.isEnvironmentRegion(i)
            nodes = ME.Regions{i};
            Pregion = ME.PRegion(i)/ME.PRegionTime;
            ME.m(nodes) = ...
                (power_factor*engine_Pressure/Pregion + ...
                (1-power_factor))*ME.m(nodes);
        end
    end
    ME.PRegion(:) = 0; ME.PRegionTime(:) = 0;
    for iPvOutput = ME.Model.PVOutputs
        Pregion = ME.PRegion(i)/ME.PRegionTime(i);
        ME.PRegion(i) = 0; ME.PRegionTime(i) = 0;
        ME.m(iPvOutput.RegionNodes) = ...
            (power_factor*engine_Pressure/Pregion + ...
            (1-power_factor))*ME.m(iPvOutput.RegionNodes);
        i = i + 1;
    end
end

end

% Assess cycle time and modify mechanism load accordingly
if ME.ss_condition && ME.Model.recordOnlyLastCycle
    if ME.MoveCondition == 2
        if Load_Function_is_Not_Given
            % Modify Load to approach initial speed
            % Calculate Speed
            speed = 2*pi/(ME.curTime - AdjustTime);
            dspeed = min(0.01/(max(log(Plot_Number),1)), ...
                0.5*abs((SetSpeed - speed)/SetSpeed));
            if speed < SetSpeed

```

```

        LoadRecord = LoadRecord - power_factor*dspeed;
    else
        LoadRecord = LoadRecord + power_factor*dspeed;
    end
    ME.MechanicalSystem.LoadFunction = @(Speed) LoadRecord;
end
end
end

AdjustTime = ME.curTime;
end
assignDynamic(ME,ME.Inc,false); % Initialize the dynamic function
end
end
progressbar(1);

if ~ME.Model.recordOnlyLastCycle
    Results.Data.dA = Results.Data.dA(1:n-1);
    Results.Data.t = Results.Data.t(1:n-1);
    if grab_Pressure; Results.Data.P = Results.Data.P(:,1:n-1); end
    if grab_Temperature; Results.Data.T = Results.Data.T(:,1:n-1); end
    if grab_Velocity; Results.Data.U = Results.Data.U(:,1:n-1); end
    if grab_PressureDrop; Results.Data.dP = Results.Data.dP(:,1:n-1); end
    if grab_Turbulence; Results.Data.turb = Results.Data.turb(:,1:n-1); end
    if grab_ConductionFlux; Results.Data.cond = Results.Data.cond(:,1:n-1); end
    if ME.Model.recordStatistics
        Results.Data.QEnv = Results.Data.QEnv(:,1:n-1);
        Results.Data.QSource = Results.Data.QSource(:,1:n-1);
        Results.Data.QSink = Results.Data.QSink(:,1:n-1);
        Results.Data.Flow_Loss = Results.Data.Flow_Loss(:,1:n-1);
        Results.Data.Power = Results.Data.Power(:,1:n-1);
        Results.Data.CR(:) = ME.VolMax(:)./ME.VolMin(:);
    end
end

%% Save Data
if record_data
    if ME.Model.recordStatistics
        statistics = struct(...
            'Time',Results.Data.t,...
            'Angle',Results.Data.A,...
            'Omega',Results.Data.dA,...
            'To_Environment',Results.Data.QEnv,...
            'To_Source',Results.Data.QSource,...
            'To_Sink',Results.Data.QSink,...
            'Flow_Loss',Results.Data.Flow_Loss,...
            'Power',Results.Data.Power,...
            'TotalPower',Plot_Powers,...
            'Gas_Nodes',length(ME.P)-1,...
            'Solid_Nodes',length(ME.T)-length(ME.P)+1,...
            'Gas_Faces',length(ME.Fc_V),...
            'Mixed_Faces',length(ME.Fc_R)-length(ME.Fc_V),...
            'Solid_Faces',length(ME.Fc_Cond)-length(ME.Fc_R),...
            'CR',Results.Data.CR,...
            'VMin',sum(ME.VolMin(:)),...
            'VMax',sum(ME.VolMax(:)));
    end
    if ME.Model.recordStatistics
        if nargin > 4
            if isempty(ME.Model.outputPath)
                save([options.title '_Statistics'],'statistics');
            else
                save([ME.Model.outputPath '\' options.title '_Statistics'],'statistics');
            end
        else
            if isempty(ME.Model.outputPath)
                save([ME.Model.name '_Statistics'],'statistics');
            else
                save([ME.Model.outputPath '\' ME.Model.name '_Statistics'],'statistics');
            end
        end
    end
end
end

```

```

end
if nargin > 4
    if ~isempty(ME.Model.Sensors)
        for iSensor = ME.Model.Sensors; iSensor.plotData(true,options.title); end
    end
    if ~isempty(ME.Model.PVoutputs)
        for iPVoutput = ME.Model.PVoutputs; iPVoutput.plotData(true,options.title); end
    end
else
    if ~isempty(ME.Model.Sensors)
        for iSensor = ME.Model.Sensors; iSensor.plotData(true,ME.Model.name); end
    end
    if ~isempty(ME.Model.PVoutputs)
        for iPVoutput = ME.Model.PVoutputs; iPVoutput.plotData(true,ME.Model.name); end
    end
end
end
if Load_Function_is_Not_Given
    ME.MechanicalSystem.LoadFunction = function_handle.empty;
end
%% Clean up
clear assignDynamic;
clear implicitSolve;
% clear dUFunc;
clear KValue;
clear solve_loops;
end

```

```
function implicitSolve(ME,initialize) %#ok<INUSD>
```

```

persistent indf;
persistent indS;
persistent indmf;
persistent indn;
persistent sindn;
persistent indnminus;
persistent indmfminus;
persistent totalNodes;
persistent isDynVol;
persistent dvind;
persistent lenf;
persistent lenn;
persistent nd0;
persistent nd1;
persistent nd2;
persistent nd3;
persistent nd1mf;
persistent nd2mf;
persistent Cfcs;
persistent Cnd1;
persistent Cnd2;
persistent Pi;
persistent Ti;
persistent rhoEnv;
persistent PEnv;
persistent uEnv;
persistent TEnv;
persistent Fcrho;
persistent Fcu;
persistent Fcmu;
persistent FcT;
persistent FcP;
persistent theta_FL;
persistent Q;
persistent Qtd;
persistent ReCritComparitor;
persistent F2C;
persistent ExFc;
persistent ExR1;
persistent ExR2;
persistent ExLfc;
persistent ExLC;

```



```

persistent ExLN;
persistent FlowTimeStep;
persistent RecordStatistics;
persistent facesES;
persistent sgnES;
persistent facesEG;
persistent sgnEG;
persistent facesSr;
persistent sgnSr;
persistent facesSi;
persistent sgnSi;
persistent Ci;
persistent Cs;
persistent Qi;
persistent Qs;
persistent C1;
persistent C2;
persistent C3;
persistent C4;
persistent CT;
persistent nlambda;
% persistent dP_dt;
if nargin == 2
    % dP_dt = zeros(length(ME.Regions),1);
    ME.Fc_V_averager = zeros(size(ME.Fc_V));
    ME.Fc_turb_averager = ME.Fc_V_averager;
    ME.assignDynamic(ME.Inc);
    indf = (1:length(ME.Fc_U))';
    if ~isempty(ME.Fc_DynShear_Factor)
        indS = ME.Fc_DynShear_Factor(1,:);
    else
        indS = [];
    end
    Cs = ME.m(ME.Fc_Nd(ME.FcApprox,2))./ME.dT_dU(ME.Fc_Nd(ME.FcApprox,2));
    Cs(isinf(Cs)) = 1e6;
    indmf = ME.Mix_Fc;
    indn = (1:length(ME.P))';
    totalNodes = length(ME.T);
    lenf = length(indf);
    lenn = length(indn);
    sindn = (lenn+1:totalNodes);
    indnminus = (1:lenn-1)';
    nd0 = ME.Fc_Nd03(indf,1);
    nd1 = ME.Fc_Nd(indf,1);
    nd2 = ME.Fc_Nd(indf,2);
    nd3 = ME.Fc_Nd03(indf,2);
    nd1mf = ME.Fc_Nd(indmf,1);
    nd2mf = ME.Fc_Nd(indmf,2);
    isDynVol = logical(ME.isDynVol(indn));
    dvind = indn(isDynVol);
    elements = ME.dT_dU(nd2mf)>0;
    indmfminus = indmf(elements);
    nd2mf = nd2mf(elements);
    Ti = zeros(size(ME.T));
    Pi = zeros(size(ME.P));
    rhoEnv = ME.rho(lenn);
    PEnv = ME.P(end);
    uEnv = ME.u(lenn);
    TEnv = ME.T(lenn);
    Q = zeros(totalNodes,1);
    Qtd = Q;
    Cfcs = ME.Cond_Fcs;
    Cnd1 = ME.Cond_Nds1;
    Cnd2 = ME.Cond_Nds2;
    ReCritComparitor = [zeros(1,lenn-1); ...
        11.5*ones(1,lenn-1)];
    ExR1 = ME.ExplicitNorm(:,2);
    ExR2 = ME.ExplicitNorm(:,3);
    F2C = ME.Fc2Col;
    ExFc = ME.ExplicitNorm(:,1);
    ExLFC = ME.ExplicitLeak(:,1);

```

```

ExLC = ME.ExplicitLeak(:,2);
ExLN = ME.ExplicitLeak(:,3);
ME.Fc_V = zeros(lenf,1);
Fcu = zeros(lenf,1);
Fcrho = Fcu;
Fcmu = Fcu;
FcT = Fcu;
FcP = Fcu;
theta_FL = Fcu;
FlowTimeStep = 1;
RecordStatistics = ME.Model.recordStatistics;
if RecordStatistics
    facesES = ME.ToEnvironmentSolid(1,:);
    sgnES = ME.ToEnvironmentSolid(2,:);
    facesEG = ME.ToEnvironmentGas(1,:);
    sgnEG = ME.ToEnvironmentGas(2,:);
    facesSr = ME.ToSource(1,:); sgnSr = ME.ToSource(2,:);
    facesSi = ME.ToSink(1,:); sgnSi = ME.ToSink(2,:);
    ME.E_ToEnvironment = 0;
    ME.E_ToSource = 0;
    ME.E_ToSink = 0;
    ME.E_Flow_Loss = 0;
end
return
end
t = 0; done = false;
ME.Fc_V_averager(:) = 0;
ME.Fc_turb_averager(:) = 0;
while ~done
    hmax = min(ME.dt_max-t, ME.Solid_dt_max(ME.Inc));

    %% Assign Properties
    mi = ME.m(indn);
    ui = [ME.u; ME.T(sindn)./ME.dT_dU(sindn)];
    signU = sign(ME.Fc_U);
    Tnew = zeros(size(ME.vol(indn)));
    dm_dt = Tnew;
    unew = Tnew;
    mnew = Tnew;

    inc = ME.Inc + t/ME.dt_max;
    % if inc == 1
    %     fprintf('start');
    % end
    time = ME.curTime + t;
    ME.assignDynamic(inc);
    ME.vol(ME.vol<=0) = 1e-8;
    Areai = ME.Fc_Area(indf);

    % Density - Upwinding
    rhoi = mi./ME.vol(indn);
    rhoi(end) = rhoEnv;

    % Pressure
    n = length(ME.Regions);
    for i = 1:n
        nodes = ME.Regions{i};
        uTemp = ui(nodes);
        TTemp = ME.u2T{i}(uTemp);
        ME.k(nodes) = ME.kFunc{i}(TTemp);
        ME.mu(nodes) = ME.muFunc{i}(TTemp);
        ME.dT_dU(nodes) = ME.dT_duFunc{i}(uTemp);
        ME.dh_dT(nodes) = ME.dh_dTFunc{i}(TTemp);
        Ti(nodes) = TTemp;
    end
    Ti(sindn) = ME.T(sindn);
    ME.Fc_turb = 0.5*(ME.turb(nd1) + ME.turb(nd2));

    Pi = Ti(indn).*rhoi.*ME.Rs;
    % fprintf([num2str(std(Pi(1:end-1))) '\n']);
    Pi(end) = PEnv;

```

```

FcP = 0.5*(Pi(nd1) + Pi(nd2));
% Thermal Conductivity
ME.k(dvind) = ME.k(dvind) + ...
    0.021*rhoi(dvind).*ME.Dh(dvind).*...
    (ME.Rs(dvind) + 1./ME.dT_dU(dvind)).*sqrt(ME.turb(dvind));
%
ME.Fc_k = (ME.k(nd1) + ME.k(nd2))/2;
ME.k(ME.k>100) = 100;
% Viscosity
ME.mu(dvind) = ME.mu(dvind) + ...
    0.021*rhoi(dvind).*ME.Dh(dvind).*sqrt(ME.turb(dvind));
%
Fcmu = (ME.mu(nd1) + ME.mu(nd2))/2;
% Do flux limiting on the int energy, then calculate T from u2T
forward = (ME.Fc_V > 0);
Fcu(forward) = ui(nd1(forward));
Fcu(~forward) = ui(nd2(~forward));
theta_FL(forward) = (ui(nd1(forward))-ui(nd0(forward)))/...
    (ui(nd2(forward))-ui(nd1(forward)));
theta_FL(~forward) = (ui(nd2(~forward))-ui(nd3(~forward)))/...
    (ui(nd1(~forward))-ui(nd2(~forward)));
theta_FL(isnan(theta_FL)) = 1;
theta_FL(theta_FL > 1) = 1;
theta_FL(theta_FL < -1) = -1;
Fcu = Fcu + ((theta_FL.^2 + theta_FL)/(theta_FL.^2 + 1)).*...
    (ME.Fc_A.*ui(nd0) + ME.Fc_B.*ui(nd1) + ...
    ME.Fc_C.*ui(nd2) + ME.Fc_D.*ui(nd3) - Fcu);
n = length(ME.Regions);
for i = 1:n
    faces = ME.RegionFcs{i};
    FcT(faces) = ME.u2T{i}(Fcu(faces));
    ME.Fc_k(faces) = ME.kFunc{i}(FcT(faces));
    Fcmu(faces) = ME.muFunc{i}(FcT(faces));
end
Fcrho(indf) = 2.*FcP(indf)/((ME.Rs(nd1(indf)) + ME.Rs(nd2(indf))).*FcT(indf));
ME.PR = abs((1./ME.dT_dU(indn) + ME.Rs).*ME.mu./ME.k);
ME.Fc_PR = 0.5*(ME.PR(nd1)+ME.PR(nd2));

% Parameters used by outside calculation
ME.Fc_RE = abs(ME.Fc_U.*Fcrho.*ME.Fc_Dh./Fcmu);
ME.Fc_RE(ME.Fc_RE==0) = 1e-8;
ME.getWeight();

% Assign Node Reynold's Number
area = zeros(lenn-1,1);
ME.RE = area;
for i = 1:3:length(ME.TransNet)-2
    ME.RE(ME.TransNet{i+1}) = ME.RE(ME.TransNet{i+1}) + ...
        ME.Fc_RE(ME.TransNet{i+2}).*Areai(ME.TransNet{i+2});
    area(ME.TransNet{i+1}) = area(ME.TransNet{i+1}) + ...
        Areai(ME.TransNet{i+2});
end
ME.RE = ME.RE./(area);
ME.RE(isnan(ME.RE)) = 1e-8;

%% Calculate Flow Independent Energy Flux to Nodes
invConv = ME.Dh(indnminus)/(ME.k(indnminus).*ME.Nusselt());
ME.Fc_Cond(indmf) = ME.Fc_Area(indmf)/(ME.Fc_R(indmf)' + invConv(ndlmf));
ME.Fc_Cond(indf) = ME.NkFunc().*ME.Fc_k(indf).*Areai./ME.Fc_Cond_Dist(indf);
ME.Fc_Cond(indS) = ME.Fc_Cond(indS) + abs(...
    (ME.dA/4)*(1./ME.dT_dU(indS)).*...
    ME.Fc_Shear_Factor(indS).*Fcrho(indS).*Areai(indS));

Q(:) = 0;
Qfc = ME.Fc_Cond(Cfcs).*(Ti(Cnd1) - Ti(Cnd2));
for i = 1:3:length(ME.CondNet)-2
    Q(ME.CondNet{i+1}) = ...
        Q(ME.CondNet{i+1}) + ME.CondNet{i}.*Qfc(ME.CondNet{i+2});
end

fcs = ME.FcApprox;
Ci = mi(ME.Fc_Nd(fcs,1))./ME.dT_dU(ME.Fc_Nd(fcs,1));
C1 = Ci + Cs;

```

```

CT = (Ti(ME.Fc_Nd(fcs,1)).*Ci + Ti(ME.Fc_Nd(fcs,2)).*Cs)./Cl - Ti(ME.Fc_Nd(fcs,1));
nlambda = -ME.Fc_Cond(fcs).*(1./Cs + 1./Ci);
Qs = Q(ME.Fc_Nd(fcs,2));

%
%
ME.MaxFourier = 0.025;
ME.MaxCourant = 0.025;

% Assign TimeStep
if ~isempty(ME.Fc_dx)
if ~isempty(ME.Fc_Cond)
hmax = min([hmax FlowTimeStep ...
ME.MaxFourier*...
min(min(mi(nd1)./(ME.dT_dU(nd1).*ME.Fc_Cond(indf))), ...
min(mi(nd2)./(ME.dT_dU(nd2).*ME.Fc_Cond(indf)))) ...
ME.MaxFourier*min(mi(nd1mf)./(ME.dT_dU(nd1mf).*ME.Fc_Cond(indmf))) ...
ME.MaxFourier*min(ME.m(nd2mf)./(ME.dT_dU(nd2mf).*ME.Fc_Cond(indmfminus))))]);
else
hmax = min([hmax FlowTimeStep ...
ME.MaxFourier*min(mi(nd1mf)./(ME.dT_dU(nd1mf).*ME.Fc_Cond(indmf))) ...
ME.MaxFourier*min(ME.m(nd2mf)./(ME.dT_dU(nd2mf).*ME.Fc_Cond(indmfminus))))]);
end
else
if ~isempty(ME.Fc_Cond)
hmax = min([hmax ...
ME.MaxFourier*...
min(min(mi(nd1)./(ME.dT_dU(nd1).*ME.Fc_Cond(indf))), ...
min(mi(nd2)./(ME.dT_dU(nd2).*ME.Fc_Cond(indf))))]);
end
end
% hmax = 1e-4;

RegionPressure = zeros(n,1);
%
%
%
%
dm_region = RegionPressure;
m_region = RegionPressure;
V_region = RegionPressure;
dV_region = RegionPressure;
E_region = RegionPressure;

% Calculate Explicit Volume Flow - Normal
if ~isempty(ME.ExplicitNorm)
ME.Fc_dP(ExFc) = Pi(nd1(ExFc)) - Pi(nd2(ExFc));
x = sign(ME.Fc_dP(ExFc)).*...
sqrt(2*(Areai(ExFc).^2).*abs(ME.Fc_dP(ExFc))./Fcrho(ExFc));
V = ME.Fc_V(ExFc);
V_max = 350*Areai(ExFc);
V(V > V_max) = V_max(V > V_max);
iteration = 1;
while iteration < 100
oldV = V;
K = ME.KValue(ExFc);
V = x./sqrt(K);
V(V > V_max) = V_max(V > V_max);
V(isnan(V)) = 1;
if CustomRSSQ(oldV-V) < 1e-8; break; end
end
ME.Fc_V(ExFc) = V;
for i = 1:length(ExR1)
E = ME.Fc_V(ExFc(i)).*FcP(ExFc(i));
E_region(ExR1(i)) = E_region(ExR1(i)) - E;
E_region(ExR2(i)) = E_region(ExR2(i)) + E;
end
end

% Calculate Explicit Volume Flow - Leak
if ~isempty(ME.ExplicitLeak)
dP = Pi(nd1(ExLFC)) - Pi(nd2(ExLFC));
ME.Fc_V(ExLFC) = sign(dP).*ExLC.*(abs(dP)).^ExLN;
for i = 1:length(ExLR1)
if ME.Fc_V(ExLFC(i)) > 0; x = Pi(nd2(ExLFC(i)));
else; x = Pi(nd1(ExLFC(i)));
end
end

```

```

        E = ME.Fc_V(ExLFC(i)).*x;
        E_region(ExLR1(i)) = E_region(ExLR1(i)) - E;
        E_region(ExLR2(i)) = E_region(ExLR2(i)) + E;
    end
end

% Assume that flow can't change that much
if ~isempty(ME.Fc_dx); h = min(hmax, 0.8*FlowTimeStep);
else; h = hmax;
end

% Boundary Work
P0 = zeros(length(ME.Regions),1);
for i = 1:length(ME.Regions)
    if ME.isEnvironmentRegion(i)
        P0(i) = PEnv;
    else
        nodes = ME.Regions{i};
        ni = nodes(1);
        P0(i) = Ti(ni)*ME.R(i)*mi(ni)/ME.vol(ni);
        dV_dt_region = sum(ME.dV_dt(nodes));
        M_region = sum(mi(nodes));
        E_region(i) = E_region(i) - P0(i)*dV_dt_region;
        Q(nodes) = (mi(nodes)/M_region).*E_region(i);
        nodes = ME.Regions{i}(isDynVol(ME.Regions{i}));
        nodes = ME.Regions{i}(:);
        P0(i) = sum(Ti(nodes).*ME.R(i).*mi(nodes))/sum(ME.vol(nodes));
        for ni = nodes
            Q(ni) = Q(ni) - ME.dV_dt(ni)*P0(i);
        end
    end
end

%% Setup Variables for Volume Flux Solving
not_done = true;
while not_done
    i = 1;
    Qi = Q(ME.Fc_Nd(fcs,1));
    for nd = ME.Fc_Nd(fcs,1)'
        data = ME.Faces{nd,1};
        mdot = ME.Fc_V(data(:,1)).*double(data(:,2)).*Fcrho(data(:,1));
        dm_dt(nd) = sum(mdot);
        Qi(i) = Qi(i) + ...
            sum(mdot.*(Fcu(data(:,1)) + P0(i)./Fcrho(data(:,1)))) - ...
            P0(i).*ME.dV_dt(nd);
        Qi(i) = Qi(i) + ...
            sum(mdot.*(Fcu(data(:,1)) + (P0(i) + dP_dt(i,ME.Inc))./Fcrho(data(:,1)))) - ...
            P0(i).*ME.dV_dt(nd) - h*(ME.vol(nd) + 0.5*ME.dV_dt(nd)).*dP_dt(i,ME.Inc);
    end
    i = i + 1;
end
C2 = (Qi + Qs)./C1;
C3 = (Ci.*Qs - Cs.*Qi)./(ME.Fc_Cond(fcs).*C1.^2);
C3(isnan(C3)) = 0;
C3(isinf(C3)) = 0;
C4 = (Ti(ME.Fc_Nd(fcs,1)) - Ti(ME.Fc_Nd(fcs,2)))./(C3.*C1);
C4(isnan(C4)) = 0;
C4(isinf(C4)) = 0;
Qtd(:) = 0;
Qtemp = -(Ci.*(CT - Cs.*C3.*(1 - (1 + C4)).*exp(nlambda*h))...
    + (Ci.*C2 - Qi)*h);
for i = 1:length(ME.FcApprox)
    fc = ME.FcApprox(i);
    % 1st is gas, it can have multiple connections
    Qtd(ME.Fc_Nd(fc,1)) = Qtd(ME.Fc_Nd(fc,1)) - Qtemp(i);
    % 2nd is solid, it can only have 1 connection
    Qtd(ME.Fc_Nd(fc,2)) = Qtemp(i);
end

not_done = false;
Vnew = ME.vol(indn) + h*ME.dV_dt(indn);

```

```

for ind = find(Vnew<=0); Vnew(ind) = 1e-8; end

%% Enter Volume Flux Solving Loop
for i = 1:length(ME.Regions)
    if ~isempty(ME.RegionFcs{i})
        faces = ME.ActiveRegionFcs{i};
        nodes = ME.Regions{i};
        A = zeros(ME.RegionFcCount(i));
        b = zeros(ME.RegionFcCount(i),1);
        if ME.isEnvironmentRegion(i)
            % Vdot Rows
            for row = 1:length(faces)
                ni = nodes(row);
                b(row) = Vnew(ni)*PEnv/ME.R(i) - ...
                    Ti(ni)*mi(ni) + ME.dT_dU(ni)*(h*Q(ni) + Qtd(ni) ...
                        - PEnv.*(Vnew(ni)-ME.vol(ni)));
                data = ME.Faces{ni,1};
                for p = 1:size(data,1)
                    fc = data(p,1);
                    X = h*double(data(p,2))*Fcrho(fc)*(...
                        Ti(ni) + ME.dT_dU(ni)*...
                            (Fcu(fc) - ui(ni) + PEnv/Fcrho(fc)));
                    if data(p,3); b(row) = b(row) - ME.Fc_V(fc)*X;
                    else; A(row,F2C(fc)) = A(row,F2C(fc)) + X;
                    end
                end
            end
            if isempty(ME.RegionLoops{i})
                As = sparse(A);
                V = As\b;
                ME.Fc_V(faces) = V(F2C(faces));
                ME.Fc_U(faces) = ME.Fc_V(faces)./ME.Fc_Area(faces) +
ME.Fc_Vel_Factor(faces)*ME.dA;
            else
                solve_loops(ME,i,F2C,length(faces)+1,A,b,Fcrho,Fcmu,time);
            end
        else
            % Solve for Vdot
            for row = 1:length(faces)
                ni = nd1(faces(row));
                nj = nd2(faces(row));
                b(row) = ...
                    (Ti(ni)*mi(ni) + ME.dT_dU(ni)*(h*Q(ni) + Qtd(ni) ...
                        - h*P0(i).*ME.dV_dt(ni))/Vnew(ni) - ...
                    (Ti(nj)*mi(nj) + ME.dT_dU(nj)*(h*Q(nj) + Qtd(nj) ...
                        - h*P0(i).*ME.dV_dt(nj))/Vnew(nj);
                    b(row) = ...
                    (Ti(ni)*mi(ni) + ME.dT_dU(ni)*(h*Q(ni) + Qtd(ni) ...
                        - (P0(i) + 0.5*h*dP_dt(i,ME.Inc)).*(Vnew(ni)-ME.vol(ni)) ...
                        - h*(ME.vol(ni)).*dP_dt(i))/Vnew(ni) - ...
                    (Ti(nj)*mi(nj) + ME.dT_dU(nj)*(h*Q(nj) + Qtd(nj) ...
                        - (P0(i) + 0.5*h*dP_dt(i,ME.Inc)).*(Vnew(nj)-ME.vol(nj)) ...
                        - h*(ME.vol(nj)).*dP_dt(i))/Vnew(nj);
                data = ME.Faces{ni,1};
                for p = 1:size(data,1)
                    fc = data(p,1);
                    X = h*double(data(p,2))*Fcrho(fc)*(...
                        Ti(ni) + ME.dT_dU(ni)*(Fcu(fc) - ui(ni) ...
                            + P0(i)/Fcrho(fc))/Vnew(ni);
                    X = h*double(data(p,2))*Fcrho(fc)*(...
                        Ti(ni) + ME.dT_dU(ni)*(Fcu(fc) - ui(ni) ...
                            + (P0(i) + 0.5*h*dP_dt(i,ME.Inc))/Fcrho(fc))/Vnew(ni);
                    if data(p,3); b(row) = b(row) + ME.Fc_V(fc)*X;
                    else; A(row,F2C(fc)) = A(row,F2C(fc)) - X;
                    end
                end
            end
            data = ME.Faces{nj,1};
            for p = 1:size(data,1)
                fc = data(p,1);
                X = h*double(data(p,2))*Fcrho(fc)*(...

```



```

if ME.isEnvironmentRegion(i); nodes = nodes(nodes~=lenn); end
for nd = nodes'
    data = ME.Faces(nd,1);
    mdot = ME.Fc_V(data(:,1)).*double(data(:,2)).*Fcrho(data(:,1));
    dm_dt(nd) = sum(mdot);
    Q(nd) = Q(nd) + sum(mdot.*(Fcu(data(:,1)) + P0(i)./Fcrho(data(:,1))));
end
mnew(nodes) = ME.m(nodes) + h*dm_dt(nodes);
unew(nodes) = (mi(nodes).*ui(nodes) + Qtd(nodes) + h*(Q(nodes)...
    - P0(i).*ME.dV_dt(nodes)))./mnew(nodes);
Tnew(nodes) = ME.u2T{i}(unew(nodes));
ME.P(nodes) = ME.R(i).*mnew(nodes).*Tnew(nodes)./Vnew(nodes);
% Fix The parameters to ensure that the pressure is constant
if ME.isEnvironmentRegion(i)
    Pavg = PEnv; % dP_dt(i) = 0;
else
    Pavg = sum(ME.P(nodes).*ME.vol(nodes))/sum(ME.vol(nodes));
    if ME.Inc == 1 && t == 0
        dP_dt(i) = dP_dt(i);
    else
        dP_dt(i) = (Pavg - P0(i))/h;
    end
end
Tnew(nodes) = Pavg./ME.P(nodes).*Tnew(nodes);
end
ME.P(lenn) = PEnv;

%% TURBULENCE ITEMS
if lenf ~= 0
    change = indf(sign(ME.Fc_U(indf))~=signU);
    ME.Fc_to(change) = ME.curTime + t;
    changed_nodes = false(lenn,1);
    changed_nodes(nd1(change)) = true;
    changed_nodes(nd2(change)) = true;
    changed_nodes(isDynVol) = false;
    ME.to(changed_nodes) = ME.curTime + t;
    ME.turb(changed_nodes) = 0;
    % Define (for those that care) the critical reynolds number
    ME.Va(indnminus) = ME.dA*rhoi(indnminus).*(ME.Dh(indnminus).^2)./ME.mu(indnminus);
    ReCritComparator(1,:) = (sqrt(ME.Va(indnminus))./...
        (0.075+0.112.*(ME.curTime + t - ME.to(1:lenn-1))))';
    ME.REcrit(indnminus) = 200*max(ReCritComparator);
    TurbTime = 0;
    steps = ME.Fc_K12 > 0;
    while TurbTime < h
        ME.Fc_turb = 0.5*(ME.turb(nd1) + ME.turb(nd2));
        h_turb = h - TurbTime;
        dturb_dt = zeros(lenn,1);
        % Turbulence Transport
        for i = indf(steps)'
            n1 = nd1(i);
            n2 = nd2(i);
            % dturb_dt = turb*(dKE_dt / KE - dm_dt / m)
            if ME.Fc_V(i) > 0
                % Leaving n1
                % Leaving a variable volume space ... No change to kappa
                if ~isDynVol(n1)
                    dturb_dt(n1) = dturb_dt(n1) + (- ME.Fc_turb(i) + ME.turb(n1));
                end
                % Entering n2
                if isDynVol(n2)
                    dKE_dm = 0.5*(ME.Fc_V(i)/Areai(i))^2;
                    dturb_dt(n2) = dturb_dt(n2) + (dKE_dm - ME.turb(n2));
                else
                    dturb_dt(n2) = dturb_dt(n2) + (1 - ME.turb(n2));
                end
            end
        else
            % Entering n1
            if isDynVol(n1)
                dKE_dm = 0.5*(ME.Fc_V(i)/Areai(i))^2;
                dturb_dt(n1) = dturb_dt(n1) + (dKE_dm - ME.turb(n1));
            end
        end
    end
end

```



```

else
    dturb_dt(n1) = dturb_dt(n1) + (1 - ME.turb(n1));
end
% Leaving n2
% Leaving a variable volume space ... No change to kappa
if ~isDynVol(n2)
    dturb_dt(n2) = dturb_dt(n2) + (- ME.Fc_turb(i) + ME.turb(n2));
end
end
end
for i = indf(~steps)'
    % dturb_dt = turb*(dKE_dt / KE - dm_dt / m)
    % dturb_dt = turb * (dKE_dm / ke - 1) * dm_dt / m
    % dturb_dt = .... * (dKE_dm / ke - 1) * ..... / ..
    dKE_dm = ME.Fc_turb(i);
    dturb_dt(nd1(i)) = dturb_dt(nd1(i)) - ...
        (dKE_dm - ME.turb(nd1(i))) * sign(ME.Fc_V(i));
    dturb_dt(nd2(i)) = dturb_dt(nd2(i)) + ...
        (dKE_dm - ME.turb(nd2(i))) * sign(ME.Fc_V(i));
end
dturb_dt = dturb_dt .* (dm_dt ./ mi);

%% Turbulence Decay/Generation Within Nodes
for i = 1:lenn-1
    if isDynVol(i)
        % Variable Volume
        % F. J. Cantelmi, Measurement and Modeling of In-Cylinder Heat Transfer
        % with Inflow-Produced Turbulence, MS Thesis, Virginia Polytechnic Institute
        % and State University, June (1995)
        dturb_dt(i) = dturb_dt(i) - ...
            5.8*(abs(ME.turb(i) + dturb_dt(i)*h)^(3/2))/ME.Dh(i) - ...
            ME.turb(i)/mi(i)*dm_dt(i);
    else
        if ME.RE(i) > ME.REcrit(i)
            % Generate
            dturb_dt(i) = dturb_dt(i) + ...
                (0.008*ME.dA*ME.RE(i)/ME.Va(i))*(1-ME.turb(i));
        else
            % Decay
            dturb_dt(i) = dturb_dt(i) - ...
                (0.25*ME.dA*2300/ME.Va(i))*abs(ME.turb(i))^(3/2);
        end
    end
end
end
% Test - To limit timestep
d_turb = zeros(size(ME.turb(indn)));
for i = indn'
    if isDynVol(i)
        d_turb(i) = h_turb*dturb_dt(i);
    else
        d_turb(i) = max(0,min(1,ME.turb(i) + ...
            h_turb*dturb_dt(i))) - ME.turb(i);
    end
end
max_d_turb = max(abs(d_turb(~isDynVol(indn))));
if max_d_turb > 0.1; h_turb = h_turb*0.1/max_d_turb; end
% Assign Values
ME.turb = ME.turb + h_turb*dturb_dt;
ME.turb(ME.turb<0) = 0;
for i = indn'
    if ME.turb(i) > 1 && ~isDynVol(i)
        ME.turb(i) = 1;
    end
end
ME.turb(lenn) = 0;
TurbTime = TurbTime + h_turb;
end
end
if ME.Model.recordOnlyLastCycle
    for i = 1:length(ME.Regions)

```

```

        if ~ME.isEnvironmentRegion(i)
            nodes = ME.Regions{i};
            ME.PRegion(i) = ME.PRegion(i) + ME.P(nodes(1))*h;
        end
    end
    ME.PRegionTime = ME.PRegionTime + h;
    i = 1;
    for iPvOutput = ME.Model.PVoutputs
        ME.PRegion(i) = ME.PRegion(i) + ME.P(iPvOutput.RegionNodes(1))*h;
        ME.PRegionTime(i) = ME.PRegionTime(i) + h;
        i = i + 1;
    end
end

% Mass Change
ME.m(indn) = mnew(indn);

% Internal Energy Change
ME.u(indn) = unew(indn);

% Temperature
ME.T(indn) = Tnew(indn);
ME.T(sindn) = ME.T(sindn) + ...
    ME.dT_dU(sindn).*(h*Q(sindn) + Qtd(sindn))./ME.m(sindn);

% Environment
ME.m(lenn) = inf;
ME.u(lenn) = uEnv;
ME.T(lenn) = TEnv;

% Basic Boundary Work
ME.CycledE = ME.CycledE + h*(0.5*sum(ME.dV_dt.*(Pi + ME.P)));

% Parameters used by functions that are called after each
% ... angular increment.
if lenf ~= 0
    ME.Fc_V_averager = ME.Fc_V_averager + h*ME.Fc_V;
    ME.Fc_turb_averager = ME.Fc_turb_averager + h*ME.Fc_turb;
end
t = t + h;
if t >= ME.dt_max
    done = true;
    % Setup for Flow Loss Calculation
    ME.Fc_V = ME.Fc_V_averager / ME.dt_max;
    ME.Fc_turb = ME.Fc_turb_averager / ME.dt_max;
    for i = 1:length(ME.Regions)
        if isempty(ME.RegionLoops{i})
            faces = ME.ActiveRegionFcS{i};
            ME.Fc_RE(faces) = abs(ME.Fc_U(faces).*Fcrho(faces).*ME.Fc_Dh(faces)./Fcmu(faces));
            ME.Fc_RE(ME.Fc_RE==0) = 1e-7;
            ME.getWeight(faces);
            ME.KpU_2A(faces) = ME.KValue(faces).*Fcrho(faces).*ME.Fc_U(faces)./Areai(faces);
        end
    end
end

if ME.ss_condition || ~ME.continuetoSs
    % Get cycle time for averaging the effective conductance and
    % ... conductance temperatures
    ME.CycleTime = ME.CycleTime + ME.dt_max;

    % Calculate Effective Conduction for mixed faces
    ME.CondEff(indmf) = ME.CondEff(indmf) + ...
        ME.dt_max * ME.Fc_Cond(indmf);

    % Calculate the Effective Conduction * Temperature of mixed faces
    ME.CondTempEff(indmf) = ME.CondTempEff(indmf) + ...
        ME.dt_max * ME.Fc_Cond(indmf).* ME.T(ndlmf);

    % Test The reynold's number
    if isempty(ME.EffRE); ME.EffRE = zeros(size(indn)); end
    ME.EffRE = ME.EffRE + ...

```

```

%           (ME.dt_max/sum(ME.vol(indnminus))) * sum(ME.Dh(indnminus).*ME.vol(indnminus));
end

% Record statistics
if ME.Model.recordConductionFlux
    ME.CondFlux(:) = 0;
    for i = 1:3:length(ME.CondNet)-2
        ME.CondFlux(ME.CondNet{i+1}) = ...
            ME.CondFlux(ME.CondNet{i+1}) + ...
            abs(ME.CondNet{i}.*Qfc(ME.CondNet{i+2}));
    end
    if any(~isreal(Qtd(:)))
        fprintf('err');
    end
    ME.CondFlux = (ME.CondFlux + real(Qtd(:)))./ME.vol;
end
if RecordStatistics
    ME.E_ToEnvironment = ME.E_ToEnvironment + ...
        ME.dt_max*(sum(Qfc(facesES).*sgnES) + ...
        sum(sgnEG.*(Qfc(facesEG) + ...
        ME.Fc_V(facesEG).*rhoi(facesEG).*ui(facesEG))));
    ME.E_ToSource = ME.E_ToSource + ...
        ME.dt_max*sum(Qfc(facesSr).*sgnSr);
    ME.E_ToSink = ME.E_ToSink + ...
        ME.dt_max*sum(Qfc(facesSi).*sgnSi);
    if length(ME.VolMin) < length(ME.Regions)
        ME.VolMin = 100000*ones(size(ME.Regions));
        ME.VolMax = zeros(size(ME.Regions));
    end
    for i = 1:length(ME.Regions)
        if ~ME.isEnvironmentRegion(i)
            nodes = ME.Regions{i};
            Vol = sum(ME.vol(nodes));
            ME.VolMin(i) = min(ME.VolMin(i),Vol);
            ME.VolMax(i) = max(ME.VolMax(i),Vol);
        else
            ME.VolMin(i) = 0;
            ME.VolMax(i) = 0;
        end
    end
    % Qfc = ME.Fc_Cond(Cfcs).*(Ti(Cnd1) - Ti(Cnd2));
    TRatio = Ti(Cnd1)./Ti(Cnd2);
    TRatio(TRatio>1) = 1./TRatio(TRatio>1);
    ME.ExergyLossStatic = ME.ExergyLossStatic + ...
        ME.dt_max*sum(abs(Qfc(ME.StaticFaces).*(1-TRatio(ME.StaticFaces))));
    ME.ExergyLossShuttle = ME.ExergyLossShuttle + ...
        ME.dt_max*sum(abs(Qfc(ME.ShuttleFaces).*(1-TRatio(ME.ShuttleFaces))));
end
end
end
end

function solve_loops(ME,region,F2C,startrow,A,b,Fcrho,Fcmu,time)
persistent recordValues;
persistent recordTimes;
if isempty(recordValues)
    recordValues = cell(length(ME.Regions),1);
    recordTimes = recordValues;
    for ind1 = 1:length(ME.Regions)
        recordValues{ind1} = [];
        recordTimes{ind1} = [];
    end
end
end

% Loop Definitions
loop = ME.RegionLoops{region};
Ind = ME.RegionLoops_Ind{region};
Nloops = size(Ind, 2);

% UnCollapsed References
rows = startrow:startrow + Nloops-1;

```

```

% Predict Values at this time-step
if size(recordValues{region},2) == 3
    % Quadratically Extrapolate
    y0 = recordValues{region}(:,1);
    y1 = recordValues{region}(:,2);
    y2 = recordValues{region}(:,3);
    x0 = recordTimes{region}(1);
    x1 = recordTimes{region}(2);
    x2 = recordTimes{region}(3);
    prediction = ...
        y0*(((time-x1)*(time-x2)) / ((x0-x1)*(x0-x2))) + ...
        y1*(((time-x0)*(time-x2)) / ((x1-x0)*(x1-x2))) + ...
        y2*(((time-x0)*(time-x1)) / ((x2-x0)*(x2-x1)));
end

skipLoop = false(Nloops,1);
% Define loops that participate
for p = 1:Nloops
    if Ind(3, p) && ME.Fc_Area(Ind(3, p)) == 0
        % The Area has gone to 0, therefore the volume flow rate is 0
        A(rows(p), F2C(Ind(3, p))) = 1;
        skipLoop(p) = true;
    else
        % The entry in "A" will be a 1, to set the volume flow rate to the
        % ... value defined in "b". The Last entry of the loop is the
        % ... only one that cannot be a part of another loop.
        A(rows(p), F2C(loop(2,Ind(2,p)))) = 1;
        if size(recordValues,2) == 3
            b(rows(p)) = prediction(p);
        else
            b(rows(p)) = ME.Fc_V(loop(2, Ind(2, p)));
        end
    end
end

% Calculate inverse of matrix
Ainv = inv(A);

% Eliminate the rows that are not useful outputs
indl = 1:length(skipLoop);
rows(skipLoop) = [];
indl(skipLoop) = [];
x = Ainv*b;

% Initialize Solving Loop
iteration = 1;
max_iterations = 300;
fn = ones(length(indl),1);

tol = 1e-8;
if ~isempty(indl)

    % Newton's Method
    J = zeros(length(indl));

    while iteration < max_iterations

        for i = 1:length(indl)
            Sgni = loop(3,Ind(1,indl(i)):Ind(2,indl(i)))';
            Fcsi = loop(2,Ind(1,indl(i)):Ind(2,indl(i)));
            for j = 1:length(indl)
                DeltaV = Ainv(:,rows(j));
                if i == j
                    [dfi_dxj, fni] = getCost(...
                        ME,x(F2C(Fcsi)), Sgni, Fcsi, Fcmu(Fcsi), ...
                        Fcrho(Fcsi), DeltaV(F2C(Fcsi)));
                    J(i,j) = dfi_dxj;
                    fn(i) = fni;
                else
                    [dfi_dxj] = getCost(...

```

```

                ME,x(F2C(Fcsi)), Sgni, Fcsi, Fcmu(Fcsi), ...
                Fcrho(Fcsi), DeltaV(F2C(Fcsi)));
            J(i,j) = dfi_dxj;
        end
    end
end

% Test Convergence
if sum(abs(fn)) < tol; break; end

% x = x + J\(-f); - Calculate the shift in x
x = x + Ainv(:,rows)*(J\(-fn));

iteration = iteration + 1;
end
end

if iteration == max_iterations
    fprintf('XXX Failed to Converge 300 iterations. XXX\n');
else
    fprintf(['Converged in ' num2str(iteration) ' iterations. \n']);
end

% Record for extrapolation
if size(recordValues{region},2) == 3
    recordValues{region}(:,1:2) = recordValues{region}(:,2:3);
    recordTimes{region}(1:2) = recordTimes{region}(2:3);
    recordValues{region}(:,end) = x(rows(:));
    recordTimes{region}(end) = time;
else
    recordValues{region}(:,end+1) = x(rows(:));
    recordTimes{region}(end+1) = time;
end

ME.Fc_V(ME.RegionFcs{region}) = x(F2C(ME.RegionFcs{region}));
end

%{
function dP = p_drop(ME,fc)
    nd1 = ME.Fc_Nd(fc,1);
    nd2 = ME.Fc_Nd(fc,2);
    rho1 = ME.m(nd1)/ME.vol(nd1);
    rho2 = ME.m(nd2)/ME.vol(nd2);
    rho = (rho1 + rho2)/2;
    mu1 = ME.muFunc(ME.T(nd1));
    mu2 = ME.muFunc(ME.T(nd2));
    mu = (mu1 + mu2)/2;
    U = ME.Fc_V(fc)./ME.Fc_Area(fc) + ME.Fc_Vel_Factor(fc)*ME.dA;
    RE = abs(rho*U*ME.Fc_Dh(fc)./mu) + 1e-7;
    ME.getWeight(fc);
    [K,~] = ME.KValue(fc);
    dP = K*rho*abs(U)*U;
end
%}

function [derivative, cost] = getCost(ME, Vnew, Sgn, Fcs, Fcmu, Fcrho, DeltaV)
    ME.Fc_U(Fcs) = Vnew./ME.Fc_Area(Fcs) + ME.Fc_Vel_Factor(Fcs)*ME.dA;
    ME.Fc_RE(Fcs) = abs(ME.Fc_U(Fcs).*Fcrho.*ME.Fc_Dh(Fcs)./Fcmu) + 1e-7;
    ME.getWeight(Fcs);
    if nargin == 2
        [K, derv] = ME.KValue(Fcs);
        cost = sum(Sgn.*K.*Fcrho.*abs(ME.Fc_U(Fcs)).*ME.Fc_U(Fcs));
        derivative = sum(...
            (DeltaV.*Sgn.*Fcrho.*abs(ME.Fc_U(Fcs))./ME.Fc_Area(Fcs)).*(...
            sign(ME.Fc_U(Fcs)).*ME.Fc_RE(Fcs).*derv + 2*K));
    else
        [K, derv] = ME.KValue(Fcs);
        derivative = sum(...
            (DeltaV.*Sgn.*Fcrho.*abs(ME.Fc_U(Fcs))./ME.Fc_Area(Fcs)).*(...
            sign(ME.Fc_U(Fcs)).*ME.Fc_RE(Fcs).*derv + 2*K));
    return;
end

```

```

end
end

function Forces = Iteration_Solve(ME)
% Inc = next position, iterate up to this position, where dynamics
% ... are calculated

%% Step 2: Start Recursive Algorithm
ME.implicitSolve();
if ME.stop
    fprintf('Simulation Finished Prematurely. (In Iteration_Solve)\n');
end

%% Step 5: Collect Information for Dynamics, in the form of "Forces"
% Pressure Boundaries
Forces = ME.CalcForces();
end

function assignAvgDynamic(ME)
if ~isempty(ME.Dynamic)
    p = mean(ME.Dynamic);
    if ~isempty(ME.DynDh)
        ME.Dh(ME.DynDh(1,:)) = p(ME.DynDh(2,:));
    end
    % Dynamic Volume
    if ~isempty(ME.DynVol)
        ME.vol(ME.DynVol(1,:)) = p(ME.DynVol(2,:));
    end
    % Dynamic Area
    if ~isempty(ME.Fc_DynArea)
        ME.Fc_Area(ME.Fc_DynArea(1,:)) = p(ME.Fc_DynArea(2,:));
    end
    % Dynamic Conductance
    if ~isempty(ME.Fc_DynCond)
        ME.Fc_Cond(ME.Fc_DynCond(1,:)) = p(ME.Fc_DynCond(2,:));
    end
    % Dynamic Distance
    if ~isempty(ME.Fc_DynDist)
        ME.Fc_Dist(ME.Fc_DynDist(1,:)) = p(ME.Fc_DynDist(2,:));
    end
    if ~isempty(ME.Fc_DynCond_Dist)
        ME.Fc_Cond_Dist(ME.Fc_DynCond_Dist(1,:)) = ...
            p(ME.Fc_DynCond_Dist(2,:));
    end
    % Dynamic dx for Courant Calculation
    if ~isempty(ME.Fc_Dyndx)
        ME.Fc_dx(ME.Fc_Dyndx(1,:)) = p(ME.Fc_Dyndx(2,:));
    end
end
end

function Assign_Engine_Pressure(ME, P)
for i = 1:length(ME.Regions)
    if ~ME.isEnvironmentRegion(i)
        Nds = ME.Regions{i};
        ME.m(Nds) = P*(ME.vol(Nds)./ME.T(Nds))./ME.R(i);
    else
        Nds = ME.Regions{i};
        ME.m(Nds) = ME.P(end)*(ME.vol(Nds)./ME.T(Nds))./ME.R(i);
    end
end
end

function assignDynamic(ME,Inc,initialize)
persistent A;
persistent B;
persistent C;
persistent D;
persistent DynLength;
persistent IncBase;
if nargin == 3

```

```

if initialize == false
if ~isempty(ME.Dynamic)
% Define A,B,C,D for all variables
DynLength = size(ME.Dynamic,1);
IncBase = floor(Inc);
if IncBase >= 2
V0 = ME.Dynamic(IncBase - 1,:);
V1 = ME.Dynamic(IncBase,:);
if IncBase < DynLength - 1
V2 = ME.Dynamic(IncBase+1,:);
V3 = ME.Dynamic(IncBase+2,:);
elseif IncBase < DynLength
V2 = ME.Dynamic(IncBase+1,:);
V3 = ME.Dynamic(2,:);
else
V2 = ME.Dynamic(2,:);
V3 = ME.Dynamic(3,:);
end
else
V2 = ME.Dynamic(IncBase+1,:);
V3 = ME.Dynamic(IncBase+2,:);
if IncBase >= 1
V0 = ME.Dynamic(DynLength-1,:);
V1 = ME.Dynamic(1,:);
else
V0 = ME.Dynamic(DynLength-2,:);
V1 = ME.Dynamic(DynLength-1,:);
end
end
end
dV1 = (V2 - V0) / (2 * ME.A_Inc / ME.dA_old);
dV2 = (V3 - V1) / (2 * ME.A_Inc / ME.dA);
A = ME.dt_max * (dV1 + dV2) - 2 * (V2 - V1);
B = ME.dt_max * (-2*dV1 - dV2) + 3 * (V2 - V1);
C = ME.dt_max * dV1;
D = V1;
end
end
return;
end
if ~isempty(ME.Dynamic)
Inc_p = Inc-IncBase;
if Inc_p > 1; frac = 1; else; frac = Inc_p; end
point = A*frac^3 + B*frac^2 + C*frac + D;

% Dynamic Velocity Factor
if ~isempty(ME.Fc_DynVel_Factor)
ME.Fc_Vel_Factor(ME.Fc_DynVel_Factor(1,:)) = ...
point(ME.Fc_DynVel_Factor(2,:));
end
if ~isempty(ME.Fc_DynShear_Factor)
ME.Fc_Shear_Factor(ME.Fc_DynShear_Factor(1,:)) = ...
point(ME.Fc_DynShear_Factor(2,:));
end

point(point<0) = 0;
% Dynamic Area
if ~isempty(ME.Fc_DynArea)
ME.Fc_Area(ME.Fc_DynArea(1,:)) = point(ME.Fc_DynArea(2,:));
end
% Dynamic Conductance
if ~isempty(ME.Fc_DynCond)
ME.Fc_Cond(ME.Fc_DynCond(1,:)) = point(ME.Fc_DynCond(2,:));
end
% Dynamic Fc_A
if ~isempty(ME.Fc_DynA)
ME.Fc_A(ME.Fc_DynA(1,:)) = point(ME.Fc_DynA(2,:));
end
% Dynamic Fc_B
if ~isempty(ME.Fc_DynB)
ME.Fc_B(ME.Fc_DynB(1,:)) = point(ME.Fc_DynB(2,:));
end

```

```

end
% Dynamic Fc_C
if ~isempty(ME.Fc_DynC)
    ME.Fc_C(ME.Fc_DynC(1,:)) = point(ME.Fc_DynC(2,:));
end
% Dynamic Fc_D
if ~isempty(ME.Fc_DynD)
    ME.Fc_D(ME.Fc_DynD(1,:)) = point(ME.Fc_DynD(2,:));
end
if any(isnan(ME.Fc_A)) || any(isnan(ME.Fc_B)) || ...
    any(isnan(ME.Fc_C)) || any(isnan(ME.Fc_D))
    fprintf('err');
end

point(point<1e-8) = 1e-8;
% Dynamic Volume
if ~isempty(ME.DynVol)
    ME.vol(ME.DynVol(1,:)) = point(ME.DynVol(2,:));
    ME.dV_dt(ME.DynVol(1,:)) = (1/ME.dt_max)*(...
        3*A(ME.DynVol(2,:))*frac^2 + ...
        2*B(ME.DynVol(2,:))*frac + ...
        C(ME.DynVol(2,:)));
end
% Dynamic Active Times for ShearContacts
if ~isempty(ME.SC_Active)
    ME.Shear_Contact(6,ME.SC_Active(1,:)) = ...
        round(point(ME.SC_Active(2,:)));
end
% Dynamic K12
if ~isempty(ME.Fc_DynK12)
    ME.Fc_K12(ME.Fc_DynK12(1,:)) = point(ME.Fc_DynK12(2,:));
end
if ~isempty(ME.Fc_DynK21)
    ME.Fc_K21(ME.Fc_DynK21(1,:)) = point(ME.Fc_DynK21(2,:));
end

point(point<1e-4) = 1e-4;
% Dynamic Dh (node)
if ~isempty(ME.DynDh)
    ME.Dh(ME.DynDh(1,:)) = point(ME.DynDh(2,:));
end
% Dynamic Dh (face)
if ~isempty(ME.Fc_DynDh)
    ME.Fc_Dh(ME.Fc_DynDh(1,:)) = point(ME.Fc_DynDh(2,:));
end
% Dynamic Distance
if ~isempty(ME.Fc_DynDist)
    ME.Fc_Dist(ME.Fc_DynDist(1,:)) = point(ME.Fc_DynDist(2,:));
end
if ~isempty(ME.Fc_DynCond_Dist)
    ME.Fc_Cond_Dist(ME.Fc_DynCond_Dist(1,:)) = ...
        point(ME.Fc_DynCond_Dist(2,:));
    if any(ME.Fc_Cond_Dist==0)
        ME.Fc_Cond_Dist(ME.Fc_Cond_Dist==0) = 0.0001;
    end
end
% Dynamic dx for Courant Calculation
if ~isempty(ME.Fc_Dyndx)
    ME.Fc_dx(ME.Fc_Dyndx(1,:)) = point(ME.Fc_Dyndx(2,:));
end
end
end

function Forces = CalcForces(ME)
    % Distribute pressure losses

    fcs = 1:length(ME.Fc_U);
    nds = 1:length(ME.P);
    nd1 = ME.Fc_Nd(fcs,1);
    nd2 = ME.Fc_Nd(fcs,2);
    rhoi = ME.m(nds)./ME.vol(nds);

```



```

rhoi(end) = ME.rho(end);
Fcrho = 0.5*(rhoi(nd1) + rhoi(nd2));
for i = 1:length(ME.Regions)
    nodes = ME.Regions{i};
    if ~isempty(ME.ActiveRegionFcs{i})
        faces = ME.ActiveRegionFcs{i};
        % Calculate KpU_2A
        ME.Fc_U(faces) = ME.Fc_V(faces)./ME.Fc_Area(faces) - ...
            ME.Fc_Vel_Factor(faces)*ME.dA;
        ME.Fc_RE(faces) = ...
            abs(2*ME.Fc_U(faces).*Fcrho(faces).*ME.Fc_Dh(faces)./...
                (ME.mu(nd1(faces)) + ME.mu(nd2(faces))));
        ME.Fc_RE(ME.Fc_RE==0) = 1e-7;
        ME.getWeight();
        len = length(faces) + 1;
        A = ME.A_Press{i};
        b = zeros(len,1);
        A(len,:) = ME.vol(nodes);
        if ME.isEnvironmentRegion(i)
            b(len) = ME.P(end);
            A(len,len) = 1e8; % Some large value that is not infinity
        else
            b(len) = ME.R(i)*sum(ME.vol(nodes))*...
                ME.T(nodes(1)).*ME.m(nodes(1))./ME.vol(nodes(1));
        end
        ME.Fc_dP(faces) = ME.KValue(faces).*Fcrho(faces).*...
            abs(ME.Fc_U(faces)).*ME.Fc_U(faces);
        b(1:len-1) = ME.Fc_dP(faces);
        A = sparse(A);
        ME.P(nodes) = A\b;
        ME.dP(nodes) = ME.P(nodes) - ME.R(i)*ME.T(nodes(1))*ME.m(nodes(1))./ME.vol(nodes(1));
    else
        if ~ME.isEnvironmentRegion(i)
            ME.P(nodes) = ...
                ME.R(i)*ME.T(nodes(1)).*ME.m(nodes(1))./ME.vol(nodes(1));
        end
        ME.dP(nodes) = 0;
    end
end

end
ME.E_Flow_Loss = ME.E_Flow_Loss + ...
    ME.dt_max * sum(abs(ME.Fc_V(fcs)).*(ME.P(nd1)-ME.P(nd2))));

% Make forces
if ~isempty(ME.MechanicalSystem)
    Forces = cell(1,length(ME.MechanicalSystem.Converters));
else
    Forces = cell(0);
end
if ~isempty(Forces)
    for i = 1:length(Forces)
        Forces{i} = zeros(1,length(ME.MechanicalSystem.Converters(i).Stroke));
    end
    for i = 1:size(ME.Press_Contact,2)
        conv = ME.Press_Contact(1,i);
        subconv = ME.Press_Contact(2,i);
        area = ME.Press_Contact(3,i);
        index = ME.Press_Contact(4,i);
        Forces{conv}(subconv) = Forces{conv}(subconv) + area.*ME.P(index);
    end
    for i = 1:size(ME.Shear_Contact,2)
        if ME.Shear_Contact(6,i)
            conv = ME.Shear_Contact(1,i);
            subconv = ME.Shear_Contact(2,i);
            area = ME.Shear_Contact(3,i);
            ind1 = ME.Shear_Contact(4,i);
            ind2 = ME.Shear_Contact(5,i);
            Forces{conv}(subconv) = Forces{conv}(subconv) + ...
                area*(ME.P(ind1)-ME.P(ind2));
        end
    end
end
end

```

```

end
end

function getWeight(ME,faces)
    if nargin == 1; faces = 1:length(ME.Fc_U); end
    W = zeros(length(faces),1);

    % Ignore turbulence transport
    ind = ~ME.useTurbulenceFc(faces);
    W(ind) = (ME.Fc_RE(faces(ind))-2300)/1700;
    W(W>1) = 1;
    W(W<0) = 0;
    W = W.*W.*(3 - 2*W);

    % Use turbulence transport
    ind = ~ind;
    W(ind) = ME.Fc_turb(faces(ind));
    W(W>1) = 1;
    W(W<0) = 0;
    ME.Fc_W(faces) = W;
end

function [K, derv] = KValue(ME,faces)
    K = zeros(length(faces),1);
    if nargin == 2
        derv = K;
    end
    for i = 1:length(faces)
        fc = faces(i);
        if ME.Fc_K12(fc) > 0
            if ME.Fc_V(fc) > 0
                K(i) = ME.Fc_K12(fc)/2;
            else
                K(i) = ME.Fc_K21(fc)/2;
            end
        else
            if ME.Fc_W(fc) == 0
                if nargin == 2
                    K(i) = ME.Fc_fFunc_l{fc}(ME.Fc_RE(fc))*ME.Fc_Dist(fc)/(ME.Fc_Dh(fc)*2);
                    derv(i) = (ME.Fc_fFunc_l{fc}(ME.Fc_RE(fc) + 1e-8)*ME.Fc_Dist(fc)/(ME.Fc_Dh(fc)*2) -
- ...
                        K(i))/1e-8;
                else
                    K(i) = ME.Fc_fFunc_l{fc}(ME.Fc_RE(fc))*ME.Fc_Dist(fc)/(ME.Fc_Dh(fc)*2);
                end
            elseif ME.Fc_W(fc) == 1
                K(i) = ME.Fc_fFunc_t{fc}(ME.Fc_RE(fc))*ME.Fc_Dist(fc)/(ME.Fc_Dh(fc)*2);
                derv(i) = (ME.Fc_fFunc_t{fc}(ME.Fc_RE(fc) + 1e-8)*ME.Fc_Dist(fc)/(ME.Fc_Dh(fc)*2) -
...
                        K(i))/1e-8;
            else
                K(i) = ((1-ME.Fc_W(fc))*ME.Fc_fFunc_l{fc}(ME.Fc_RE(fc)) + ...
                    ME.Fc_W(fc)*ME.Fc_fFunc_t{fc}(ME.Fc_RE(fc)))*ME.Fc_Dist(fc)/(ME.Fc_Dh(fc)*2);
                derv(i) = (((1-ME.Fc_W(fc))*ME.Fc_fFunc_l{fc}(ME.Fc_RE(fc) + 1e-8) + ...
                    ME.Fc_W(fc)*ME.Fc_fFunc_t{fc}(ME.Fc_RE(fc) + 1e-8))*ME.Fc_Dist(fc)/(ME.Fc_Dh(fc)*2) -
- ...
                        K(i))/1e-8;
            end
        end
    end
end
end

function Nk = NkFunc(ME)
    Nk = zeros(size(ME.Fc_U));
    Nkt = Nk;
    % Laminar Functions
    for i = 1:length(ME.Fc_NkFunc_l)
        fcs = ME.Fc_NkFunc_l_el{i};
        if nargin(ME.Fc_NkFunc_l{i}) == 1
            Nk(fcs) = (1-ME.Fc_W(fcs)).*ME.Fc_NkFunc_l{i}(ME.Fc_RE(fcs));
        else

```

```

        Nk(fcs) = (1-ME.Fc_W(fcs)).*...
            ME.Fc_NkFunc_l{i}(ME.Fc_RE(fcs),ME.Fc_PR(fcs));
    end
end
% Turbulent Functions
for i = 1:length(ME.Fc_NkFunc_t)
    fcs = ME.Fc_NkFunc_t_el{i};
    if nargin(ME.Fc_NkFunc_t{i}) == 1
        Nkt(fcs) = ME.Fc_W(fcs).*ME.Fc_NkFunc_t{i}(ME.Fc_RE(fcs));
    else
        Nkt(fcs) = ME.Fc_W(fcs).*...
            ME.Fc_NkFunc_t{i}(ME.Fc_RE(fcs),ME.Fc_PR(fcs));
    end
end
Nk = Nk + Nkt;
Nk(Nk<1) = 1; % Nothing can be worse than pure conduction
end

function Nu = Nusselt(ME)
    Nu = zeros(length(ME.P)-1,1);
    Nut = Nu;
    ME.RE = abs(ME.RE);
    W = (ME.RE-2300)/1700;
    W(W<0) = 0; W(W>1) = 1;
    W = W.*(W.*(3 - 2*W) - 1);
    W(ME.useTurbulenceNd) = ME.turb(ME.useTurbulenceNd);
    W(W<0) = 0; W(W>1) = 1;
    % Laminar Functions
    for i = 1:length(ME.NuFunc_l)
        nds = ME.NuFunc_l_el{i};
        if nargin(ME.NuFunc_l{i}) == 1
            Nu(nds) = (1-W(nds)).*ME.NuFunc_l{i}(ME.RE(nds));
        else
            Nu(nds) = (1-W(nds)).*ME.NuFunc_l{i}(ME.RE(nds),ME.PR(nds));
        end
    end
end
% Turbulent Function
for i = 1:length(ME.NuFunc_t)
    nds = ME.NuFunc_t_el{i};
    if nargin(ME.NuFunc_t{i}) == 1
        Nut(nds) = W(nds).*ME.NuFunc_t{i}(ME.RE(nds));
    else
        Nut(nds) = W(nds).*ME.NuFunc_t{i}(ME.RE(nds),ME.PR(nds));
    end
end
Nu = Nu + Nut;
% Nu(Nu<1) = 1;% Pure Conduction Nusselt Number
end

end

end

```

## Result

The result is a class that includes the following functionality:

A function for animating colored node plots. Input properties are produced in the model.run function.

A function for animating velocity or pressure drop plots, centered on faces.

A function for capturing a snapshot.

```
classdef Result < handle
    %UNTITLED Summary of this class goes here
    % Detailed explanation goes here

    properties
        % For Display
        Model Model;

        XDATA
        YDATA
        Cmap
        Data
        % Data.t
        % Data.T
        % Data.P
        % Data.rho
        % Data.U

        OriginalAxes;
        Fig;
        Axes;
        GraphicsObjects;
    end

    methods

        function this = Result()
            end

        % Plot
        function animateNode(this,propertyname,cornerpnts,bodypnts,frequency,~,~,input_title)
            %% Currently this is restricted to constructs lying on the vertical axis
            if isfield(this.Data,propertyname)
                data = this.Data.(propertyname);

                start = 1;
                % With Each column in data
                % Get coordinates

                h = figure();
                try
                    set(h,'color','w');
                    a = gca;
                    axis tight manual;
                    a.XLim = oldaxes.XLim;
                    a.YLim = oldaxes.YLim;
                    a.XAxis.TickLabelFormat = '%.2f';
                    a.YAxis.TickLabelFormat = '%.2f';
                    switch propertyname
                        case 'T'; colorLabel = 'Temperature (K)';
                        case 'P'; colorLabel = 'Pressure (Pa)';
                        case 'dP'; colorLabel = 'Pressure Buildup (Pa)';
                    end
                catch
                    % Error handling
                end
            end
        end
    end
end
```

```

        case 'turb'; colorLabel = 'Proportion of Fully Turbulence (0-1)';
        case 'cond'
            colorLabel = 'Natural Logarithm of Sum of absolute Power Exchange per unit Volume
(ln(W/m^3))';
            data = log(data+1);
        end
        xlabel('X (m)');
        ylabel('Y (m)');
        timepnts = this.Data.t;

        if nargin > 7
            if ~isempty(this.Model.outputPath)
                filename = [this.Model.outputPath '\' input_title '_Animated ' propertyname
'.gif'];
            else
                filename = [input_title '_Animated ' propertyname '.gif'];
            end
        else
            if ~isempty(this.Model.outputPath)
                filename = [this.Model.outputPath '\' this.Model.name '_Animated ' propertyname
'.gif'];
            else
                filename = [this.Model.name '_Animated ' propertyname '.gif'];
            end
        end
        cmap = jet(100);
        colormap(cmap);
        data(data==0) = NaN();
        vals = linspace(min(min(data(start:end,:))),max(max(data(start:end,:))),7);
        mapper =
linspace(min(min(data(start:end,:))),max(max(data(start:end,:))),size(cmap,1));
        if vals(1) == vals(end)
            fprintf('ERR: minimum == maximum (in result.animateNode) \n');
            fprintf(['... Error ocured with property: ' propertyname '\n']);
            try
                close(h);
            catch
            end
            return;
        end
        if isnan(vals(1)) || isnan(vals(end))
            try
                close(h);
            catch
            end
            return;
        end
        caxis([vals(1) vals(end)]);

        if mapper(1) ~= mapper(2)
            data(isnan(data)) = 0;
            N_XData = zeros(4,size(data,1));
            N_YData = N_XData;
            B_XData = zeros(4,length(bodypnts));
            B_YData = B_XData;
            TextHandle = text(a.XLim(1)+0.01*(a.XLim(2)-a.XLim(1)),...
                a.YLim(2)-0.05*(a.YLim(2)-a.YLim(1)), '');
            hcb = colorbar('Ticks',vals,'Limits',[vals(1) vals(end)]);
            ylabel(hcb, colorLabel);
            yt=get(hcb,'Ticks');
            switch propertyname
                case 'T'
                    set(hcb,'XTickLabel',strtrim(cellstr(num2str(yt,'%1f'))));
                case 'P'
                    set(hcb,'XTickLabel',strtrim(cellstr(num2str(yt,'%2e'))));
                case 'turb'
                    set(hcb,'XTickLabel',strtrim(cellstr(num2str(yt,'%2f'))));
                case 'cond'
                    set(hcb,'XTickLabel',strtrim(cellstr(num2str(yt,'%2f'))));
                case 'dP'
                    set(hcb,'XTickLabel',strtrim(cellstr(num2str(yt,'%1e'))));
            end
        end
    end
end

```

```

end
C = zeros(size(data,1),1);
%C = zeros(size(data,1),1,3);
for i = 1:size(data,2)-1
    angindex = 1+mod(i,Frame.NTheta-1);
    if i == 1
        proceed = true;
        FirstTime = true;
    else
        FirstTime = false;
        t = t + timepnts(i) - timepnts(i-1);
        if t > frequency
            proceed = true;
        else
            proceed = false;
        end
    end
end
if proceed
    if true || ~any(data(:,i)==0) || strcmp(propertyname,'turb')
        t = 0;
        n = 1;
        if FirstTime
            for item = start:size(data,1)
                %C(n,1,:) = interp1(mapper,cmap,data(item,i));
                C(n) = data(item,i);
                switch cornerpnts{item}(1,1)
                    case 1 % Static Position
                        % Cut off the first column
                        p = cornerpnts{item}(:,2:5);
                        N_XData(:,n) = [p(1,4) - p(1,1); p(1,4) + p(1,2); ...
                            p(1,4) + p(1,1); p(1,4) - p(1,2)];
                        N_YData(:,n) = [p(2,4) - p(2,1); p(2,4) + p(2,2); ...
                            p(2,4) + p(2,1); p(2,4) - p(2,2)];
                        if all(p(:,3) == 0)
                            % Origin Centered
                            n = n + 1;
                        else
                            % Ring Shaped
                            N_XData(:,n+1) = N_XData(:,n) + p(1,3);
                            N_YData(:,n+1) = N_YData(:,n) + p(2,3);
                            C(n+1) = C(n);
                            n = n + 2;
                        end
                    case 2 % One Dimension Stretch
                        % Cut off the first column
                        temp = 4+angindex;
                        p = [cornerpnts{item}(:,2:4) ...
                            cornerpnts{item}(:,temp)];
                        N_XData(:,n) = [p(1,1); ...
                            p(1,1) + p(1,2); ...
                            p(1,1) + p(1,2) + p(1,4); ...
                            p(1,1) + p(1,4)];
                        N_YData(:,n) = [p(2,1); p(2,1) + p(2,2); ...
                            p(2,1) + p(2,2) + p(2,4); ...
                            p(2,1) + p(2,4)];
                        if all(p(:,3) == 0)
                            % Origin Centered
                            n = n + 1;
                        else
                            % Ring Shaped
                            N_XData(:,n+1) = N_XData(:,n) + p(1,3);
                            N_YData(:,n+1) = N_YData(:,n) + p(2,3);
                            C(n+1) = C(n);
                            n = n + 2;
                        end
                    case 3 % Two Dimension Stretch
                        temp = 2+angindex;
                        p = [cornerpnts{item}(1:2,temp) ...
                            cornerpnts{item}(1:2,2) ...
                            cornerpnts{item}(3:4,2) ...
                            cornerpnts{item}(3:4,temp)];
                end
            end
        end
    end
end

```

```

N_XData(:,n) = [p(1,1); p(1,1) + p(1,2); ...
               p(1,1) + p(1,2) + p(1,4); ...
               p(1,1) + p(1,4)];
N_YData(:,n) = [p(2,1); p(2,1) + p(2,2); ...
               p(2,1) + p(2,2) + p(2,4); ...
               p(2,1) + p(2,4)];
if all(p(:,3) == 0)
    % Origin Centered
    n = n + 1;
else
    % Ring Shaped
    N_XData(:,n+1) = N_XData(:,n) + p(1,3);
    N_YData(:,n+1) = N_YData(:,n) + p(2,3);
    C(n+1) = C(n);
    n = n + 2;
end
case 4 % Translation
temp = 4+angindex;
p = [cornerpnts{item}(:,2:4) ...
     cornerpnts{item}(:,temp)];
N_XData(:,n) = [p(1,4) - p(1,1); p(1,4) + p(1,2); ...
               p(1,4) + p(1,1); p(1,4) - p(1,2)];
N_YData(:,n) = [p(2,4) - p(2,1); p(2,4) + p(2,2); ...
               p(2,4) + p(2,1); p(2,4) - p(2,2)];
if all(p(:,3) == 0)
    % Origin Centered
    n = n + 1;
else
    % Ring Shaped
    N_XData(:,n+1) = N_XData(:,n) + p(1,3);
    N_YData(:,n+1) = N_YData(:,n) + p(2,3);
    C(n+1) = C(n);
    n = n + 2;
end
end
end
for b = 1:length(bodypnts)
    if ndims(bodypnts{b}) == 2 %ok<ISMAT>
        B_XData(:,b) = bodypnts{b}(1,:);
        B_YData(:,b) = bodypnts{b}(2,:);
    else
        B_XData(:,b) = bodypnts{b}(1,:,angindex);
        B_YData(:,b) = bodypnts{b}(2,:,angindex);
    end
end
C(n+1:end) = [];
%C(n+1:end,1,:) = zeros(0,1,3);
N_XData(:,n+1:end) = zeros(4,0);
N_YData(:,n+1:end) = zeros(4,0);
PatchHandle = patch(N_XData,N_YData,real(C),'LineStyle','none');
PatchHandleBodies =
patch(B_XData,B_YData,zeros(length(bodypnts),1),'EdgeColor','k',...
      'FaceColor','none','LineWidth',1);
else
    for item = start:size(data,1)
        C(n) = data(item,i);
        switch cornerpnts{item}(1,1)
            case 1 % Static Position
                if all(cornerpnts{item}(:,4) == 0)
                    % Origin Centered
                    n = n + 1;
                else
                    % Ring Shaped
                    C(n+1) = C(n);
                    n = n + 2;
                end
            case 2 % One Dimension Stretch
                % Cut off the first column
                temp = 5+mod(i,Frame.NTheta-1);
                p = [cornerpnts{item}(:,2:4) ...
                    cornerpnts{item}(:,temp)];

```

```

N_XData(3:4,n) = [p(1,1) + p(1,2) + p(1,4); ...
                 p(1,1) + p(1,4)];
N_YData(3:4,n) = [p(2,1) + p(2,2) + p(2,4); ...
                 p(2,1) + p(2,4)];
if all(p(:,3) == 0)
    % Origin Centered
    n = n + 1;
else
    % Ring Shaped
    N_XData(:,n+1) = N_XData(:,n) + p(1,3);
    N_YData(:,n+1) = N_YData(:,n) + p(2,3);
    C(n+1) = C(n);
    n = n + 2;
end
case 3 % Two Dimension Stretch
temp = 3+mod(i,Frame.NTheta-1);
p = [cornerpnts{item}(1:2,temp) ...
     cornerpnts{item}(1:2,2) ...
     cornerpnts{item}(3:4,2) ...
     cornerpnts{item}(3:4,temp)];
N_XData(:,n) = [p(1,1); p(1,1) + p(1,2); ...
                p(1,1) + p(1,2) + p(1,4); ...
                p(1,1) + p(1,4)];
N_YData(:,n) = [p(2,1); p(2,1) + p(2,2); ...
                p(2,1) + p(2,2) + p(2,4); ...
                p(2,1) + p(2,4)];
if all(p(:,3) == 0)
    % Origin Centered
    n = n + 1;
else
    % Ring Shaped
    N_XData(:,n+1) = N_XData(:,n) + p(1,3);
    N_YData(:,n+1) = N_YData(:,n) + p(2,3);
    C(n+1) = C(n);
    n = n + 2;
end
case 4 % Translation
temp = 5+mod(i,Frame.NTheta-1);
p = [cornerpnts{item}(:,2:4) ...
     cornerpnts{item}(:,temp)];
N_XData(:,n) = [p(1,4) - p(1,1); p(1,4) + p(1,2); ...
                p(1,4) + p(1,1); p(1,4) - p(1,2)];
N_YData(:,n) = [p(2,4) - p(2,1); p(2,4) + p(2,2); ...
                p(2,4) + p(2,1); p(2,4) - p(2,2)];
if all(p(:,3) == 0)
    % Origin Centered
    n = n + 1;
else
    % Ring Shaped
    N_XData(:,n+1) = N_XData(:,n) + p(1,3);
    N_YData(:,n+1) = N_YData(:,n) + p(2,3);
    C(n+1) = C(n);
    n = n + 2;
end
end
end
for b = 1:length(bodypnts)
    if ndims(bodypnts{b}) == 2 %#ok<ISMAT>
        %B_XData(:,b) = bodypnts{b}(1,:);
        %B_YData(:,b) = bodypnts{b}(2,:);
    else
        temp = 1 + mod(i,Frame.NTheta-1);
        B_XData(:,b) = bodypnts{b}(1,:,temp);
        B_YData(:,b) = bodypnts{b}(2,:,temp);
    end
end
end
set(PatchHandle,'XData',N_XData);
set(PatchHandle,'YData',N_YData);
set(PatchHandleBodies,'XData',B_XData);
set(PatchHandleBodies,'YData',B_YData);

```



```

set(PatchHandle,'CData',real(C));
set(TextHandle,'String',[num2str(round(timepnts(i),2)) ' seconds']);
drawnow;

% Capture the plot as an image
try
    frame = getframe(h);
catch
    return;
end
im = frame2im(frame);
[imind,cm] = rgb2ind(im,256);

% Write to the GIF File
try
    if i == 1
        imwrite(imind,cm,filename,'gif','Loopcount',inf,'DelayTime',0);
    else
        imwrite(imind,cm,filename,'gif','WriteMode','append','DelayTime',0);
    end
catch
    fprintf('XXX GIF write error XXX\n');
    fprintf(['... propertyname = ' propertyname 'XXX\n']);
end
else
    fprintf('XXX: Not ~any(data(:,i)==0) || strcmp(propertyname,"turb") in
Result.AnimateNode\n');
    fprintf('... Error occured in the animation generation, where it is expected
that ... \n');
    fprintf('... properties, other than Turbulence should not have a value of 0.
\n');
    fprintf(['... propertyname = ' propertyname 'XXX\n']);
    break;
end
end
end
end
catch
end
try
    close(h);
catch
end
end
end
end

```

```

function animateFace(this,propertyname,cornerpnts,bodypnts,frequency,~,~,input_title)
%% Currently this is restricted to constructs lying on the vertical axis
if isfield(this.Data,propertyname)
    data = this.Data.(propertyname);
    start = 1;
    % With Each column in data
    % Get coordinates

    h = figure();
    try
        set(h,'color','w');
        a = gca;
        axis tight manual;
        a.XLim = oldaxes.XLim;
        a.YLim = oldaxes.YLim;
        xlabel('X (m)');
        ylabel('Y (m)');
        timepnts = this.Data.t;

        if nargin > 7
            if ~isempty(this.Model.outputPath)
                filename = [this.Model.outputPath '\\' input_title '_Animated ' propertyname
'.gif'];
            else
                filename = [input_title '_Animated ' propertyname '.gif'];
            end
        end
    end
end

```

```

        end
    else
        if ~isempty(this.Model.outputPath)
            filename = [this.Model.outputPath '\\' this.Model.name '_Animated ' propertyname
'.gif'];
        else
            filename = [this.Model.name '_Animated ' propertyname '.gif'];
        end
    end
end
minimum = 0;
var = abs(data(start:end,:));
maximum = prctile(var(:),100);
%vals = linspace(min(min(data(start:end,:)),max(max(data(start:end,:))),7);
%mapper =
linspace(min(min(data(start:end,:)),max(max(data(start:end,:))),size(cmap,1));
if minimum == maximum
    fprintf('ERR: minimum == maximum (in result.animateFace) \n');
    fprintf(['... Error ocured with property: ' propertyname '\n']);
    try
        close(h);
    catch
    end
    return;
end

data(isnan(data)) = 0;
F_XData = zeros(1,size(data,1));
F_YData = F_XData;
F_UxData = F_XData;
F_UyData = F_XData;
B_XData = zeros(4,length(bodypnts));
B_YData = B_XData;
TextHandle = text(a.XLim(1)+0.01*(a.XLim(2)-a.XLim(1)),...
    a.YLim(2)-0.05*(a.YLim(2)-a.YLim(1)),');
hold on;
switch propertyname
    case 'U'
        base_size = 0.1;
    case 'dP'
        base_size = 20;
end
for i = 1:size(data,2)-1
    angindex = 1+mod(i,Frame.NTheta-1);
    if i == 1; proceed = true; FirstTime = true;
    else
        FirstTime = false;
        t = t + timepnts(i) - timepnts(i-1);
        if t > frequency; proceed = true;
        else; proceed = false;
        end
    end
end
if proceed
    t = 0;
    if FirstTime
        for item = start:size(data,1)
            value = data(item,i)*base_size/maximum;
            p = cornerpnts{item};
            if size(p,2) < 3
                % It is a static face
                F_UxData(item) = p(1,1)*value;
                F_UyData(item) = p(2,1)*value;
                F_XData(item) = p(1,2);
                F_YData(item) = p(2,2);
            else
                % It is a dynamic face
                F_UxData(item) = p(1,1)*value;
                F_UyData(item) = p(2,1)*value;
                F_XData(item) = p(1,1+angindex);
                F_YData(item) = p(2,1+angindex);
            end
        end
    end
end
end
end

```

```

for b = 1:length(bodypnts)
    if ndims(bodypnts{b}) == 2 %#ok<ISMAT>
        B_XData(:,b) = bodypnts{b}(1,:);
        B_YData(:,b) = bodypnts{b}(2,:);
    else
        B_XData(:,b) = bodypnts{b}(1,:,angindex);
        B_YData(:,b) = bodypnts{b}(2,:,angindex);
    end
end
switch propertyname
    case 'U'
        QuiverHandle = quiver(...
            F_XData,F_YData,...
            F_UxData,F_UyData,...
            'Color','k');
    case 'dP'
        if exist('QuiverHandle','var')

            end
            for x = 1:length(F_XData)
                QuiverHandle(x) = plot(...
                    F_XData(x),...
                    F_YData(x),...
                    'Marker','o',...
                    'MarkerEdgeColor','b',...
                    'MarkerFaceColor','b',...
                    'MarkerSize',sqrt(F_UxData(x)^2 + F_UyData(x)^2)+1e-8,...
                    'LineStyle','none');
            end
        end
    PatchHandleBodies = patch(...
        B_XData,B_YData,zeros(length(bodypnts),1),...
        'EdgeColor','k',...
        'FaceColor','none',...
        'LineWidth',1);
else
    for item = start:size(data,1)
        value = data(item,i)*base_size/maximum;
        p = cornerpnts{item};
        if size(p,2) < 3
            % It is a static face
            F_UxData(item) = p(1,1)*value;
            F_UyData(item) = p(2,1)*value;
        else
            % It is a dynamic face
            F_UxData(item) = p(1,1)*value;
            F_UyData(item) = p(2,1)*value;
            F_XData(item) = p(1,1+angindex);
            F_YData(item) = p(2,1+angindex);
        end
    end
end
for b = 1:length(bodypnts)
    if ndims(bodypnts{b}) == 2 %#ok<ISMAT>
        %B_XData(:,b) = bodypnts{b}(1,:);
        %B_YData(:,b) = bodypnts{b}(2,:);
    else
        temp = 1 + mod(i,Frame.NTheta-1);
        B_XData(:,b) = bodypnts{b}(1,:,temp);
        B_YData(:,b) = bodypnts{b}(2,:,temp);
    end
end
end
switch propertyname
    case 'U'
        set(QuiverHandle,'XData',F_XData);
        set(QuiverHandle,'YData',F_YData);
        set(QuiverHandle,'UData',F_UxData);
        set(QuiverHandle,'VData',F_UyData);
    case 'dP'
        for x = 1:length(F_XData)
            set(QuiverHandle(x),'XData',F_XData(x));

```

```

        set(QuiverHandle(x),'YData',F_YData(x));
        set(QuiverHandle(x),'MarkerSize',...
            sqrt(F_UxData(x)^2 + F_UyData(x)^2)+1e-8);
    end
end
set(PatchHandleBodies,'XData',B_XData);
set(PatchHandleBodies,'YData',B_YData);
set(TextHandle,'String',[num2str(round(timepnts(i),2)) ' seconds']);
drawnow;

% Capture the plot as an image
try
    frame = getframe(h);
catch
    fprintf('ERR: Failed to Get Frame in Result.AnimateFace \n')
    fprintf(['... Error ocured with property: ' ...
        propertyname '\n']);
    try
        close(h);
    catch
        end
    return;
end
im = frame2im(frame);
[imind,cm] = rgb2ind(im,256);

% Write to the GIF File
try
    if i == 1
        imwrite(imind,cm,filename,'gif','Loopcount',inf,'DelayTime',0);
    else
        imwrite(imind,cm,filename,'gif','WriteMode','append','DelayTime',0);
    end
catch
    fprintf('XXX GIF write error XXX\n');
    fprintf(['... propertyname = ' propertyname 'XXX\n']);
end
end
end
catch
end
try
    close(h);
catch
end
end
end

function Start2DPlot(this,property,t)
if isempty(this.Axes)
    this.OriginalAxes = gca;
else
    this.Close2DPlot();
end
this.Fig = figure();
this.Axes = gca;
this.GraphicsObjects(length(XData,1)) = patch();
this.Cmap = colormap();
differences = abs(this.Data.t - t);
[value1, index1] = min(differences);
[value2, index2] = min(differences ~= value1);
if index1 < index2
    index2 = index2 + 1;
end
scalar = abs(value1/(value1-value2))*this.Data.(property)(index2,:) + abs(value2/(value1-
value2))*this.Data.(property)(index1,:);
ulimit = max(scalar);
llimit = min(scalar);
mapper = linspace(llimit,ulimit,size(cmap,1));

%% LOOP

```

```

for i = 1:length(this.XDATA,1)
    rgb = interp1(mapper,cmap,scalar(i));
    this.GraphicsObjects(i) = fill(this.XDATA(i,:),this.YDATA(i,:),rgb);
end
end

function Close2DPlot(this)
    % Delete the current figure
    close(this.Fig);
    delete(this.Fig);
    delete(this.Axes);
end

function getSnapshot(this,Model,name)
    if isempty(name); return; end
    if ~isfield(this.Data,'T'); return; end
    % Find Snapshot position
    N = Frame.NTheta-1;
    LEN = size(this.Data.T,2);
    if LEN == N
        ind = LEN;
    else
        if mod(LEN,N) == 0
            ind = LEN;
        else
            ind = LEN - mod(LEN,N) + 1;
        end
    end
end

while all(this.Data.T(:,ind) == 0)
    ind = ind - 1;
    if ind == 0
        return;
    end
end

% Define number of cells
n = 0;
for iGroup = Model.Groups
    n = n + length(iGroup.Bodies);
end
BData(n) = BodyData();
Snapshot = struct('Name',name,'Data',BData);
index = 1;
for iGroup = Model.Groups
    for iBody = iGroup.Bodies
        % Get a new ID for bodies without one
        if iBody.ID == 0; iBody.ID = Model.getBodyID(); end

        %% Create XData and YData vectors
        XData = zeros(length(iBody.Nodes),1);
        YData = XData;
        AltXData = XData;
        AltYData = YData;

        %% Get X & Y Data
        i = 1;
        Alti = 1;
        for k = 1:length(iBody.Nodes)
            Nd = iBody.Nodes(k);
            if isfield(Nd.data,'matl') && ...
                Nd.data.matl.Phase ~= iBody.matl.Phase
                AltXData(Alti) = (Nd.xmin + Nd.xmax)/2;
                Alti = Alti + 1;
            else
                XData(i) = (Nd.xmin + Nd.xmax)/2;
                i = i + 1;
            end
        end
        XData = unique(XData(1:i-1));
        AltXData = unique(AltXData(1:Alti-1));
    end
end

```

```

i = 1;
Alti = 1;
for k = 1:length(iBody.Nodes)
    Nd = iBody.Nodes(k);
    if isfield(Nd.data,'matl') && ...
        Nd.data.matl.Phase ~= iBody.matl.Phase
        AltYData(Alti) = (Nd.ymin(1) + Nd.ymax(1))/2;
        Alti = Alti + 1;
    else
        YData(i) = (Nd.ymin(1) + Nd.ymax(1))/2;
        i = i + 1;
    end
end
YData = unique(YData(1:i-1));
AltYData = unique(AltYData(1:Alti-1));

%% Assign T array
TData = zeros(length(YData),length(XData));
AltTData = zeros(length(AltYData),length(AltXData));
if iBody.matl.Phase == enumMaterial.Gas
    TurbData = TData;
else
    TurbData = zeros(0,0);
end
end
PData = 0;
for k = 1:length(iBody.Nodes)
    Nd = iBody.Nodes(k);
    if isfield(Nd.data,'matl') && ...
        Nd.data.matl.Phase ~= iBody.matl.Phase
        i = find(AltYData == (Nd.ymin(1) + Nd.ymax(1))/2);
        j = find(AltXData == (Nd.xmin + Nd.xmax)/2);
        AltTData(i, j) = this.Data.T(Nd.index,ind);
    else
        i = find(YData == (Nd.ymin(1) + Nd.ymax(1))/2);
        j = find(XData == (Nd.xmin + Nd.xmax)/2);
        TData(i, j) = this.Data.T(Nd.index,ind);
        if ~isempty(TurbData) && isfield(this.Data,'turb')
            TurbData(i,j) = this.Data.turb(Nd.index,ind);
        end
        if iBody.matl.Phase == enumMaterial.Gas
            if isfield(this.Data,'SnapShot_P')
                PData = this.Data.SnapShot_P(Nd.index);
            end
        end
    end
end
end

if length(XData) > 1 && XData(1) > XData(2)
    XData = flip(XData); TData = flip(TData,2);
    TurbData = flip(TurbData,2);
end
if length(YData) > 1 && YData(1) > YData(2)
    YData = flip(YData); TData = flip(TData,1);
    TurbData = flip(TurbData,1);
end
if length(AltXData) > 1 && AltXData(1) > AltXData(2)
    AltXData = flip(AltXData); AltTData = flip(AltTData,2);
end
if length(AltYData) > 1 && AltYData(1) > AltYData(2)
    AltYData = flip(AltYData); AltTData = flip(AltTData,1);
end

%% Normalize positions
[~,~,x1,x2] = iBody.limits(enumOrient.Vertical);
[y1,y2,~,~] = iBody.limits(enumOrient.Horizontal);
XData = (XData - x1)/(x2-x1);
YData = (YData - y1(1))/(y2(1)-y1(1));
AltXData = (AltXData - x1)/(x2-x1);
AltYData = (AltYData - y1(1))/(y2(1)-y1(1));

% Assign the Body

```

```
        Snapshot.Data(index) =  
BodyData(iBody.ID,XData,YData,TData,PData,AltXData,AltYData,AltTData,TurbData);  
        % End  
        index = index + 1;  
    end  
    end  
    this.Model.addSnapshot(Snapshot);  
end  
  
end  
  
end
```

## BodyData

Bodydata is a class that represents a single body inside of a snapshot. This class contains the following functionality:

A constructor.

A function for applying the data to the nodes of a body.

```
classdef BodyData < handle

properties
    ID;
    XData;
    YData;
    TData;
    PData;
    AltXData;
    AltYData;
    AltTData;
    TurbData;
end

methods
    function this = BodyData(...
        iID,iXData,iYData,...
        iTData,iPData,...
        iAltXData,iAltYData,...
        iAltTData,iTurbData)
        if nargin > 0; this.ID = iID; end
        if nargin > 1; this.XData = iXData; end
        if nargin > 2; this.YData = iYData; end
        if nargin > 3; this.TData = iTData; end
        if nargin > 4; this.PData = iPData; end
        if nargin > 5; this.AltXData = iAltXData; end
        if nargin > 6; this.AltYData = iAltYData; end
        if nargin > 7; this.AltTData = iAltTData; end
        if nargin > 8; this.TurbData = iTurbData; end
    end

    function [success] = applyBody(this,iBody)
        success = false;
        if isempty(this.TData) && isempty(this.AltTData)
            return;
        else
            if iBody.ID == this.ID
                % It is a valid pair, Gas-Gas
                % ... Over all the nodes of the body
                if iBody.isDiscretized()
                    % Process the stored data into a grid
                    Xs = zeros(length(this.YData)+2,length(this.XData)+2);
                    Ys = Xs;
                    for r = 1:length(this.YData)+2; Xs(r,:) = [0; this.XData; 1]'; end
                    for c = 1:length(this.XData)+2; Ys(:,c) = [0; this.YData; 1]; end
                    PaddedT = CustExpandArray(this.TData);
                    FT = griddedInterpolant(Xs',Ys',PaddedT','linear','linear');
                    if ~isempty(this.TurbData)
                        PaddedTurb = CustExpandArray(this.TurbData);
                        FTurb = griddedInterpolant(...
                            Xs',Ys',PaddedTurb','linear','linear');
                    end
                end
                if ~isempty(this.AltXData)
                    AltXs = zeros(...
                        length(this.AltYData)+2,length(this.AltXData)+2);
                end
            end
        end
    end
end
```



```

    AltYs = AltXs;
    for r = 1:length(this.AltYData)+2
        AltXs(r,:) = [0; this.AltXData; 1]';
    end
    for c = 1:length(this.AltXData)+2
        AltYs(:,c) = [0; this.AltYData; 1];
    end
    PaddedAltT = CustExpandArray(this.AltTData);
    FAltT = griddedInterpolant(...
        AltXs',AltYs',PaddedAltT','linear','linear');
end

success = true;
[~,~,x1,x2] = iBody.limits(enumOrient.Vertical);
[y1,y2,~,~] = iBody.limits(enumOrient.Horizontal);
y1 = y1(1);
y2 = y2(1);
for Nd = iBody.Nodes
    % Consider what type of node it is
    assignAltTemp = false;
    assignTemp = false;
    assignTurb = false;
    if isfield(Nd.data,'matl')
        if strcmp(Nd.data.matl.name,'Constant Temperature') || ...
            strcmp(Nd.data.matl.name,'Perfect Insulator')
            assignTemp = false;
        elseif Nd.data.matl.Phase == iBody.matl.Phase
            assignTemp = true;
            if iBody.matl.Phase == enumMaterial.Gas
                if ~isempty(this.PData) && this.PData ~= 0
                    Nd.data.P = this.PData;
                end
            end
            assignTurb = ~isempty(this.TurbData);
        end
    else
        if isempty(this.AltTData)
            assignTemp = true;
        else
            assignAltTemp = true;
        end
    end
end
else
    assignTemp = true;
    if iBody.matl.Phase == enumMaterial.Gas
        if ~isempty(this.PData) && this.PData ~= 0
            Nd.data.P = this.PData;
        end
    end
    assignTurb = ~isempty(this.TurbData);
end
end
cx = ((Nd.xmax + Nd.xmin)/2 - x1)/(x2-x1);
cy = ((Nd.ymin(1) + Nd.ymax(1))/2 - y1)/(y2-y1);
T = Nd.data.T;
Turb = 0;
if assignTemp && ~isempty(this.TData)
    T = FT(cx,cy);
    if assignTurb; Turb = FTurb(cx,cy); end
    %{
    if length(this.XData) == 1
        if length(this.YData) == 1
            % No interpolation
            T = this.TData(1,1);
            if assignTurb
                Turb = this.TurbData(1,1);
            end
        else
            % Linear Interpolation
            try
                T = interp1(this.YData,this.TData,cy,'pchip','extrap');
                if assignTurb
                    Turb = interp1(this.YData,this.TurbData,cy,'pchip','extrap');
                end
            catch
            end
        end
    end
end

```

```

        end
    catch
        this.TData = [];
        this.TurbData = [];
        success = false;
        return;
    end
end
else
    if length(this.YData) == 1
        % Linear Interpolation
        try
            T = interp1(this.XData,this.TData,cx,'pchip','extrap');
            if assignTurb
                Turb = interp1(this.XData,this.TurbData,cx,'pchip','extrap');
            end
        catch
            this.TData = [];
            this.TurbData = [];
            success = false;
            return;
        end
    else
        % Double Linear Interpolation
        try
            T2 = interp1(this.YData,this.TData,cy,'pchip','extrap');
            T = interp1(this.XData,T2',cx,'pchip','extrap');
            if assignTurb
                Turb2 = interp1(this.YData,this.TurbData,cy,'pchip','extrap');
                Turb = interp1(this.XData,Turb2',cx,'pchip','extrap');
            end
        catch
            this.TData = [];
            this.TurbData = [];
            success = false;
            return;
        end
    end
end
end
%}
elseif assignAltTemp && ~isempty(this.AltTData)
    T = FAltT(cx,cy);
    %{
    if length(this.AltXData) == 1
        if length(this.AltYData) == 1
            % No interpolation
            T = this.AltTData(1,1);
        else
            % Linear Interpolation
            try
                T = interp1(this.AltYData,this.AltTData,cy,'pchip','extrap');
            catch
                this.AltTData = [];
                success = false;
                return;
            end
        end
    end
else
    if length(this.AltYData) == 1
        % Linear Interpolation
        try
            T = interp1(this.AltXData,this.AltTData,cx,'pchip','extrap');
        catch
            this.AltTData = [];
            success = false;
            return;
        end
    else
        % Double Linear Interpolation
        try
            T2 = interp1(this.AltYData,this.AltTData,cy,'pchip','extrap');

```



## **G.7. ListObjs**

## List Object

ListObj represents objects in the property drop down. It is a class with the following functionality:

A constructor.

A function which is called when the entry in the listbox is clicked.

A function that gets the objects children, given its type.

A function that gets the string that is displayed in the listbox editor.

```
classdef ListObj < handle
    %LISTOBJ Summary of this class goes here
    % Detailed explanation goes here

    properties
        MODE = '';
        lvl int8;
        isExpanded logical = false;
        Parent;
        Child; % Various
        Info; % Various
        Subs ListObj;
    end

    methods
        function this = ListObj(MODE, lvl, Parent, Child, info)
            if nargin > 0
                this.MODE = MODE;
                this.lvl = lvl;
                this.Parent = Parent;
                this.Child = Child;
                if nargin > 4
                    this.Info = info;
                end
            end
        end

        function on_click(this)
            switch this.MODE
                case 'Editstr'
                    % Bring up user form inputdlg
                    newvalue = get(this.Parent, this.Child);
                    if isempty(newvalue); newvalue = ''; end
                    newvalue = inputdlg(['Property: ' this.Child ': '], ...
                        'Edit Properties', 1, {newvalue});
                    if ~isempty(newvalue) && isa(newvalue{1}, 'char')
                        set(this.Parent, this.Child, newvalue{1});
                    end
                case 'Editnum'
                    % Bring up user form inputdlg
                    newvalue = inputdlg(['Edit the value of ' this.Child ' in ' this.Info], ...
                        'Edit Properties', 1, ...
                        {num2str(get(this.Parent, this.Child))});
                    number = SymbolicMath(newvalue{1});
                    if isnan(number)
                        msgbox('Invalid formula: Ensure that your formula is complete and avoids
scientific notation.');
```

```

        set(this.Parent,this.Child,number);
    end
    case 'Editnumconvert'
        % Bring up user form inputdlg
        newvalue = (inputdlg(['Edit the value of ' this.Child ' in '
this.Info{2}],...
        'Edit Properties',1,...
        {num2str(round(get(this.Parent,this.Child)*this.Info{1})))));
        number = SymbolicMath(newvalue{1});
        if isnan(number)
            msgbox('Invalid formula: Ensure that your formula is complete and avoids
scientific notation.');
```

```

        else
            set(this.Parent,this.Child,number/this.Info{1});
        end
    case {'Expandobj', 'Expandlist'}
        this.isExpanded = ~this.isExpanded;
    case 'Configureobj'
        % the Parent has a child that has a parameter labeled 'Source'
        % this source is used as an input into the child's constructor
        % Bring up a user form
        if ischar(this.Child)
            Item = get(this.Parent,this.Child);
            Item.Modify();
        else
            this.Child.Modify();
        end
    case 'Pickobj'
        % Info is a objarray
        Item = get(this.Parent,this.Child);
        objs = this.Info;
        names = {'...'};
        for index = length(objs):-1:1
            names{index+1} = objs(index).name;
        end
        if ~isempty(Item)
            for index = 1:length(objs)
                if Item == objs(index)
                    break;
                end
            end
        end
        index = listdlg('ListString',names,...
            'SelectionMode','single',...
            'InitialValue',index); % ADD WIDTH, HEIGHT CONSTRAINTS
        if index == 1
            set(this.Parent,this.Child,[]);
        else
            set(this.Parent,this.Child,objs(index-1));
        end
    case 'Pickfunction'
        % Info is a folder name
        % Bring up a user form listdlg from folder: Item
        % Get index of current value
        temp = get(this.Parent,this.Child);
        if isempty(temp)
            Item = '';
        else
            Item = func2str(temp);
        end
        files = dir(this.Info);
        names = {files.name};
        names = names{3:end}; % Remove the first couple
        if ~iscell(names)
            names = {names};
        end
        for index = size(names,1):-1:1
            names{index} = names{index}(1:end-2);
        end
        for index = 1:length(names)
            if strcmp(names{index},Item); break; end
        end
    end
end

```

```

        end
        index = listdlg('ListString',names,...
            'SelectionMode','single',...
            'InitialValue',index);
        if isempty(index)
            set(this.Parent,this.Child,function_handle.empty);
        else
            set(this.Parent,this.Child,str2func(names{index}));
        end
    case 'Function'
        functions(this.Parent,this.Child);
    case 'Deleteobj'
        this.Parent.deReference();
    case 'NamedList'
        names = get(this.Parent,this.Child);
        if ~isempty(names)
            [indx, tf] = listdlg(...
                'PromptString',['Select ' this.Child ' to Remove'],...
                'ListString',names);
            if tf
                answers = false(length(names),1);
                answers(indx) = true;
                set(this.Parent,this.Child,answers);
            end
        end
    end
end
end

function [objects] = getObjs(this,expanded)
    if ischar(this.Child)
        if ~strcmp(this.Child,'Deleteobj')
            Item = get(this.Parent,this.Child);
            end
            Text = this.Child;
        else
            Item = this.Child;
            Text = class(this.Child);
        end
        slvl = this.lvl+1;
        if nargin == 2
            switch this.MODE
                case 'Expandobj'
                    if isempty(Item)
                        objects = this;
                    else
                        switch class(Item)
                            case 'Model'
                                objects = [this; ...
                                    ListObj('Editstr',slvl,Item,'Name'); ...
                                    ListObj('Expandlist',slvl,Item,'Groups'); ...
                                    ListObj('Expandlist',slvl,Item,'Bridges'); ...
                                    ListObj('Expandlist',slvl,Item,'Leaks'); ...
                                    ListObj('Expandlist',slvl,Item,'Sensors'); ...
                                    ListObj('Expandlist',slvl,Item,'PVoutputs'); ...
                                    ListObj('NamedList',slvl,Item,'SnapShots'); ...
                                    ListObj('NamedList',slvl,Item,'NonConnections'); ...
                                    ListObj('NamedList',slvl,Item,'Custom Minor Losses'); ...
                                    ListObj('Expandlist',slvl,Item,'Lin. to Rot.
Mechanisms'); ...
                                    ListObj('Expandlist',slvl,Item,'Optimization Studies');
...
                                    ListObj('Expandobj',slvl,Item,'Initial Internal
Conditions'); ...
                                    ListObj('Expandobj',slvl,Item,'External Conditions'); ...
                                    ListObj('Editnum',slvl,Item,'Engine Temperature','K');
...
                                    ListObj('Editnum',slvl,Item,'Engine Pressure','Pa'); ...
                                    ListObj('Editnum',slvl,Item,'Minimum Speed','Hz'); ...
                                    ListObj('Expandobj',slvl,Item,'Mechanical System'); ...
                                    ListObj('Expandobj',slvl,Item,'Mesher'); ...
                                    ListObj('Editnum',slvl,Item,'Max Courant Final'); ...

```

```

        ListObj('Editnum',slvl,Item,'Max Fourier Final'); ...
        ListObj('Editnum',slvl,Item,'Max Courant Converging');
...
        ListObj('Editnum',slvl,Item,'Max Fourier Converging']);
case 'Group'
    objects = [this; ...
        ListObj('Editstr',slvl,Item,'Name'); ...
        ListObj('Expandobj',slvl,Item,get(Item,'Position')); ...
        ListObj('Expandlist',slvl,Item,'Bodies'); ...
        ListObj('Expandlist',slvl,Item,'Connections'); ...
        ListObj('Expandlist',slvl,Item,'Relation Managers'); ...
        ListObj('Deleteobj',slvl,Item,['X] Delete']);
case 'Body'
    objects = [this; ...
        ListObj('Editstr',slvl,Item,'Name'); ...
        ListObj('Expandobj',slvl,Item,'Bottom Connection'); ...
        ListObj('Expandobj',slvl,Item,'Top Connection'); ...
        ListObj('Expandobj',slvl,Item,'Inner Connection'); ...
        ListObj('Expandobj',slvl,Item,'Outer Connection'); ...
        ListObj('Configureobj',slvl,Item,'Material'); ...
        ListObj('Editnum',slvl,Item,'Temperature'); ...
        ListObj('Editnum',slvl,Item,'Pressure'); ...
        ListObj('Editnum',slvl,Item,'Radial
Divides','divisions'); ...
        ListObj('Editnum',slvl,Item,'Axial Divides','divisions');
...
ListObj('Pickobj',slvl,Item,'RefFrame',Item.Group.Model.RefFrames); ...
        ListObj('Configureobj',slvl,Item,'Change Matrix');...
        ListObj('Expandobj',slvl,Item,'Expand Matrix');...
        ListObj('Pickfunction',slvl,Item,'Radial Discretization
Function','Function - Discretization'); ...
        ListObj('Pickfunction',slvl,Item,'Axial Discretization
Function','Function - Discretization'); ...
        ListObj('Deleteobj',slvl,Item,['X] Delete']);
case 'Connection'
    objects = [this; ...
        ListObj('Editnum',slvl,Item,'x','m'); ...
ListObj('Pickobj',slvl,Item,'RefFrame',Item.Group.Model.RefFrames); ...
        ListObj('Expandlist',slvl,Item,'Bodies'); ...
        ListObj('Deleteobj',slvl,Item,['X] Delete'); ...
        ListObj('Expandlist',slvl,Item,'Isolated Bodies'); ...
        ListObj('Function',slvl,Item,'Add Bodies To Not Join');
...
        ListObj('Function',slvl,Item,'Remove Bodies To Not
Join']);
case 'Bridge'
    objects = [this; ...
        ListObj('Expandobj',slvl,Item,'Connection 1'); ...
        ListObj('Expandobj',slvl,Item,'Connection 2'); ...
        ListObj('Expandobj',slvl,Item,'Body 1'); ...
        ListObj('Expandobj',slvl,Item,'Body 2'); ...
        ListObj('Deleteobj',slvl,Item,['X] Delete']);
case 'LeakConnection'
    objects = [this; ...
        ListObj('Expandobj',slvl,Item,'Connection 1'); ...
        ListObj('Expandobj',slvl,Item,'Connection 2'); ...
        ListObj('Expandobj',slvl,Item,'Object 1'); ...
        ListObj('Expandobj',slvl,Item,'Object 2'); ...
        ListObj('Pickfunction',slvl,Item,'LeakFunc','Function -
Leakage'); ...
        ListObj('Deleteobj',slvl,Item,['X] Delete']);
case 'Environment'
    objects = [this; ...
        ListObj('Editstr',slvl,Item,'Name'); ...
        ListObj('Editnum',slvl,Item,'Pressure','Pa'); ...
        ListObj('Editnum',slvl,Item,'Temperature','K'); ...
        ListObj('Editnum',slvl,Item,'h','W/mK'); ...
        ListObj('Configureobj',slvl,Item,'Gas']);
case 'Position'

```



```

        objects = [this; ...
        ListObj('Editnum',slvl,Item,'x','m'); ...
        ListObj('Editnum',slvl,Item,'y','m'); ...
        ListObj('Editnumconvert',slvl,Item,'Theta',{180/pi;
'degrees'}})];
    case 'Matrix'
        objects = [this; ...
        ListObj('Configureobj',slvl,Item,'Material'); ...
        ListObj('Pickfunction',slvl,Item,'Laminar Friction
Function','Function - Laminar Friction'); ...
        ListObj('Pickfunction',slvl,Item,'Turbulent Friction
Function','Function - Turb Friction'); ...
        ListObj('Pickfunction',slvl,Item,'Laminar Nusselt
Function','Function - Laminar Nusselt'); ...
        ListObj('Pickfunction',slvl,Item,'Turbulent Nusselt
Function','Function - Turb Nusselt'); ...
        ListObj('Pickfunction',slvl,Item,'Laminar Streamwise
Cond. Enhancement','Function - Laminar Cond Enhancement'); ...
        ListObj('Pickfunction',slvl,Item,'Turbulent Streamwise
Cond. Enhancement','Function - Turb Cond Enhancement'); ...
        ListObj('Editnum',slvl,Item,'Source Temperature','K');
...
        ListObj('Deleteobj',slvl,Item,['X] Delete'));
    case 'Mesher'
        objects = [this; ...
        ListObj('Editnum',slvl,Item,'Nodes through Oscillation
Depth'); ...
        ListObj('Editnum',slvl,Item,'Maximum Node Thickness');
...
        ListObj('Editnum',slvl,Item,'Maximum Growth Rate'); ...
        ListObj('Editnum',slvl,Item,'Heat Exchanger Fin
Divisions'); ...
        ListObj('Editnum',slvl,Item,'Minimum Solid Time Step');
...
        ListObj('Editnum',slvl,Item,'Gas Entrance Exit N'); ...
        ListObj('Editnum',slvl,Item,'Gas Maximum Size'); ...
        ListObj('Editnum',slvl,Item,'Gas Minimum Size')];
    case 'Sensor'
        objects = [this; ...
        ListObj('Editstr',slvl,Item,'Name'); ...
        ListObj('Editnum',slvl,Item,'Samples'); ...
        ListObj('Deleteobj',slvl,Item,['X] Delete'));
    case 'PVoutput'
        objects = [this; ...
        ListObj('Editstr',slvl,Item,'name'); ...
        ListObj('Pickobj',slvl,Item,'Source
Body/Region',Item.Model.BodyList); ...
        ListObj('Deleteobj',slvl,Item,['X] Delete'));
    case 'MechanicalSystem'
        objects = [this; ...
        ListObj('Editnum',slvl,Item,'Flywheel Inertia'); ...
        ListObj('Editnum',slvl,Item,'Drive Train Weight'); ...
        ListObj('Editnum',slvl,Item,'Drive Train Normal Friction
Coefficient'); ...
        ListObj('Pickfunction',slvl,Item,'Load
Function','Function - Load Function');
    case 'RelationManager'
        objects = [this; ...
        ListObj('Editstr',slvl,Item,'Name'); ...
        ListObj('Expandlist',slvl,Item,'Relations')];
    case 'Relation'
        objects = [this; ...
        ListObj('Editstr',slvl,Item,'Name'); ...
        ListObj('Expandobj',slvl,Item,'Connection1'); ...
        ListObj('Expandobj',slvl,Item,'Connection2'); ...
ListObj('Pickobj',slvl,Item,'Frame',Item.manager.Group.Model.RefFrames); ...
        ListObj('Deleteobj',slvl,Item,['X] Delete'));
    case 'OptimizationScheme'
        objects = [this; ...
        ListObj('Editstr',slvl,Item,'Name'); ...

```

```

                                ListObj('NamedList',slvl,Item,'DOFs']);
        end
    end
    case 'Expandlist'
        objs = get(this.Parent,Text);
        LEN = length(objs);
        objects(LEN+1,1) = ListObj();
        switch class(objs)
            case {'LinRotMechanism', 'Material'}
                % Modify rather than expand
                for i = LEN:-1:1
                    objects(i+1) = ListObj('Configureobj',slvl,Item,objs(i));
                end
            otherwise
                for i = LEN:-1:1
                    objects(i+1) = ListObj('Expandobj',slvl,Item,objs(i));
                end
            end
        end
        objects(1) = this;
    otherwise
        objects = this;
    end
end
else % do not expand
    objects = this;
end
end

function [output] = getString(this)
    starter = repmat(' . . . ',1,this.lvl);
    if ischar(this.Child)
        if ~strcmp(this.Child,['X Delete']) && ~strcmp(this.MODE,'Function')
            Item = get(this.Parent,this.Child);
        end
        Text = this.Child;
    else
        Item = this.Child;
        Text = class(this.Child);
    end
    switch this.MODE
        case 'Editstr'
            output = [starter Text ': [' Item ']]];
        case 'Editnum'
            output = [starter Text ': [' num2str(Item) ' ' this.Info ']]];
        case 'Editnumconvert'
            output = [starter Text ': [' num2str(Item*this.Info{1}) ' (' this.Info{2}
')]'];
        case 'Expandobj'
            if isvalid(Item)
                output = [starter Text ' (' class(Item) '): [' Item.name ']]];
            else
                output = [starter Text ' (' class(Item) '): [X Deleted Object]]];
            end
        case 'Expandlist'
            objs = get(this.Parent,Text);
            if ~isempty(objs)
                objs = objs(isvalid(objs));
            end
            if length(objs) < 1; output = [starter Text '[empty]'];
            else; output = [starter Text '[...]'];
            end
        case 'Configureobj'
            output = [starter Text ': [' Item.name ']]];
        case 'Pickobj'
            % info is the list of frames
            output = [starter class(Item) ': ' Item.name];
        case 'Pickfunction'
            if isempty(Item)
                output = [starter Text ': [...]'];
            else
                output = [starter Text ': [@' func2str(Item) ']]];
            end
    end
end

```

```
        case 'Function'
            output = [starter Text];
        case 'Deleteobj'
            output = [starter Text];
        case 'NamedList'
            if isempty(Item); output = [starter Text '[empty]'];
            else; output = [starter Text '[...]'];
            end
        end
    end
end

function [indicator] = isExpandable(this)
    switch this.MODE
        case {'Expandobj', 'Expandlist'}
            indicator = true;
        otherwise
            indicator = false;
        end
    end
end
end
end
```

## Make Code

This function makes a string code that stores the state of what is displayed on the property listbox. This was introduced so that the structure could be recreated after edits.

```
function [ Code ] = MakeCode( ListObjs, ClickedIndex)
Code = '';
lvl = 0;
n = ones(1,16); % Current Index on Level
i = 1;
while i <= length(ListObjs)
    % Handle Step downs and step ups
    if ListObjs(i).lvl > lvl
        % Step Down
        % ... Parent      n(lvl+1)
        % ... ... Child  n(lvl+2) = 1
        if isempty(Code); Code = [num2str(int8(n(lvl+1)-1)) '['];
        elseif Code(end) == '['; Code = [Code num2str(int8(n(lvl+1)-1)) '['];
        else; Code = [Code ',' num2str(int8(n(lvl+1)-1)) '['];
        end
        lvl = lvl + 1;
        % Iterate the "Child" node
        n(lvl+1) = 2;
    elseif ListObjs(i).lvl < lvl
        % Step Up
        % ... ... Child    n(lvl)
        % ... Next-Parent n(lvl+1)
        while lvl > ListObjs(i).lvl
            Code = [Code '']];
            n(lvl+1) = 1;
            lvl = lvl - 1;
        end
        % Iterate the "Next-Parent" node
        n(lvl+1) = n(lvl+1) + 1;
    else
        % Iterate the node
        n(lvl+1) = n(lvl+1) + 1;
    end
end

% Handle the click
if nargin == 2 && ClickedIndex == i
    if ListObjs(i).isExpandable()
        if i < length(ListObjs)
            if ListObjs(i+1).lvl > ListObjs(i).lvl
                % This one is already expanded, collapse
                i = i + 1;
                while i <= length(ListObjs) && ListObjs(i).lvl > lvl
                    i = i + 1;
                end
                % Iterate the level forward, we are skipping to another node
                n(lvl+1) = n(lvl+1) + 1;
            else
                % This one should be expanded
                if isempty(Code); Code = num2str(int8(n(lvl+1)-1));
                elseif Code(end) == '['; Code = [Code num2str(int8(n(lvl+1)-1))];
                else; Code = [Code ',' num2str(int8(n(lvl+1)-1))];
                end
            end
        end
        % This one should be expanded
        if isempty(Code); Code = num2str(int8(n(lvl+1)-1));
        elseif Code(end) == '['; Code = [Code num2str(int8(n(lvl+1)-1))];
        else; Code = [Code ',' num2str(int8(n(lvl+1)-1))];
        end
    end
end
end
```

```
    end
    i = i + 1;
end
while lvl > 0
    Code = [Code ''];
    lvl = lvl - 1;
end
Code = strrep(Code, '[', '');
if ~isempty(Code) && Code(end) == '['
    Code(end) = '';
end
end
```

## Read Code

This reads the code and produces the active list that is displayed on the property listbox.

```
function [ newListObjs ] = ReadCode( Code, ListObjs )
i = 1;
num = 0;
newListObjs = ListObj.empty;
close = false;
while i <= length(Code)
    k = i+1;
    while k <= length(Code) ...
        && Code(k) ~= '[' ...
        && Code(k) ~= ']' ...
        && Code(k) ~= ','
            k = k + 1;
    end
    onum = num;
    num = int16(str2double(Code(i:k-1)));
    if num ~= onum + 1
        if length(ListObjs) >= num-1
            newListObjs = [newListObjs; ListObjs(onum+1:num-1)];
        else
            return;
        end
    end
    if num > length(ListObjs) || num < 1
        return;
    end
    Internals = ListObjs(num).getObjs(true);
    newListObjs = [newListObjs; Internals(1)];
    if length(Internals) > 1
        if k <= length(Code)
            switch Code(k)
                case '['
                    % Enter Recursion on Contents with contents of element "num"
                    i = k + 1;
                    lvlcount = -1;
                    while i <= length(Code) && lvlcount < 0
                        switch Code(i)
                            case '['; lvlcount = lvlcount - 1;
                            case ']' ; lvlcount = lvlcount + 1;
                        end
                        i = i + 1;
                    end
                    if i < length(Code)
                        if Code(i) == ','
                            i = i + 1;
                            newCode = Code(k+1:i-3);
                        else
                            newCode = Code(k+1:i-2);
                        end
                    end
                    else
                        newCode = Code(k+1:i-2);
                    end
                    newListObjs = [newListObjs; ReadCode(newCode, Internals(2:end))];
                case ']'
                    % End of Recursion Layer
                    newListObjs = [newListObjs; Internals(2:end)];
                    return;
                case ','
                    % New expansion, add elements between expanded elements
                    newListObjs = [newListObjs; Internals(2:end)];
            end
        else
            newListObjs = [newListObjs; Internals(2:end)];
        end
    end
end
```

```
end
  i = max(i,k);
end
if num < length(ListObjs)
  newListObjs = [newListObjs; ListObjs(num+1:end)];
end
end
```

## Reset Code

This resets the code to be just the default entry (model).

```
function [Code] = ResetCode(Code)
if ~isempty(Code)
    if Code(1) == '1'
        % Model may maintain its expansion
        if length(Code) > 1
            if Code(2) == '['
                i = 3; lvlcount = -1;
                while i < length(Code) && lvlcount < 0
                    switch Code(i)
                        case '['; lvlcount = lvlcount - 1;
                        case ']' ; lvlcount = lvlcount + 1;
                    end
                    i = i + 1;
                end
                if length(Code) > i; Code(i+1:end) = ''; end
            end
        end
    else
        % Everything else is presumed to have changed
        Code = '';
    end
end
end
```



## Selection List Data

Simple class that stores the list-objects and state code of the property list-box.

```
classdef SelectionListData < handle
    properties
        ListObjs;
        Code;
    end
end
end
```

## **G.8. Geometry**

## Line 1D

Line 1D is a class whose purpose is to represent a group of 1D line segments. This line segment can lie upon any axis, provided that a distance along the axis can be measured. Line 1D contains the following functionality:

A constructor.

A function (**DestroyIntersept**) that finds and destroys the intercept between two line 1D objects (which is acted upon in both participating objects). This may split one of the objects resulting in the addition of an other row to the bounds matrix.

A second function (**Subtract**) that removes the area covered by the second input line1D from the current object.

A third function (**MergeAndAppend**) adds the another line1D object to the current object.

The last function (**CreateLine2Ds**) creates a series of line2D given an orientation and extra position. (assuming axial or radial directions)

```
classdef Line1D < handle
    %Line1D Summary of this class goes here
    % Detailed explanation goes here

    properties
        bounds;
    end

    methods
        function this = Line1D(lb,ub)
            if nargin == 2
                this.bounds(1,1) = lb;
                this.bounds(1,2) = ub;
            end
        end

        function DestroyIntersept(this,other)
            n1 = 1; n2 = 1;
            s1 = 1; s2 = 1;
            notdone = true;
            while notdone

                notdone = false;

                for thisrow = s1:size(this.bounds,1)
                    lb1 = this.bounds(thisrow,1);
                    ub1 = this.bounds(thisrow,2);
                    for otherrow = s2:size(other.bounds,1)
                        lb2 = other.bounds(otherrow,1);
                        ub2 = other.bounds(otherrow,2);
                        if lb1 < lb2
```

```

if ub1 > ub2% && lb1 < lb2
    % Split this, delete other
    other.bounds(otherrow,1:2) = inf;
    extra1(n1,1) = ub2;
    extra1(n1,2) = this.bounds(thisrow,2);
    this.bounds(thisrow,2) = lb2;
    n1 = n1 + 1;
elseif ub1 < ub2% && lb1 < lb2
    % this ----- -> ----xx
    % other ----- ->    xx-----
    % Chop
    this.bounds(thisrow,2) = lb2;
    other.bounds(otherrow,1) = ub1;
else % ub1 == ub2 && lb1 < lb2
    % chop this, delete other
    other.bounds(otherrow,1:2) = inf;
    this.bounds(thisrow,2) = lb2;
end
elseif lb1 > lb2
if ub1 > ub2% && lb1 > lb2
    % this ----- ->    xx-----
    % other----- -> ----xx
    % Chop
    this.bounds(thisrow,1) = ub2;
    other.bounds(otherrow,2) = lb1;
elseif ub1 < ub2% && lb1 > lb2
    % Split other, delete this
    this.bounds(thisrow,1:2) = inf;
    extra2(n2,1) = ub1;
    extra2(n2,2) = other.bounds(otherrow,2);
    other.bounds(otherrow,2) = lb1;
    n2 = n2 + 1;
else % ub1 == ub2 && lb1 > lb2
    % chop other, delete this
    this.bounds(thisrow,1:2) = inf;
    other.bounds(otherrow,2) = lb1;
end
else % lb1 == lb2
if ub1 > ub2 % && lb1 == lb2
    % Chop this, delete other
    this.bounds(thisrow,1) = ub2;
    other.bounds(otherrow,1:2) = inf;
elseif ub1 < ub2 % && lb1 == lb2
    % Chop other, delete this
    this.bounds(thisrow,1:2) = inf;
    other.bounds(otherrow,1) = ub1;
else % ub1 == ub2 && lb1 == lb2
    % delete other, delete this
    this.bounds(thisrow,1:2) = inf;
    other.bounds(otherrow,1:2) = inf;
end
end
end
end

% Clean up both
r = 1:size(this.bounds,1);
this.bounds(isinf(this.bounds(r,1)), :) = [];
r = 1:size(other.bounds,1);
other.bounds(isinf(other.bounds(r,1)), :) = [];

if ~(size(this.bounds,1) == 0 || size(other.bounds,1) == 0)
if n1 > 1
    s1 = size(this.bounds,1)+1;
    s2 = 1;
    this.bounds = [this.bounds; extra1];
    notdone = true;
end
end

```

```

        if ~notdone && n2 > 1
            s1 = 1;
            s2 = size(other.bounds,1)+1;
            other.bounds = [other.bounds; extra2];
            notdone = true;
        end
    end

end

end

function Subtract(this,other)
    n1 = 1;
    s1 = 1;
    notdone = true;
    while notdone

        notdone = false;

        for thisrow = s1:size(this.bounds,1)
            lb1 = this.bounds(thisrow,1);
            ub1 = this.bounds(thisrow,2);
            for otherrow = 1:size(other.bounds,1)
                lb2 = other.bounds(otherrow,1);
                ub2 = other.bounds(otherrow,2);
                if ~(lb1 > ub2 || lb2 > ub1)
                    if lb1 < lb2
                        if ub1 > ub2% && lb1 < lb2
                            % Split this, delete other
                            extral(n1,1) = ub2;
                            extral(n1,2) = this.bounds(thisrow,2);
                            this.bounds(thisrow,2) = lb2;
                            n1 = n1 + 1;
                        elseif ub1 < ub2% && lb1 < lb2
                            % this ----- -> ----xx
                            % other ----- ->    xx-----
                            % Chop
                            this.bounds(thisrow,2) = lb2;
                        else % ub1 == ub2 && lb1 < lb2
                            % chop this, delete other
                            this.bounds(thisrow,2) = lb2;
                        end
                    elseif lb1 > lb2
                        if ub1 > ub2% && lb1 > lb2
                            % this ----- ->    xx-----
                            % other----- -> ----xx
                            % Chop
                            this.bounds(thisrow,1) = ub2;
                        elseif ub1 < ub2% && lb1 > lb2
                            % Split other, delete this
                            this.bounds(thisrow,1:2) = inf;
                        else % ub1 == ub2 && lb1 > lb2
                            % chop other, delete this
                            this.bounds(thisrow,1:2) = inf;
                        end
                    else % lb1 == lb2
                        if ub1 > ub2 % && lb1 == lb2
                            % Chop this, delete other
                            this.bounds(thisrow,1) = ub2;
                        elseif ub1 < ub2 % && lb1 == lb2
                            % Chop other, delete this
                            this.bounds(thisrow,1:2) = inf;
                        else % ub1 == ub2 && lb1 == lb2
                            % delete other, delete this
                            this.bounds(thisrow,1:2) = inf;
                        end
                    end
                end
            end
        end
    end
end

```

```

        end
    end
end

% Clean up both
r = 1:size(this.bounds,1);
this.bounds(isinf(this.bounds(r,1)),:) = [];

if ~(size(this.bounds,1) == 0)
    if n1 > 1
        this.bounds = [this.bounds; extra1];
    end
end

end
end

function MergeAndAppend(this,other)
    removeThis = false(1,length(this.bounds,1));
    removeOther = false(1,length(other.bounds,1));
    % See if the endpoints match up, then merge, it is assumed that the
    % lines have been subjected to DestroyIntercepts already
    % ... Add the merged lines to "this"
    for i = 1:length(this.bounds,1)
        for j = 1:length(other.bounds,1)
            if ~removeOther(j)
                if this.bounds(i,1) == other.bounds(j,2)
                    this.bounds(i,1) = other.bounds(j,1);
                    removeOther(j) = true;
                    i = i - 1; %#ok<FXSET>
                elseif this.bounds(i,2) == other.bounds(j,1)
                    this.bounds(i,2) = other.bounds(j,2);
                    removeOther(j) = true;
                    i = i - 1; %#ok<FXSET>
                end
            end
        end
    end
end
% See if the modified entries of "this" can be merged
for i = 1:length(this.bounds,1)-1
    if ~removeThis(i)
        for j = i+1:length(this.bounds,1)
            if ~removeThis(j)
                if this.bounds(i,1) == this.bounds(j,2)
                    this.bounds(i,1) = this.bounds(j,1);
                    removeThis(j) = true;
                    i = i - 1; %#ok<FXSET>
                elseif this.bounds(i,2) == this.bounds(j,1)
                    this.bounds(i,2) = this.bounds(j,2);
                    removeThis(j) = true;
                    i = i - 1; %#ok<FXSET>
                end
            end
        end
    end
end
% Remove entries
this.bounds(removeOther,:) = [];
this.bounds = [this.bounds; other.bounds];
end

function Line2DOutput = CreateLine2Ds(this,Orient,x)
    Line2DOutput(1,length(this.bounds,1)) = Line2DChain();
    if Orient == enumOrient.Vertical
        for i = 1:length(this.bounds,1)
            Line2DOutput(i) = ...
                Line2DChain(x,this.bounds(i,1),x,this.bounds(i,2));
        end
    end
end

```

```
    else
      for i = 1:length(this.bounds,1)
        Line2DOutput(i) = ...
          Line2DChain(this.bounds(i,1),x,this.bounds(i,2),x);
      end
    end
  end
end
end
end
```

## Line 2D Chain

Line 2D chain is a class that represents a set of connected points in 2D space. It contains the following functionality:

A constructor.

A function (**attemptToMerge**), which tries to link up the beginning and end points of two line 2D chains, returning if the merging was successful. This is used by the solid-environment boundary detection algorithm.

A function (**get.isFinished**) which checks if both the start and end points are on the middle axis, or if they are equal to each other. If so then nothing can be added to the loop.

A series of functions that retrieve particular defining points from the arrays. (such as the ends or an extracted series of x or y data from the point lists).

```
classdef Line2DChain < handle
%LINECHAIN Summary of this class goes here
% Detailed explanation goes here

properties
    Pnts Pnt2D = Pnt2D.empty;
end

properties (Dependent)
    XData double;
    YData double;
    Start Pnt2D;
    End Pnt2D;
    isFinished logical;
end

methods
function this = Line2DChain(x1,y1,x2,y2)
    if nargin > 0
        this.Pnts = [Pnt2D(x1,y1) Pnt2D(x2,y2)];
    end
end
function successful = attemptToMerge(this,other)
    successful = false;
    if this.Pnts(end) == other.Pnts(1)
        this.Pnts = [this.Pnts other.Pnts(2:end)]; successful = true;
    elseif this.Pnts(end) == other.Pnts(end)
        this.Pnts = [this.Pnts other.Pnts(end-1:-1:1)]; successful = true;
    elseif this.Pnts(1) == other.Pnts(end)
        this.Pnts = [other.Pnts this.Pnts(2:end)]; successful = true;
    elseif this.Pnts(1) == other.Pnts(1)
        this.Pnts = [other.Pnts(end:-1:2) this.Pnts]; successful = true;
    end
end
function isFinished = get.isFinished(this)
    isFinished = (this.Pnts(1).x == 0 && this.Pnts(end).x == 0) || ...
        (this.Pnts(1) == this.Pnts(end));
end
function Start = get.Start(this)
```



```
    Start = this.Pnts(1);  
end  
function End = get.End(this)  
    End = this.Pnts(end);  
end  
function XData = get.XData(this)  
    XData = zeros(1,length(this.Pnts));  
    for i = 1:length(this.Pnts)  
        XData(i) = this.Pnts(i).x;  
    end  
end  
function YData = get.YData(this)  
    YData = zeros(1,length(this.Pnts));  
    for i = 1:length(this.Pnts)  
        YData(i) = this.Pnts(i).y;  
    end  
end  
end  
end  
end
```

## Loop Node

Loop Node is a class that serves to facilitate the finding of loops in the gas network. It contains the following functionality:

A constructor.

A recursive function (**get.lvl**) used to determine the length of a chain of loopNodes.

```
classdef LoopNode
    %LOOPNODE Summary of this class goes here
    % Detailed explanation goes here

    properties
        parent LoopNode;
        parentFc Face;
        Nd Node;
    end

    properties (Dependent)
        lvl int8;
    end

    methods
        function this = LoopNode(iparent, iparentFc, iNd)
            if nargin == 1
                this.Nd = iparent;
            elseif nargin == 3
                this.parent = iparent;
                this.parentFc = iparentFc;
                this.Nd = iNd;
            end
        end

        function lvl = get.lvl(this)
            if ~isempty(this.parent)
                lvl = this.parent.lvl + 1;
            else
                lvl = 1;
            end
        end
    end
end
```

## Pnt 2D

Pnt2D is a class that is the basic x, y data structure of most of the geometry functions. It contains additional functionality:

Constructor.

A **rotate** function which takes a rotation matrix and updates its positions.

A **shift** function that adds a length 2 position vector to the x, y position.

```
classdef Pnt2D < handle
    %PNT2D Summary of this class goes here
    % Detailed explanation goes here

    properties
        x double;
        y double;
    end

    methods
        function this = Pnt2D(x,y)
            if nargin == 0
                return;
            end
            this.x = x;
            this.y = y;
        end

        function isequal = eq(Pnt1,Pnt2)
            isequal = (Pnt1.x == Pnt2.x && Pnt1.y == Pnt2.y);
        end

        function rotate(this, RotationMatrix)
            newx = RotationMatrix(1,1)*this.x + RotationMatrix(1,2)*this.y;
            this.y = RotationMatrix(2,1)*this.x + RotationMatrix(2,2)*this.y;
            this.x = newx;
        end

        function shift(this, PositionVector)
            this.x = PositionVector(1) + this.x;
            this.y = PositionVector(2) + this.y;
        end
    end
end
```

## G.1. Enum

### Enumeration of Face Types

```
classdef enumFType
    enumeration
        Mix,...
        Gas,...
        Leak,...
        Solid,...
        Environment,...
        MatrixTransition
    end
end
```

### Enumeration of Types of Materials (Phase)

```
classdef enumMaterial
    enumeration
        Gas, Solid, Liquid
    end
end
```

### Enumeration of Types of Matrix

```
classdef enumMatrix
    enumeration
        WovenScreen,
        RandomFiber,
        PackedSphere,
        StackedFoil,
        HeatExchanger
    end
end
```

### Enumeration of Movement States

```
classdef enumMove
    enumeration
        Moving, Static, Stretching
    end
end
```

### Enumeration of Node Types

```
classdef enumNType
```

```
enumeration
    SVGN,... Static Volume Gas Node
    VVGN,...Variable Volume Gas Node
    SAGN,...Shearing Annular Gas Node
    SN,...Solid Node
    EN % Environment Node
end
end
```

## Enumeration of Orientations

```
classdef enumOrient
    enumeration
        Vertical, Horizontal
    end
end
```

## Enumeration of Shapes

```
classdef enumShape
    enumeration
        Cylinder, Annulus, Cuboid
    end
end
```

## G.2. Startup

This function automatically runs whenever this project is opened in MATLAB, due to being named “startup”.

```
function startup()
    addpath(...
        'enum',...
        'Helper Function',...
        'Saved Files',...
        'Geometry',...
        'MinorElements',...
        'MajorElements',...
        'Mechanical',...
        'Mechanical/Mechanical Helper',...
        'Function - Turb Nusselt',...
        'Function - Turb Friction',...
        'Function - Turb Cond Enhancement',...
        'Function - Leakage',...
        'Function - Laminar Nusselt',...
        'Function - Laminar Friction',...
        'Function - Laminar Cond Enhancement',...
        'Function - Discretization',...
        'Function - Load Function',...
        'GUI',...
        'Simulation',...
        'ListObjs',...
        'Motion',...
        'Test_Running',...
        'Relations',...
        'Optimization',...
        'Output Helpers'...
    );

    % mex anyEq.c -largeArrayDims;
end
```

## G.1. Helper Function

Add Row:

adds a number (N) rows to a 2D cell matrix

```
function Output = AddRow(Input,N)
Output = Input;
for i = size(Output,1)+1:N+size(Output,1)
    for j = 1:size(Output,2)
        Output{i,j} = '';
    end
end
end
```

Annular Conduction:

Determines the conductance from the center of a node to a radius “r” given the node is made up of a material “matl” and the interaction is over a length of “L”.

```
function [ U ] = AnnularConduction(Node,r,L,matl)
if Node.xmin ~= 0
    mid_r = sqrt(Node.xmin*Node.xmax);
    if mid_r < r
        U = (2*pi*matl.ThermalConductivity/log(r/mid_r)).*L;
    else
        U = ((2*pi*matl.ThermalConductivity)/log(mid_r/r)).*L;
    end
else
    % The Constant comes from 1/log(1/0.570524), which is the center
    % ... Non-dimensional radius of: Resistance*Area of a cylinder.
    U = 2*pi*matl.ThermalConductivity*1.781896.*L;
end
end
```

Arcsin:

Modifies the regular discontinuous output of the asin function to be continuous over a given array of sequential values.

```
function [output] = asin_omni(input)
intermittent = zeros(size(input));
for i = 1:length(input)
    intermittent(i) = asin(input(i));
end
count = 0;
output = zeros(size(input));
i = 2;
output(1) = intermittent(1);
d = diff(intermittent);
while i < length(input)+1 && count < 100
    while i < length(input)+1 && d(i-1) >= 0
        output(i) = intermittent(i) + count*pi;
        i = i + 1;
    end
end
```

```

end
count = count + 1;
while i < length(input)+1 && d(i-1) <= 0
    output(i) = count*pi - intermittent(i);
    i = i + 1;
end
count = count + 1;
end
end
end

```

### Assess Triad:

Assess whether or not any of the faces of a triad can be permanently closed, and what the gain or closing each face would be. The use of this is that each triad can be scored, then the faces can be closed via a greedy algorithm. Each triad is reassessed after any faces are closed, as they may interact significantly.

```

function [ scores ] = assessTriad( Triad, derefinement_factor)
a = Triad(1);
b = Triad(2);
c = Triad(3);
modification = sqrt(derefinement_factor);
threshold = 0.1/modification;

scores = [0, 0, 0];

canCloseA = canClose(a);
canCloseB = canClose(b);
canCloseC = canClose(c);

if canCloseA
    for i = 1:Frame.NTheta-1
        Aa = getArea(a,i);
        if Aa == 0; continue; end
        Ab = getArea(b,i);
        if Ab == 0; continue; end
        Ac = getArea(c,i);
        if Ac == 0; continue; end
        r_a = Aa/min(Ab,Ac);
        if r_a < threshold; scores(1) = scores(1) + 1; end
    end
end

if canCloseB
    for i = 1:Frame.NTheta-1
        Aa = getArea(a,i);
        if Aa == 0; continue; end
        Ab = getArea(b,i);
        if Ab == 0; continue; end
        Ac = getArea(c,i);
        if Ac == 0; continue; end
        r_b = Ab/min(Aa,Ac);
        if r_b < threshold; scores(2) = scores(2) + 1; end
    end
end

if canCloseC
    for i = 1:Frame.NTheta-1
        Aa = getArea(a,i);
        if Aa == 0; continue; end
        Ab = getArea(b,i);
        if Ab == 0; continue; end
    end
end

```



```

    Ac = getArea(c,i);
    if Ac == 0; continue; end
    r_c = Ac/min(Aa,Ab);
    if r_c < threshold; scores(3) = scores(3) + 1; end
end
end
end

```

**Atan:**

Modified to provide a continuous output over a set of sequential input values.

```

function [input] = atanSmooth(input)
input = atan(input);
for i = 2:length(input)
    if abs(input(i-1) - input(i)) > 3
        input(i:end) = input(i:end) + sign(input(i-1)-input(i))*pi;
    end
end
end

```

**Can Close:**

determines if a face can be closed without dividing the region

```

function [itcan] = canClose( fc )
if isfield(fc.data,'Area')
    if all(fc.data.Area > 0)
        % Path from one side to the other
        [itcan] = canPathTo(fc, fc.Nodes(1), fc.Nodes(2));
    else; itcan = true;
    end
else
    itcan = false;
end
end

function [canPath, visited] = canPathTo(visited, target, start)
canPath = false;
for fc = start.Faces
    if (fc.Type == enumFType.Gas || ...
        fc.Type == enumFType.MatrixTransition) && ...
        ~any(fc == visited)
        % Make sure the face is traversible
        if all(fc.data.Area > 0)
            if fc.Nodes(1) == start; i = 2; else; i = 1; end
            % Test for completion
            if fc.Nodes(i) == target
                canPath = true; return;
            else
                % Continue Searching
                for fci = fc.Nodes(i).Faces
                    [canPath, visited] = ...
                        canPathTo([visited fc], target, fc.Nodes(i));
                    if canPath; return; end
                end
            end
        end
    end
end
end
end
end
end

```

## Collapse vector:

collapses vector that are repetitive to save effort later.

```
function [ Input ] = CollapseVector( Input )
    if all(Input(1) - 1e-8 < Input) && all(Input(1) + 1e-8 > Input)
        Input = Input(1);
    end
end
```

## Combine Conduction in Series:

combines a pair of conductance's as if they were attached in series.

```
function [ U ] = combine_Conduction_Series( U1, U2 )
if U1 == 0
    U = U2;
else
    if U2 == 0
        U = 0;
    else
        U = 1/(1/U1+1/U2);
    end
end
end
```

## Copy Class:

Copies a handle object, property for property. Children objects are not copied, by default.

```
function newObj = CopyClass(existingObj)
    newObj = feval(class(existingObj));
    props = properties(existingObj);
    for i = 1:length(props)
        newObj.(props{i}) = existingObj.(props{i});
    end
end
```

## Custom Expand Array

Expands array by padding it in all directions

```
function [newarray] = CustExpandArray(array)
    newarray = zeros(size(array) + [2 2]);
    % Identical core of the array
    newarray(2:(1+size(array,1)),2:(1+size(array,2))) = array(:,:);
    % Four Edges
    newarray(1,2:(1+size(array,2))) = array(1,:);
    newarray(end,2:(1+size(array,2))) = array(end,:);
    newarray(2:(1+size(array,1)),1) = array(:,1);
    newarray(2:(1+size(array,1)),end) = array(:,end);
    % Four Corners
    newarray(1,1) = newarray(1,2);
    newarray(1,end) = newarray(1,end-1);
    newarray(end,1) = newarray(end,2);
    newarray(end,end) = newarray(end,end-1);
end
```

## Custom RSSQ

Calculates the root sum of squares of a vector. This function exists but is the only function that comes from an add-on library to MATLAB.

```
function [out] = CustomRSSQ(in)
    out = sqrt(sum(in.*in));
end
```

## Debug\_loop plot:

Plots all loops on the GUI window

```
function [ ] = debug_loopPlot( Model, Closed)
h = figure();
if nargin > 1
    if length(Closed) == 1
        if ~Closed
            for fc = Model.Faces
                if fc.Type == enumFType.Gas || fc.Type == enumFType.MatrixTransition
                    c1 = fc.Nodes(1).minCenterCoords();
                    c2 = fc.Nodes(2).minCenterCoords();
                    line([c1.x; c2.x], [c1.y; c2.y]);
                end
            end
        end
    else
        for fc = Model.Faces
            if fc.Type == enumFType.Gas || fc.Type == enumFType.MatrixTransition
                if ~Closed(fc.index)
                    c1 = fc.Nodes(1).minCenterCoords();
                    c2 = fc.Nodes(2).minCenterCoords();
                    line([c1.x; c2.x], [c1.y; c2.y]);
                end
            end
        end
    end
else
    for fc = Model.Faces
        if fc.Type == enumFType.Gas || fc.Type == enumFType.MatrixTransition
            if all(fc.data.Area > 0)
                c1 = fc.Nodes(1).minCenterCoords();
                c2 = fc.Nodes(2).minCenterCoords();
                line([c1.x; c2.x], [c1.y; c2.y]);
            end
        end
    end
end
close(h);
end
```

## Distance to Rectangle

Returns the distance that a point “P” composed of Px and Py is from a rectangle with center “Cx, Cy”, width and height.

```

function [ d ] = Dist2Rect( Px,Py,Cx,Cy,width,height )
% 9 cases
w = width/2;
h = height/2;
if Px < Cx+w
    if Px > Cx-w
        if Py < Cy+h
            if Py > Cy-h % Bounded, Bounded (Inside)
                d = 0;
            else % Under, Bounded
                d = ((Cy-h)-Py)^2;
            end
        else % Above, Bounded
            d = (Py-(Cy+h))^2;
        end
    else
        if Py < Cy+h % Under Top Surface, Left
            if Py > Cy-h % Bounded, Left
                d = (Cx-w-Px)^2;
            else % Under, Left
                d = ((Cx-w)-Px)^2+((Cy-h)-Py)^2;
            end
        else % Above, Left
            d = ((Cx-w)-Px)^2+(Py-(Cy+h))^2;
        end
    end
end
else
    if Py < Cy+h
        if Py > Cy-h % Bounded, Right
            d = (Px-(Cx+w))^2;
        else % Under, Right
            d = ((Cy-h)-Py)^2+(Px-(Cx+w))^2;
        end
    else % Above, Right
        d = (Py-(Cy+h))^2+(Px-(Cx+w))^2;
    end
end
end
end

```

## Distance for comparison

Calculates the square distance, avoiding the square root.

```

function [d] = Dist4Compare(A,B)
d = (A(1)-B(1))^2+(A(2)-B(2))^2;
end

```

## Distort Position Vector

Distorts a set of x and y vectors, representing a set of points by a rotation and then a shift in 2D space.

```

function [oXData,oYData] = DistortPositionVectors(iXData,iYData,shift,rotate)
if nargin > 3 && all(size(rotate) == [2 2])
    oXData = rotate(1,1)*iXData + rotate(1,2)*iYData;
    oYData = rotate(2,1)*iXData + rotate(2,2)*iYData;
if length(shift) == 2
    oXData = shift(1) + oXData;
    oYData = shift(2) + oYData;
end
end

```

```

    end
    return;
end
if nargin > 2 && length(shift) == 2
    oXData = shift(1) + iXData;
    oYData = shift(2) + iYData;
end
end
end

```

## Face Motion

Given a face calculates (if any) the shear and velocity factors and if possible returns these values in addition to a new shear contact in the event of a shear. It may return more than 1 shear contact.

```

function [ V, S, SContact ] = FaceMotion(Fc)
V = [];
S = [];
SContact = ShearContact.empty;
% Only if the face is horizontal and is gas
if Fc.Orient ~= enumOrient.Horizontal; return; end
n1 = Fc.Nodes(1);
n2 = Fc.Nodes(2);
if n1.Type == enumNType.SN || n2.Type == enumNType.SN; return; end

% factor = omega/N [rad/step]
h = 2*pi/(Frame.NTheta-1);

%% Self Motion
% Produce V_middle - of units [m/rad]
if all(n1.ymin == n2.ymax)
    % Oriented towards the negative
    sgn = 1;
    V_middle = zeros(1,Frame.NTheta);
    y = n1.ymin;
    if ~isscalar(y)
        V_middle(1) = (y(2)-y(end-1))/(2*h);
        V_middle(2:end) = (y(3:end)-y(1:end-2))/(2*h);
    end
elseif all(n1.ymax == n2.ymin)
    % Oriented towards the positive
    sgn = -1;
    V_middle = zeros(1,Frame.NTheta);
    y = n2.ymin;
    if ~isscalar(y)
        V_middle(1) = (y(2)-y(end-1))/(2*h);
        V_middle(2:end-1) = (y(3:end)-y(1:end-2))/(2*h);
        V_middle(end) = V_middle(1);
    end
else
    % They are connected, but through a bridge.
    return;
end

%% Adjacent Motion
% Find the adjacent surfaces motion
% For each frame of motion detect whether or not the face is adjacent to a
% ... solid surface.
Bodies_1_inside = Body.empty;
Area_1_inside = cell_of_zeros(length(n1.Faces));
Bodies_1_outside = Body.empty;
Area_1_outside = cell_of_zeros(length(n1.Faces));
Bodies_2_inside = Body.empty;
Area_2_inside = cell_of_zeros(length(n2.Faces));
Bodies_2_outside = Body.empty;
Area_2_outside = cell_of_zeros(length(n2.Faces));

```

```

% Test the connections of node 1
for fc = n1.Faces
    if fc.Orient == enumOrient.Vertical
        % Only if it is a mixed face will it populate "nd"
        nd = [];
        if fc.Nodes(1).Type == enumNType.SN; nd = fc.Nodes(1);
        elseif fc.Nodes(2).Type == enumNType.SN; nd = fc.Nodes(2);
        end
        if ~isempty(nd)
            found = false;

            if nd.xmin < n1.xmin % It is closer to the axis than the original node
                for i = 1:length(Bodies_1_inside)
                    if Bodies_1_inside(i) == nd.Body
                        Area_1_inside{i} = Area_1_inside{i} + fc.data.Area;
                        found = true;
                    end
                end
            end
            if ~found
                Bodies_1_inside(end+1) = nd.Body;
                Area_1_inside(length(Bodies_1_inside)) = ...
                    Area_1_inside(length(Bodies_1_inside)) + fc.data.Area;
            end
        else % It is farther from the axis than the original node
            for i = 1:length(Bodies_1_outside)
                if Bodies_1_outside(i) == nd.Body
                    Area_1_outside{i} = Area_1_outside{i} + fc.data.Area;
                    found = true;
                end
            end
            if ~found
                Bodies_1_outside(end+1) = nd.Body;
                Area_1_outside(length(Bodies_1_outside)) = ...
                    Area_1_outside(length(Bodies_1_outside)) + fc.data.Area;
            end
        end
    end
end
end

% Test the connections of node 2
for fc = n2.Faces
    if fc.Orient == enumOrient.Vertical
        % Only if it is a mixed face will it populate "nd"
        nd = [];
        if fc.Nodes(1).Type == enumNType.SN
            nd = fc.Nodes(1);
        elseif fc.Nodes(2).Type == enumNType.SN
            nd = fc.Nodes(2);
        end
        if ~isempty(nd)
            found = false;
            if nd.xmin < n2.xmin % It is closer to the axis than the original node
                for i = 1:length(Bodies_2_inside)
                    if Bodies_2_inside(i) == nd.Body
                        Area_2_inside{i} = Area_2_inside{i} + fc.data.Area;
                        found = true;
                    end
                end
            end
            if ~found
                Bodies_2_inside(end+1) = nd.Body;
                Area_2_inside(length(Bodies_2_inside)) = ...
                    Area_2_inside(length(Bodies_2_inside)) + fc.data.Area;
            end
        else % It is farther from the axis than the original node
            for i = 1:length(Bodies_2_outside)
                if Bodies_2_outside(i) == nd.Body
                    Area_2_outside{i} = Area_2_outside{i} + fc.data.Area;
                    found = true;
                end
            end
        end
    end
end
end

```

```

        end
        if ~found
            Bodies_2_outside(end+1) = nd.Body;
            Area_2_outside{length(Bodies_2_outside)} = ...
                Area_2_outside{length(Bodies_2_outside)} + fc.data.Area;
        end
    end
end
end
end

% Determine inner absolute speeds
% ... If no bodies are shared, then speed is equal to the face velocity to
% ... represent a low shear condition.
V_inner = V_middle; i = 1;
shared_inner = false(size(Bodies_1_inside));
for iBody = Bodies_1_inside
    j = 1;
    for oBody = Bodies_2_inside
        if iBody == oBody
            shared_inner(i) = true;
            overlap = and(Area_1_inside{i} > 0, Area_2_inside{j} > 0);
            motion = get_motion(iBody);
            if length(overlap) == 1 && overlap
                V_inner = motion;
                break;
            else
                V_inner(overlap) = motion(overlap);
            end
        end
        j = j + 1;
    end
    i = i + 1;
end

% Determine outer absolute speeds
% ... If no bodies are shared, then speed is equal to the face velocity to
% ... represent a low shear condition.
shared_outer = false(size(Bodies_1_outside));
V_outer = V_middle; i = 1;
for iBody = Bodies_1_outside
    j = 1;
    for oBody = Bodies_2_outside
        if iBody == oBody
            shared_outer(i) = true;
            overlap = and(Area_1_outside{i} > 0, Area_2_outside{j} > 0);
            motion = get_motion(iBody);
            if length(overlap) == 1 && overlap
                V_outer = motion;
                break;
            else
                V_outer(overlap) = motion(overlap);
            end
        end
        j = j + 1;
    end
    i = i + 1;
end

% Assign V and S vectors
if n1.xmin == 0 && n2.xmin == 0
    V = (V_middle - V_outer);
    S = [];
else
    V = V_middle - (V_inner + V_outer)/2;
    S = abs(V_inner - V_outer);
end

Frames = Frame.empty;
N = length(Bodies_1_inside(shared_inner)) + ...

```

```

    length(Bodies_1_outside(shared_outer));
ActiveTimes = cell(N, 1);
for i = 1:N; ActiveTimes{i} = 0; end
for i = 1:length(Bodies_1_inside)
    if shared_inner(i)
        Frm = Bodies_1_inside(i).get('RefFrame');
        if ~isempty(Frm)
            found = false;
            for j = 1:length(Frames)
                if Frames(j) == Frm
                    found = true;
                    % Add active Points to array
                    ActiveTimes{j} = or(ActiveTimes{j}, ...
                        and(Area_1_inside{i} > 0, Area_2_inside{i} > 0));
                    break;
                end
            end
            if ~found
                Frames(end+1) = Frm;
                ActiveTimes{length(ActiveTimes),1} = ...
                    and(Area_1_inside{i} > 0, Area_2_inside{i} > 0);
            end
        end
    end
end
for i = 1:length(Bodies_1_outside)
    if shared_outer(i)
        Frm = Bodies_1_outside(i).get('RefFrame');
        if ~isempty(Frm)
            found = false;
            for j = 1:length(Frames)
                if Frames(j) == Frm
                    found = true;
                    % Add active Points to array
                    ActiveTimes{j} = or(ActiveTimes{j}, ...
                        and(Area_1_outside{i} > 0, Area_2_outside{i} > 0));
                    break;
                end
            end
            if ~found
                Frames(end+1) = Frm;
                ActiveTimes{length(ActiveTimes),1} = ...
                    and(Area_1_outside{i} > 0, Area_2_outside{i} > 0);
            end
        end
    end
end

% So we now have the reference frames that move past the face, as well as
% ... the times that they are active for.
if isempty(Frames)
    SContact = ShearContact.empty;
else
    SContact(length(Frames)) = ShearContact(0);
    Converters = n1.Body.Group.Model.Converters;
    i = 1;
    for Frm = Frames
        for Converter_id = 1:length(Converters)
            if Frm.Mechanism == Converters(Converter_id)
                break;
            end
        end
        if n1.ymin(1) < n2.ymin(1)
            SContact(i) = ShearContact(...
                Converter_id, Frm.MechanismIndex, Fc.data.Area/2, n1, n2, ActiveTimes{i});
        else
            SContact(i) = ShearContact(...
                Converter_id, Frm.MechanismIndex, Fc.data.Area/2, n2, n1, ActiveTimes{i});
        end
        i = i + 1;
    end
end

```



```

end

% Currently the gas velocity is oriented towards the negative direction
% ... Correct
if all(V < 1e-4) && all(V > -1e-4)
    V = [];
else
    V = sgn*V;
end

if all(S < 1e-4)
    S = [];
end
end

function [a] = cell_of_zeros(len)
    a = cell(len,1);
    for i = 1:len; a{i} = 0; end
end

function [motion] = get_motion(iBody)
    [y, ~, ~, ~] = iBody.limits(enumOrient.Horizontal);
    % factor = omega/N [rad/step]
    h = 2*pi/(Frame.NTheta-1);
    motion = zeros(1,Frame.NTheta);
    if ~isscalar(y)
        motion(1) = (y(2)-y(end-1))/(2*h);
        motion(2:end-1) = (y(3:end)-y(1:end-2))/(2*h);
        motion(end) = motion(1);
    end
end
end

```

## Find Closest 2 Nodes

Determines, from a set of nodes the set of 2 nodes that are closest to a position.

```

function [ iNodes ] = findClosest2(loc, iNodes )
    selection = zeros(1,2);
    d = zeros(1,4);
    for i = 1:length(iNodes)
        pnts = iNodes.minCenterCoords;
        d(i) = (pnts.x - loc.x)^2 + (pnts.y - loc.y)^2;
    end
    n = 1;
    while true
        dmin = d(1);
        k = 1;
        for i = 2:length(d)
            if dmin > d(i)
                dmin = d(i);
                k = i;
            end
        end
        if dmin == inf
            iNodes = iNodes(selection(1:n));
            return;
        end
        d(k) = inf;
        selection(n) = k;
        n = n + 1;
        if n == 3
            iNodes = iNodes(selection);
            return;
        end
    end
end
end

```

end

## Find Closest 4 Nodes

Determines, from a set of nodes the set of 4 nodes that are the closest.

```
function [oNodes,Interp] = findClosest4(loc,Body)
% Find distance to all nodes
% Find the members of the body who's material matches the phase of the
% body material.
iNodes = Body.Nodes;
Phase = Body.matl.Phase;
include = true(size(iNodes));
for i = 1:length(iNodes)
    if ~isvalid(iNodes(i))
        include(i) = false;
        continue;
    end
    if isfield(iNodes(i).data,'matl') && iNodes(i).data.matl.Phase ~= Phase
        include(i) = false;
        continue;
    end
end
iNodes = iNodes(include);
if isempty(iNodes)
    oNodes = [];
    Interp = [];
    return;
end
dist2 = zeros(length(iNodes),1);
i = 1;
for Nd = iNodes
    NodeCenter = Nd.minCenterCoords;
    dist2(i) = (NodeCenter.x-loc.x)^2+(NodeCenter.y-loc.y)^2;
    if dist2(i) == 0
        oNodes = Nd.index;
        Interp = 1;
        return;
    end
    i = i + 1;
end

% Sort the dist and node array
tNodes = Node.empty;
if length(iNodes) > 3
    limit = 4;
else
    limit = 2;
end
if ~isempty(iNodes)
    notdone = true;
    while notdone
        d1 = min(dist2);
        if d1 == Inf || length(tNodes) == limit
            [Interp, tNodes] = interpCoefficients(loc,tNodes);
            oNodes = zeros(1,length(tNodes));
            for i = 1:length(tNodes)
                oNodes(i) = tNodes(i).index;
            end
            return;
        end
        MinNodes = iNodes(dist2==d1);
        maxlen = min(length(MinNodes),limit - length(tNodes));
        tNodes(end+1:end+maxlen) = MinNodes(1:maxlen);
        dist2(dist2==d1) = Inf;
    end
end
end
```

```

Interp = [];
end

```

## Find String In Cell

Finds, in a cell vector, a particular string and returns the index.

```

function [index] = FindStringInCell(iCell,iString)
index = 0;
for i = 1:length(iCell)
    if strcmp(iCell{i},iString)
        index = i;
        break;
    end
end
end
end

```

## Func 2 Lookup

Unused: in favor of vectorized anonymous functions. This function takes a function, a range of values and a tolerance and computes an adaptive lookup table that covers the prescribed range of inputs.

```

function [x,v] = Func2Lookup(Func,xmin,xmax,vTol)
tic;
N = 200; n = 3;
xs = zeros(N,1); vs = xs; edges = xs; active = xs;
xs(1) = xmin; vs(1) = Func(xmin); edges(1) = 2; active(1) = true;
xs(2) = xmax; vs(2) = Func(xmax); edges(2) = 0; %active(2) = false;
while(any(active))
    for i = 1:n-1
        if active(i)
            % Test 5 points
            dx = (xs(edges(i))-xs(i))/6;
            m = (vs(edges(i))-vs(i))/(xs(edges(i))-xs(i));
            xs(n) = xs(i) + 3*dx;
            vs(n) = Func(xs(n));
            if abs(vs(n) - ((xs(n) - xs(i))*m + vs(i))) > vTol
                edges(n) = edges(i);
                edges(i) = n;
                active(n) = true;
                n = n + 1;
                break;
            else
                xs(n) = xs(i) + 2*dx;
                vs(n) = Func(xs(n));
                if abs(vs(n) - ((xs(n) - xs(i))*m + vs(i))) > vTol
                    edges(n) = edges(i);
                    edges(i) = n;
                    active(n) = true;
                    n = n + 1;
                    break;
                else
                    xs(n) = xs(i) + 4*dx;
                    vs(n) = Func(xs(n));
                    if abs(vs(n) - ((xs(n) - xs(i))*m + vs(i))) > vTol
                        edges(n) = edges(i);
                        edges(i) = n;
                        active(n) = true;
                    end
                end
            end
        end
    end
end

```

```

        n = n + 1;
        break;
    else
        xs(n) = xs(i) + dx;
        vs(n) = Func(xs(n));
        if abs(vs(n) - ((xs(n) - xs(i))*m + vs(i))) > vTol
            edges(n) = edges(i);
            edges(i) = n;
            active(n) = true;
            n = n + 1;
            break;
        else
            xs(n) = xs(i) + 5*dx;
            vs(n) = Func(xs(n));
            if abs(vs(n) - ((xs(n) - xs(i))*m + vs(i))) > vTol
                edges(n) = edges(i);
                edges(i) = n;
                active(n) = true;
                n = n + 1;
                break;
            else
                active(i) = false;
            end
        end
    end
end
end
end
end
if n > N
    break;
end
end
if n > N
    break;
end
end
x = zeros(n-1,1);
v = zeros(n-1,1);
x(1) = xs(1);
v(1) = vs(1);
next = edges(1);
for i = 2:n-1
    x(i) = xs(next);
    v(i) = vs(next);
    next = edges(next);
end
plot(x,v,'o',xmin:0.01:xmax,Func(xmin:0.01:xmax));
toc;
end

```

## Function Table

Unused: in favor of vectorized anonymous functions. Determines the set of unique functions and calculates a set length lookup table.

```

function [Indexes, Lookups] = Function_Table(...
    Tol, min_x, max_x, Functions, N)
LEN = length(Functions);
Lookups = zeros(LEN,N,2);
Indexes = zeros(LEN,1);
isUnique = false(size(Indexes));
n = 1;
for i = 1:LEN
    if ~isempty(Functions{i}) && Indexes(i) == 0
        isUnique(i) = true;
    end
end

```

```

    Indexes(i) = n;
    for j = i+1:LEN
        if ~isempty(Functions{j}) && Indexes(j) == 0
            if Indexes(j) == 0 && Functions{i}(0.5) == Functions{j}(0.5) && Functions{i}(1) ==
Functions{j}(1)
                Indexes(j) = n;
            end
        end
    end
    end
    n = n + 1;
end
end

for i = 1:LEN
    f = Functions{i};
    if ~isempty(f)
        if isUnique(i)
            n = Indexes(i);
            entry = 2;
            delta = 1;
            Lookups(n,1,1) = min_x;
            Lookups(n,1,2) = f(min_x);
            x2 = 1e-8;
            y2 = Lookups(n,1,2);
            while x2 < max_x && entry <= N
                % Use the Tolerance to construct a lookup table from the function
                x1 = x2;
                y1 = y2;
                max_delta = inf;
                min_delta = 1e-8;
                delta = delta * 1.1;
                locating = true;
                Tries = 1;
                while locating && Tries < 10
                    x2 = Lookups(n,entry-1,1) + delta;
                    y2 = f(x2);
                    if delta == min_delta || delta == max_delta
                        locating = false;
                    else
                        ymid = f((x1 + x2)/2);
                        err = ((y1 + y2) - 2*ymid)/ymid;
                        if abs(err) > Tol
                            factor = 0.99*sqrt(Tol/abs(err));
                            max_delta = delta*(1 + factor)/2;
                            delta = max(min_delta,delta*factor);
                        elseif abs(err) < Tol*0.9
                            factor = sqrt(Tol/abs(err));
                            min_delta = delta*(1 + factor)/2;
                            delta = min(max_delta,delta*factor);
                        else
                            locating = false;
                        end
                        Tries = Tries + 1;
                    end
                end
                if Tries == 10
                    x2 = (min_delta + max_delta)/2;
                    y2 = f(x2);
                end
                Lookups(n,entry,1) = x2;
                Lookups(n,entry,2) = y2;
                entry = entry + 1;
            end
        end
    end
end
end
if n < LEN
    Lookups = Lookups(1:n, :, :);
end
end
end

```

## Get Any Fc Loop of Size

Used in debugging, found loops of size 2 which were subsequently fixed.

```
function [success, visited] = getAnyFcLoopOfSize(visited, target, start, max_length)
    success = false;
    for fc = start.Faces
        if visited(end) ~= fc && (fc.Type == enumFType.Gas || ...
            fc.Type == enumFType.MatrixTransition)
            % Make sure the face is traversible
            if fc.Nodes(1) == start; i = 2; else; i = 1; end
            % Test for completion
            if fc.Nodes(i) == target
                success = true; visited = [visited fc]; return;
            else
                % Length Check
                if length(visited) + 1 == max_length
                    success = false; return;
                else
                    % Continue Searching
                    for fci = fc.Nodes(i).Faces
                        [success, new_visited] = getAnyFcLoopOfSize(...
                            [visited fc], target, fc.Nodes(i), max_length);
                        if success
                            visited = new_visited;
                            return;
                        end
                    end
                end
            end
        end
    end
end
end
end
end
end
end
```

## Get Area Percent Horizontal

Calculates the percentage that an offset circle of offset “x” and diameter “d” covers a centered ring of inner radius “r1” and outer radius “r2”.

```
function [ area_perc ] = GetAreaPercentHorizontal( x, r1, r2, d )
%GETAREAPERCENTHORIZONTAL Summary of this function goes here
%   Calculates the percentage that an offset circle covers a ring between
%   r1 and r2
x = abs(x);
if d == 0 || x - d/2 > r2 || x + d/2 < r1; area_perc = 0; return; end
if x-d/2 < -r2 && x+d/2 > r2; area_perc = 1; return; end

c_r1 = max([x-d/2 r1]);
c_r2 = min([x+d/2 r2]);

N = 100;
r = linspace(c_r1,c_r2,N);
r = (r(1:end-1)+r(2:end))/2; % Center the radius's
dr = (c_r2-c_r1)/(N-1);
area = 0;

if x > d/2
    for ri = r
        % Center is outside of circle, angle can never be larger than pi/2
        area = area + 2*dr*ri*acos((ri^2+x^2-(d/2)^2)/(2*ri*x));
    end
end
```

```

end
else
for ri = r
if d/2 >= ri + x
% Full Circle
area = area + 2*dr*ri*pi;
else
% Center is inside of circle
area = area + 2*dr*ri*acos((ri^2+x^2-(d/2)^2)/(2*ri*x));
end
end
end
area_perc = area/(pi*(r2^2-r1^2));
end

```

## Get Area Percent Mix

Calculates the percentage that a circle of vertical offset “x” and diameter “d” covers the side of a thin annular shell of radius “r” and extents “y1” and “y2”.

```

function [ PercArea ] = GetAreaPercentMix(r, x, y1, y2, d )
%GETAREAPERCENTMIX Summary of this function goes here
% Calculates the percentage that an circle at x of diameter d covers a
% strip between y1 and y2
Total_Area = 2*pi*r*(y2-y1);
c_y1 = max([x-d/2 y1]);
c_y2 = min([x+d/2 y2]);
if c_y1 >= c_y2; PercArea = 0; return; end
N = max([2 floor(100*(c_y2-c_y1)/d)]);
y = linspace(c_y1,c_y2,N);
y = (y(1:end-1)+y(2:end))/2;
dy = (c_y2-c_y1)/(N-1);
area = 0;
for yi = y
area = area + 2*dy*min([r sqrt((d/2)^2-(x-yi)^2)]);
end
PercArea = area/Total_Area;
end

```

## Get Center of Overlap Region

From 2 overlaps find and return the center of overlapping segment.

```

function [c] = getCenterOfOverlapRegion(min1,min2,max1,max2)
if isscalar(min1)
if isscalar(min2)
temp1 = max(min1,min2);
else
temp1 = max([min1(ones(size(min2))); min2]);
end
else
if isscalar(min2)
temp1 = max([min2(ones(size(min1))); min1]);
else
temp1 = max([min1; min2]);
end
end
if isscalar(max1)
if isscalar(max2)
temp2 = min(max1,max2);
else

```

```

    temp2 = min([max1(ones(size(max2))); max2]);
end
else
    if isscalar(max2)
        temp2 = min([max2(ones(size(max1))); max1]);
    else
        temp2 = min([max1; max2]);
    end
end
end
c = (temp1 + temp2)/2;
end

```

## Get Face Type

From two nodes of types “Type1” and “Type2”, get the face type that would be produced.

```

function NType = getFaceType(Type1,Type2)
    if Type1 ~= Type2
        NType = enumFType.Mix;
    else
        NType = Type1;
    end
end

```

## Get First Derivative

Get first derivative with respect to angle, given a vector that is assumed to cover the entire set of angles.

```

function [Var] = getFirstDer(Pos)
    Var(length(Pos)) = (Pos(2)-Pos(end-1));
    Var(2:end-1) = (Pos(3:end)-Pos(1:end-2));
    Var(1) = Var(end);
    denominator = (4*pi/(Frame.NTheta-1));
    Var = (Var/denominator)';
end

```

## Get Proper Name

Opens up a generic userform that requires a name, and validates this name.

```

function [ answer ] = getProperName( ObjectName )
    answer{1} = '/';
    trial = 0;
    while regexp(answer{1},'[/\*:?"<>|]', 'once')
        if trial > 0
            msgbox(['You cannot have any of the following ' ...
                'characters [/\*:?"<>|] in a file name']);
        end
        trial = trial + 1;
        answer = inputdlg(['Enter a descriptive name for the ' ObjectName],...
            'Name(filename,title,etc...):',[1 50]);
        if isempty(answer{1})
            answer = '';
            return;
        end
    end
end

```



```

end
answer = answer{1};
end

```

## Get Second Derivative

Get second derivative with respect to angle, given a vector that is assumed to cover the entire set of angles.

```

function [Var] = getSecondDer(Pos)
    Var(length(Pos)) = (Pos(2)-2*Pos(end)+Pos(end-1));
    Var(2:end-1) = (Pos(3:end)-2*Pos(2:end-1)+Pos(1:end-2));
    Var(1) = Var(end);
    denominator = ((2*pi)/(Frame.NTheta-1))^2;
    Var = (Var/denominator)';
end

```

## Get Triad

Get the triad associated with node n1.

```

function [ Triad ] = GetTriad( n1 )
%GETTRIAD Summary of this function goes here
% Detailed explanation goes here
Triad = cell(0);
count = 1;
for f1 = n1.Faces
    if isGasFace(f1)
        if f1.Nodes(1) == n1; n2 = f1.Nodes(2); else; n2 = f1.Nodes(1); end
        for f2 = n2.Faces
            if isGasFace(f2) && f2 ~= f1
                if f2.Nodes(1) == n2; n3 = f2.Nodes(2); else; n3 = f2.Nodes(1); end
                if n1 == n3; continue; end
                for f3 = n3.Faces
                    if isGasFace(f3) && f3 ~= f2 && f2 ~= f1
                        if f3.Nodes(1) == n1 || f3.Nodes(2) == n1
                            Triad{count} = [f1 f2 f3];
                            count = count + 1;
                            break;
                        end
                    end
                end
            end
        end
    end
end
end
end
end
end
end
end

function isit = isGasFace(fc)
    isit = fc.Type == enumFType.Gas || fc.Type == enumFType.MatrixTransition;
end

```

## Holder (shell class)

Unused class that holds a value type object such that it can be referenced via just the pointer without copying.

```

classdef Holder < handle
    %HOLDER Summary of this class goes here
    % Detailed explanation goes here

    properties
        vars cell;
    end

    methods
        function obj = Holder(ivars)
            obj.vars = ivars;
        end
    end
end
end

```

## iif

### Inline if condition.

```

function out = iif(cond,a,b)
%IIF implements a ternary operator

% pre-assign out
out = repmat(b,size(cond));

out(cond) = a;
end

```

## Interp Coefficients

Provides the interpolation coefficients given a set of (at most) 4 nodes to calculate the value of a property at a location. This can pair down the set to 1 or 2 nodes as well.

```

function [ Interp, iNodes ] = interpCoefficients(loc, iNodes)
%INTERPCOEFFICIENTS Summary of this function goes here
% Detailed explanation goes here

% Given 4 or less nodes
notdone = true;
while notdone
    switch length(iNodes)
        case 0
            fprintf('ERR: No nodes provided for interpolation into interpCoefficients.m');
            Interp = [];
            iNodes = Node.empty;
            notdone = false;
        case 3
            fprintf('ERR: Invalid number of nodes provided for interpolation into
interpCoefficients.m');
            Interp = [];
            iNodes = Node.empty;
            notdone = false;
        case 1
            Interp = 1;
            notdone = false;
        case 2
            % Ratio of distances
            NC1 = iNodes(1).minCenterCoords;
            NC2 = iNodes(2).minCenterCoords;
            if iNodes(1).xmin == 0; NC1.x = 0; end
            if iNodes(2).xmin == 0; NC2.x = 0; end
            vec = Pnt2D(NC2.x-NC1.x,NC2.y-NC1.y);

```

```

mag = sqrt(vec.x^2+vec.y^2);
vec.x = vec.x/mag;
vec.y = vec.y/mag;
pnt = Pnt2D(loc.x-NC1.x,loc.y-NC1.y);
dot = pnt.x*vec.x + pnt.y*vec.y;
d1 = dot; % Distance to NC1
d2 = mag - dot; % Distance to NC2
Interp(2) = d1/(d1+d2);
Interp(1) = d2/(d1+d2);
notdone = false;
case 4
% Bilinear Interpolation
% Determine if it is between any two points and pick the pair with the
% lowest collective distance
pairs = zeros(3,6);
P(4) = iNodes(4).minCenterCoords;
P(1) = iNodes(1).minCenterCoords;
P(2) = iNodes(2).minCenterCoords;
P(3) = iNodes(3).minCenterCoords;
if iNodes(1).xmin == 0; P(1).x = 0; end
if iNodes(2).xmin == 0; P(2).x = 0; end
if iNodes(3).xmin == 0; P(3).x = 0; end
if iNodes(4).xmin == 0; P(4).x = 0; end
isvertical = true;
x = P(1).x;
for i = 2:4
    if P(i).x ~= x
        isvertical = false;
        break;
    end
end
end

% If they are all vertical then pick one on either side
if isvertical

    % They are all vertically aligned, pick 2
    % Find the closest that is greater than
    greaterthan = 0;
    d = inf;
    for i = 1:4
        if loc.y <= P(i).y
            if P(i).y - loc.y < d
                d = P(i).y - loc.y;
                greaterthan = i;
            end
        end
    end
end

% Find the closest that is less than
lessthan = 0;
d = inf;
for i = 1:4
    if loc.y >= P(i).y
        if loc.y - P(i).y < d
            d = loc.y - P(i).y;
            lessthan = i;
        end
    end
end

% Fill in the gaps with a next closest
if lessthan ~= 0 && greaterthan ~= 0
    [ Interp, iNodes ] = interpCoefficients(loc, iNodes([lessthan; greaterthan]));
else
    if lessthan == 0
        greaterthan2 = 0;
        d = inf;
        for i = 1:4
            if i ~= greaterthan
                if loc.y <= P(i).y
                    if P(i).y - loc.y < d

```

```

        d = P(i).y - loc.y;
        greaterthan2 = i;
    end
end
end
end
    [ Interp, iNodes ] = interpCoefficients(loc, iNodes([greaterthan2;
greaterthan]));
else
    lessthan2 = 0;
    d = inf;
    for i = 1:4
        if i ~= lessthan
            if loc.y >= P(i).y
                if loc.y - P(i).y < d
                    d = loc.y - P(i).y;
                    lessthan2 = i;
                end
            end
        end
    end
    [ Interp, iNodes ] = interpCoefficients(loc, iNodes([lessthan2; lessthan]));
end
end
else
    ishorizontal = true;
    y = P(1).y;
    for i = 2:4
        if P(i).y ~= y
            ishorizontal = false;
            break;
        end
    end
end
if ishorizontal
    % They are all horizontally aligned, pick 2

    % They are all vertically aligned, pick 2
    % Find the closest that is greater than
    greaterthan = 0;
    d = inf;
    for i = 1:4
        if loc.x <= P(i).x
            if P(i).x - loc.x < d
                d = P(i).x - loc.x;
                greaterthan = i;
            end
        end
    end

    % Find the closest that is less than
    lessthan = 0;
    d = inf;
    for i = 1:4
        if loc.x >= P(i).x
            if loc.x - P(i).x < d
                d = loc.x - P(i).x;
                lessthan = i;
            end
        end
    end

    % Fill in the gaps with a next closest
    if lessthan ~= 0 && greaterthan ~= 0
        [ Interp, iNodes ] = interpCoefficients(loc, iNodes([lessthan; greaterthan]));
    else
        if lessthan == 0
            greaterthan2 = 0;
            d = inf;
            for i = 1:4
                if i ~= greaterthan

```

```

        if loc.x <= P(i).x
            if P(i).x - loc.x < d
                d = P(i).x - loc.x;
                greaterthan2 = i;
            end
        end
    end
end
[ Interp, iNodes ] = interpCoefficients(loc, iNodes([greaterthan2;
greaterthan]));
else
    lessthan2 = 0;
    d = inf;
    for i = 1:4
        if i ~= lessthan
            if loc.x >= P(i).x
                if loc.x - P(i).x < d
                    d = loc.x - P(i).x;
                    lessthan2 = i;
                end
            end
        end
    end
end
[ Interp, iNodes ] = interpCoefficients(loc, iNodes([lessthan2;
lessthan]));
end
else
    % Linear Interpolate/extrapolate between all 4
    Interp = zeros(1,4);
    %
https://en.wikipedia.org/wiki/Bilinear\_interpolation#:~:text=In%20mathematics%2C%20bilinear%20interpolation%20is,again%20in%20the%20other%20direction.
    x1 = min([P(1).x P(2).x P(3).x P(4).x]);
    x2 = max([P(1).x P(2).x P(3).x P(4).x]);
    y1 = min([P(1).y P(2).y P(3).y P(4).y]);
    y2 = max([P(1).y P(2).y P(3).y P(4).y]);
    Qs = 1:4;
    for i = 1:4
        if P(i).x == x1 && P(i).y == y1; Qs = take(1, i, Qs); break; end
    end
    for i = 1:4
        if P(i).x == x1 && P(i).y == y2; Qs = take(2, i, Qs); break; end
    end
    for i = 1:4
        if P(i).x == x2 && P(i).y == y1; Qs = take(3, i, Qs); break; end
    end
    factor = (1/((x2-x1)*(y2-y1)));
    Interp(Qs(1)) = factor * (x2 - loc.x) * (y2 - loc.y);
    Interp(Qs(2)) = factor * (x2 - loc.x) * (loc.y - y1);
    Interp(Qs(3)) = factor * (loc.x - x1) * (y2 - loc.y);
    Interp(Qs(4)) = factor * (loc.x - x1) * (loc.y - y1);
end
end
notdone = false;

end
end
end

function [input] = swap(index1, index2, input)
    temp = input(index1);
    input(index1) = input(index2);
    input(index2) = temp;
end

function [input] = take(value, index, input)
    for i = 1:length(input)
        if input(i) == value
            input = swap(i, index, input);
            return;
        end
    end
end

```

```

        end
    end
end

```

## Is String Numeric

Test whether the provided string is numerical, applied for all user forms that require a numerical input.

```

function [isit] = isStrNumeric(str)
%ISSTRNUMERIC Summary of this function goes here
% Detailed explanation goes here
isit = all(ismember(str, '0123456789+-.eEdD'));
end

```

## Linear Conduction

Calculates the conductance in the axial direction, through a node.

```

function [U] = LinearConduction(Node,r1,r2,matl)
%ANNULARCONDUCTION (Length, inner_Radius, outer_Radius, Material_Ref)
L = Node.ymax(1) - Node.ymin(1);
U = (2*pi*(r2*r2-r1*r1)*matl.ThermalConductivity)/L;
end

```

## Power From PV

Calculate the power using the pressure and volume data for a single cycle.

```

function [Work] = PowerFromPV(P,V)
Pavg = (P(1:end-1)+P(2:end));
dVol = (V(2:end)-V(1:end-1));
Work = 0.5*sum(Pavg.*dVol);
end

```

## Process Triad

Take a triad and based on a threshold close the prescribed face.

```

function [ ] = processTriad( Triad, derefinement_factor, index_to_close)
target = Triad(index_to_close);
others = Triad(Triad ~= target);
modification = sqrt(derefinement_factor);
threshold = 0.1/modification;
count = Frame.NTheta-1;

canCloseTarget = canClose(target);

if ~canCloseTarget; return; end

for i = 1:Frame.NTheta-1
    At = getArea(target,i);
    if At == 0; count = count - 1; continue; end
    A1 = getArea(others(1),i);
    if A1 == 0; count = count - 1; continue; end
    A2 = getArea(others(2),i);

```

```

if A2 == 0; count = count - 1; continue; end
r = At/min(A1,A2);
if r < threshold
    setArea(target,i,0);
else
    count = count - 1;
end
end
end

fprintf(['Edited ' num2str(count) ' Increments\n']);
end

```

## Propegate Active Faces

Recursive function designed to find all the independent faces of a region.

```

function [ k,Array,Visited ] = PropegateActiveFaces (Nd,Visited,k,Array)
Visited(Nd.index) = true;
if k == length(Array)
    return;
end
for Fc = Nd.Faces
    if Fc.Nodes(1).index <= length(Visited) && ...
        Fc.Nodes(2).index <= length(Visited) && ...
        isfield(Fc.data,'dx') && all(Fc.data.Area > 0)
        if ~Visited(Fc.Nodes(1).index)
            k = k + 1;
            Array(k) = Fc.index;
            [k,Array,Visited] = PropegateActiveFaces (Fc.Nodes(1),Visited,k,Array);
        elseif ~Visited(Fc.Nodes(2).index)
            k = k + 1;
            Array(k) = Fc.index;
            [k,Array,Visited] = PropegateActiveFaces (Fc.Nodes(2),Visited,k,Array);
        end
    end
end
end
end

```

## Propegate Region

Recursive function designed to find all the nodes in a region.

```

function [region] = PropegateRegion (Nd,region,n)
if region(Nd.index) == 0
    region(Nd.index) = n;
    for Fc = Nd.Faces
        % Test if it is a gas face that does not close off
        if isfield(Fc.data,'dx') && all(Fc.data.Area > 0)
            if Fc.Nodes(1) == Nd
                Nd2 = Fc.Nodes(2);
            else
                Nd2 = Fc.Nodes(1);
            end
            if Nd2.index <= length(region)
                region = PropegateRegion (Nd2,region,n);
            end
        end
    end
end
end
end
end

```

## Rotation Matrix

Calculates the rotation matrix in 2D.

```
function [ R ] = RotMatrix( THETA )
%ROTMATRIX Defines the rotation matrix for a vector by the angle THETA
R = [cos(THETA) -sin(THETA); sin(THETA) cos(THETA)];
end
```

## Shift Vector

Shift a vector, which is assumed to cover all the angular increments, by a phase. The vector can be shifted by any fractional amount by linear interpolation of the resulting vector.

```
function [ vector ] = shiftVector( vector, Phase )
N = length(vector);
temp = N*(Phase/(2*pi));
n1 = floor(temp);
frac = temp-n1;
n2 = ceil(temp);
v1 = circshift(vector,-n1);
v2 = circshift(vector,-n2);
vector = (1-frac)*v1 + frac*v2;
end
```

## Progress Bar

Created by Steve Hoelzer. Included here for completeness. Simple popup interface that allows for simple updates on the progress.

```
function progressbar(varargin)
% Description:
% progressbar() provides an indication of the progress of some task using
% graphics and text. Calling progressbar repeatedly will update the figure and
% automatically estimate the amount of time remaining.
% This implementation of progressbar is intended to be extremely simple to use
% while providing a high quality user experience.
%
% Features:
% - Can add progressbar to existing m-files with a single line of code.
% - Supports multiple bars in one figure to show progress of nested loops.
% - Optional labels on bars.
% - Figure closes automatically when task is complete.
% - Only one figure can exist so old figures don't clutter the desktop.
% - Remaining time estimate is accurate even if the figure gets closed.
% - Minimal execution time. Won't slow down code.
% - Randomized color. When a programmer gets bored...
%
% Example Function Calls For Single Bar Usage:
% progressbar % Initialize/reset
% progressbar(0) % Initialize/reset
% progressbar('Label') % Initialize/reset and label the bar
% progressbar(0.5) % Update
% progressbar(1) % Close
%
% Example Function Calls For Multi Bar Usage:
% progressbar(0, 0) % Initialize/reset two bars
% progressbar('A', '') % Initialize/reset two bars with one label
% progressbar('', 'B') % Initialize/reset two bars with one label
```



```

% progressbar('A', 'B')      % Initialize/reset two bars with two labels
% progressbar(0.3)          % Update 1st bar
% progressbar(0.3, [])      % Update 1st bar
% progressbar([], 0.3)      % Update 2nd bar
% progressbar(0.7, 0.9)     % Update both bars
% progressbar(1)            % Close
% progressbar(1, [])        % Close
% progressbar(1, 0.4)       % Close
%
% Notes:
% For best results, call progressbar with all zero (or all string) inputs
% before any processing. This sets the proper starting time reference to
% calculate time remaining.
% Bar color is choosen randomly when the figure is created or reset. Clicking
% the bar will cause a random color change.
%
% Demos:
%   % Single bar
%   m = 500;
%   progressbar % Init single bar
%   for i = 1:m
%       pause(0.01) % Do something important
%       progressbar(i/m) % Update progress bar
%   end
%
%   % Simple multi bar (update one bar at a time)
%   m = 4;
%   n = 3;
%   p = 100;
%   progressbar(0,0,0) % Init 3 bars
%   for i = 1:m
%       progressbar([],0) % Reset 2nd bar
%       for j = 1:n
%           progressbar([],[],0) % Reset 3rd bar
%           for k = 1:p
%               pause(0.01) % Do something important
%               progressbar([],[],k/p) % Update 3rd bar
%           end
%           progressbar([],j/n) % Update 2nd bar
%       end
%       progressbar(i/m) % Update 1st bar
%   end
%
%   % Fancy multi bar (use labels and update all bars at once)
%   m = 4;
%   n = 3;
%   p = 100;
%   progressbar('Monte Carlo Trials','Simulation','Component') % Init 3 bars
%   for i = 1:m
%       for j = 1:n
%           for k = 1:p
%               pause(0.01) % Do something important
%               % Update all bars
%               frac3 = k/p;
%               frac2 = ((j-1) + frac3) / n;
%               frac1 = ((i-1) + frac2) / m;
%               progressbar(frac1, frac2, frac3)
%           end
%       end
%   end
%
% Author:
%   Steve Hoelzer
%
% Revisions:
% 2002-Feb-27 Created function
% 2002-Mar-19 Updated title text order
% 2002-Apr-11 Use floor instead of round for percentdone
% 2002-Jun-06 Updated for speed using patch (Thanks to waitbar.m)
% 2002-Jun-19 Choose random patch color when a new figure is created
% 2002-Jun-24 Click on bar or axes to choose new random color

```

```

% 2002-Jun-27 Calc time left, reset progress bar when fractiondone == 0
% 2002-Jun-28 Remove extraText var, add position var
% 2002-Jul-18 fractiondone input is optional
% 2002-Jul-19 Allow position to specify screen coordinates
% 2002-Jul-22 Clear vars used in color change callback routine
% 2002-Jul-29 Position input is always specified in pixels
% 2002-Sep-09 Change order of title bar text
% 2003-Jun-13 Change 'min' to 'm' because of built in function 'min'
% 2003-Sep-08 Use callback for changing color instead of string
% 2003-Sep-10 Use persistent vars for speed, modify titlebarstr
% 2003-Sep-25 Correct titlebarstr for 0% case
% 2003-Nov-25 Clear all persistent vars when percentdone = 100
% 2004-Jan-22 Cleaner reset process, don't create figure if percentdone = 100
% 2004-Jan-27 Handle incorrect position input
% 2004-Feb-16 Minimum time interval between updates
% 2004-Apr-01 Cleaner process of enforcing minimum time interval
% 2004-Oct-08 Seperate function for timeleftstr, expand to include days
% 2004-Oct-20 Efficient if-else structure for sec2timestr
% 2006-Sep-11 Width is a multiple of height (don't stretch on widescreens)
% 2010-Sep-21 Major overhaul to support multiple bars and add labels
%
persistent progfig progdata lastupdate
% Get inputs
if nargin > 0
    input = varargin;
    ninput = nargin;
else
    % If no inputs, init with a single bar
    input = {0};
    ninput = 1;
end
% If task completed, close figure and clear vars, then exit
if input{1} == 1
    if ishandle(progfig)
        delete(progfig) % Close progress bar
    end
    clear progfig progdata lastupdate % Clear persistent vars
    drawnow
    return
end
% Init reset flag
resetflag = false;
% Set reset flag if first input is a string
if ischar(input{1})
    resetflag = true;
end
% Set reset flag if all inputs are zero
if input{1} == 0
    % If the quick check above passes, need to check all inputs
    if all([input{:}] == 0) && (length([input{:}]) == ninput)
        resetflag = true;
    end
end
% Set reset flag if more inputs than bars
if ninput > length(progdata)
    resetflag = true;
end
% If reset needed, close figure and forget old data
if resetflag
    if ishandle(progfig)
        delete(progfig) % Close progress bar
    end
    progfig = [];
    progdata = []; % Forget obsolete data
end
% Create new progress bar if needed
if ishandle(progfig)
else % This strange if-else works when progfig is empty (~ishandle() does not)

    % Define figure size and axes padding for the single bar case
    height = 0.03;

```

```

width = height * 8;
hpad = 0.02;
vpad = 0.25;

% Figure out how many bars to draw
nbars = max(ninput, length(progdata));

% Adjust figure size and axes padding for number of bars
heightfactor = (1 - vpad) * nbars + vpad;
height = height * heightfactor;
vpad = vpad / heightfactor;

% Initialize progress bar figure
left = (1 - width) / 2;
bottom = (1 - height) / 2;
progfig = figure(...
    'Units', 'normalized',...
    'Position', [left bottom width height],...
    'NumberTitle', 'off',...
    'Resize', 'off',...
    'MenuBar', 'none' );

% Initialize axes, patch, and text for each bar
left = hpad;
width = 1 - 2*hpad;
vpadttotal = vpad * (nbars + 1);
height = (1 - vpadtotal) / nbars;
for ndx = 1:nbars
    % Create axes, patch, and text
    bottom = vpad + (vpad + height) * (nbars - ndx);
    progdata(ndx).progaxes = axes( ...
        'Position', [left bottom width height], ...
        'XLim', [0 1], ...
        'YLim', [0 1], ...
        'Box', 'on', ...
        'ytick', [], ...
        'xtick', [] );
    progdata(ndx).progpatch = patch( ...
        'XData', [0 0 0 0], ...
        'YData', [0 0 1 1] );
    progdata(ndx).progtext = text(0.99, 0.5, '', ...
        'HorizontalAlignment', 'Right', ...
        'FontUnits', 'Normalized', ...
        'FontSize', 0.7 );
    progdata(ndx).proglabel = text(0.01, 0.5, '', ...
        'HorizontalAlignment', 'Left', ...
        'FontUnits', 'Normalized', ...
        'FontSize', 0.7 );
    if ischar(input{ndx})
        set(progdata(ndx).proglabel, 'String', input{ndx})
        input{ndx} = 0;
    end

    % Set callbacks to change color on mouse click
    set(progdata(ndx).progaxes, 'ButtonDownFcn', {@change_color, progdata(ndx).progpatch})
    set(progdata(ndx).progpatch, 'ButtonDownFcn', {@change_color, progdata(ndx).progpatch})
    set(progdata(ndx).progtext, 'ButtonDownFcn', {@change_color, progdata(ndx).progpatch})
    set(progdata(ndx).proglabel, 'ButtonDownFcn', {@change_color, progdata(ndx).progpatch})

    % Pick a random color for this patch
    change_color([], [], progdata(ndx).progpatch)

    % Set starting time reference
    if ~isfield(progdata(ndx), 'starttime') || isempty(progdata(ndx).starttime)
        progdata(ndx).starttime = clock;
    end
end

% Set time of last update to ensure a redraw
lastupdate = clock - 1;

```

```

end
% Process inputs and update state of progdata
for ndx = 1:ninput
    if ~isempty(input{ndx})
        progdata(ndx).fractiondone = input{ndx};
        progdata(ndx).clock = clock;
    end
end
end
% Enforce a minimum time interval between graphics updates
myclock = clock;
if abs(myclock(6) - lastupdate(6)) < 0.01 % Could use etime() but this is faster
    return
end
end
% Update progress patch
for ndx = 1:length(progdata)
    set(progdata(ndx).progpatch, 'XData', ...
        [0, progdata(ndx).fractiondone, progdata(ndx).fractiondone, 0])
end
end
% Update progress text if there is more than one bar
if length(progdata) > 1
    for ndx = 1:length(progdata)
        set(progdata(ndx).progtxt, 'String', ...
            sprintf('%ld%', floor(100*progdata(ndx).fractiondone)))
    end
end
end
% Update progress figure title bar
if progdata(1).fractiondone > 0
    runtime = etime(progdata(1).clock, progdata(1).starttime);
    timeleft = runtime / progdata(1).fractiondone - runtime;
    timeleftstr = sec2timestr(timeleft);
    titlebarstr = sprintf('%2d%    %s remaining', ...
        floor(100*progdata(1).fractiondone), timeleftstr);
else
    titlebarstr = ' 0%';
end
end
set(progfig, 'Name', titlebarstr)
% Force redraw to show changes
drawnow
% Record time of this update
lastupdate = clock;
% -----
function changecolor(h, e, progpatch) %#ok<INUSL>
% Change the color of the progress bar patch
% Prevent color from being too dark or too light
colormin = 1.5;
colormax = 2.8;
thiscolor = rand(1, 3);
while (sum(thiscolor) < colormin) || (sum(thiscolor) > colormax)
    thiscolor = rand(1, 3);
end
set(progpatch, 'FaceColor', thiscolor)
% -----
function timestr = sec2timestr(sec)
% Convert a time measurement from seconds into a human readable string.
% Convert seconds to other units
w = floor(sec/604800); % Weeks
sec = sec - w*604800;
d = floor(sec/86400); % Days
sec = sec - d*86400;
h = floor(sec/3600); % Hours
sec = sec - h*3600;
m = floor(sec/60); % Minutes
sec = sec - m*60;
s = floor(sec); % Seconds
% Create time string
if w > 0
    if w > 9
        timestr = sprintf('%d week', w);
    else
        timestr = sprintf('%d week, %d day', w, d);
    end
end
end

```

```

elseif d > 0
    if d > 9
        timestr = sprintf('%d day', d);
    else
        timestr = sprintf('%d day, %d hr', d, h);
    end
elseif h > 0
    if h > 9
        timestr = sprintf('%d hr', h);
    else
        timestr = sprintf('%d hr, %d min', h, m);
    end
elseif m > 0
    if m > 9
        timestr = sprintf('%d min', m);
    else
        timestr = sprintf('%d min, %d sec', m, s);
    end
else
    timestr = sprintf('%d sec', s);
end

```

## Symbolic Math

Takes a string of mathematical algebra and returns the calculated answer, used in enhancing input boxes with math functionality.

```

function [NumberOutput] = SymbolicMath(StringInput)
% Initialize
StringInput(StringInput==' ') = [];
Intermediates = zeros(3,1);
collapsed = false(size(StringInput));
count = 1;

% Brackets & Basic Numbers
level = 0; basic_number_start = 0;
for i = 1:length(StringInput)
    if strcmp(StringInput(i),'(')
        if level == 0
            start = i;
        end
        level = level + 1;
    elseif strcmp(StringInput(i),')')
        level = level - 1;
        if level == 0
            Intermediates(1,count) = start;
            Intermediates(2,count) = i;
            Intermediates(3,count) = SymbolicMath(StringInput((start+1):(i-1)));
            collapsed(start:i) = true;
            count = count + 1;
        end
    end
end
if level == 0 && ismember(StringInput(i), '0123456789.eE')
    if basic_number_start == 0
        basic_number_start = i;
    end
    if i == length(StringInput)
        Intermediates(1,count) = basic_number_start;
        Intermediates(2,count) = i;
        Intermediates(3,count) = ...
            str2double(StringInput(basic_number_start:i));
        collapsed(basic_number_start:i) = true;
        basic_number_start = 0;
        count = count + 1;
    end
end

```

```

else
    if basic_number_start ~= 0
        Intermediates(1,count) = basic_number_start;
        Intermediates(2,count) = i - 1;
        Intermediates(3,count) = ...
            str2double(StringInput(basic_number_start:(i-1)));
        collapsed(basic_number_start:(i-1)) = true;
        basic_number_start = 0;
        count = count + 1;
    end
end
end
if level ~= 0
    NumberOutput = NaN;
    return;
end

% Exponents
if strcmp(StringInput(1),'^') || strcmp(StringInput(end),'^')
    NumberOutput = NaN;
    return;
end
for i = 2:(length(StringInput)-1)
    if ~collapsed(i)
        if strcmp(StringInput(i),'^')
            % Find the intermediate before and after and merge them
            before = 0;
            after = 0;
            for ind = 1:(count-1)
                if Intermediates(2,ind) == i-1
                    before = ind;
                elseif Intermediates(1,ind) == i+1
                    after = ind;
                end
            end
            end
            if before > 0 && after > 0
                Intermediates(3,before) = Intermediates(3,before) ^ ...
                    Intermediates(3,after);
                Intermediates(2,before) = Intermediates(2,after);
                Intermediates(1,after) = -1;
                Intermediates(2,after) = -1;
            else
                NumberOutput = NaN;
                return;
            end
            collapsed(i) = true;
        end
    end
end

% Division
if strcmp(StringInput(1),'/') || strcmp(StringInput(end),'/')
    NumberOutput = NaN;
    return;
end
for i = 2:(length(StringInput)-1)
    if ~collapsed(i)
        if strcmp(StringInput(i),'/')
            % Find the intermediate before and after and merge them
            before = 0;
            after = 0;
            for ind = 1:(count-1)
                if Intermediates(2,ind) == i-1
                    before = ind;
                elseif Intermediates(1,ind) == i+1
                    after = ind;
                end
            end
            end
            if before > 0 && after > 0
                Intermediates(3,before) = Intermediates(3,before) / ...

```

```

        Intermediates(3,after);
        Intermediates(2,before) = Intermediates(2,after);
        Intermediates(1,after) = -1;
        Intermediates(2,after) = -1;
    else
        NumberOutput = NaN;
        return;
    end
    collapsed(i) = true;
end
end
end

% Multiplication
if strcmp(StringInput(1),'*') || strcmp(StringInput(end),'*')
    NumberOutput = NaN;
    return;
end
for i = 2:(length(StringInput)-1)
    if ~collapsed(i)
        if strcmp(StringInput(i),'*')
            % Find the intermediate before and after and merge them
            before = 0;
            after = 0;
            for ind = 1:(count-1)
                if Intermediates(2,ind) == i-1
                    before = ind;
                elseif Intermediates(1,ind) == i+1
                    after = ind;
                end
            end
            if before > 0 && after > 0
                Intermediates(3,before) = Intermediates(3,before) * ...
                    Intermediates(3,after);
                Intermediates(2,before) = Intermediates(2,after);
                Intermediates(1,after) = -1;
                Intermediates(2,after) = -1;
            else
                NumberOutput = NaN;
                return;
            end
            collapsed(i) = true;
        end
    end
end

% Addition
if strcmp(StringInput(end),'+')
    NumberOutput = NaN;
    return;
end
for i = 1:(length(StringInput)-1)
    if ~collapsed(i)
        if strcmp(StringInput(i),'+')
            % Find the intermediate before and after and merge them
            before = 0;
            after = 0;
            for ind = 1:(count-1)
                if Intermediates(2,ind) == i-1
                    before = ind;
                elseif Intermediates(1,ind) == i+1
                    after = ind;
                end
            end
            if after > 0
                if before > 0
                    Intermediates(3,before) = Intermediates(3,before) + ...
                        Intermediates(3,after);
                    Intermediates(2,before) = Intermediates(2,after);
                    Intermediates(1,after) = -1;
                end
            end
        end
    end
end

```

```

        Intermediates(2,after) = -1;
    else
        % Intermediates(3,after) = Intermediates(3,after);
        Intermediates(1,after) = i;
    end
    else
        NumberOutput = NaN;
        return;
    end
    collapsed(i) = true;
end
end
end

% Subtraction
if strcmp(StringInput(end),'-')
    NumberOutput = NaN;
    return;
end
for i = 1:(length(StringInput)-1)
    if ~collapsed(i)
        if strcmp(StringInput(i),'-')
            % Find the intermediate before and after and merge them
            before = 0;
            after = 0;
            for ind = 1:(count-1)
                if Intermediates(2,ind) == i-1
                    before = ind;
                elseif Intermediates(1,ind) == i+1
                    after = ind;
                end
            end
            if after > 0
                if before > 0
                    Intermediates(3,before) = Intermediates(3,before) - Intermediates(3,after);
                    Intermediates(2,before) = Intermediates(2,after);
                    Intermediates(1,after) = -1;
                    Intermediates(2,after) = -1;
                else
                    Intermediates(3,after) = - Intermediates(3,after);
                    Intermediates(1,after) = i;
                end
            end
            else
                NumberOutput = NaN;
                return;
            end
            collapsed(i) = true;
        end
    end
end

% Assess conclusion
if ~all(collapsed)
    NumberOutput = NaN;
    return;
end
for i = 1:count-1
    if Intermediates(1,i) ~= -1
        NumberOutput = Intermediates(3,i);
        return;
    end
end
end
end

```

## Is Function



Determines if a particular object of unknown type is a function. Usually this is for differentiating between an error and a correctly assigned function.

```

function [TF, ID] = isfunction(FUN)
% ISFUNCTION - true for valid matlab functions
%
% TF = ISFUNCTION(FUN) returns 1 if FUN is a valid matlab function, and 0
% otherwise. Matlab functions can be strings or function handles.
%
% [TF, ID] = ISFUNCTION(FUN) also returns an identifier ID. ID can take the
% following values:
%     1 : FUN is a function string
%     2 : FUN is a function handle
%     0 : FUN is not a function, but no further specification
%    -1 : FUN is not a function but a script
%    -2 : FUN is not a valid function m-file (e.g., a matfile)
%    -3 : FUN does not exist (as a function)
%    -4 : FUN is not a function but something else (a variable)
%
% FUN can be a cell array, TF and ID will then be arrays, the same size
% as FUN
%
% Examples:
%     tf = isfunction('lookfor')
%         % tf = 1
%     [tf, id] = isfunction(@isfunction, 'sin','qrqtrwxy',1:4, @clown.jpg)
%         % -> tf = [ 1  1  0  0  0 ]
%         %     id = [ 2  1 -2 -4 -3 ]
%
% See also FUNCTION, SCRIPT, EXIST,
%           ISA, WHICH, NARGIN, FUNCTION_HANDLE

% version 3.2 (apr 2018)
% (c) Jos van der Geest
% Matlab File Exchange Author ID: 10584
% email: samelinoa@gmail.com
%
% History:
% 1.0 (dec 2011) created for strings only
% 2.0 (apr 2013) accepts cell arrays
% 3.0 (feb 2014) implemented identifier based on caught error
% 3.1 (feb 2014) added lots of help and documentation, inspired to post on
%               FEX by a recent Question/Answer thread
% 3.2 (apr 2018) spell check and contact info

if ~iscell(FUN)
    % we use cellfun, so convert to cells
    FUN = {FUN} ;
end
ID = cellfun(@local_isfunction,FUN) ; % get the identifier for each "function"
TF = ID > 0 ; % valid matlab functions have a positive identifier

% = = = = =

function ID = local_isfunction(FUNNAME)
try
    nargin(FUNNAME) ; % nargin errors when FUNNAME is not a function
    ID = 1 + isa(FUNNAME, 'function_handle') ; % 1 for m-file, 2 for handle
catch ME
    % catch the error of nargin
    switch (ME.identifier)
        case 'MATLAB:nargin:isScript'
            ID = -1 ; % script
        case 'MATLAB:narginout:notValidMfile'
            ID = -2 ; % probably another type of file, or it does not exist
        case 'MATLAB:narginout:functionDoesNotExist'
            ID = -3 ; % probably a handle, but not to a function
        case 'MATLAB:narginout:BadInput'
    
```

```
        ID = -4 ; % probably a variable or an array
    otherwise
        ID = 0 ; % unknown cause for error
end
end
```

## G.2. Other Functions

### Wall Smart Discretize

Takes a body, a mesher (set of discretize options) and an orientation over which the discretization is taking place and returns the set of boundary positions which will divide the body. This function looks at whether or not each surface of the body is exposed to a gas body, in which case it will generate the surface nodes. The rest of the body is either the maximum thickness or growing from the surface nodes.

```
function [x] = Wall_Smart_Discretize(Body,Mesher,Orient)
%UNTITLED2 Summary of this function goes here
% Detailed explanation goes here
material = Body.matl;
if Body.matl.Phase == enumMaterial.Gas
% Gas Body
N_entrance = Mesher.Gas_Entrance_Exit_N;
maximum_growth = Mesher.maximum_growth;
maximum_thickness = Mesher.Gas_Maximum_Size;
minimum_thickness = Mesher.Gas_Minimum_Size;
% Derived Values
shifti = [];
switch Orient
case enumOrient.Vertical
[~,~,inside_dim, outside_dim] = Body.limits(Orient);
case enumOrient.Horizontal
[inside_dim,outside_dim,~,~] = Body.limits(Orient);
if ~isscalar(inside_dim)
shifti = inside_dim - inside_dim(1);
inside_dim = inside_dim(1);
end
if ~isscalar(outside_dim)
%shifto = outside_dim - outside_dim(1);
outside_dim = outside_dim(1);
end
end
Distance = outside_dim - inside_dim;
Transition_Distance = Distance * 0.15;
x = inside_dim;
thickness = Transition_Distance / double(N_entrance);
if thickness > maximum_thickness
N = ceil(Distance/maximum_thickness);
x = linspace(inside_dim,outside_dim,N+1);
else
while thickness < minimum_thickness && N_entrance > 1
N_entrance = N_entrance - 1;
thickness = Transition_Distance / double(N_entrance);
end
if N_entrance == 1
thickness = min(minimum_thickness, Distance/2);
if Distance/2 - thickness < thickness
x = [inside_dim (inside_dim + outside_dim)/2 outside_dim];
else
x = [inside_dim inside_dim + thickness];
marker = 2;
while (x(end) < Distance/2 + inside_dim)
thickness = min(maximum_thickness, maximum_growth * thickness);
x(end+1) = x(end) + thickness;
end
% Adjust it at the end
Current_Distance = x(end) - x(marker);
Expected_Distance = Distance/2 + inside_dim - x(marker);
```

```

        x((marker+1):end) = (x((marker+1):end) - x(marker))*...
            (Expected_Distance/Current_Distance) + x(marker);
        % Flip it
        x = [x outside_dim-(flip(x(1:end-1))-inside_dim)];
    end
else
    for i = 1:N_entrance
        x(end+1) = x(end) + thickness;
    end
    marker = length(x);
    while (x(end) < Distance/2 + inside_dim)
        thickness = min(maximum_thickness, maximum_growth * thickness);
        x(end+1) = x(end) + thickness;
    end
    % Adjust it at the end
    Current_Distance = x(end) - x(marker);
    Expected_Distance = Distance/2 + inside_dim - x(marker);
    x((marker+1):end) = (x((marker+1):end) - x(marker))*...
        (Expected_Distance/Current_Distance) + x(marker);
    % Flip it
    x = [x outside_dim-(flip(x(1:end-1))-inside_dim)];
end
end
else
    % Solid Body
    min_ang_frequency = Body.Group.Model.engineSpeed;
    oscillation_depth_N = Mesher.oscillation_depth_N;
    maximum_thickness = Mesher.maximum_thickness;
    maximum_growth = Mesher.maximum_growth;
    shifti = [];
    switch Orient
        case enumOrient.Vertical
            [~,~,inside_dim, outside_dim] = Body.limits(Orient);
            inside_exp = Mesher.isInsideRadiiExposed(Body);
            outside_exp = Mesher.isOutsideRadiiExposed(Body);
        case enumOrient.Horizontal
            [inside_dim,outside_dim,~,~] = Body.limits(Orient);
            inside_exp = Mesher.isBottomExposed(Body);
            outside_exp = Mesher.isTopExposed(Body);
            if ~isscalar(inside_dim)
                shifti = inside_dim - inside_dim(1);
                inside_dim = inside_dim(1);
            end
            if ~isscalar(outside_dim)
                %shifto = outside_dim - outside_dim(1);
                outside_dim = outside_dim(1);
            end
        end
    end
    if Body.matl.dT_du == -1
        alpha = 1000000;
    else
        % output requirements, x must have the min and maximum point
        alpha = material.thermaldiffusivity;
    end
end

% Using the 5% amplitude condition
xdepth = 3*sqrt(2*alpha/min_ang_frequency);
xtotal = outside_dim - inside_dim;
if inside_exp
    if outside_exp
        if xtotal < 2*xdepth
            % Discretize the entire depth to the near wall standards
            N = ceil(xtotal/(xdepth/double(oscillation_depth_N)));
            x = linspace(inside_dim,outside_dim,N+1);
        else
            % Grow from both ends (calc with half then mirror)
            N_max = ceil(0.5*xtotal/(xdepth/double(oscillation_depth_N)));
            x = [linspace(inside_dim,inside_dim + xdepth,oscillation_depth_N+1) ...
                zeros(1,N_max-oscillation_depth_N)];
            i = oscillation_depth_N+1;
            while x(i) < inside_dim + xtotal/2

```

```

        i = i + 1;
        x(i) = x(i-1) + min([maximum_growth*(x(i-1)-x(i-2)) ...
            maximum_thickness]);
    end
    x(i) = inside_dim + xtotal/2;
    x(i+1:2*i-1) = outside_dim - (flip(x(1:i-1))-inside_dim);
    if length(x) > 2*i - 1
        x(2*i:end) = [];
    end
end
else
    if xtotal < xdepth
        % Discretize the entire depth to the near wall standards
        N = ceil(xtotal/(xdepth/double(oscillation_depth_N)));
        x = linspace(inside_dim,outside_dim,N+1);
    else
        % Grow from inside end
        N_max = ceil(xtotal/(xdepth/double(oscillation_depth_N)));
        x = [linspace(inside_dim,inside_dim + xdepth,oscillation_depth_N+1) ...
            zeros(1,N_max-oscillation_depth_N)];
        i = oscillation_depth_N+1;
        while x(i) < outside_dim
            i = i + 1;
            x(i) = x(i-1) + min([maximum_growth*(x(i-1)-x(i-2)) ...
                maximum_thickness]);
        end
        x(i) = outside_dim;
        if length(x) > i
            x(i+1:end) = [];
        end
    end
end
else
    if outside_exp
        if xtotal < xdepth
            % Discretize the entire depth to the near wall standards
            N = ceil(xtotal/(xdepth/double(oscillation_depth_N)));
            x = linspace(inside_dim,outside_dim,N+1);
        else
            % Grow from outside end, use mathematics from the other
            % direction, then flip afterwards
            N_max = ceil(xtotal/(xdepth/double(oscillation_depth_N)));
            x = [linspace(inside_dim,inside_dim + xdepth,oscillation_depth_N+1) ...
                zeros(1,N_max-oscillation_depth_N)];
            i = oscillation_depth_N+1;
            while x(i) < outside_dim
                i = i + 1;
                x(i) = x(i-1) + min([maximum_growth*(x(i-1)-x(i-2)) ...
                    maximum_thickness]);
            end
            x(i) = outside_dim;
            if length(x) > i
                x(i+1:end) = [];
            end
            % Flip
            x = inside_dim + (outside_dim - x);
        end
    else
        % Discretize the entire depth to the minimum standards
        N = ceil(xtotal/maximum_thickness);
        x = linspace(inside_dim,outside_dim,N+1);
    end
end
end
if isempty(shifti)
    x = transpose(x);
else
    temp = transpose(x);
    x = zeros(length(x),length(shifti));
    for r = 1:length(temp)
        x(r,:) = shifti(:);
    end
end

```

```
end
for c = 1:length(shifti)
    x(:,c) = x(:,c) + temp;
end
end
end
```

## Tube Bank Friction

Determining the constant coefficient for friction across a staggered tube bank.

```
function [ Cf ] = TubeBankFriction( Xt,Xl,do )
%TUBE BANK FRICTION 300 -> Re -> 15,000
% f = Cf*Re^-0.18
Xt_Xl = Xt/Xl;
Xl_do = Xl/do;
a = -0.108*Xt_Xl^2+0.3137*Xt_Xl-0.2335;
b = 0.7298*Xt_Xl^2-1.296*Xt_Xl+1.0343;
c = -0.2129*Xt_Xl^2+0.5613*Xt_Xl-0.7471;
Cf = a*Xl_do^2+b*Xl_do+c;
end
```

## Aligned Tube Bank Conduction

Unused, defines the constant and exponent of a function for calculating convection for flow over a set of bare, aligned tubes.

```
function [ Const, Exponent ] = AlignedTubeBankConduction( Xt,Xl,do )
%ALIGNED TUBE BANK CONDUCTION Const * Re ^ Exponent = Nst*Npr^(2/3)
Xt_Xl = Xt/Xl;
Xt_do = Xt/do;
Xl_do = Xl/do;
Const = (0.118*Xt_Xl+0.252);
Exponent = (-0.0125*Xt_Xl-0.433*Xl+0.0765*Xt-0.0892);
end
```

## Tube Bank Convection Constant

Defines the constant and exponent of a function for calculating convection for flow over a set of bare, staggered tubes.

```
function [ Ch ] = TubeBankConvectiveConstant(Xt,Xl,do,Nr)
%TUBE BANK NUSSELT Nst*Npr^0.667 = [Ch]*NRe^-0.4
Xt_Xl = Xt/Xl;
Xt_do = Xt/do;
a = -0.1548*Xt_Xl+0.0591;
b = 0.5437*Xt_Xl-0.0373;
c = -1.9244*Xt_Xl^3+0.68562*Xt_Xl^2-7.9841*Xt_Xl+2.772;
Ch = a*Xt_do^2 + b*Xt_do + c;
if nargin > 3
    Ch = Ch*(1-1/(Nr^1.112+0.918353));
end
end
```

## G.3. Relationships

### Enum Relation

```
classdef enumRelation
    enumeration
        Constant,...
        StackedShift,... % Unused
        Fixed,... % Unuseable
    end
end
```

```

        AreaConstant,...
        Scaled,...
        LowestScaled,...
        Stroke,...
        Piston,...
        Width
    end
end
end

```

## Relation

```

classdef Relation < handle
    % Relation - Class
    % ... -> name - String
    % ... -> mode - enumRelation
    % ... -> con1 - Connection
    % ... -> con2 - Connection
    % ... -> frame - Frame, associated with a mechanism with stroke

    properties
        name;
        mode enumRelation;
        con1 Connection;
        con2 Connection;
        frame Frame;
        manager RelationManager;
    end

    methods
        function this = Relation(manager,name,mode,con1,con2,frame)
            this.manager = manager;
            this.name = name;
            this.mode = mode;
            this.con1 = con1;
            this.con2 = con2;
            if nargin > 5
                this.frame = frame;
            end
        end

        function deReference(this)
            for i = length(this.manager.Relations):-1:1
                if this.manager.Relations(i).con1 == this.con1 && ...
                    this.manager.Relations(i).con2 == this.con2
                    this.manager.Relations(i) = [];
                end
            end
            this.manager.isChanged = true;
        end

        function Item = get(this,PropertyName)
            switch PropertyName
                case 'Name'
                    Item = this.name;
                case 'Connection1'
                    Item = this.con1;
                case 'Connection2'
                    Item = this.con2;
                case 'Frame'
                    Item = this.frame;
                otherwise
                    fprintf(['XXX Relation GET Inteface for ' PropertyName ...
                        ' is not found XXX\n']);
            end
        end

        function set(this,PropertyName,Item)
            switch PropertyName
                case 'Name'
                    this.name = Item;
            otherwise

```



```

                fprintf(['XXX Relation SET Inteface for ' PropertyName ...
                        ' is not found XXX\n']);
            end
        end
    end
end
end

```

## RelationManager

### Manages Geometric Relationships

```

classdef RelationManager < handle

properties
    Group Group; % Group that this relation grid refers to
    Orient enumOrient;
    Relations Relation;
    Grid logical = [];
    Grid_modes cell;

    isChanged logical = false;
end

properties (Dependent)
    name;
end

methods
    %% Relation - Class
    % ... -> name - String
    % ... -> mode - enumRelation
    % ... -> con1 - Connection
    % ... -> con2 - Connection
    % ... -> frame - Frame, associated with a mechanism with stroke

    %% Grid Construction
    function this = RelationManager(Group, Orient)
        if nargin > 1
            this.Group = Group;
            this.Orient = Orient;
        end
    end

    function Item = get(this,PropertyName)
        switch PropertyName
            case 'Name'
                Item = this.name;
            case 'Relations'
                Item = this.Relations;
            otherwise
                fprintf(['XXX Group GET Inteface for ' PropertyName ...
                        ' is not found XXX\n']);
        end
    end

    function set(this,PropertyName,Item) %#ok<INUSD,INUSL>
        switch PropertyName
            otherwise
                fprintf(['XXX Group SET Inteface for ' PropertyName ...
                        ' is not found XXX\n']);
        end
    end

    function name = get.name(this)
        switch this.Orient
            case enumOrient.Horizontal
                name = 'Rel. Man. handling horizontal connections';
            case enumOrient.Vertical
                name = 'Rel. Man. handling vertical connections';
        end
    end
end

```

```

end
end

function update(this)
    this.isChanged = false;
    % In essence, recreate the grid from the number of connections
    % ... in group, then append each relation one by one into the
    % ... grid
    this.Grid = false(1,length(this.Group.Connections));
    this.Grid_modes = cell(0);
    keep = true(size(this.Relations));
    for i = 1:length(this.Relations)
        if keep(i)
            for j = i+1:length(this.Relations)
                if keep(j)
                    if this.Relations(i).con1 == ...
                        this.Relations(j).con1 && ...
                        this.Relations(i).con2 == ...
                        this.Relations(j).con2
                        keep(j) = false;
                    end
                end
            end
        end
    end
    this.Relations(~keep) = [];
    for relation = this.Relations
        this.appendGrid(relation);
    end
end

function appendGrid(this, new_relation)
    % this.Relations(end+1) = new_relation;
    % So to make it here, there are no invalidities.
    ind1 = new_relation.con1.index;
    ind2 = new_relation.con2.index;

    % Adding to each other's groups
    group1 = find(this.Grid(:,ind1)==true);
    group2 = find(this.Grid(:,ind2)==true);
    allready = false;
    previous = 0;
    for i = 1:length(group2)
        group1(end+1) = group2(i);
    end
    if ~isempty(group1)
        for i = group1
            if this.Grid_modes{i} == new_relation.mode
                if allready
                    this.merge_rows(i,previous);
                    return;
                end
                % Simply append ind2 to this group
                this.Grid(i,ind1) = true;
                this.Grid(i,ind2) = true;
                allready = true;
                previous = i;
            end
        end
        if allready; return; end
    end

    % Forming new groups
    % To make it here, both entities are not part of a group that
    % ... has the same mode, thus, it forms a new group
    row = this.get_new_group_row();
    this.Grid(row,ind1) = true;
    this.Grid(row,ind2) = true;
    this.Grid_modes{row} = new_relation.mode;
end

```

```

function merge_rows(this,row1,row2)
    for i = 1:size(this.Grid,2)
        this.Grid(row1,i) = this.Grid(row1,i) || ...
            this.Grid(row2,i);
    end
    this.Grid(row2,:) = false;
    this.Grid_modes{row2} = enumRelation.empty;
end

function index = get_new_group_row(this)
    % See if there is an empty row
    for i = 1:size(this.Grid,1)
        if all(this.Grid(i,)==false)
            index = i;
            return;
        end
    end
    % else make a new row
    index = size(this.Grid,1) + 1;
end

%% Relationship Adding
function yesno = isNewRelationValid(this, ind1, ind2)
    yesno = true;
    % Make sure that grid is actually large enough
    if ~(size(this.Grid,2) >= ind1 && size(this.Grid,2) >= ind2)
        this.update();
    end
    % If one of the connections does not have any existing
    % ... relations then it is an automatic pass
    if ~any(this.Grid(:,ind1)); yesno = true; return; end
    if ~any(this.Grid(:,ind2)); yesno = true; return; end
    % If we got to this point then we have to find if the two
    % ... indexes are connected in some way
    groups1 = find(this.Grid(:,ind1)==true);
    groups2 = find(this.Grid(:,ind2)==true);
    for i = groups1
        for j = groups2
            % Test to see if "i" is patheable to "j"
            target = j;
            start = i;
            [found, ~] = are_rows_connected(...
                this.Grid,target,start,false(size(this.Grid,1),1));
            if found
                yesno = false; return;
            end
        end
    end
end

function success = addRelation(this, name, mode, con1, con2, frame)
    success = false;
    if this.isChanged; this.update(); end
    if this.Orient ~= con1.Orient; return; end
    if con1.Orient ~= con2.Orient; return; end
    % There cannot be an existing thing between con1 and con2
    if ~this.isNewRelationValid(con1.index, con2.index)
        return;
    end
    if nargin > 5
        newRelation = Relation(this,name,mode,con1,con2,frame);
    else
        newRelation = Relation(this,name,mode,con1,con2);
    end
    this.Relations(end+1) = newRelation;
    this.update();
    success = true;
end

%% Relationship Editing
function [success, visitedgroups, shifts, data] = Edit(...

```

```

    this, con, shift, visitedgroups, shifts, data)
if this.isChanged; this.update(); end
%% Get all the shifts that happened due to this edit
if isa(con,'Connection')
    ind = con.index;
else
    ind = con;
end
if isempty(this.Grid)
    this.update();
end
if nargin < 4
    visitedgroups = false(size(this.Grid,1),1);
    shifts = zeros(size(this.Group.Connections));
    shifts(ind) = shift;
    data = struct();
end
groups = find(this.Grid(:,ind)==true);
for i = groups'
    if ~visitedgroups(i)
        cons = find(this.Grid(i,:)==true);
        [var, data] = this.getShifts(cons, i, ind, shift, data);
        if isempty(var); success = false; return; end
        shifts(cons) = var;
        visitedgroups(i) = true;
        for j = cons
            if shifts(j) ~= 0
                [success, visitedgroups, shifts, data] = ...
                    this.Edit(j,shifts(j),visitedgroups,...
                        shifts, data);
                if ~success; return; end
            end
        end
    end
end
success = true;

%% Apply all those shifts and test each body
if nargin < 4
    for i = 1:length(this.Group.Connections)
        this.Group.Connections(i).x = ...
            this.Group.Connections(i).x + shifts(i);
    end
    if isfield(data,'frames') && isfield(data,'frameshift')
        for i = 1:length(data.frames)
            for iLRM = this.Group.Model.Converters
                for RefFrame = iLRM.Frames
                    if RefFrame == data.frames(i)
                        Mech = RefFrame.Mechanism;
                        Mech.dont_propegate = true;
                        Mech.set('Stroke',...
                            Mech.get('Stroke')+data.frameshift(i));
                    end
                end
            end
        end
    end
    for iBody = this.Group.Bodies; iBody.update(); end
    for iBody = this.Group.Bodies
        if ~iBody.isValid
            for i = 1:length(this.Group.Connections)
                this.Group.Connections(i).x = ...
                    this.Group.Connections(i).x - shifts(i);
            end
            if isfield(data,'frames') && isfield(data,'frameshift')
                for i = 1:length(data.frames)
                    for iLRM = this.Group.Model.Converters
                        for RefFrame = iLRM.Frames
                            if RefFrame == data.frames(i)
                                Mech = RefFrame.Mechanism;
                                Mech.dont_propegate = true;
                            end
                        end
                    end
                end
            end
        end
    end
end

```



```

baseind = root;
for i = length(cons):-1:1
    if this.Group.Connections(cons(i)).x < curmin
        curmin = this.Group.Connections(cons(i)).x;
        baseind = i;
    end
end
if baseind == root
    % Compacts to the other side
    curmax = -inf;
    for i = length(cons):-1:1
        if this.Group.Connections(cons(i)).x > curmax
            curmax = this.Group.Connections(cons(i)).x;
        end
    end
    for i = length(cons):-1:1
        C1o = this.Group.Connections(root).x;
        C2o = this.Group.Connections(cons(i)).x;
        shifts(i) = shift * ...
            (curmax - C2o)/(curmax - C1o);
    end
else
    % Scale everything relative to the base
    for i = length(cons):-1:1
        C1o = this.Group.Connections(root).x;
        C2o = this.Group.Connections(cons(i)).x;
        shifts(i) = shift * ...
            (C2o - curmin) / (C1o - curmin);
    end
end
switch this.Grid_modes{group}
case enumRelation.LowestScaled
    return;
case enumRelation.Stroke
    % Get largest shift and edit the value of the
    % ... stroke in the same direction
    sgn = 1;
case enumRelation.Piston
    % Get largest shift and edit the value of the
    % ... stroke in the opposite direction
    sgn = -1;
end
curmax = max(shifts);
for rel = this.Relations
    if rel.mode == enumRelation.Stroke || ...
        any(cons == rel.con1.index) || ...
        any(cons == rel.con2.index)
        if isfield(data,'frame')
            data.frame(end+1) = rel.frame;
            data.frameshift(end+1) = sgn * curmax;
        else
            data.frame = rel.frame;
            data.frameshift = sgn * curmax;
        end
    end
end
case enumRelation.Width
switch length(cons)
case 4
    xvals = zeros(4,1);
    for i = 1:4
        xvals(i) = ...
            this.Group.Connections(cons(i)).x;
    end
    % Determine order translation
    % location of first connection
    try
        indexes = zeros(4,1);
        temp = min(xvals);
        indexes(1) = find(xvals==temp);
        temp = min(xvals(xvals>temp));
    end
end

```

```

indexes(2) = find(xvals==temp);
temp = min(xvals(xvals>temp));
indexes(3) = find(xvals==temp);
temp = max(xvals);
indexes(4) = find(xvals==temp);
if root == cons(indexes(1))
    % Middle should remain the same but shift up or
    % ... down
    shifts(indexes(1)) = shift;
    shifts(indexes(2)) = shift/2;
    shifts(indexes(3)) = shift/2;
    shifts(indexes(4)) = 0;
elseif root == cons(indexes(2))
    % Boundaries should remain the same, but the
    % center should stretch
    shifts(indexes(1)) = 0;
    shifts(indexes(2)) = shift;
    shifts(indexes(3)) = -shift;
    shifts(indexes(4)) = 0;
elseif root == cons(indexes(3))
    % Boundaries should remain the same, but the
    % center should stretch
    shifts(indexes(1)) = 0;
    shifts(indexes(2)) = -shift;
    shifts(indexes(3)) = shift;
    shifts(indexes(4)) = 0;
else
    % Middle should remain the same but shift up or
    % ... down
    shifts(indexes(1)) = 0;
    shifts(indexes(2)) = shift/2;
    shifts(indexes(3)) = shift/2;
    shifts(indexes(4)) = shift;
end
catch
    fprintf('err');
end
case 6
xvals = zeros(6,1);
for i = 1:6
    xvals(i) = ...
        this.Group.Connections(cons(i)).x;
end
% Determine order translation
% location of first connection
indexes = zeros(6,1);
temp = min(xvals);
indexes(1) = find(xvals==temp);
temp = min(xvals(xvals>temp));
indexes(2) = find(xvals==temp);
temp = min(xvals(xvals>temp));
indexes(3) = find(xvals==temp);
temp = min(xvals(xvals>temp));
indexes(4) = find(xvals==temp);
temp = min(xvals(xvals>temp));
indexes(5) = find(xvals==temp);
temp = max(xvals);
indexes(6) = find(xvals==temp);
if root == cons(indexes(1))
    shifts(indexes(1)) = shift;
    shifts(indexes(2)) = shift/2;
    shifts(indexes(3)) = shift/2;
    shifts(indexes(4)) = shift/2;
    shifts(indexes(5)) = shift/2;
    shifts(indexes(6)) = 0;
elseif root == cons(indexes(2))
    shifts(indexes(1)) = 0;
    shifts(indexes(2)) = shift;
    shifts(indexes(3)) = 0;
    shifts(indexes(4)) = 0;
    shifts(indexes(5)) = -shift;

```

```

        shifts(indexes(6)) = 0;
    elseif root == cons(indexes(3))
        shifts(indexes(1)) = 0;
        shifts(indexes(2)) = 0;
        shifts(indexes(3)) = shift;
        shifts(indexes(4)) = -shift;
        shifts(indexes(5)) = 0;
        shifts(indexes(6)) = 0;
    elseif root == cons(indexes(4))
        shifts(indexes(1)) = 0;
        shifts(indexes(2)) = 0;
        shifts(indexes(3)) = -shift;
        shifts(indexes(4)) = shift;
        shifts(indexes(5)) = 0;
        shifts(indexes(6)) = 0;
    elseif root == cons(indexes(5))
        shifts(indexes(1)) = 0;
        shifts(indexes(2)) = -shift;
        shifts(indexes(3)) = 0;
        shifts(indexes(4)) = 0;
        shifts(indexes(5)) = shift;
        shifts(indexes(6)) = 0;
    elseif root == cons(indexes(6))
        shifts(indexes(1)) = 0;
        shifts(indexes(2)) = shift/2;
        shifts(indexes(3)) = shift/2;
        shifts(indexes(4)) = shift/2;
        shifts(indexes(5)) = shift/2;
        shifts(indexes(6)) = shift;
    end
    otherwise
        fprintf(['XXX A width mate is not working ' ...
            'because it does not contain 4' ...
            'elements XXX\n']);
    end
end
end
end
if any(isnan(shifts))
    fprintf('err');
end
end
end

```

```

function color = getColor(this, index)
    if this.Group.isChanged; this.Group.update(); end
    count = 1;
    colors = ...
        [0.67, 0, 0; ...
        1, 0.33, 0.33; ...
        1, 0.67, 0; ...
        1, 1, 0.33; ...
        0, 0.67, 0; ...
        0.33, 1, 0.33; ...
        0.33, 1, 1; ...
        0, 0.67, 0.67; ...
        0, 0.67, 0; ...
        0.33, 0.33, 1; ...
        1, 0.33, 1; ...
        0.67, 0, 0.67; ...
        0.67, 0.67, 0.67; ...
        0.33, 0.33, 0.33];
    color = [0, 0, 0];
    for iGroup = this.Group.Model.Groups
        for RMan = iGroup.RelationManagers
            if RMan.isChanged; RMan.update(); end
            if RMan ~= this
                for i = 1:size(RMan.Grid,1)
                    if any(RMan.Grid(i,:) == true)
                        count = count + 1;
                    end
                end
            end
        end
    else

```



```

        for i = 1:size(RMan.Grid,1)
            if i == index
                while (count > 14)
                    count = count - 14;
                end
                color = colors(count,:);
                return;
            end
            if any(RMan.Grid(i,:) == true)
                count = count + 1;
            end
        end
    end
end
end
end

function Label = getLabel(this, mode, con1, con2)
    Label = '';
    for i = 1:length(this.Grid_modes)
        if this.Grid_modes{i} == mode
            % Find a relation that matches
            for Rel = this.Relations
                if Rel.mode == mode && (con1 == Rel.con1 || con1 == Rel.con2 || ...
                    con2 == Rel.con1 || con2 == Rel.con2)
                    Label = Rel.name;
                    return;
                end
            end
        end
    end
end
end
end
end
end

function [yesno, checked] = are_rows_connected(grid,target,start,checked)
if target == start; yesno = true; return; end
yesno = false;
cols = find(grid(start,:) == true);
for i = cols
    rows = find(grid(:,i) == true);
    rows(checked(rows)) = [];
    for j = rows
        if j ~= start
            checked(j) = true;
            [yesno, checked] = are_rows_connected(grid,target,j,checked);
            if yesno; return; end
        end
    end
end
end
end

function [el] = custmink(array,k)
el = -inf;
for i = 1:k
    el = min(array(array > el));
end
end

function [el] = custmaxk(array,k)
el = +inf;
for i = 1:k
    el = max(array(array < el));
end
end
end

```

## G.4. Optimization

### Load Model

```
function Model = load_Model(name)
    newfile = [pwd '\Saved Files\' name];
    File = load(newfile,'Model');
    Model = File.Model;
    Model.AxisReference = gca;

    Model.showPressureAnimation = false;
    Model.recordPressure = false;
    Model.showTemperatureAnimation = false;
    Model.recordTemperature = false;
    Model.showVelocityAnimation = false;
    Model.recordVelocity = false;
    Model.showTurbulenceAnimation = false;
    Model.recordTurbulence = false;
    Model.recordOnlyLastCycle = true;
    Model.recordStatistics = true;
    Model.outputPath= '';
    Model.warmUpPhaseLength = 0;
    Model.animationFrameTime = 0.05;
end
```

### Gradient Ascent

```
function [History] = GradientAscent(...
    Model, ...
    OptimizationSchemeID)
    History = [];
    % Find the Folder "Test_Running" and allow the user to select a test set
    % ... which generates the appropriate structure.
    files = dir('Test_Running');
    names = {files.name};
    names(1:2) = [];
    if ~iscell(names); names = {names}; end
    for index = size(names,1):-1:1; names{index} = names{index}(1:end-2); end
    index = listdlg('PromptString','Pick the test running conditions',...
        'ListString',names,...
        'SelectionMode','single',...
        'InitialValue',index);
    if ~isempty(index)
        if strfind(names{index},'.m')
            func = str2func(names{index}(1:end-2));
        else
            func = str2func(names{index});
        end
        RunConditions = func();
    else
        msgbox('A run condition struct is required to run the model.');
```

```
        return;
    end

    % Pick Optimizing Variable -> output: options
    options = struct('OptimizedProperty','');
    names = {
        'Max Power';
        'Max Thermo Power Per Unit Engine Volume';
        'Max Efficiency';
        'Max West Number'};
    index = listdlg('ListString',names,...
        'SelectionMode','single',...
        'InitialValue',1);
```

```

if ~isempty(index)
    options.OptimizedProperty = names{index};
else
    msgbox('You must select a parameter to be optimized');
    return;
end
% RunConditions = struct that lays out the test conditions in the style of
% ... test set running.
if isfield(RunConditions,'PressureBounds')
    mod_pressure = ~isempty(RunConditions.PressureBounds);
    if length(RunConditions.PressureBounds) == 1
        options.MinPressure = 101325;
        options.MaxPressure = RunConditions.PressureBounds(2);
    elseif mod_pressure
        options.MinPressure = RunConditions.PressureBounds(1);
        options.MaxPressure = RunConditions.PressureBounds(2);
    end
end
else
    mod_pressure = false;
end
if isfield(RunConditions,'SpeedBounds')
    mod_speed = ~isempty(RunConditions.SpeedBounds);
    if length(RunConditions.SpeedBounds) == 1
        options.MinSpeed = 0.2;
        options.MaxSpeed = RunConditions.SpeedBounds(2);
    elseif mod_speed
        options.MinSpeed = RunConditions.SpeedBounds(1);
        options.MaxSpeed = RunConditions.SpeedBounds(2);
    end
end
else
    mod_speed = false;
end
originalname = replace(Model.name,' - Optimized','');
sets = RunConditions;
for optrial = 1:length(sets)
    RunConditions = sets(optrial);

    % Load the specificied model
    if isempty(RunConditions.title)
        Model.name = originalname;
        NewModel = [Model.name ' - Optimized'];
    else
        Model.name = RunConditions.title;
        NewModel = RunConditions.title;
    end

    RunConditions.title = NewModel;
    RunConditions.Model = NewModel;
    addpath('..\runs\');
    addpath(cd);

    found = false;
    for i = 1:length(Model.OptimizationSchemes)
        if Model.OptimizationSchemes(i).ID == OptimizationSchemeID
            Study = Model.OptimizationSchemes(i);
            Names = cell(1,length(Model.OptimizationSchemes(i).IDs));
            Objects = cell(1,length(Model.OptimizationSchemes(i).IDs));
            found = true;
            for k = 1:length(Model.OptimizationSchemes(i).IDs)
                Names{k} = Model.OptimizationSchemes(i).Names{k};
                switch Model.OptimizationSchemes(i).Classes{k}
                    case 'Connection'
                        for iGroup = Model.Groups
                            for iCon = iGroup.Connections
                                if iCon.ID == Model.OptimizationSchemes(i).IDs{k}
                                    Objects{k} = iCon;
                                    break;
                                end
                            end
                        end
                    if Objects{k} == iCon; break; end
                end
            end
        end
    end
end

```

```

                case 'LinRotMechanism'
                    for iLRM = Model.Converters
                        if iLRM.ID == Model.OptimizationSchemes(i).IDs{k}
                            Objects{k} = iLRM;
                            break;
                        end
                    end
                end
            end
        end
        Fields = Model.OptimizationSchemes(i).Fields;
        break;
    end
end

if mod_pressure
    pressure_ind = length(Objects) + 1;
end
if mod_speed
    speed_ind = length(Objects) + mod_pressure + 1;
end

if ~found
    fprintf('XXX Model or Optimization Scheme is not found. XXX\b');
    return;
end

% Process the model
Model.name = NewModel;
% Open up memory, keeping only the last Snapshot
Model.SnapShots(1:end-1) = [];
% Recording Options
Model.showLivePV = false;
Model.showPressureAnimation = false;
Model.recordPressure = true;
Model.showTemperatureAnimation = false;
Model.recordTemperature = true;
Model.showVelocityAnimation = false;
Model.recordVelocity = false;
Model.showTurbulenceAnimation = false;
Model.recordTurbulence = true;
Model.showConductionAnimation = false;
Model.recordConductionFlux = false;
Model.showPressureDropAnimation = false;
Model.recordPressureDrop = false;
Model.recordOnlyLastCycle = true;
Model.recordStatistics = true;
Model.warmUpPhaseLength = 0;
Model.deRefinementFactorInput = 1;
Model.RelationOn = true;
save(Model.name, 'Model');

% Initialize Recording
% ... Struct that provides the class-field names and value, as well as goal
h1 = figure();
h2 = figure();

% Adadelata parameters
extra = mod_pressure + mod_speed;
shifts = zeros(length(Objects) + extra, 1);
take_a_break = false(length(shifts), 1);
gradient = shifts;
% gamma = 0.6;
tol = 1e-6;
optimizing = true;
maxiterations = 30;
iteration = 1;
L = 0.001;
Scale = 1;
local_scale = 1;
%EPara2 = ones(size(shifts))*0.01;
%EGrad2 = ones(size(shifts));

```

```

% Define History
History = struct();
[History.Score, success, ~, ~] = RunSubFunction(Model,RunConditions,options);
if ~success
    fprintf('XXX Failed to run the first test, corrupted snapshot or unsolveable geometry
XXX\n');
    return;
end
History.Names = cell(size(Objects));
History.IDs = zeros(size(Objects));
for i = 1:length(Objects)
    History.Names{i} = Names{i};
    History.IDs(i) = Objects{i}.ID;
end
if mod_pressure
    History.Names{pressure_ind} = 'Pressure (Atm)';
end
if mod_speed
    History.Names{speed_ind} = 'Speed (Hz)';
end
History.data = zeros(length(Objects) + mod_pressure + mod_speed,0);
if ~isempty(Study.History)
    iteration = iteration + 1;
    maxiterations = maxiterations + 1;
    for i = 1:length(History.Names)
        for j = 1:length(Study.History.Names)
            if strcmp(History.Names{i},Study.History.Names{j})
                % EPara2(i) = Study.History.EPara2(j);
                % EGrad2(i) = Study.History.EGrad2(j);
                % gradient(i) = Study.History.gradient(j);
                break;
            end
        end
    end
    Scale = Study.History.Scale;
end
count = 1;
for i = 1:length(Objects)
    History.data(i,count) = Objects{i}.get(Fields{i});
end
if mod_pressure
    History.data(pressure_ind,count) = RunConditions.EnginePressure/101325;
end
if mod_speed
    History.data(speed_ind,count) = RunConditions.rpm/60;
end

while optimizing && iteration < maxiterations
    iteration = iteration + 1;
    % Calculate local gradient - output shifts - using Adadelta
    for i = 1:length(Objects)
        if ~take_a_break(i)
            [gradient(i), History] = ...
                getShiftObject(gradient(i),Objects{i},Fields{i},History,...
                    Model,RunConditions,options);
        end
    end
    if mod_pressure
        ind = pressure_ind;
        if ~take_a_break(ind)
            [gradient(ind), History, RunConditions] = getShiftRunCon(...
                gradient(ind),History,Model,RunConditions,...
                'EnginePressure',options.MinPressure,options.MaxPressure,options);
        end
    end
    if mod_speed
        ind = speed_ind;
        if ~take_a_break(ind)
            [gradient(ind), History, RunConditions] = getShiftRunCon(...

```

```

        gradient(ind),History,Model,RunConditions,...
        'rpm',options.MinSpeed,options.MaxSpeed,options);
    end
end

if max(abs(gradient)) < tol
    optimizing = false;
end

% Adadelata algorithm - Shifts
for i = 1:length(Objects)
%   shifts(i) = sqrt((EPara2(i) + 1e-8) / ...
%       (EGrad2(i) + 1e-8)) * gradient(i);
    shifts(i) = L*gradient(i);%/sqrt(EGrad2(i) + 1e-8);
end
if mod_pressure
%   shifts(i) = sqrt((EPara2(pressure_ind) + 1e-8) / ...
%       (EGrad2(pressure_ind) + 1e-8)) * gradient(pressure_ind);
    shifts(pressure_ind) = gradient(pressure_ind);%/sqrt(EGrad2(pressure_ind) + 1e-8);
end
if mod_speed
%   shifts(speed_ind) = sqrt((EPara2(speed_ind) + 1e-8) / ...
%       (EGrad2(speed_ind) + 1e-8)) * gradient(speed_ind);
    shifts(speed_ind) = gradient(speed_ind);%/sqrt(EGrad2(speed_ind) + 1e-8);
end
if max(abs(shifts)) > Scale*0.005
    shifts = Scale*shifts*0.005/max(abs(shifts));
elseif max(abs(shifts)) < Scale*0.002
    shifts = Scale*shifts*0.002/max(abs(shifts));
end

for i = 1:length(shifts)
    fprintf([num2str(shifts(i)) ' ']);
    if i == length(shifts)
        fprintf('\n');
    end
end

power_backup = History.Score(end);
increasing = true;
stepcount = 1;
trial = 1;
fprintf('Starting Uphill Climb \n');
while increasing
    % Make a step
    backup = zeros(1,length(Objects)+mod_pressure+mod_speed);
    for i = 1:length(Objects)
        fprintf(['Shifting Object: ' num2str(i) '\n']);
        backup(i) = Objects{i}.get(Fields{i});
        if ~take_a_break(i)
            newValue = Objects{i}.get(Fields{i}) + shifts(i);
            if isa(Objects{i},'Connection')
                for iBody = Objects{i}.Bodies
                    for iCon = iBody.Connections
                        if iCon.Orient == Objects{i}.Orient && iCon ~= Objects{i}
                            if sign(iCon.x - Objects{i}.x) == sign(shifts(i))
                                if ~iCon.IsFixedTo(Objects{i})
                                    shifts(i) = sign(shifts(i)).*min(abs(shifts(i)), ...
                                        abs(0.33*(iCon.x - Objects{i}.x)));
                                end
                            end
                        end
                    end
                end
            end
        end
        newValue = Objects{i}.get(Fields{i}) + local_scale * shifts(i);
        try_ = 0;
        while Objects{i}.get(Fields{i}) ~= newValue
            Objects{i}.set(Fields{i},newValue);
            if Objects{i}.get(Fields{i}) ~= newValue
                shifts(i) = shifts(i) / 2;
                newValue = Objects{i}.get(Fields{i}) + local_scale * shifts(i);
            end
        end
    end
end

```

```

        try_ = try_ + 1;
        if try_ > 3; shifts(i) = 0; break; end
        else; break;
        end
    end
    end
    take_a_break(i) = (~take_a_break(i) && abs(local_scale * shifts(i)) < tol);
end
end
if mod_pressure
    fprintf(['Shifting Pressure: ' num2str(i) '\n']);
    backup(pressure_ind) = RunConditions.EnginePressure;
    RunConditions.EnginePressure = min(options.MaxPressure,...
        max(options.MinPressure,RunConditions.EnginePressure + local_scale *
shifts(pressure_ind)));
end
if mod_speed
    fprintf(['Shifting Speed: ' num2str(i) '\n']);
    backup(speed_ind) = RunConditions.rpm;
    RunConditions.rpm = min(options.MaxSpeed,...
        max(options.MinSpeed,RunConditions.rpm + local_scale * shifts(speed_ind)));
end

% Discretize & Run using a single run
[Power, success, ShaftPower, statistics] = RunSubFunction(Model,RunConditions,options);

if ~success || isnan(Power)
    fprintf('Simulation Failed\n');
    for i = 1:length(Objects); Objects{i}.set(Fields{i},backup(i)); end
    if mod_pressure; RunConditions.EnginePressure = backup(pressure_ind); end
    if mod_speed; RunConditions.rpm = backup(speed_ind); end
    Power = power_backup;
    break;
end
if all(take_a_break(i))
    fprintf('Simulation Stalled\n');
    break;
end

% Test the step
fprintf('Testing the Next Step \n');
increasing = Power > power_backup*1.002;
if increasing
    if local_scale == 1
        % Continue stepping
        power_backup = Power;
        stepcount = stepcount + 1;
    else
        increasing = false;
        local_scale = 1;
    end
else
    % Undo the shift
    for i = 1:length(Objects); Objects{i}.set(Fields{i},backup(i)); end
    if mod_pressure; RunConditions.EnginePressure = backup(pressure_ind); end
    if mod_speed; RunConditions.rpm = backup(speed_ind); end
    Power = power_backup;
    save(Model.name,'Model');
    if stepcount == 1 && trial < 3
        % We overstepped this point, but the gradient is still valid
        % shifts = shifts;
        trial = trial + 1;
        increasing = true;
        local_scale = local_scale / 2;
    else
        local_scale = 1;
        stepcount = 1;
        increasing = false;
    end
end
end
end
end

```

```

if ~isfield(History,'Score'); History.Score = []; end
count = length(History.Score) + 1;

History.Score(count) = Power;

figure(h1);
plot(History.Score);
xlabel('Trial');
switch options.OptimizedProperty
    case 'Max Power'
        ylabel('Power [W]');
        title('Trend in Power during gradient ascent');
    case 'Max Thermo Power Per Unit Engine Volume'
        ylabel('Thermo Power [W] per m^3');
        title('Trend in Power Density during gradient ascent');
    case 'Max Efficiency'
        ylabel('Efficiency [%]');
        title('Trend in Efficiency during gradient ascent');
    case 'Max West Number'
        ylabel('West Number');
        title('Trend in West Number during gradient ascent');
end

for i = 1:length(Objects)
    History.data(i,count) = Objects{i}.get(Fields{i});
end
if mod_pressure
    History.data(pressure_ind,count) = RunConditions.EnginePressure/101325;
end
if mod_speed
    History.data(speed_ind,count) = RunConditions.rpm/60;
end

figure(h2);
xlabel('Trial');
ylabel('Position (m)');
for i = 1:size(History.data,1)
    plot(History.data(i,:));
    hold on;
end
legend(History.Names,'Location','northwest')
hold off;
if count > 1
    if History.Score(count) - History.Score(count-1) < 1e-2
        fprintf('Simulation Stalled\n');
        break;
    end
end
end

% History.EPara2 = EPara2;
% History.EGrad2 = EGrad2;
History.Scale = Scale;
History.gradient = gradient;
Study.History = History;
save(Model.name,'Model');

% Record Matrix of Body Sizes
for iGroup = Model.Groups
    for iBody = iGroup.Bodies
        if iBody.matl.Phase == enumMaterial.Gas
            if isempty(iBody.customname); field = ['Body_' num2str(iBody.ID)];
            else; field = replace(iBody.customname,' ','_');
            end
            [~,~,x1,x2] = iBody.limits(enumOrient.Vertical);
            [~,~,y1,y2] = iBody.limits(enumOrient.Horizontal);
            sets(optrial).(field) = [pi*(x2^2-x1^2) y2(1)-y1(1)];
            sets(optrial).Score = Power;
        end
    end
end
end
end

```



```

sets(optrial).Score = Power;
sets(optrial).Converged = success;
sets(optrial).statistics = statistics;
sets(optrial).HistoryScore = History.Score;
sets(optrial).HistoryDOF = History.data;
sets(optrial).HistoryNames = History.Names;
save(['Optimization_set_' replace(replace(date, ' ', '_'), ':', '-')], 'sets');
end
end

function [Parameter, success, ShaftPower, statistics] = RunSubFunction(Model, RunConditions,
options)
[success] = Model.Run(RunConditions);
addpath(['..\runs\' RunConditions.title]);
load([RunConditions.title '_Statistics'],'statistics');
ShaftPower = mean(statistics.Power);
switch options.OptimizedProperty
case 'Max Power'
Parameter = ShaftPower;
case 'Max Thermo Power Per Unit Engine Volume'
ThermoPower = 0;
for iPv = Model.PVoutputs
ThermoPower = ThermoPower + iPv.Power;
end
Vol = mean(statistics.VMax);
if ThermoPower < 0 % To prevent convergence on an incorrect sense of optimal
Parameter = ThermoPower;
else
Parameter = ThermoPower/Vol;
end
statistics.Power = ThermoPower;
case 'Max Efficiency'
Q = -sum(statistics.To_Source);
Parameter = ShaftPower/Q;
case 'Max West Number'
Wo = ShaftPower;
P = RunConditions.EnginePressure;
dV = statistics.VMax - statistics.VMin;
V = mean(dV(dV~=0));
f = RunConditions.rpm/60;
TH = RunConditions.SourceTemp;
TK = RunConditions.SinkTemp;
Parameter = Wo/(P*V*f)*(TH + TK)/(TH - TK);
end
statistics.Power = mean(statistics.Power);
statistics.Time = [];
statistics.Angle = [];
statistics.Omega = [];
statistics.TotalPower = [];
statistics.To_Environment = sum(statistics.To_Environment);
statistics.To_Source = sum(statistics.To_Source);
statistics.To_Sink = sum(statistics.To_Sink);
statistics.Flow_Loss = sum(statistics.Flow_Loss);
%{
Power = 0;
for PV = Model.PVoutputs
str = [PV.name '_' Model.name];
str = replace(str, ':', '-');
load(['..\runs\' RunConditions.title '\' str '.mat'], 'data');
pV = zeros(size(data.DependentVariable,1)+1,1); pP = pV;
for i = 1:size(data.DependentVariable,2)
pV(1:end-1) = data.IndependentVariable(:,i);
pV(end) = data.IndependentVariable(1,i);
pP(1:end-1) = data.DependentVariable(:,i);
pP(end) = data.DependentVariable(1,i);
Power = Power + PowerFromPV(pP,pV);
end
end
%}
end
end

```

```

function [grad, History] = getShiftObject(grad,obj,fld,History,...
    Model,RunConditions,options)
if grad < 0
    modshift = -0.001;
else
    modshift = 0.001;
end
grad = 0;
backup = obj.get(fld);
newValue = backup + modshift;
try_ = 0;
while obj.get(fld) ~= newValue
    obj.set(fld,newValue);
    if obj.get(fld) == newValue
        [Power, success, ~, ~] = RunSubFunction(Model,RunConditions,options);
        % Undo
        obj.set(fld,backup);
        if ~success
            modshift = -modshift / 2;
            newValue = backup + modshift;
            obj.set(fld,newValue);
            try_ = try_ + 1;
            if try_ > 8; break; end
        else
            grad = (Power - History.Score(end))/modshift;
            return;
        end
    else
        modshift = -modshift / 2;
        newValue = backup + modshift;
        obj.set(fld,newValue);
        try_ = try_ + 1;
        if try_ > 8; break; end
    end
end
if grad == 0
    fprintf('err');
end
end

function [grad, History, RunCon] = getShiftRunCon(grad,History,...
    Model,RunCon,Field,MinVal,MaxVal,options)
success = false;
backup = RunCon.(Field);
while ~success
    if strcmp(Field,'EnginePressure')
        modshift = 100;
    elseif strcmp(Field,'rpm')
        modshift = 0.1;
    end
    RunCon.(Field) = min(MaxVal,max(MinVal,backup + modshift));
    [Power, success, ~, ~] = RunSubFunction(Model,RunCon,options);
    if ~success
        modshift = modshift/2;
        RunCon.(Field) = backup;
    end
end
grad = (Power - History.Score(end))/modshift;
% if sign(grad) == sign(modshift)
%     % Take advantage of it
%     History.Score(end) = statistics.TotalPower(end);
% else
%     % Undo the small change
%     RunCon.(Field) = RunCon.(Field) - modshift;
% end
end
end

```

## Optimization Scheme

```
classdef OptimizationScheme < handle

    properties
        Model;
        name;
        ID;
        Names;
        Classes;
        IDs;
        Fields;
        History;
    end

    methods
        function this = OptimizationScheme(Model)
            if nargin > 0
                this.name = getProperName('Optimization Study');
                this.Model = Model;
                this.ID = Model.getOptimizationStudyID();
            end
        end

        function AddObj(this, obj, field)
            len = length(this.Names)+1;
            this.Names{len} = getProperName('Degree of Freedom');
            if strcmp(this.Names{len},'')
                this.Names{len} = [...
                    class(obj) ' - ' num2str(obj.ID) ' - ' field];
            end
            this.Classes{len} = class(obj);
            this.IDs{len} = obj.ID;
            this.Fields{len} = field;
        end

        function Item = get(this,PropertyName)
            switch PropertyName
                case 'Name'
                    Item = this.name;
                case 'DOFs'
                    Item = this.Names;
                otherwise
                    fprintf(['XXX Optimization Study GET Inteface for ' PropertyName ...
                        ' is not found XXX\n']);
            end
        end

        function set(this,PropertyName,Item)
            switch PropertyName
                case 'Name'
                    this.name = Item;
                case 'DOFs'
                    for i = length(Item):-1:1
                        if Item(i)
                            this.Names(i) = [];
                            this.Classes(i) = [];
                            this.IDs(i) = [];
                            this.Fields(i) = [];
                        end
                    end
                otherwise
                    fprintf(['XXX Optimization Study SET Inteface for ' PropertyName ...
                        ' is not found XXX\n']);
            end
        end
    end
end
end
```

