. . . for Solving and Optimizing Engineering Models

Sage
User's Guide

SCFusion Model Class

David Gedeon

Electronic Version for Acrobat Reader

Sage v13 Edition

Gedeon Associates
16922 South Canaan Road
Athens, OH 45701

February 1, 2025

Preface

User's Guide and Model Classes

This manual includes the Sage User's Guide and Model-Class Reference Guide for the SCFusion model class introduced in Sage version 13. The SCFusion model class (Stirling-Cycle Fusion) subsumes previous stirling-cycle, pulse-tube and low-T cooler model classes into a single model class going forward. It is backward compatible with those previous model classes in that it can read *.stl, *.ptb and *.ltc model files, automatically converting them to *.scfn files. SCFusion uses a 4-letter file extension because 3-letter extension like *.scf were already spoken for by other applications.

Manual Divisions

Parts I and II document the Sage software distribution files, the graphical user interface and general structure common to all model components running under the Sage modeling and optimization framework.

Part III documents the model component classes available in the SCFusion model class, including those software classes that have been deprecated under version 13. Deprecated means that they are no longer recommended, usually because they have been superseded by classes with improved functionality, but they are still available for backward file compatibility.

New in Version 13

On the software side, the Sage user interface and model components have continued to evolve since the previous version. Changes since version 12 documented in this manual are:

Sage GUI The Sage form now groups the EditForm and DisplayForm within its client area instead of spawning them as independent windows, making it easier to locate and manage them.

Explore Custom Variables Dialog The display can now be filtered to show only inputs, outputs, recasts, CAD-tagged or log-tagged variables. New

display icons make it easier to distinguish among the different types of customized variables.

Solve Status Dialog Now monitors line-step reductions that prevent solved variables stepping beyond allowed limits. For example temperatures in a 4 K cooler going negative. Identifies the responsible variable and the required step reduction, thereby pointing you toward the part of the model causing convergence difficulty.

Copy-and-Paste Model component copy and paste tools now support copying from one Sage instance and pasting into another. Previously it only worked by opening the source model to copy, then opening the destination model to paste, without closing the Sage application.

Grid Plots For 2D plots a new *animation* feature allows you to scroll through 2D plot curves sequentially — i.e. to plot individual position data curves at successive time nodes or individual time data curves at successive position nodes.

Independent Isothermal Heat Sources New isothermal-surface and line-heat-source components set temperature according to independent temperature-distribution inputs that can be recast in terms of higher-level user-defined inputs as a means for changing temperature boundary conditions of several lower-level components at once. Previous isothermal heat source components set the temperature according to the non-recastable Tinit temperature distribution inherited from the parent component.

Interpolated Components A new descendant of the simple transducer component interpolates the transduction force coefficient $C_f(x)$ (a function of position) from a set of cubic-spline data-pair inputs (x_i, C_{f_i}). A new interpolated spring component does a similar thing for spring force $F(x)$ from data-pair inputs (x_i, F_i).

Volumetric Compressor A new component added for modeling positive-displacement compressors of the type used in GM cryocoolers. Specifies inlet volumetric flow rate and clearance volume as inputs, rather than mass flow rate, thereby relating more directly to the specifications of an actual compressor and automatically adjusting mass flow rate as a function of upstream and downstream pressures, like a real compressor.

New Refprop Fluids added to the default gas data file: argon, carbon dioxide, methane, propane, isobutane, water. Existing helium, hydrogen and nitrogen are reformulated with improved bubble-line smoothing (below).

Bubble Line Property discontinuities at the bubble line, where vapor slams into the liquid phase, have plagued Sage convergence since the first *refprop* fluids were introduced in version 8 (2011). Improvements to $P(v)$ isotherm smoothing now allow Sage models to converge better through the bubble

line, allowing Sage to model a complete vapor-compression refrigeration cycle using a fluid like isobutane.

Duct Turbulence Model Revised and calibrated to experimental data, now with separate oscillatory and steady-flow branches and less spurious noise for highly non-sinusoidal flows common in GM cryocoolers and such.

Containers An issue within Parallel and Multi-Length containers prevented distributed conductors or conductive surfaces of annulus flow paths within composite piston-cylinder components from making y heat-transfer connections with other components of the container. This has been fixed.

Soft Ferromagnetic Materials To minimize glitches in iron-core solenoid inductance as a function of coil current, revised approximate magnetization function to eliminate slope discontinuity.

Moving Magnetic Containers Abandoned magnetic force scaling as a means of conserving energy relative to the energy entering from surrounding poles. Electromagnetic actuator models now converge more reliably at the cost of a small energy leak. Fixed a minor issue with magnetic flux distribution in voice-coil actuators that produced erroneous induction values.

New in Version 12

Changes since version 11 are:

PV power flows New built-in variables PVNeg and PVPos calculate average PV power flow rates at the negative and positive inlets of gas domain components.

Complex Nusselt numbers In solid and gas domains that calculate phase-shifted heat transfer and flow friction using correlations derived from simplified linear equations with complex exponential solutions, the formulation has been revised to filter out spurious higher harmonics. This is especially significant in low-temperature cryocooler models with highly non-sinusoidal temperature solutions and low solid heat capacities where heat-transfer higher harmonics without any physical basis could destabilize the solution or produce incorrect results.

Time-Grid Heat Connections New time-ring heat-source and heater components allow you to impose time-varying temperature or heat flow boundary conditions on the positive and negative ends of thick-wall, thin-wall or rigorous-surface thermal solids. Such connections impose temperature continuity at each time node, rather than just time-mean temperature continuity, which can be useful if the solid heat capacity is relatively low, allowing large time-varying temperatures.

Finding Example Models A new Help → Example Models menu item makes it easy to find example models distributed with the Sage application. It launches Windows File Explorer, open to the folder where example models and documentation are stored.

Low Temperature Regenerators The solid energy equation for the *quasi-adiabatic surfaces* that represent regenerator matrices now resolves instantaneous changes in specific heat rather than the time-averaged value used previously. This improves accuracy for regenerators operating below 10 K where solid specific heat is both small and highly variable with temperature.

Optimization Status Dialog Now includes a column of *attempted step* values displayed next to optimized values at every iteration. Potentially useful for spotting variables causing slow or erratic optimization progress.

CAD variables User-defined input and output variables can now be marked as CAD variables and written to a tab-delimited text file with the new File → Save CAD variables menu item. CAD variables are separate from logged variables tagged to appear in Optimization and Mapping logs. CAD variables are intended for driving key dimensions of CAD solid models.

Fourier series inputs Coefficients are now entered within cells of a string grid, rather than as individual data pairs. There is a new dialog for entering discrete function values rather than coefficients, and a means to toggle between coefficient and discrete entry formats.

Solid data The list of available thermal-solid materials contains some new materials like 6063 aluminum, nylon, kapton, and common low-temperature regenerator materials Er-Ni, Er3-Co, Er3-Ni, Ho-Cu2, Gd2-O2-S (GOS). Thermal properties for existing material have been updated according to NIST data at cryogenic temperatures. Copy and pasting columns of property data is now supported in the SCFProp utility used for translating property data into Sage format.

Renaming Model Components In-line editing capability now supported for model component names displayed in Edit Window.

Entering Property Data Solid or gas properties that vary with temperature like conductivity or specific heat are now entered within cells of a string grid, rather than as individual data pairs. Copy and pasting data from a spreadsheet is now supported.

Stick-Slip Dampers New stick-slip damper components approximate the frictional drag of a sliding object moving over a surface.

Grid Plots Now shown as modeless dialogs so you can view more than one plot at a time and keep the plot windows open while doing other things — e.g. monitoring solved variables during the solution process. A new

option adds point markers at grid node values, giving a better indication of actual solved values for time grids. If there is more than one grid in the selected component you can select the one to be plotted from a listbox. Previously only the default grid was available.

Diode A new diode component extends the options for modeling linear alternator electrical loads.

Solve Status Dialog Scroll bars added to view long exception error messages.

Explore Custom Variables Dialog There is now a *View Interpolation* button in the dialog to display the values of cubic-spline or Fourier series recast variables as continuous functions.

Generic Cylinders The gas-to-wall heat transfer formulation now converges more reliable for the case of an isolated volume without gas flow connections (e.g. a simple gas spring model) and more closely approximates the theoretical value.

RefpropToSage Revised pressure smoothing in the two-phase region from exponential smoothing at bubble- and dew-point corners to linear smoothing at dew-point corner. Updated *refprop* gases in GasLTC.dta.

An exhaustive list of software improvements starting from the earliest days of Sage, is found in the file UpgradeHistory.pdf. This file is located in the Docs\UpgradeHistory subdirectory under your Sage installation directory (default c:\Program Files (x86)\Gedeon\Sage[x]) or may be downloaded from the Sage website at www.sageofathens.com. Each software improvement documented there is generally associated with a revision in this reference manual.

Part I

Getting Started with Sage Software

Chapter 1

Installation

1.1 Computer Requirements

Sage runs under the Microsoft Windows operating system, most recently including Windows 10 and 11. A high resolution ($\geq$ 17 in) display monitor is convenient for editing complex models.

1.2 Installing

The correct installation procedure will update the Windows Registry and ensure that you can easily un-install your application later. Sage software is distributed as a single executable setup file (e.g. SageSCF.exe). You can run the setup file using the Add/Remove Programs utility located in the Windows Control Panel. Or just double-click on the file in Windows File Explorer or the equivalent. Then follow the on-screen instructions.

The Sage files that are installed on your computer are selected according to the license ID and product key you enter in the "enter license information" dialog that pops up during the installation process.

1.3 Un-Installing

The automatic un-installing process will remove all installed files and update the Windows Registry. Files created by you, either knowingly (data files) or invisibly (program initialization files) will remain in place. So afterwards, you might want to use Windows File Explorer to manually delete files from the installation directory or whatever directory you stored data files in.

Removal Activate Add/Remove Programs in the Control Panel, select the Sage application from the listbox and follow the prompts.

1.4 Files

The default installation directory is c:\Program Files (x86)\Gedeon\Sage[x]. Within the installation directory are a number of subdirectories. The first two listed below are part of the normal executable software distribution. The last two are part of the DLL or source-code distribution, licensed separately.

\Apps Contains executable program files and sample data files in subdirectories [ModelClass]\Bin and [ModelClass]\Samples, where [ModelClass] is the name of the installed application. You can and should store your own data files in whatever directory you choose. For example, you might use a main directory ..\Sagework, with subdirectories as needed to keep things organized.

\Docs contains document files (generally in Adobe Acrobat pdf format) including manual, technical notes, source-code instructions, the Sage upgrade history, and so forth.

\DLL Contains dynamic-link libraries and documentation in the [ModelClass] subdirectories.

\Source Contains source code common to all Sage applications in the Dialogs, Models, and Units subdirectories and for particular model classes in the SageApps\[ModelClass] subdirectories.

Each Sage application remembers from session to session such things as window dimensions, scroll positions, file folders. Some of this information is stored in an application-specific file named [ModelClass].ini, stored in the Windows CSIDL_LOCAL_APPDATA folder. Some is stored in a project-specific file AName.*in, in the same folder where your data file is stored, where AName is the name of your input file and .*in is the file extension particular to the application you are running. The application-specific file is updated whenever the program closes. The project-specific file when you save the data file. Both are regenerated automatically with default values if they are lost.

Chapter 2

Overview

2.1 What is Sage

Sage is a graphical interface that supports simulation and optimization of an underlying class of engineering models. The underlying model class represents something like a spring-mass-damper resonant system, a stirling-cycle machine, or anything else that has been properly coded to work with Sage.

The model classes of Sage are not just fixed-geometry models. Each may contain an unlimited number of variations or instances. A model instance, or just plain *model* for short, is a particular collection of component building blocks, connected and assembled in a particular way, with particular data values, forming a complete system representing whatever it is you are trying to simulate. In other words, you don't just add numerical data values within the confines of a presumed geometry. You may modify the geometry too. Each particular instance of a given model class resides in its own disk file with a unique name but common file extension particular to the Sage application you are running.

Each model class comes with its own executable file [ModelClass].exe for dealing with its own instances. Running a model-class executable file brings up the common Sage graphical interface which allows you to:

- create new or read existing model files
- enter numerical data
- edit model geometry
- specify optimization problems
- solve, map or optimize the model
- view, save or print a listing

These functions are all controlled by menu commands or toolbar controls.

2.2 What are Models

Models are more than the sum of their component building blocks. The way the components are organized and connected together is important too.

2.2.1 Models as Trees

Within Sage, model components are organized logically in a hierarchical tree structure. For example, the root model-component of a stirling machine contains a number of sub-components representing pistons, heat exchangers, and the like. These sub-components may themselves contain sub-sub-components. And so forth. The natural way to organize this in terms of child (sub) components branching off of their parent components — as trees in computer-science parlance — not unlike the directory structure of your computer file system.

The tree-structured point of view is especially convenient for organizing a model's file-storage stream or output listing. It does not reveal much, however, about the boundary-interconnections among model components, which are crucial to understanding the functioning of the model as a whole.

2.2.2 Models as Interconnected Systems

An alternate way to present models is through their boundary interconnections, which are the abstractions by which quantities like fluid flow, force, heat flux, etc., pass from one model component to another. A special form, known as the *edit* form, presents the model from this point of view. In the edit form, each model component is represented by an icon, with sibling components (belonging to a common parent component) grouped on the same page of the form. Boundary connections among components are indicated graphically by matching numbered arrows attached to the individual model components. In this way it is possible to understand the physical connections among components. An analogy would be this: A catalog of parts, even if tree-structured, tells us little about how an automobile works. We also need to know that the wheels are connected to the engine through clutch, gearbox and differential, before we begin to understand the whole machine. So it is with Sage. To understand your model you must take some time to delve through its interconnections.

2.3 Numerical Input and Output

Model components are self-contained entities. As such they manage their own inputs and outputs. Continuing with the automobile analogy: If you want to know what a wheel is doing, ask the wheel.

In Sage, if you want to specify input data for a model component, you do so directly within that model component. And if you want to find the output for a model component, you look within that same component. One ramification of this is that output listings are organized differently than you may be used to.

Instead of finding all similar quantities from the whole model listed together, you find a sequence of component sub-listings following each other in hierarchical order. A table of contents at the beginning makes it easy to navigate through the listing. Once you get the hang of it you will find it quite easy to home in on a particular component of interest and ignore the rest.

2.4 Solving, Mapping and Optimizing

An important thing to do with models is solve them. After you modify a model's numerical inputs, some of its numerical outputs may no longer be valid. This is because models are defined in terms of implicit relationships among variables which must be iteratively solved. Solving is a menu activated process that brings numerical outputs back into sync for the whole model hierarchy simultaneously.

You can also map your model, another menu-activated process available after you have selected a number of input variables to be automatically stepped over a range of values. The stepping sequence is that which would be produced by a nested loop structure. After each step, the model is automatically solved and selected outputs are stored in a disk file for later inspection. More details on mapping are in chapter 5.

Yet another menu-activated process is optimization, which is what you do after you have specified an optimization problem — involving optimized variables, constraints and an objective function. Unlike mapping, which is an exhaustive investigation of a broad area, an optimization is more like a locally-guided walk to the top of a hill. At each step of the way the model is solved and selected outputs are stored in a disk file. More details on optimization are in chapter 6

You can find out more about what's going on behind the scenes during solving and optimizing in chapter 11.

Part II

Sage General Reference

Chapter 3

Menu Commands

The important thing to know about menu commands is that they are often keyed to the model component currently active. An active model component is the one currently selected in the display or edit form, or whose caption within the edit form is highlighted. You can select multiple model components in the edit form by holding down the shift key while mouse clicking on them or dragging a selection rectangle over them. In that case the active model component is the one most recently selected.

Summarized in this chapter are only those menu commands unique to Sage. Common Windows menu commands are not listed.

3.1 File

The file commands generally have something to do with operations involving disk files, generally for the entire model as a whole.

3.1.1 File|New

Creates a new model instance belonging to a particular model class and puts up an empty edit form ready to accept components from the palette.

3.1.2 File|Open

Opens a existing model instance file.

3.1.3 File|Save

Saves the current model instance under the existing file name.

3.1.4 File|Save As

Allows you to change the file name of the current model instance before saving. Generally used when you want to make a change but save the old model too. As of version 8 the "Save as type" selection list in the save-as dialog allows you to save under a previous stream format, currently limited to version 7. The resulting output file can be read by the previous version of Sage but any new model component classes or variables added since that version will not be included in the stream. If your model contains components not supported in the previous version you will be asked to manually remove them before continuing. Once saved as a previous version subsequent saves will also be in that previous version until you change the type selection in the save-as dialog.

3.1.5 File|Listing

Opens a dialog that shows a text listing of input and output values for all components of the model in its present state. Components are listed in hierarchical tree order with a table of contents at the top. The dialog includes check boxes that allow you to select various categories of output to appear in the listing, search capability, text formatting functionality and menu items for printing the listing or saving it to a file in rich-text format.

3.1.6 File|Save Solution Grid

Creates a file containing detailed solution variables for any computational grids used by the model focused in the display or edit form and its connectors. Includes the grids of child model components (components that appear in lower-level windows of the edit form) and their connectors.

3.1.7 File|Save Log Variables

Creates a file containing all user-defined inputs and outputs selected to appear in log files. You so designate a user variable by checking the Write to Log file check-box in the user-variable input dialog or in the Tools|Explore Custom Variables dialog. Log-file tagged user-variables are indicated with a * prefix in the selection dialog list box of the Specify|User Inputs or Specify|User Variables dialogs.

3.1.8 File|Save CAD Variables

Similar to File|Save Log Variables, except for user-defined inputs and outputs with the CAD variable check-box selected in the user-variable input dialog or in the Tools|Explore Custom Variables dialog. CAD variables are intended for driving key dimensions of CAD solid models. CAD variables are indicated with a ^ prefix in the selection dialog list box of the Specify|User Inputs or Specify|User Variables dialogs.

3.1.9 File|Save Embedded Properties

Displays a selection list of all materials (those with unique names) embedded in the entire model and allows you to save the selected material's properties to an individual data file. This is useful if your default property data file (*see* chapter 28) does not contain that particular material. You can then use the SCFProp utility to append the saved data file to the appropriate property data file where it will then be available in the input selection list for any named material variable of the model.

3.2 Display

The display commands pertain to the display form which appears to the right within the main Sage form. The display form shows textual information for selected model components.

3.2.1 Display|Add Page

Adds to the display form a page (or pages) for an particular model component selected from a model-tree selection dialog.

3.2.2 Display|Remove Page

Removes the currently selected page from the display form. Removing a page does not affect the underlying model component that was displayed.

3.2.3 Display|Remove All

Clears all pages in the display form.

3.2.4 Display|Print Display

Prints the contents of the currently selected page in the display form.

3.3 Edit

The edit commands pertain to the edit form which appears to the left within the main Sage form.

3.3.1 Edit|Select All

Selects all model components in the edit form.

3.3.2 Edit|Up Connector

Moves up the highlighted connector arrow(s) one level in the model hierarchy toward the root, enabling you to connect it to a mate at that level. To highlight a connector arrow click on it. Hold down the shift key to highlight many.

3.3.3 Edit|Down Connector

Opposite of the up command.

3.3.4 Edit|Cut Model(s)

Removes the currently selected model components from the edit form to a clip-board of sorts, as a means of deleting them with undo capability. Available only if the selected components are not connected to other model components outside the group. Only the most recently cut model components can be pasted back.

3.3.5 Edit|Copy Model(s)

Copies the currently selected model components to the clip-board but does not remove them from the edit form. Useful for cloning model components, including their child components.

3.3.6 Edit|Paste Model(s)

Pastes model components, including child components, from the clip board to the edit form. Multiple pastings are possible to undelete or clone model components. The paste function works for pasting model components into the parent component originally copied from or a different parent — possibly in a different model file. This allows you to create a new model file by copying components from one or more existing model files. The only restriction is that the paste parent be capable of supporting the copy. In other words, the paste parent must contain seeds in its child-creation palette of the same class types as the components to be pasted.

3.3.7 Edit|Delete Model(s)

Deletes the currently selected model components in the edit form. Available only if the selected components are not connected to other model components outside the group.

3.3.8 Edit|Change Bitmap

Launches a dialog that permits loading a new bitmap image for the active model component from a disk file or restoring the default image.

3.3.9 Edit|Print Form

Prints the edit form as it appears on the display monitor.

3.4 Scan

For reviewing or specifying information for the whole model sub-tree beginning with the active model component. If the root model component is active then Scan includes the whole model hierarchy.

3.4.1 Scan|Input Values

Scans and allows modification of numerical inputs.

3.4.2 Scan|User Inputs

Scans and allows modification of user-defined inputs.

3.4.3 Scan|User Variables

Scans and allows modification of user-defined outputs.

3.4.4 Scan|Recast Variables

Scans and allows modification of recast inputs.

3.4.5 Scan|Mapped Variables

Scans and allows modification of variables stepped in the map process.

3.4.6 Scan|Optimized Variables

Scans and allows modification variables solved in the optimize process.

3.4.7 Scan|Constraints

Scans and allows modification of constraints.

3.4.8 Scan|Comments

Scans and allows modification of comments.

3.5 Specify

For specifying information for the active model component only.

3.5.1 Specify|Input Values

For numerical data input or modification.

3.5.2 Specify|User Defined Inputs

For creating and modifying special user-defined inputs. See chapter 4.

3.5.3 Specify|User Defined Variables

For creating and modifying special user-defined outputs. See chapter 4. In the definition dialog, checking the "write to log file" box tags the variable so it will appear in mapping or optimization log files.

3.5.4 Specify|Recast Variables

For recasting independent input variables as dependent variables, defined in terms of an algebraic expression involving other model variables. See chapter 4.

3.5.5 Specify|Mapped Variables

For selecting variables to be stepped in the map process.

3.5.6 Specify|Optimized Variables

For selecting variables to be solved in the optimize process.

3.5.7 Specify|Constraints

For specifying equality or inequality constraints for the optimize process.

3.5.8 Specify|Objective Function

For specifying the objective function for the optimize process.

3.5.9 Specify|Rename

For changing the name of a model component. As of v12, model component names can also be changed by clicking on the name caption beneath the model component icon in the edit window and editing the text directly.

3.5.10 Specify|Comment

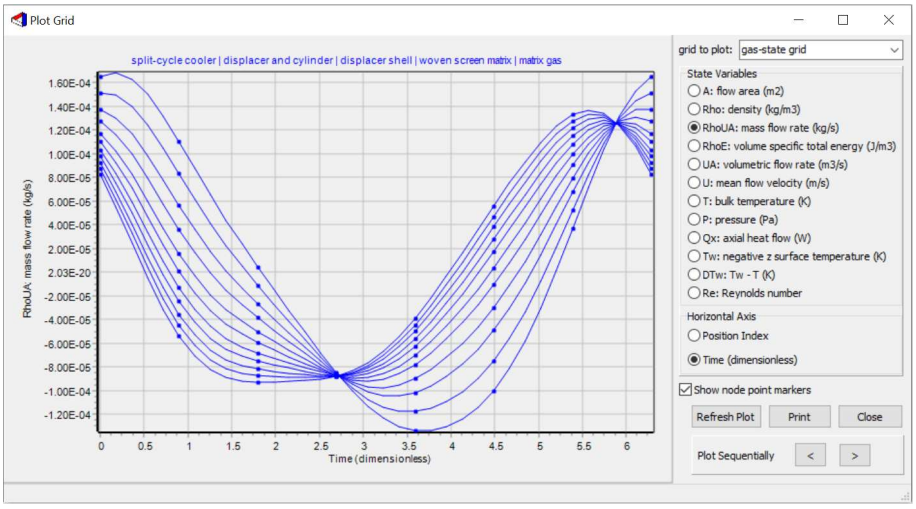
For entering an optional comment for a model component. A comment is a text string of arbitrary length, possibly containing multiple lines. Comments appear in the display window or listing.

3.5.11 Specify|Child-Model Order

For changing the order in which sibling model components appear in the model listing and certain dialog boxes.

3.5.12 Specify|Plot Solution Grid

Shows an interactive dialog that plots the current computational grid, if any, of the selected model component, as illustrated below.



You can select which of possibly several different *state* variables to plot, specify either time or position as the variable along the horizontal axis, and step through the individual curves sequentially. You can also show the grid-plot dialog via popup menus. *See* section (3.10).

3.6 Process

3.6.1 Process|Solve

Brings numerical outputs up to date with numerical inputs that may have been changed.

3.6.2 Process|Map

Carries out the mapping sequence specified by the mapped variables.

3.6.3 Process|Optimize

Carries out the optimization problem specified by the optimized variables, constraints and objective function, if any.

3.6.4 Process|Parse Solution

Parses or compiles the evaluation expressions in user-defined variables, along with other model setup tasks, without actually solving the model.

3.6.5 Process|Parse Mapping

Parses or compiles the evaluation expressions for mapped variables, without actually mapping the model.

3.6.6 Process|Parse Optimization

Parses or compiles the evaluation expressions in constraints and the objective function, along with other optimization setup tasks, without actually optimizing the model.

3.6.7 Process|Reinitialize

Resets all implicitly solved model variables to their initial values. Helpful when a change to a numerical input or model structure causes the model solver to fail to converge.

3.7 Tools

3.7.1 Tools|Explore Optimization

Lists all optimized variables in a model, grouped by model component, with subject-to constraints listed in a parallel column. Useful for understanding the optimization structure of complicated models and for finding over-constrained or infeasible optimization specifications.

3.7.2 Tools|Explore Custom Variables

Displays an interactive form containing a tree-structured view of all user-defined inputs, outputs and recast inputs in the whole model. You may click on a variable identifier in the tree view to display its value and defining information or use the Find button to locate a variable anywhere in the model by matching its identifier against the one you type in the identifier box. Other buttons allow you to trace which variables reference or depend on other user-defined variables and make editing changes. A submenu that pops up after clicking the right mouse button supports copy and paste operations. *See* section 4.10.2

3.8 Options

3.8.1 Options|Sage

Activates a self-explanatory dialog for setting options pertaining to all Sage applications.

3.8.2 Options|Model Class

Activates a self-explanatory dialog for setting options pertaining to a particular model class.

3.9 Help

3.9.1 Help|PDF manual

Shows an electronic version of the Sage User's Guide opened under Adobe Reader. It is the same as the printed User's Guide except with hyperlinked contents, index and internal cross references. All of the navigation features of a standard Windows help file are available along with better typography and formatting.

3.9.2 Help|Sample Models

Launches Windows File Explorer, open to the folder where sample models and documentation are stored for the Sage application currently running.

3.9.3 Help|About

Shows version and copyright information.

3.10 Popup Menus

The *Specify* menu items are also available under popup menus on clicking the right mouse button when positioned on the selected model component in the display window or edit window. The popup menu also includes an item for toggling the view of the selected model component from the edit window to the display window and vice-versa.

There is also a popup menu associated with connection arrows. It allows you to plot the solution grid for the connection, provided the connection is active (solid arrow) and the quantity *flowing* through the connection is solved on a grid, usually labeled with a t or x suffix, as in F_{Gt} or Q_{Gxt}. To show this popup menu, left-click to select a connector arrow then right-click the selected arrow to show the menu. See the Specify|Plot Solution Grid menu item for more information about solution grid plots.

Chapter 4

Working with Models

4.1 Data Files

Use the File|New menu command to create a new model or the File|Open command to open an existing model file. Creating a new model opens an empty edit form and fills the model-component palette within the main Sage form with potential model components. See chapter 7 for what to do after that. Opening an existing model file restores a model to its numerical state and display appearance at the time the model was last saved.

You are responsible for saving a model when you have made changes to it. Use the File|Save command or the File|Save As command if you want to save it to a new file and simultaneously change the file name for subsequent Saves. It is always a good idea to save your model before making a lot of changes to it, in case you want to revert back to the old file. Otherwise, undoing model changes is a manual process. The one *undo* feature in Sage is to re-paste a deleted (cut) model component back into the edit form, but it works only for the most recently deleted component.

When you save a model file, there are actually two files saved, a model-specific file and a Windows initialization file containing the state of the model Windows interface. The files have the same name but different file extensions. You will normally not be aware of the Windows initialization file and it will be regenerated if lost.

4.2 Viewing Model Structure

The edit form (to the left within the main application form) shows the child model components within a given parent model component, along with the connections among them. If the parent component is the root-level component, you are looking at the whole model at the highest level of abstraction. The name of the current parent component is on the selected tab at the bottom of the form. You can move up and down in the parent-child hierarchy in one of

two ways: Double-clicking on a component icon in the window will activate that child as the new parent component. Or clicking on a tab will change that model component to the parent component. The tabs are designed to hold only one pathway (from root to terminal component) of the model tree at a time.

Keep in mind that many menu commands are keyed to the model component that is currently active. Within the edit form, the active model component is the one whose icon caption bar is highlighted. Or, if no component icons are highlighted, it is the parent model component itself. Highlighting a model component is just a matter of clicking on its icon. Click in the form client area between child-component icons to activate the parent component. Or click on the parent's tab.

There are two purposes for the edit form. The first is passive display of the existing model structure to help you understand what the model is all about. You may print the edit form as it appears on your display monitor, with the Edit|Print command. The second purpose is to support interactive modification of the model structure. This is the subject of chapter 7.

4.3 Viewing Model Data

Proceeding from the general to the specific, you may view your model's data with listings, display-form pages or solution-grid files.

4.3.1 Listings

The format in which model files are stored on disk is a special binary file stream understood by Sage alone. It is not appropriate for human readable hard copy. For readable copy, you will need an output listing which you may either preview, print or save-to-file using menu sub-commands Listing Preview, Print Listing or Save Listing under the File|Listing... main menu item. An output listing contains current numerical inputs and outputs for the entire model hierarchy. There is a table-of-contents header at the beginning of an output listing which numbers its various sections (one for each model component) using a notation that conveys the model-tree hierarchy. Using this identification number it is easy to scan through the listing to the location of the model-component you are interested in.

You can select among various categories of display using the File|Listing Preview command, then checking the appropriate boxes at the top of the preview dialog. For example, to display a model's inputs only, check the *inputs* box. To display a model's optimization structure only, check the *optimized*, *constraints* and *objective function* boxes. And so forth.

The File|Print Listing command produces a non-formatted printout without margins or intelligent page breaks. For a formatted printout you should instead save the listing to a file using the File|Save Listing command. The result is an ASCII text file which you may then format and print as you see fit with your favorite word processor or text printer.

4.3.2 Display Form

The display form contains the same textual information as a listing except you include in the form only those model components you are interested in. The display for a single model component is known as a page, and each page has a corresponding tab at the bottom of the form. You display different pages by clicking on the corresponding tab. To add a new page to the form use the Display|Add Page command or highlight a component in the edit form and select Show in Display Window in the right-click popup menu. To remove a page, select that page then use the Display|Remove Page command. To print a page use the Display|Print command. A display-form page contains the complete textual output for a model component. There is no way to restrict the display by categories.

Although the pages of the display form are read-only (do not support direct editing operations) the Scan and Specify menu items are keyed to the model component that is currently active — the one whose display page is selected.

4.3.3 Solution Grid Variables

The standard numerical outputs that appear in the listing or display form are usually single numerical values representing discrete quantities — integrals or averages, that sort of thing. Often, a model component has an underlying computational solution grid behind the outputs. You can inspect this grid via the File|Save Solution Grid command which produces an ASCII format disk file containing all the numerical solution grids for the model component currently active in the edit form or display form and all its child model components.

You should keep in mind that not all model components contain solution grids. Generally, the highest level components (those visible in the root page of the edit form) do not. Solution grids are usually found only in lower-level components of a model class. To see which components contain grids you will have to read the documentation for the model class you are working with.

An output file may contain several grids each containing several individual state variables. The easiest way to explain this is with an actual example, such as the following which show the solution grid for a time-ring reciprocating mass:

```
resonant system | reciprocator
position displacement grid
X: displacement (m)
-7.569E-04 -3.784E-04 3.784E-04 7.569E-04 3.784E-04 -3.784E-04
Xd: velocity (m/s)
-1.720E-20 2.471E-01 2.471E-01 1.668E-20 -2.471E-01 -2.471E-01
Xdd: acceleration (m/s2)
1.076E+02 5.378E+01 -5.378E+01 -1.076E+02 -5.378E+01 5.378E+01
```

The solution values for the three state variables of the position displacement grid are listed. The individual variable values appear as tab-delimited fields within

a line or record. For the present case — a pure time grid — the entire grid of values fits in a single record, starting at time zero and equal-spaced throughout the periodic interval. For a pure space grid it would be much the same, starting at the negative domain endpoint and ending at the positive domain endpoint, also equal-spaced. For a space-time grid, there would be a sequence of records, each record corresponding to the time nodes at a fixed spatial position. The first record would consist of the time nodes at the negative domain endpoint, and so forth. In this example there is only one grid, the position displacement grid.

You can inspect a solution grid file with any text editor or word processor. Or you can open it with a spreadsheet program, such as Excel, where the tab-delimited fields should arrange themselves into neat rows and columns. Besides looking at your data, you can then plot it or operate upon it in other ways as you see fit.

4.4 Numerical Input

Once you have created a model structure or read an existing model file, you will generally want to change one or more numerical inputs.

To change a numerical input, first select the model component in which it resides by mouse-clicking the appropriate tab in the display or edit form, or clicking the appropriate icon in the edit form. Then select Specify|Input Variables from the main menu or right-click context menu. You will be presented with a list of input variables pertaining to that particular model component only. To change a variable value click on that variable and an appropriate input dialog will appear, geared to the format of the individual variable. It is here that you actually enter the value. Sorry, but you cannot directly edit a variable value within the display form.

Sometimes you want to change or inspect numerical inputs for more than just a single model component. This is possible with the Scan|Input Variables menu command. Using this option you are presented with a list of all model-component names requiring numerical input for the active model and its entire sub-tree. Clicking on a model-component name brings up the appropriate input-specifying dialog. Often, scanning input variables this way will be much more convenient than individually selecting model components one at a time. When you want to inspect input variables for the entire model hierarchy, just scan the root model component.

4.5 System of Units

It is possible to change the dimensional units displayed for the variables of your model. To do so, make the appropriate selections in the dimensions page of the model-class options dialog available under the Options|Model Class menu item. For example, if you change the length dimension from m (meters) to

in (inches), all variables whose dimensions were previously listed as (m) are immediately converted to (in). Also affected are any variables with derived dimensions involving length. For example, variables in (m/s) are converted to (in/s).

Changing dimensional units does not affect your model's solution, nor does affect the internally-stored values for any built-in variables, which are always in SI units (International System) or dimensionless. It merely changes your model's appearance by means of a value-conversion layer of software built into the visual interface. So it is safe to change dimensions back and forth as often as required. No information will be lost.

But, changing dimensions is not entirely without subtle consequences. This is because the value-conversion layer of software also affects the internal values of any user-defined variable (section 4.8), constraint or objective function (chapter 6) whose value derives from a string expression referencing a dimensional variable. For example, if you have defined a constraint like $X \geq 0.04$, its value is based on the current dimensional value of X. Anything else would be confusing. But as a result, the constraint may be satisfied if X has units of (m) and violated if it has units of (in)! Similarly, in a mapping specification (chapter 5), the range you specify applies to the dimensional value of the mapped variable, as you would expect. So the consequences of mapping X over a range $[0.010, 0.020]$, depend on the current dimensions of X. Sage gives you the responsibility for making sure that your mapping and optimization specifications survive a dimensional change. To be safe, it is a good idea to settle on a system of units at the beginning of a project and stick with it, only changing it from time to time to compare with engineering drawings in other units or communicate with actual engineers who prefer to speak in other units.

The state of your model's dimensional units is saved in the project-specific initialization file logically paired with your model's input file (same name). This initialization file is updated whenever you save your model and read whenever you load your model. So exiting Sage without saving your model, or just reloading it, will abandon any temporary dimensional changes you might have made.

4.6 Solving

After you have modified the model structure or changed numerical inputs, you will notice that the warning *not solved* appears after the *Outputs* heading in the display form. This means you can no longer trust the values displayed. They are displayed anyway as an aid to diagnostics during the solution process.

To re-validate numerical outputs you must solve the model with the Process| Solve menu command. This command initiates an iterative process that may take a significant amount of time to finish. In some cases the solver may not even converge.

To keep you informed during the solving process, Sage puts up a status dialog which displays useful information and allows you to stop or pause the

process if things are not going well. The goal is for the solver to drive the model's RMS error function (measure of how close the model's implicit function values are to zero) to some target value. When the RMS error fails to go down after a reasonable number of iterations (30–40 or so), you have trouble. You can inspect the individual components of the RMS error function and possibly diagnose problems using the techniques described in section 4.7.

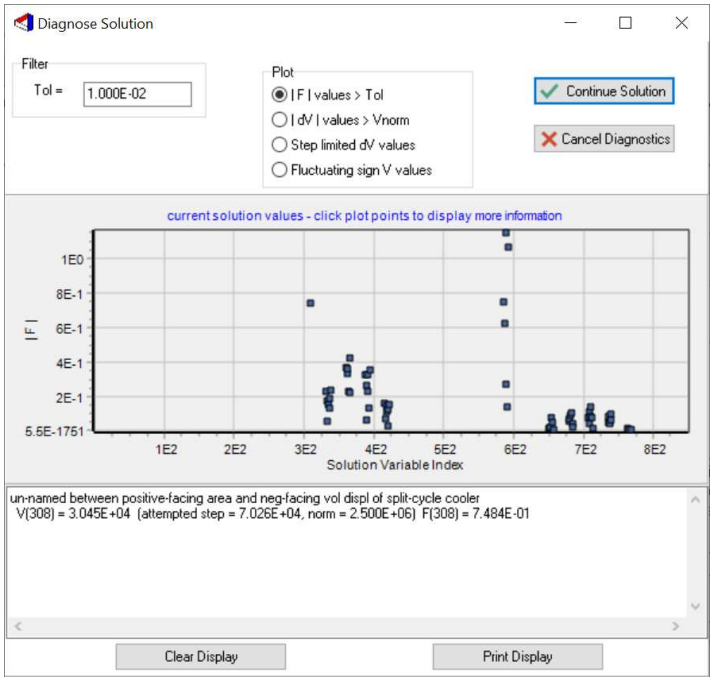
Sometimes a model will not converge for a good physical reason, such as a missing boundary condition or bad initial conditions. The first and easiest thing to try is to stop the current solving process, re-initialize all variables with the Process|Reinitialize command then try again. This often works if the problem is due to remnants of a previous solution being incompatible with a drastic change in model structure or numerical input. Sage never throws away solution information unless you tell it to. It assumes previous solutions are reasonable initial conditions for subsequent solutions, which pays big dividends when you or its optimizer make only small changes to numerical inputs. If re-initializing doesn't work, it may be that the initial values are too far from the converged solution. Most model classes give you some control over this by including a number of constant inputs which are used to set initial conditions. These range from normalization values for key physical dimensions (usually in the root model component) to initial temperature distributions and the like. The idea is to set any such constants to reasonable values for your particular model-class instance. Documentation for individual model classes should offer hints about setting these constants.

If converge continues to fail, it may be that you have a physically absurd model. Helping you to understand why, is the reason Sage displays numerical outputs, even if not solved. In fact, after each solver iteration, sage updates its numerical outputs. Variables tending to zero or infinity will often give you vital clues about why your model is not converging. A good physical understanding of your model is important here. If all else fails, you may have to throw away your recent changes and resort to the last working version of your model file. You did save one, right?

4.7 Solver Diagnostics

In the event Sage's solver fails to converge and the methods discussed in section 4.6 don't help, you may want to take a look behind the scenes by checking the

solver diagnostic dialog box under the Options|Sage menu command.



The result is a diagnostic dialog displayed after each Jacobian-matrix factorization, usually every solution iteration for a non-converging model. This dialog allows you to graphically inspect implicit variables and system function values for model components that may be causing convergence problems. In your model, each implicitly solved variable V is associated with a function component F, to be driven to zero during the solution process.[1] By selecting the appropriate radio button in the dialog you can display function values that fail to get small, extreme variable steps, out-of-range variable steps or fluctuating-sign variable values. Clicking on a plot point displays information about the corresponding V and F values, in the memo box at the end of the dialog. V values are SI dimensioned values. F values are dimensionless and normalized (scaled to a range of values on the order of one).

In any of the plots, if there are only a few consistently large values they may point you to an area of your model that is causing convergence problems. By focusing your attention on that part of the model you may be able to understand what the problem is and how to correct it. If you isolate the problem to a computational grid you may want to explore that grid in detail using the

[1] For example, the implicit variable Xddot (acceleration) within the reciprocating mass solution grid, at some particular time, is associated with a function component representing the residual force of Newton's second law of motion "mass × acceleration − summation of forces", at the same time.

File|Save Solution Grid command. Diagnostic plot options are:

$|F|$ values > Tol With this button clicked you see a plot of all system function values whose absolute value exceeds the current Tol value in the tolerance edit box. Keep an eye out for F values that remain large from one iteration to the next.

$|dV|$ values > Vnorm Shows all *extreme* solution variable steps requested (but not necessarily taken) by the previous iteration of the nonlinear solver in its attempt to simultaneously zero all the F's. Generally, for a well behaved solution the dV steps are smaller than the normalization values V_{norm} and nothing is plotted. Otherwise it may be a sign that the model equations are ill conditioned or indeterminate. The solver imposes limitations to prevent actually taking extreme steps but this plot shows the raw step values prior to any limitations. If you see any normalized step values $>>$ 1 it means that these variables are only weakly determined. Sometimes this happens when starting from initial solution values or after the solution is re-initialized and then improve when the solution evolves a bit. If the problem persists then you might want to change the model structure to more strongly determine the solution variables displayed in the plot. For example, impose stronger thermal anchoring, position anchoring, etc.

Step limited dV values Shows all variables where the solver-recommended step exceeded the allowed limit. For example, a step into negative territory for a variable restricted to positive values, or a too-large steps for a variable whose steps are limited to stabilize solution convergence.

Fluctuating sign V values This plot is available for the second solve iteration and higher. It plots all solution variable values that have changed sign since the previous iteration. Directional variables (e.g. mass flow rate) sometimes get stuck in an oscillating loop where they hunt back and forth near zero. This is because certain function components may depend discontinuously on the direction of such variables. For example, the gas thermal energy equation for a computational cell depends on the temperature of the fluid flowing through the cell boundary, which changes discontinuously when the flow direction changes. Variables in this plot may help you identify such problems. Slightly shifting the reference phase angle of whatever component is driving the solution (e.g. phase angle of moving piston) can remedy problems of this type. Such phase shifting can prevent direction reversal times from coinciding with nodes in the time grid.

4.8 User Variables

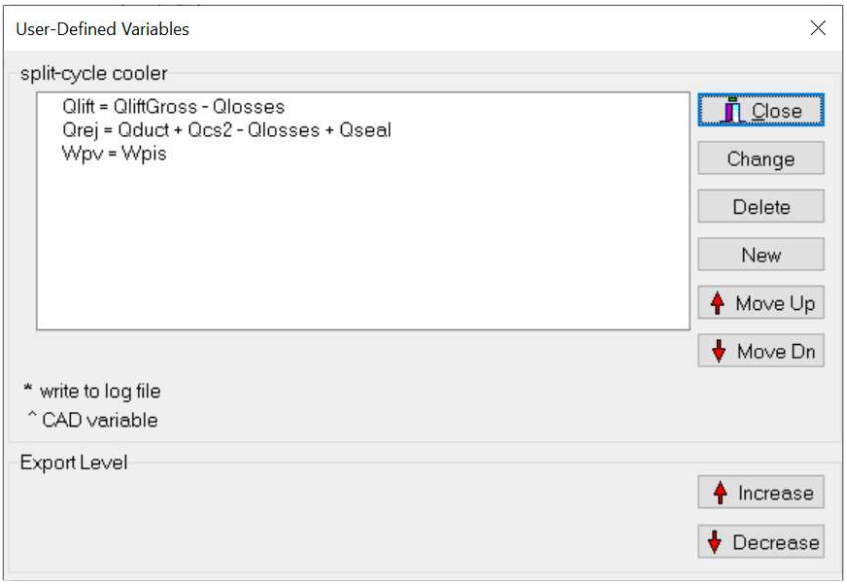
Sometimes the numerical outputs programmed into model components may not be sufficient for your needs. You may find you are always having to add together

a number of individual outputs, from one or more model components, to get the answer you are interested in. User-defined variables can help.

User-defined variables are special output variables that you yourself add to model components and define by entering algebraic expressions in terms of other variables known to that model component. They become part of the model component and appear in the display window and output listing. They may also appear in log files that accompany mappings or optimizations. Variables referenceable in the defining expression are any of a component's own input or output variables, provided they have a numerical type, as well as any user-defined variables in its model sub-tree, provided their export level is high enough. Sounds complicated but it really isn't. You just have to be a bit organized.

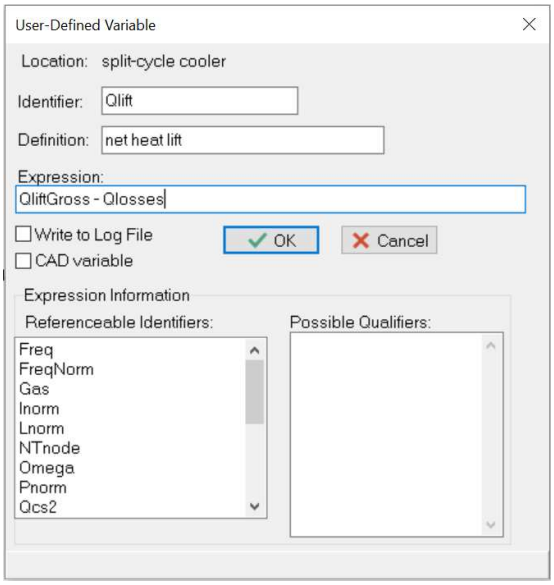
For example, you may want to define a variable named Qin (net heat input) whose value is Qh + Qparasitic, where Qh and Qparasitic are names of two referenceable variables, in this case other user-defined variables. All you do is activate the model component in which you want your variable to reside and select the Specify|User Defined Variables menu item. A dialog will then open, allowing you to create a New variable, Change an existing one, and so forth.

(**note** you can also specify user-defined variables for the entire model in a structured way by selecting the Tools|Explore Custom Variables menu item)



With this dialog you can also set the selected user-variable's display order (Move Up, Move Dn buttons) and Increase or Decrease its export level (visibility for purposes of referencing by other user-defined algebraic expressions, *see* below). When you click on the New or Change buttons A sub-dialog prompts you for

the vital information that defines the variable:



Within the sub-dialog you have the opportunity to enter your variable's identifier, a defining comment and an algebraic expression. The identifier is the name used to reference the variable in other algebraic expressions of your model, similar to the variable identifier in a programming language. The defining comment is only for human use, to help you, and others who might read it, keep track of the variables purpose. More information about referenceable variables and acceptable syntax for the algebraic expression can be found in chapters 8 and 9. You may also designate your variable to appear in mapping or optimization log files or CAD files by checking the *write to log file* box or *CAD variable* box.

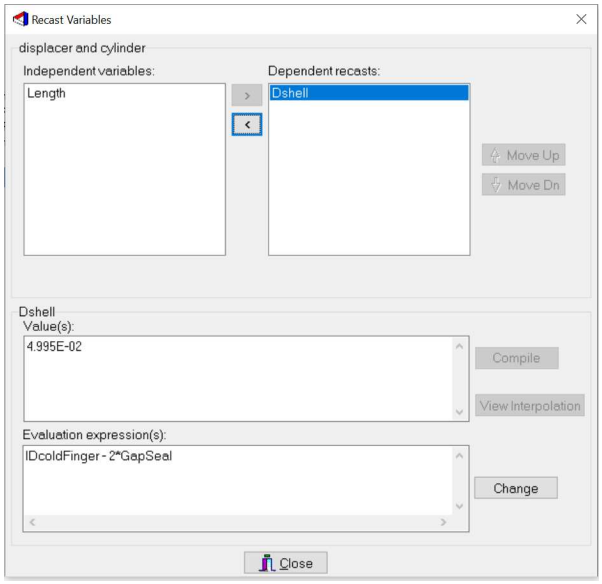
The export level, defined in the second dialog above, is the highest-level parent model component in which the variable will be referenceable. It is not automatically as high as possible to avoid potential name conflicts with variables in other branches of the model tree. Variables are automatically referenceable in the model component in which they are defined and any lower-level child model.

Writing algebraic expressions for user variables is a bit like programming in a typical computer language. It is possible to make grammatical syntax errors that render your expression unreadable by Sage. To see if you have done this, you may use the Process|Parse Solution menu command. Or you may just try to solve the model without taking this step. If Sage cannot parse your algebraic expression it will put up a dialog box showing your expression with the cursor at the offending location, giving you an opportunity to change it. You will find a terse comment describing the problem within the status bar at the bottom of the dialog box.

4.9 Recast Variables

Sage model components are designed for general use within a number of possible different hardware configurations. For example, a "piston-and-cylinder" component may represent a piston within a stand-alone pressure wall or a displacer within a cylinder that is the inner wall of an annular regenerator. In the later case the length of the displacer is related to the length of the regenerator but the two lengths are specified independently in the Sage model. You can eliminate the need for entering both inputs independently by *recasting* the displacer length as a dependent variable, defined in terms of regenerator length. In general you can recast any real-valued independent input variable (or variable with real parts) as a dependent variable so that Sage calculates its value during the solution process in terms of a user-defined expression involving other model variables. Recast variables appear in the Sage listing or display windows under the heading Recasts and their calculated values appear under the Outputs heading.

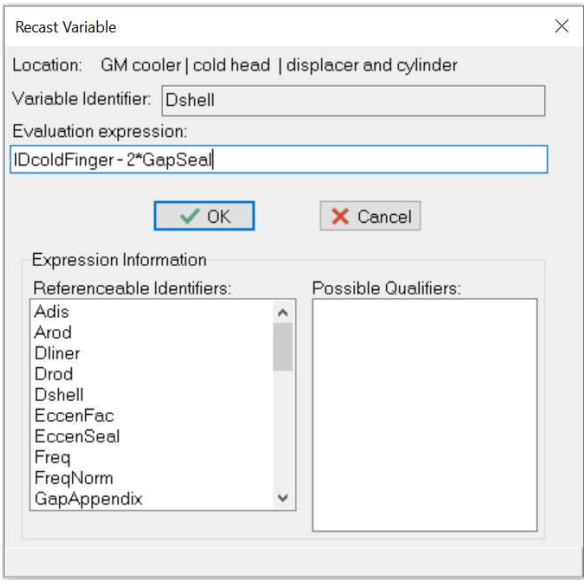
To recast an independent input variable, activate the model component in which it reside and select the Specify|Recast Variables menu command. A dialog opens, showing variables eligible for recasting and those already recast.
(**note** you can also specify recast variables for the entire model in a structured way by selecting the Tools|Explore Custom Variables menu item)



The pertinent information for the selected recast variable is displayed at the bottom of the dialog. Using the Change button you can revise the defining algebraic expression (*see* below). Using the Move Up or Move Dn buttons you can change the display order in the list box and also in the Sage listing or dis-

play window. When you change the defining algebraic expression Sage has to parse the expression before it can calculate its numerical value. Pressing the Compile button initiates the parsing process. If successful the calculated value is displayed in the value box. If not, a dialog pops up indicating the reason for the problem and allowing you an opportunity to fix it. The ViewInterpolation button allows you to see intermediate values for more complicated recast variables like cubic-splines or Fourier series (*see* below).

The > and < buttons move a selected variable from the "independent variables" to the "dependent recasts" list or the other way. Moving a variable to the "dependent recasts" list immediately opens its definition-dialog, similar to the dialog used to specify user-defined variables:



Moving a variable to the "independent variables" list restores it to its original state, except that its value becomes the most recent value it had as a recast variable.

In the above example the displacer Length is recast to equal Ldis, which is a user-defined input defined at the root level (*see* section 4.10).

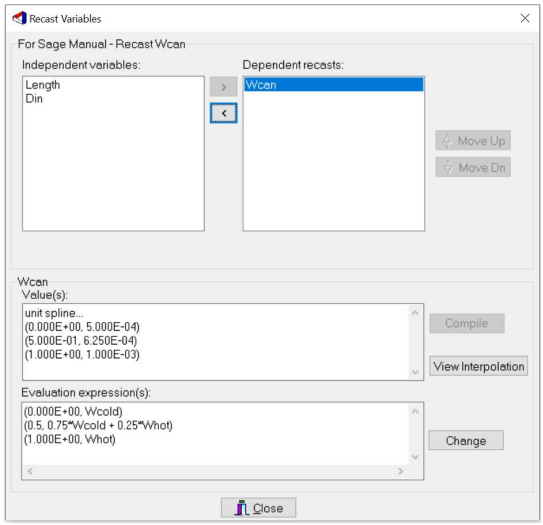
Recasting is not the only way to implement geometric model constraints. It is also possible to do so during the optimization process. Using recast variables however, eliminates explicit constraints and associated variables from an optimization structure, making it easier to understand and run faster. Moreover, recast variables are updated as part of the solution process and do not require re-optimizing the model each time you change an associated input variable. There are also advantages for the mapping process because a single mapped variable can affect other inputs that you have recast as dependents, allowing you to map along a *curve* in model space rather than just along the input *coordinate*

directions.

4.9.1 What Variables are Recastable?

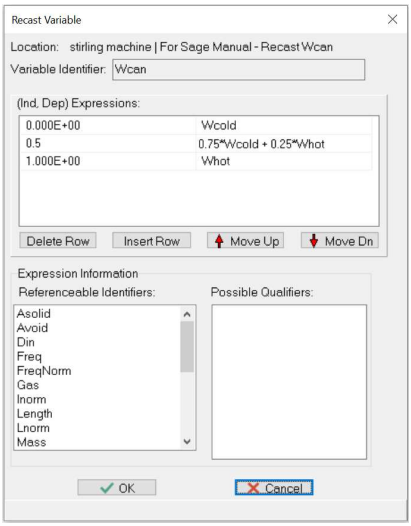
In addition to recasting single-valued real input variables, you can also recast the real-valued parts of more complicated variable types like Fourier series and cubic-splines data pairs. Provided they are what Sage recognizes as true independent variables rather than constants. Constants are not recastable because they used for normalizing variables during solving and optimizing and must not change during either process. Constants are usually single-valued real inputs (e.g. Tnorm) although some cubic-spline variables (e.g. Tinit of heat exchangers) are also implemented as constants.

The dialog below illustrates recasting the wall thickness distribution Wcan of a tubular-cone canister to a cubic-spline with discrete values expressed in terms of user-defined inputs Wcold and Whot.



You enter expressions for multi-valued input variables like WCan in a dialog

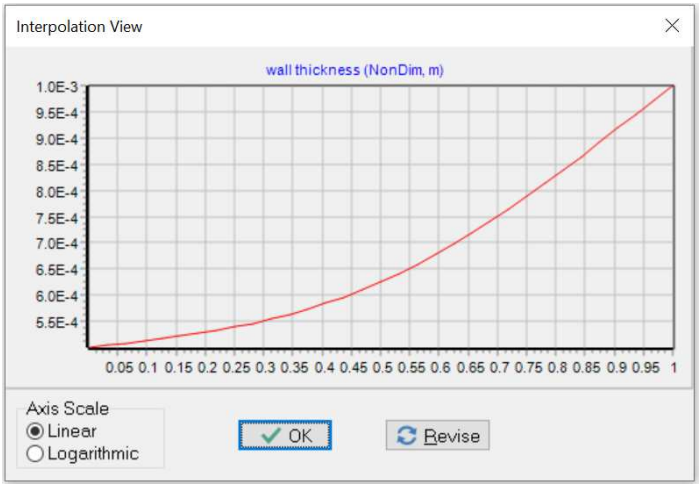
like this:



Expressions for the independent and dependent parts of discrete interpolation points are entered in the cells of the *string grid* control of the dialog. Just mouse-click on the desired cell to make changes, taking care, in this case, that the independent values are entered in increasing order from 0 to 1.

The Delete Row, Insert Row, Move Up and Move Dn buttons act on both independent and dependent cells simultaneously for the selected row of the string grid. The selected row is the one you have most recently edited or clicked on with the mouse.

Clicking the ViewInterpolation button produces a dialog like this:



4.9.2 Note on Dimensional Units

A recast variable retains its original dimensional units and Sage assumes the defining expression is also in those same units. So, for example, if you change the displayed units from meters (m) to inches (in) in the Options|Model Class dialog then the defining expression for any recast variable with the dimension of "length" must scale by a factor of 39.37. If the defining expression is a numerical constant, then you must change that constant manually. If, as in the above example, the defining expression references another variable, then the scaling is automatic, provided the referenced variable has the same dimensional units as the recast variable. But if the referenced variable has different dimensional units then the defining expression will scale incorrectly. The best practice then for implementing defining expressions is to reference a variable with the correct dimensional units in the expression. But doing so is up to you. There is no mechanism in place to enforce dimensional consistency in the defining expression.

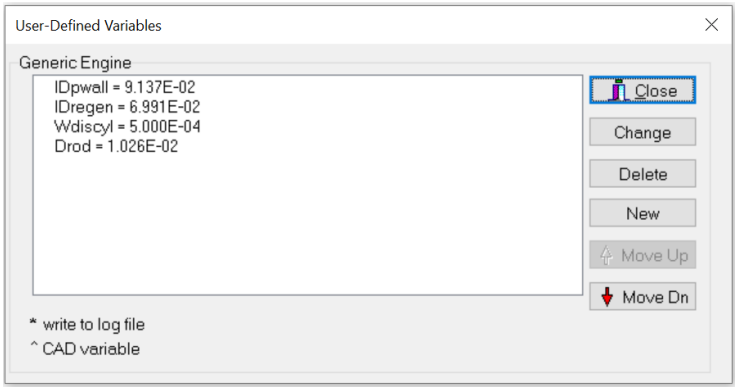
4.10 User Inputs

So that you may take full advantage of the recast-variables feature (section 4.9) Sage allows you to add custom user-defined input variables to model components. A user-defined input is intended for referencing in the defining expressions of a number of distinct recast input variables at lower levels of the model tree. That way a single user-defined input value can directly assign the values of what were previously several independent inputs. A user-defined input becomes part of the model component it is defined in and behaves just any other input for purposes of model solving, mapping or optimizing. Currently Sage allows only single-valued real user-defined inputs.

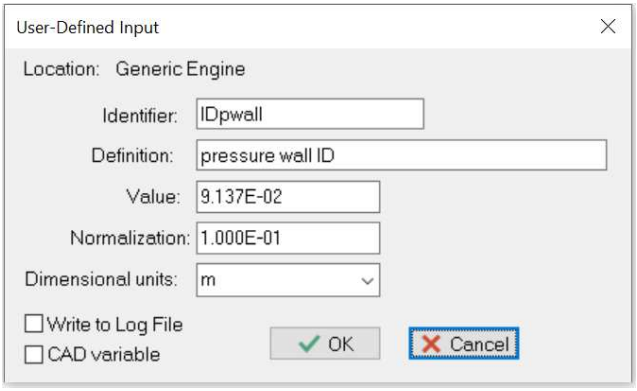
To create a user-defined input variable, activate the model component in which it is to reside and select the Specify|User Inputs menu command. A dialog opens, showing any user-defined inputs already present with buttons for creating new ones or modifying existing ones.

(**note** you can also specify user-defined inputs for the entire model in a struc-

tured way by selecting the Tools|Explore Custom Variables menu item)



The New button creates a new input. The Change button allows you to revise the defining properties of an existing input (the one selected) or the Delete button allows you to remove it from the model component. The Move Up or Move Dn buttons change the display order in the list box and also in the Sage listing or display window. Pressing the Change or New buttons opens the dialog for setting the defining fields:



The Identifier field is the name that will appear in the listing and display window and by which the input may be referenced in algebraic expressions within your model. The Definition field is a short description of what the input means. It also appears in the listing and display window but only for your convenience. The Value field is the numerical value that propagates through the model solution according to the other variables that reference the input. This is the value you normally see in the listing and display windows for an input variable. You can also assign it with the Specify|Input Values dialog just as for a built-in input variable. The normalization field supplies the *scale* of the input. It is used for setting step size during numerical differencing operations

in the solution or optimization processes. The Dimensional Units are selected from a list and tie the value and normalization fields to the current dimensions selected in the Options|Model Class dialog. For example, if you change the displayed units from meters (m) to inches (in), the Value and Normalization fields automatically scale by a factor of 39.37. When you reference a user-defined input in the defining expression of a recast input variable (or any other algebraic expression) the referenced value also automatically scales according to the currently selected dimensional units.

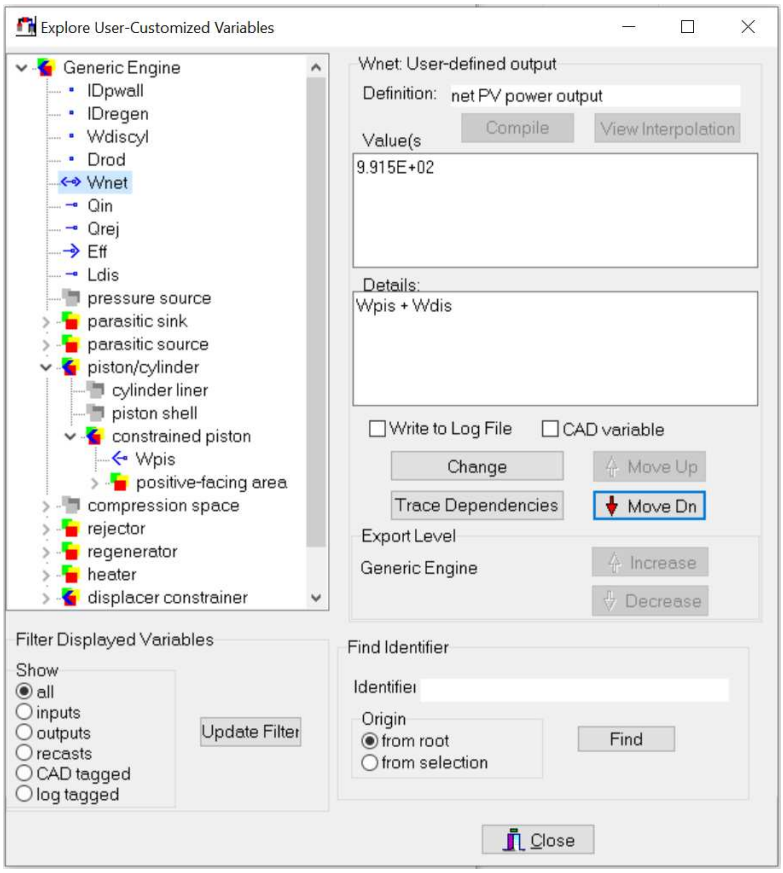
4.10.1 Identifier Visibility

As with a built-in input, the identifier of a user-defined input is visible from any lower-level child component in the model tree and may be referenced there in any algebraic expression. There is no provision for increasing the visibility to a higher level (parent model component) as there is for user-defined dependent variables. If you need to reference a user-defined input at a higher level then just define it at that level or higher in the first place.

4.10.2 Exploring Custom Variables

The Tools|Explore Custom Variables menu command opens up an interactive visual form where you can work with all the user-customized variables (user variables, recast variables and user inputs) in your entire model simultaneously. The key to doing this is a tree-structured listing of model components and

user-defined variables located at the left of the form, as illustrated here:



The user-customized variables, or *custom variables* for short, are listed as nodes under the model components in which they are defined. The displayed information, buttons and sub-menu items of the interactive form are keyed to the selected custom variable or model component in the tree view. You select a custom variable by clicking on its identifier in the tree view or by using the Find button to locate a variable anywhere in the model by matching its identifier against the one you type in the identifier box.

The Trace Dependencies button allows you to trace all the custom variables that depend on or are referenced by the selected custom variable. In the example above, the user-variable Wnet is selected. It references two other user-defined variables Wpis and Wdis, which are indicated by left-facing arrows in the display. It is itself referenced by another user-defined variable Eff, which is indicated by a right-facing arrow.

Several editing options are available in the panel to the right in the form. Many of these operations are also available under the Specify|User Inputs, User

Variables, or Recast Variables menu items but they are repeated here for convenience. You can change a variables identifier, definition or expression. You can change the order in which it is listed and for user variables change its export level. The export level is the level above the parent model component in which the variable identifier may be referenced by an expression in another custom variable, constraint or whatever.

After you change a custom variable all custom variables that may reference it are disabled for evaluation purposes and the value box displays "not compiled". The Compile button allows you re-parse all the custom variables, after which the value box once again displays numerical values.

There is also a submenu available that supports new, delete, cut, copy and paste operations. The submenu pops up by positioning the mouse cursor within the tree view then clicking the right button. When a model component is selected the New submenu item, allows you to insert a new user input, variable or recast variable. When a user input or variable is selected the Cut or Copy operations are available. Recast variables are not copyable because they are tied to specific built-in variables of the model. Copy and paste operations use the Windows clipboard by encoding a user input or variable defining data as a tab-delimited text string. It is possible to copy user inputs or variables from one model to another model running under a second Sage application. In order to paste a user input or variable that has been cut or copied it is necessary to first select the model component in the tree view into which the variable will be pasted.

Chapter 5

Mapping

There are times when you may want to investigate a sequence of solutions over a discrete range of one or more input variables. This is the process of mapping. Mapping is something you might want to do to achieve a better understanding of your model's sensitivity to various inputs. But it is not the same as optimization, the topic of the next chapter.

Anyone with any programming experience will understand the sequence of solutions generated in a mapping as the result of a nested loop structure. Except no programming is required (on your part) to do a mapping in Sage. All you do is select one or more input variables to be mapped using the Specify|Mapped Variables command then run the mapping with the Process|Map command. The rest is automatic.

Prior to specifying a mapped variable you must first activate the model component containing the variable by clicking its tab in the display or edit form, or clicking on its edit-form icon. Mappable variables are selected from a list of any drivable variable within the model component as described in chapter 10 — usually a real-valued independent variable, rarely the real part of a composite variable. For each mapped variable selected, you also specify the first and last values the variable will take, the number of iterations to be made within the mapping interval and whether the variable should be mapped in equal steps or equal ratios. The total number of solutions performed in the whole mapping process is the product of the iteration counts for the individual mapped variables. Evidently, for complicated models that take a long time to solve, you will want to think carefully about how many variables to map and their iteration counts.

When mapping initiates you are prompted for the name of a disk file in which the mapping results will be stored in ASCII format. This file will contain a sequence of lines or records containing tab-delimited values for the mapped variables as well as any user-defined variables tagged for appearance in log files. You select such user-defined variables by checking the "write to log file" box in the user-variable definition dialog available under the Specify|User Variables or Tools|Explore Custom Variables menu items.

During the mapping process, solutions are generated automatically until the mapping is finished. A mapping status dialog shows you the current mapping iteration and solution progress within that iteration. Using the buttons of the status dialog you can abort the mapping at any time, after which the mapping log file is closed with the results so far. You can also view the mapping progress in the display form, as usual, while the mapping is in progress.

Chapter 6

Optimizing

The standard mathematical definition of a general nonlinear programming problem looks like this:

> minimize a real-valued objective function $F(\boldsymbol{x})$, where $\boldsymbol{x}$ is a vector of n real variables, subject to the equality or inequality constraints
>
> $$c_i(\boldsymbol{x}) = 0,\ i \in \mathcal{E} \quad (6.1)$$
> $$c_i(\boldsymbol{x}) \geq 0,\ i \in \mathcal{N} \quad (6.2)$$
>
> where $\mathcal{E}$ and $\mathcal{N}$ are disjoint index sets.

which may leave you cold. In Sage you deal with optimization as a natural extension of the engineering model you already understand, in terms of things you quite naturally want to do. The objective function and any constraints fall out naturally. And after you have specified a few optimization problems you might want to note that they can, after all, be cast into the above rigorous mathematical form.

6.1 Specifying Optimization Variables

Although optimization variables pertain to the model as a whole, you specify them one group at a time within each model component. To flag a variable as an optimization variable, first activate the model component containing it (click its tab in the display or edit form, or click on its edit-form icon), then select Specify|Optimized Variables from the menu. As with mappable variables, optimizable variables are selected from a list of any drivable variable within the model component as described in chapter 10 — usually a real-valued independent variable, rarely the real part of a composite variable.

6.2 Specifying Constraints

Constraints, too, pertain to the model as a whole but are specified individually within a particular model component. To specify a constraint, first activate the model component that will contain it (click its tab in the display or edit form, or click on its edit-form icon), then select Specify|Constraints from the menu. Constraints are specified in the form $E_1 \leq E_2$, $E_1 = E_2$, $E_1 \geq E_2$, where $E1$ and $E2$ are two algebraic expressions you enter, similar to those entered for user-defined variables. More information about entering algebraic expression can be found in chapter 9.

Satisfying your constraints is the job of Sage's optimization driver. It does this by tweaking the optimization variables to force the left-hand-side expression minus the right-hand-side expression ($E_1 - E_2$) to be zero, in the case of an equality constraint, or merely non-negative or non-positive in the case of an inequality constraint. Equality constraints are always said to be *active* — meaning *in force* or *to be reckoned with*. Inequality constraints are active only when they start out violated or the optimization driver tends to violate the constraint during the course of the optimization. A useful rule of thumb is that each active constraint reduces by one the degrees of freedom available in the search domain to minimize or maximize the objective function.

Constraints are very useful things. You can invoke simple inequality constraints to keep an optimized variable in bounds — such as "X $\leq$ C" or "X $\geq$ C". Or, you can invoke more complicated constraints — such as PowerOut = 100, where PowerOut is a user-defined variable you have defined in terms of built-in outputs. You can even invoke constraints without an objective function present. However, when you do this it is necessary to specify the same number of optimization variables as active constraints. Go ahead and be liberal imposing constraints. Create as many as you feel you need. If physically sensible and reasonably linear, they are no great burden on Sage's optimizer.

6.3 Specifying the Objective Function

Like all the other elements of an optimization problem, the objective function pertains to the model as a whole but is specified within a particular model component. There can be only one objective function, which means that if you create a new one, the old one disappears. To specify the objective function, first activate the model component in which it is to appear (click its tab in the display or edit form, or click on its edit-form icon), then select Specify|Objective Function from the menu. You will be presented with dialog containing information about the existing objective function, if any, and the means to change it. The objective function is specified in the form "minimize E" or "maximize E", where E is an algebraic expression similar to those entered for user-defined variables or constraints. More information about entering algebraic expression can be found in chapter 9.

In the objective-function dialog box, the existing objective function is dis-

played even if it is in another model component. Where it presently resides appears in the dialog box labeled *Currently In Model Component*. Where a newly created or changed objective function will reside appears in the dialog box labeled *New Model Component*. The component in which the objective function belongs is the one where it is most natural to reference the variables in its defining expression, usually, but not always, the root model component.

As with satisfying constraints, Sage maximizes or minimizes your objective function by tweaking optimization variables according to its built-in logic. The objective function often represents some model output such as power, heat input, efficiency — typically, something that arises as the end result of a considerable amount of computation. As such, objective functions can be highly non quadratic (the ideal) and require many iterations for Sage to minimize or maximize. On the other hand, they may be relatively simple, as in the sample problems concocted at the beginning of this chapter.

6.4 Running An Optimization

Once you have specified the optimization problem, you initiate the optimization process with the Process|Optimize menu command. When optimization begins you are prompted for the name of a disk file in which the optimization results will be stored in ASCII format. This file will contain a sequence of lines or records containing tab-delimited values for the optimized variables, constraints and objective function, as well as any user-defined variables tagged for appearance in log files. You select such user-defined variables by checking the "write to log file" box in the user-variable definition dialog available under the Specify|User Variables command.

A status dialog box then appears, giving you a blow-by-blow account of the process and an opportunity to stop or pause if the need should arise. You can use the pause option to inspect the current solution state or save the model between iterations. The status dialog itself contains a good deal of information designed to keep you somewhat informed and entertained while you are sipping your coffee, or doing whatever you do to kill time. The goal of the optimizer is to drive the so-called pseudo-Lagrangian step change to a small but negative value.

The pseudo-Lagrangian step-change is a measure of the relative change over the current step of the objective function plus a weighted sum of violated constraints. The value should start out large and negative and grow ever smaller (but still always negative) as the optimization converge to the minimizer. For our present purposes, it is sufficient to think of the pseudo-Lagrangian as an approximation to the classical Lagrangian function of constrained optimization theory, which has an extreme point at the minimizer or maximizer.

One thing to keep in mind is that values printed in the status dialog pertain to the optimization problem at a somewhat higher level of abstraction than your original specification. The objective function and constraint violations displayed are normalized values. And if you have chosen to *maximize* rather

than *minimize* your objective function, its sign is switched. This is because the optimizer always *minimizes* objective functions. Minimizing $-f$ is equivalent to maximizing f.

After some up-front work to estimate evaluation precision in the objective function and constraints and to initialize the Hessian (second-derivative matrix), an optimization is carried out as a sequence of iterations. Each iteration requires a small step of each optimization variable in order to perform numerical partial derivatives of the objective function and its constraints, followed by a line search (all variables stepped simultaneously) along a direction defined by the solution to a quadratic-programming subproblem the optimizer maintains as it goes along. You can follow this in the status dialog box labeled *stepping*. Each step requires another model solution.

If all goes well, Sage will converge to a unique solution within a reasonable period of time. Satisfying any violated constraints usually requires only a few iterations, say less than ten. After that, Sage seems to concentrate more on minimizing or maximizing the objective function, which may take a good deal longer, say up to forty iterations — maybe more in some cases. After convergence you should inspect the state of your model then save it using the File|Save command if all is well.

Sometimes, due to noise in the model, the optimizer will never reach its target pseudo-Lagrangian step change. It will achieve a small value in the range of, say, (-10^{-5}) to $(-10^{-8}$ and never get lower. This is generally good enough and you should feel free to stop the optimizer whenever you think it is done. After all, you are a good deal more intelligent than Sage's optimizer and should be able to sense when your model is as optimized as it is ever likely to get. The optimizers main advantage over you is superior diligence.

6.5 Diagnostics

The most obvious symptom of a failing optimization is when the magnitude of the pseudo-Lagrangian step change fails to decrease, or even increases during the optimization process. There are other signs to look for that may help you pinpoint the problem.

6.5.1 Variable Step Limits

Keep an eye on the status-dialog box labeled *step limit for*. Ideally it should displays *none*, which is a sign that all is going smoothly and the optimizer's quadratic approximation to your optimization problem is reasonably valid. If it displays the name of one of your optimization variables instead, then the optimizer is having trouble with that variable. It has tried to step it to a forbidden value. If the problem does not clear up after a few iterations, it may be a good idea to stop and restart the optimization after a careful review of the current state of the optimization variables.

6.5.2 Line-Search Step Reductions

Another think to look for is two successive model solutions, instead of one, during the line search. You may have to be quick to notice this, depending on the time the solver spends solving your model. The first step of the line search is with the increment returned by the quadratic programming subproblem. If that step fails to produce the expected decrease in the pseudo-Lagrangian function, then a second, reduced, step is taken. When a failed search step happens near the beginning of an optimization, it usually means that either the Hessian (second-derivative) matrix, which is internally maintained and updated by Sage's optimizer, has not yet had time to fully evolve, or the pseudo-Lagrangian function is nowhere near quadratic. Usually the problem is self-correcting after a few iterations as the quadratic approximation begins to better fit the nonlinear pseudo-Lagrangian. If the problem does not go away it may indicate an ill-conditioned optimization as described below. Sometimes it helps to restart the optimization.

6.5.3 Ill-Conditioned Problems

When Sage absolutely fails to converge, it is probably suffering from an ill-conditioned optimization problem. You will want to be on guard for some of the following cases.

The most obvious example of ill-conditioning is a problem without a minimum or maximum. An example might be maximizing heat lift in a stirling cooler but forgetting to constrain input power. The size and power of the machine may go up forever. It is usually not difficult to avoid this problem, but sometimes the interaction of objective function and constraints is subtle. The symptom of a problem without a minimum or maximum is optimization variables drifting off to infinity or zero while the objective function keeps decreasing steadily.

A similar problem, but not as easy to spot, is the problem of weakly-determined optimization variables — subsets of variables with more degrees of freedom than required to solve the problem. Say, for example, you are optimizing a rectangular-channel heat exchanger with three pertinent physical variables: channel number, width and gap. And assume your objective function and constraints depend only on the two derived variables: hydraulic diameter and wetted perimeter. Then there would be no way you could optimize all three physical variables simultaneously. Nor could Sage. The best you could hope to do would be to optimize two physical variables, fixing the third. If Sage were to attempt to solve for all three, it would probably do its best, but wind up drifting around inconclusively. Whenever you select an optimization variable, take time to think of how the objective function and constraints depend upon it. There is no point adding a variable if it has no bearing on either. And if a variable is present solely to satisfy a constraint, be sure it is the only such variable.

The most ill-conditioned problem of all is one with infeasible or overdeter-

mined constraints — constraints that cannot possibly be satisfied. A simple example might be a problem with the two inequality constraints "pressure $\geq$ 10.0E5" and "pressure $\leq$ 5.0E5". No one would be that foolish, but when constraints get complicated, incompatibilities become less glaring. Sage has *big* trouble when faced with such constraints. The main symptoms are very large pseudo-Lagrangian step changes but no real improvement in constraint violations or the objective function. The objective function may *increase* dramatically (wrong way) as it takes a back seat to Sage's attempt to satisfy your constraints. If you have any experience running Sage with a well-behaved problem, you will immediately recognize the symptoms of infeasible constraints.

If the optimizer feels the problem is bad enough it will terminate the optimization process with a message box containing some information about the nature of the problem. Always check for the possibility that you have more equality constraints (plus active inequality constraints) than optimization variables. Otherwise, because of small numerical differences in the linear constraint approximations, it is rare for constraints to be truly infeasible. More likely, you will see the weird behavior described earlier than any error messages.

Occasionally Sage may suggest that your inequality constraints are infeasible, even though you have specified no inequality constraints. This is not an error. It merely reflects the status of two hidden inequality constraints added to your problem by Sage's optimizer. When this happens, you should infer that your equality constraints are really the infeasible ones.

If all else fails, after 100, or so, iterations — more than enough for all foreseeable well-conditioned optimizations — Sage's optimizer is programmed to give up, displaying its reason in a message box. The exact number of iterations before this happens may be model-class specific.

6.5.4 Multiple Extreme Points

Unconstrained optimizations generally lead to a unique solution — independent of starting conditions. There are two reasons for this: First, in most engineering models, optimization variables tend to cluster themselves into weakly coupled subsets (within model components or component subtrees). And second, within each of these subsets model outputs tend to vary smoothly, or at least without multiple peaks and valleys.

Add constraints and all bets are off. Constraints, especially nonlinear inequality constraints, tend to carve up the search domain into weird shaped, possibly disjoint, regions, with multiple local extreme points potentially found on the boundaries. Moreover, these regions are in n-dimensional space (where n is usually greater than 3) and are extremely difficult to visualize.

The problem is that Sage will happily converge to any local minimum with complete disregard for a *better* local minima lurking in some other corner of the search domain. This is a problem with any gradient-based search algorithm. The only way to find multiple solutions is to restart Sage from several different points in the search domain. But the problem is, how do you pick these starting points?

The most satisfactory approach to finding a global optimum is by way of successive approximation, building intuition about your problem as you go. Start out by solving a problem with as few constraints as possible, then add constraints one by one and see what happens. Save the data files from intermediate results in case you need to return to them as future starting points. As you add constraints you will be able to develop some intuition about where that constraint drives the solution, why it does so, whether or not there might be other local extrema, and where to look for them. Eventually you will accumulate a number of good starting points from which to start Sage for the final case where all constraints are simultaneously present.

6.5.5 Normalization Values

When specifying an optimized variable, constraint or the objective function, you can change its normalization value using an edit box in the specification dialog. Actually, what you can change is the scale factor which multiplies the default normalization value. But first, what is a normalization value and why might you need to change it?

A normalization value establishes a representative size for a *dimensioned* value, providing the necessary information to convert it to a uniform sized *dimensionless* value. A dimensioned value may be a very small number or a very large number, depending on physical units. For example, the value of a clearance seal gap might be of the order of 10^{-5} m while the value of charge pressure might be of the order 10^{6} Pa. A dimensionless value is always of uniform size, on the order of one.

The way Sage converts a dimensioned value to a dimensionless one is by dividing by the normalization value. Sage automatically maintains a default normalization value for each optimized variable, constraint or objective function. The basis for these normalization values are special input constants, usually defined in the top-level model component and usually with the suffix "norm" in the variable name, as in Lnorm. So, for example, a gap of 10^{-5} m becomes, in dimensionless form, gap / Lnorm. Normalization constants for the more-complicated expressions found in constraints and objective functions are built up as similar expressions involving basic normalization constants.

The good news is that you need not be concerned about normalization values in most cases. As long as the normalized values of all optimized variables, constraints and the objective function are within an order of magnitude or two of each other, the optimizer can successfully complete the optimization. On rare occasions, however, a normalization value can be too small or too large. If too small, then the optimizer is fooled into stepping farther than necessary when making finite-difference approximations to partial derivatives. Sometimes stepping so far that the model solution fails to converge. If the normalization value is too large, then the optimizer may not step far enough to get an accurate finite-difference result. Or it may be lulled into thinking that the optimization has converged because the objective function is changing by what it things is a very small amount.

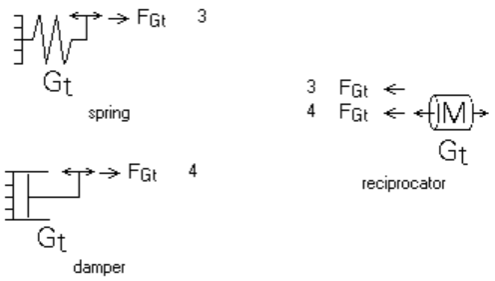
To allow you to diagnose this sort of problem, the value displays of optimized variables, constraints and the objective function (visible in display windows during optimization or after Process|Compile Optimization) include the normalization value in parenthesis, as in (norm = 1.654E-2), or some such. So, for example, if your objective is to maximize Wpv (where Wpv might be a user-defined variable for stirling-engine PV power output), you might expect trouble if Wpv is of the order 10 W, and norm is of the order 10^4. To remedy the problem, you would set the normalization scale factor for the objective function to 10^{-3}, using the edit box in the specification dialog. This would correct the value of norm to 10 (visible after another Process|Compile Optimization), resulting in a better conditioned optimization problem.

Chapter 7

Editing Model Structure

In Sage, you edit (create or modify) a model's structure within a graphical interface known as the edit form, within the main Sage form. The edit form contains a number of pages at different levels of the model hierarchy. These pages are created as explained below and selected by mouse-clicking on the tabs at the bottom of the edit form. The leftmost page always shows the child components of the root-level model component.

The edit-form root page for a simple resonant-system model might look like this:



Visible in the window are graphical icons representing child model components of the root model. Attached to the child-model icons are arrows corresponding to their boundary connections. An arrow with a number beside it indicates an active connection to another model icon within the same window. Active connections are identified by matching numbers. Arrows without numbers indicate potential connections, as yet unmade. The reason connections are indicated with numbers rather than attachment lines is to avoid cluttering up the window in complicated models.

Boundary connections can be made freely, but only among similar type connectors. Similar type connectors are indicated by the symbolic code next to the arrows. In the above example, the arrows labeled FGt indicate time-ring forces

(solved on a periodic time grid) capable of represents sinusoidally time-varying quantities as well as higher harmonics.

7.1 Basic Operations

The edit form has a basic mouse interface for certain common operations.

7.1.1 Selecting Components

Each model-component icon in the edit form has a caption bearing the name of the model component it belongs to. Clicking on a component icon highlights its caption bar, which activates that model component for pertinent menu operations. You can select multiple model components by holding down the shift key while mouse clicking on them or dragging a selection rectangle over them. In that case the active model component is the one most recently selected.

7.1.2 Positioning Icons

Holding down the left mouse button while the mouse cursor is on a component icon allows you to drag the icon anywhere within the edit form. Releasing the mouse button drops the icon at its new position. Continuing to hold down the shift key after selecting multiple model components allows you to move them as a group. You can also use the arrow keys to move selected model components.

7.1.3 Navigating the Model Tree

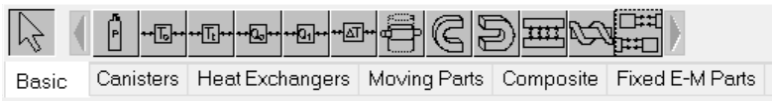
Model components can have child components, and so on for several generations. To display a component's children just double-click on its icon. This adds a new page to the edit form and a new tab at the bottom. You can re-display the parent component by clicking on its tab.

The tabs hold only a single branch pathway from the root to some lower-level component. When you change the tail of a path, by double clicking on a different component icon, the tabs corresponding to the old tail disappear.

7.2 Component Palette

When the edit form is active, a model-component palette appears near the top of the main Sage form. This palette contains buttons representing the seeds for new model components which can germinate in the current page of the edit form. The buttons of the palette are themselves arranged in tab-selected pages corresponding to different categories of model components. The available components in the palette are those which make sense in the context of the parent component, generally a small subset of the total number of components

in the model class. The component palette for the Basic tab of the root model looks like this



To create a new model component click a button in the component palette, then click again in the edit form at the position you want the component to reside. Once you've created a model component you can drag it around the edit form with the mouse, as describe above.

Considerably more difficult than the problem of creating model components is deciding which ones to create in the first place. For help with this you might want to look at the sample data files distributed with your particular model class. You can also find detailed information about each model component in other sections of this user's guide.

7.3 Connections

Making or breaking connections between model components is simply a matter of drag-and-dropping a connector to its mate.

To connect together two compatible boundary connectors, click on the first connector arrow with the left mouse button, then with the button held down position the mouse cursor on the second connector arrow and release the button. The first-clicked connector arrow establishes the type of the connection, and locks it in. The connector arrow you drag to must then be a mating arrow (opposite sense) of the same type (label).

Breaking connections is just like making them. Click on the first connector arrow with the left mouse button, then with the button held down position the mouse cursor on the mating connector arrow and release the button.

Clicking on a connection arrow for an active connection highlights the mating connector arrow at the other end of the connection. Hovering the mouse over a connection arrow displays a popup window that traces the source of the connection within the model component and also shows the current value of the quantity *flowing* through the connection if the connection is active.

7.4 Changing Connector Level

Often, the desired mate to a boundary connector resides at another location in the model tree, not displayed on the same page of the edit form. So the two boundary connectors are not immediately connectable. In this case it is always possible to trace up the model tree (toward the root) from each boundary-connector's model component to find two ancestor model components that appear within a common page. This is useful because, in Sage, you are allowed to

move boundary connector arrows up the model hierarchy. Connector arrows so moved appear attached to the ancestor model-component icon. Once they are there they may be attached to any compatible boundary connector at the new level. By doing this for our original two star-crossed connector arrows, they can be eventually joined.

To move a boundary-connector arrow *up* the model tree (toward the root) first highlight the arrow by clicking on it then click on the Inc connector level speed button in the main Sage form (red up-directed arrow) or use the Edit|Up Connector menu command. If you hold-down the shift key, you can highlight several connector arrows at the same time, after which you can move them as a group. But you can only highlight an arrow that is not already connected.

To move a boundary-connector *down* the model tree, first highlight one (or several) disconnected arrows then click on the Dec connector level speed button in the main Sage form (red down-directed arrow) or use the Edit|Down Connector menu command.

The notions of *up* and *down* may or may not make sense to you, depending on your mental picture of a model tree. For purpose of moving connectors you might want to visualize the model tree with the root at the top and branches extending downward.

7.5 Cut and Pasting Model Components

The standard Window's conventions for cutting and pasting blocks of text in a text editor apply to model components in the edit form. Cut and copy operate on the currently selected model components and insert those component into a clipboard-like temporary storage area. The clipboard is the source for one or more subsequent paste operations.

This clipboard is not the same as the standard Window's clipboard for textual information. So you will not be able to paste into another Windows application a model component cut from Sage.

7.5.1 Copy

First highlight a model component by clicking on its icon or select multiple model components by holding down the shift key while clicking on them or dragging a selection rectangle over them. Then click on the Copy speed button in the main Sage form (overlapped pages) or use the Edit|Copy Model(s) menu command. The model components, including all generations of child components, are now copied into the clipboard for subsequent pasting. Copying does not affect the existing model components so it is useful for cloning or duplicating those model component as sibling components.

This only works for model components that are not connected to other model components outside the selection group. If you want to copy model components connected externally first break any boundary connections to components out-

side the group (possibly at a higher level in the model tree) using the drag-and-drop techniques described above.

7.5.2 Cut

First select model components as for copying. Then click on the Cut speed button in the main Sage form (scissors) or use the Edit|Cut Model(s) menu command. This is just like the copy operation except the model component is deleted from the edit form.

7.5.3 Delete

First select model components as for copying. Then press the delete key or use the Edit|Delete Model(s) menu command. There is no undo option after deleting model components.

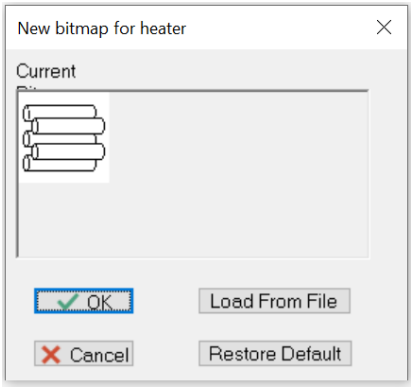
7.5.4 Paste

Following a copy or cut operation, click on the Paste speed button in the main Sage form or use the Edit|Paste Model(s) menu command to clone new model components into the edit form. Model components can only be pasted into the same parent model component they were copied from or to compatible parent having seeds in its child-creation palette of the same class types as the components to be pasted.

7.6 Changing Model Component Bitmaps

The Edit|Change Bitmap menu command allows you to change the bitmap images used to represent model components in the edit window. The command is keyed to the model component currently focused in the edit window. When you select

this command a dialog pops up, like this:



Pressing the "Load From File" button allows you to load a new image from any bitmap file (*.bmp) that you have previously created using some graphical editing software. Pressing the "Restore Default" button restores the bitmap originally assigned to the model component by Sage when it was first created.

The default bitmaps for Sage model components are all 64x64 pixel black-and-white bitmaps that were originally created with an early Borland application called Resources Workshop. Borland has stopped actively promoting Resources Workshop, but any other graphics editing software should do as well, provided it supports creating Windows compatible bitmap files (*.bmp). Bitmaps do not have to adhere to the 64x64 black-and-white format. You can load any size bitmap and even in color. You may use a bitmap rendered by CAD software or captured from a computer screen image. But keep in mind that the bitmap image will become embedded in your model file and affect its storage size accordingly.

The change-bitmap feature gives you control over the illusion of what the model component is intended to do. Do not underestimate this illusion. Appropriate bitmap images go a long way toward making a Sage model comprehensible. You may not want to change the default bitmaps for any of Sage's individual model components but submodels are another matter. If you have a model containing submodels you might find it useful to change their default images to something representative of what they actually do.

Chapter 8

Variable Types

Sage models employ a number variable types that you need to be aware of. These include the common integer and real (floating-point) variables but go on quite a bit beyond that.

8.1 Integers

Integer variables display as numerals without decimal points like this:

```
NCell           number spatial cells            2
```

Sometimes integer variable inputs are restricted to a range of values.

You may reference integer variables directly by name in user-defined algebraic expressions. They are converted to real values when you do this.

8.2 Reals

Real variables display in exponential format like this:

```
Freq            frequency (Hz)                5.500E+01
```

Sometimes real variable inputs are restricted to non-negative, strictly positive or values within some prescribed closed interval. You may enter real values in dialog boxes with or without decimal points or the E suffix.

Real variables may be referenced directly by name in user-defined algebraic expressions.

8.3 Complex

Complex variables display in an extended exponential format like this:

```
F           boundary force (N)       (-9.466, 7.633)E+01
```

The numbers in parenthesis are the real and imaginary parts and the `E+01` is the common power-of-ten multiplier. The above example is equivalent to the complex number $(-94.66, 76.33)$ or $-94.66 + 76.33i$, where i is the imaginary unit ($i = \sqrt{-1}$).

In model components with an underlying basis in linear differential equations, complex amplitudes are often used to represent sinusoidal time-varying quantities. Presuming this to be the case in the above example, physical boundary force would actually be understood as the real-part of $Fe^{i\omega t}$, or

$$\text{physical boundary force} = -94.66\cos\omega t - 76.33\sin\omega t$$

If we look at this as just the first-harmonic terms in a Fourier-series expansion $\sum(a_n \cos n\omega t + b_n \sin n\omega t)$, we note that $a_1 = \Re F$ and $b_1 = -\Im F$, where $\Re F$ and $\Im F$ are the real and imaginary parts of complex amplitude F.

Complex variables are not directly referenceable in user-defined algebraic expressions. However for a complex variable X, the following subfields are referenceable:

X.real	real part
X.imag	imaginary part
X.amp	amplitude or modulus
X.arg	argument or phase angle
X.cos	cosine coefficient a_1 of Fourier series representation
X.sin	sine coefficient b_1 of Fourier series representation

According to the preceding paragraph X.cos = X.real and X.sin = -X.imag.

8.4 Phasors

Phasors are just complex numbers displayed in polar form. A typical phasor variable would be displayed like this:

```
F           boundary force (N)       1.216E+02 cis(2.463)
```

The leading coefficient (1.216E+02 in this case) is the complex magnitude or amplitude — the length of the phasor. The number in parenthesis (2.463) is the phase angle in radians measured counterclockwise from the positive real axis. The notation "cis" reads "cosine plus i sine", where i is the complex unit. In other words, the above variable is equal to the complex number $121.6\cos(2.463) + 121.6i\sin(2.463)$, which is the same as the number in the previous example.

Under the convention that boundary force actually represents a complex amplitude for a sinusoidal time-varying quantity, we would now have

$$\text{physical boundary force} = 121.6\cos(\omega t + 2.463)$$

If we look at this as just the first-harmonic term in a Fourier-series cosine expansion $\sum(c_n \cos(n\omega t + r_n)$, we have $c_1 = |F|$ and $r_1 = \angle F$, where $|F|$ and $\angle F$ are the magnitude and phase angle of phasor amplitude F.

Phasors are indirectly referenceable in user-defined algebraic expressions through the same subfield identifiers as for complex numbers.

8.5 Fourier Series

Fourier-series variables have a multi-line display like this:

```
FTmean        x-mean temperature (K)        2.990E+02...
  ( 1.597,  0.428,  0.301)E+01 Amp
  (-2.894, -0.431,  3.142)E+00 Arg
```

The number to the right on the first line is the time-mean value and numbers in the subsequent two lines are the common-exponent amplitudes and phases of the first three harmonics in the Fourier-series cosine expansion. The above example is an abbreviation for

$$\begin{aligned} T &= 299 \\ &+ 15.97\cos(\omega t - 2.849) \\ &+ 4.28\cos(2\omega t - 0.431) \\ &+ 3.01\cos(3\omega t + 3.142) \end{aligned}$$

where t is time and ω is angular frequency. The amplitude and phase of the first harmonic in a Fourier cosine series correspond to the amplitude and phase of a complex or phasor-type variable.

An alternate, though less common, format is in terms of coefficients in a cosine-sine expansion like this

```
FTmean        x-mean temperature (K)        2.990E+02...
  (-1.529,  0.389, -0.163)E+01 Cos
  ( 0.461,  0.179, -0.253)E+01 Sin
```

which would be an abbreviation for

$$\begin{aligned} T &= 299 \\ &- 15.29\cos\omega t + 4.61\sin\omega t \\ &+ 3.89\cos 2\omega t + 1.79\sin 2\omega t \\ &- 1.63\cos 3\omega t - 2.53\sin 3\omega t \end{aligned}$$

The two representations are completely equivalent, as can be seen by applying the cosine angle-addition formula to the cosine series

$$\sum_{n=1}^{\infty} c_n \cos(n\omega t + r_n) = \sum_{n=1}^{\infty}(c_n \cos r_n)\cos n\omega t - \sum_{n=1}^{\infty}(c_n \sin r_n)\sin n\omega t$$

$$= \sum_{n=1}^{\infty} a_n \cos n\omega t + \sum_{n=1}^{\infty} b_n \sin n\omega t$$

Evidently, one can convert from cosine-only to cosine-sine series using the relationships

$$\begin{aligned} a_n &= c_n \cos r_n \\ b_n &= -c_n \sin r_n \end{aligned}$$

or conversely

$$\begin{aligned} \tan r_n &= -b_n/a_n \\ c_n^2 &= a_n^2 + b_n^2 \end{aligned}$$

However, one must use caution if the need arises to shift the reference phase in an amplitude-phase expansion. Each harmonic shifts phase by a different amount. To see this note that if $f(\tau) = \cos(n\tau + r_n)$, where $\tau = \omega t$, then

$$f(\tau - \alpha) = \cos(n(\tau - \alpha) + r_n) = \cos(n\tau + r_n - n\alpha)$$

from which it is clear that the n-th harmonic for a function phase-shifted by α is phase-shifted by an amount $n\alpha$ compared to the original ωt series.

The number of harmonics displayed is automatic in the case of an output variable. It is generally half the number of time nodes specified in the underlying computational grid, the maximum resolvable amount according to the Nyquist sampling theorem.

For an input variable, the number of harmonics is up to the user, although higher harmonics are not resolvable, and in fact will cause aliasing errors, in any computational grid containing fewer than twice as many time nodes as the number of the harmonic. Amplitudes and phases for individual harmonics are changed via dialog-box commands.

The higher-frequency harmonics in a Fourier-series output variable should be small compared to the lower-frequency harmonics. When this is the case, it suggests that the number of time nodes in your model's time grid is sufficient to resolve the solution and all is well. If this is not the case, then you may be fooling yourself. At best, the higher-frequency harmonics are simply missing from the output. At worst, they are actually showing up as erroneous contributions to low-frequency harmonics, in which case you may be in for a rude awakening in the test cell. So keep an eye on those high-frequency coefficients and increase the time nodes in your solution grid if necessary.

Fourier-series variables are not directly referenceable in user-defined algebraic expressions. However for a Fourier-series variable X, the following subfields are referenceable:

X.mean	time-mean value
X.cos.n	nth-harmonic cosine coefficient
X.sin.n	nth-harmonic sine coefficient
X.amp.n	nth-harmonic amplitude
X.arg.n	nth-harmonic phase angle

In the last four cases, n is an integer, ranging from 1 up to the maximum available harmonic index.

Interpolation You may also reference the evaluated result of a Fourier-series variable by following the variable name with an argument enclosed in parenthesis. For example, if X is a Fourier-series variable then X(0.5) returns X evaluated at $\omega t = 0.5$. The argument can be any valid expression as defined in chapter 9, not just a simple constant. The dimensional units of the argument are presumed to be the same as the current dimensional units in effect for angles (Options|Model Class dialog).

8.5.1 Discrete Representations

There are some subtle differences between Fourier-series representations of continuous functions and representations of discrete functions defined only at the nodes of a time grid. You should be aware of these in case you plan to work out the Fourier coefficients of a special function defined in terms of a table of values. The general formulation for a function $f(\tau_j)$, defined at points $\{\tau_j = 2\pi j/N : j = 0 \ldots N-1\}$ is

$$f(\tau_j) = f_m + \sum_{n=1}^{M} (a_n \cos n\tau_j + b_n \sin n\tau_j)$$

In the above formula, N corresponds to the Sage input variable NTnode and M is either $N/2$, if N is even, or $(N-1)/2$, if N is odd. Variable τ substitutes for ωt. The Fourier-series coefficient are

mean value

$$f_m = \frac{1}{N} \sum_{j=0}^{N-1} f(\tau_j)$$

cosine coefficients $n = 1..M$

$$a_n = \frac{2}{N} \sum_{j=0}^{N-1} f(\tau_j) \cos n\tau_j$$

sine coefficients $n = 1..M$

$$b_n = \frac{2}{N} \sum_{j=0}^{N-1} f(\tau_j) \sin n\tau_j$$

Except for the case N even, where a_M is half the above value and $b_M = 0$ (technically, b_M is half the above value too, however it evaluates to zero because $\sin M\tau_j$ is always zero). For Sage input variables in terms of cosine-coefficients

only, the previous conversion equations can be used to convert the a_n and b_n coefficients to c_n and r_n.

The above formulation is the real equivalent of the complex discrete Fourier transform, which says for a complex-valued function $\boldsymbol{F}$

$$\boldsymbol{F}(\tau_j) = \sum_{n=0}^{N-1} \boldsymbol{c}_n e^{in\tau_j}$$

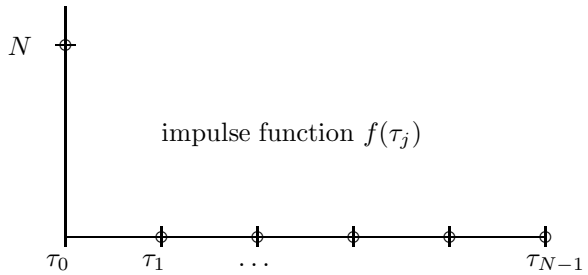
if and only if

$$\boldsymbol{c}_n = \sum_{j=0}^{N-1} \boldsymbol{F}(\tau_j) e^{-in\tau_j}$$

The real equivalent follows after substituting $(a_n, -b_n)/2$ for $\boldsymbol{c}_n$, assuming $\boldsymbol{F}$ is real and making some simplifications.

Case: Impulse Function

A simple example should make it clear how to use the above definitions to translate a tabular function to Sage inputs. In this case, the tabular function is the impulse function which takes the value N at $\tau_0 = 0$ and zero for all other τ_j. When plotted out, the impulse function looks like this:



Since this is an even function ($f(\tau_j) = f(\tau_{N-j})$), only the cosine coefficients are nonzero. And even then, most of the terms in the above summations drop out because $f(\tau_j) = 0$ for $j \geq 1$. So with a little thought, it is clear that the expansion for the impulse function for the case of N (NTnode) even is

$$f(\tau_j) = 1 + 2\cos\tau_j + 2\cos 2\tau_j + \ldots + 1\cos N/2\,\tau_j$$

The impulse function phase-shifted by α would be

$$f(\tau_j) = 1 + 2\cos(\tau_j + \alpha) + 2\cos(2\tau_j + 2\alpha) + \ldots + 1\cos(N/2\,\tau_j + N/2\,\alpha)$$

For the case of N odd, the coefficient of the last term would be 2, rather than 1 and its order would be $(N-1)/2$, rather than $N/2$.

8.5.2 Complex Solution Formulations

Sage was designed to model physical systems driven by periodic boundary conditions, often sinusoidal functions time. In many cases such systems are approximated by linear differential equations amenable to solutions in terms of complex exponential functions. Although Sage mostly solves systems governed by nonlinear differential equations on a time grid, there are cases where linearized complex solutions are useful. For example, in deriving phase-shifted heat transfer and flow-friction correlations in solid and gas domains. In such cases Sage employs a standard technique for converting between time-grid solutions and linearized complex solutions, illustrated below for the case of a solved temperature T taken to be the real part of the complex temperature variation

$$\boldsymbol{T}(t) = (T_r + iT_r)e^{i\omega t} = (T_r + iT_i)\left(\cos(\omega t) + i\sin(\omega t)\right) \tag{8.1}$$

T_r and T_i are the real and imaginary components of a complex temperature amplitude. Expanding the right-hand side of the above equation the real and imaginary parts of the complex temperature variation are

$$\Re\boldsymbol{T}(t) = T_r\cos(\omega t) - T_i\sin(\omega t) \tag{8.2}$$

and

$$\Im\boldsymbol{T}(t) = T_r\sin(\omega t) + T_i\cos(\omega t) \tag{8.3}$$

The real-part $\Re\boldsymbol{T}$ corresponds to the physical solved temperature. From the above definition of a Fourier series it is clear that the complex temperature amplitude is related to the coefficients of the Fourier-series expansion for temperature, according to

$$T_r = a_1 \tag{8.4}$$

and

$$T_i = -b_1 \tag{8.5}$$

where a_1 and b_1 are the cosine and sine coefficients of the first harmonic.

This way a nearly sinusoidal solution on a time-grid can be converted to an approximate complex amplitude using equations (8.4) and (8.5). A complex amplitude can be converted to a time grid value by evaluating equation (8.2) at the grid-node time value.

Prior to Sage version 12 (July 2021) the complex temperature variation in the above example would have been calculated differently, from the local approximation

$$\boldsymbol{T}(t) \approx (T - T_m) - i\frac{\partial T}{\partial \omega t} \tag{8.6}$$

where T is the instantaneous physical temperature and T_m is the time-mean temperature. This is valid for sinusoidal temperature variations, because then $T - Tm = \Re\boldsymbol{T}$ and by direct differentiation of equation (8.2) it follows from equation (8.3) that $\Im\boldsymbol{T} = -\frac{\partial T}{\partial \omega t}$. But for solutions with large higher harmonic

content the terms on the right-hand side would not vary sinusoidally so that any complex heat transfer or flow friction terms derived from the resulting complex temperature variation would also tend to have large higher harmonics, inconsistent with the linearized complex solutions upon which they were based. So this formulation was abandoned in version 12.

8.6 Data Pairs

Data-pair variables have a multi-line display like this:

```
ExtrmT        temperature extremes (NonDim, K) data pairs...
  ( 0.000E+00,  3.000E+02)
  ( 2.500E-01,  4.000E+02)
  ( 1.000E+00,  2.000E+02)
```

Within the parenthesis, the left number represents the independent variable and the right number the dependent variable. In this example we have the extreme temperature values occurring at dimensionless positions 0, 0.25 and 1.0, respectively. Generally speaking, data-pair variables are used as outputs.

Data-pair variables are not directly referenceable in user-defined algebraic expressions. However for a data-pair variable X, the following subfields are referenceable:

X.TData.n	nth independent value in X's defining data
X.FData.n	nth dependent value in X's defining data

where n ranges from 1 to the number of interpolation pairs defining X. For example ExtrmT.FData.2 in the above example would be 4.0E2.

The notation TData and FData to denote independent and dependent values requires some explanation. Sage's original purpose for data pairs was to store physical properties $F(T)$ as a function of temperature T. In that context the notation TData and FData makes perfect sense. Later on data pairs were used to represent other independent variables as well but the designation TData for the independent value was retained.

8.7 Cubic Splines

Cubic-spline variables are similar in outward appearance to data-pair variables. They have a multi-line display like this:

```
Tinit         initial temperature (NonDim, K) unit spline..
  ( 0.000E+00,  3.000E+02)
  ( 5.000E-01,  6.000E+02)
  ( 1.000E+00,  9.000E+02)
```

What you see are interpolation pairs through which a cubic-spline curve will be fit, with the left number representing the independent variable and the right number the dependent variable. In this example we have the specifications for a cubic spline passing through the temperatures 300K, 600K and 900K at dimensionless positions 0, 0.5 and 1.0, respectively. The number of interpolation pairs is arbitrary so long as there are at least two. Changing them is easily done via dialog box commands.

Generally speaking, cubic spline variables are used to specify things like initial temperature profiles or interpolation data for material properties. The independent variable may be dimensionless, as in this example, or itself a dimensioned quantity. This is generally clear from context. Further information for specific variables can be found in the documentation for the model component in which it resides.

Cubic-spline variables are not directly referenceable in user-defined algebraic expressions. However for a cubic-spline variable Y, the following subfields are referenceable:

Y.TData.n	nth independent value in Y's defining data
Y.FData.n	nth dependent value in Y's defining data

where n ranges from 1 to the number of interpolation pairs defining Y. For example Tinit.FData.2 in the above example would be 6.0E2;

Interpolation You may also reference the interpolated value of a cubic-spline variable by following the variable name with an argument enclosed in parenthesis. For example, if Y is a cubic-spline variable then Y(0.25) returns Y interpolated at $x = 0.25$, where x is the independent variable. The argument can be any valid expression as defined in chapter 9, not just a simple constant. The dimensional units of the argument are presumed to be the same as the current dimensional units in effect for the independent variable (Options|Model Class dialog).

TBelow Operator The TBelow operator evaluates the total duration of the independent variable T for which the dependent interpolation function F is below a given value. For example if Y(t) is a time-function cubic-spline variable then Y.TBelow(1.0) returns the total time duration for which Y is below 1.0. The TBelow operator works even if the independent variable is not time. For example, if Y(x) is a function of position then Y.TBelow has the units of length. In general, TBelow has the units currently in effect for the spline's independent variable (Options|Model Class dialog). The dimensional units of the argument are presumed to be the same as the current dimensional units in effect for the dependent variable.

8.8 Enumerated

Enumerated variables display like this:

```
Solid          wall material                SS304...
```

where the identifier on the right is one of several discrete alternatives selected in a dialog box. If you select the "name only" display option within the Sage options dialog (Options|Sage menu command) this is all you will see. If you select the "full detail" display option, enumerated variables are followed by additional display lines containing further information. For example, the above variable is followed by

```
Rhos = 7.800E+03 (kg/m3)
(T, Ks(T))... (K, W/(m K))
(3.000E+00, 1.800E-01)
(6.000E+00, 4.800E-01)
(1.000E+01, 9.000E-01)
(2.000E+01, 2.200E+00)
(4.000E+01, 4.600E+00)
(1.000E+02, 8.100E+00)
(3.000E+02, 1.470E+01)
(1.500E+03, 3.000E+01)
(T, Cs(T))... (K, J/(kg K))
(3.000E+00, 1.500E+00)
(6.000E+00, 3.000E+00)
(1.000E+01, 5.300E+00)
(2.000E+01, 1.340E+01)
(4.000E+01, 5.700E+01)
(1.000E+02, 2.210E+02)
(3.000E+02, 4.850E+02)
(1.000E+03, 6.100E+02)
(1.500E+03, 6.600E+02)
```

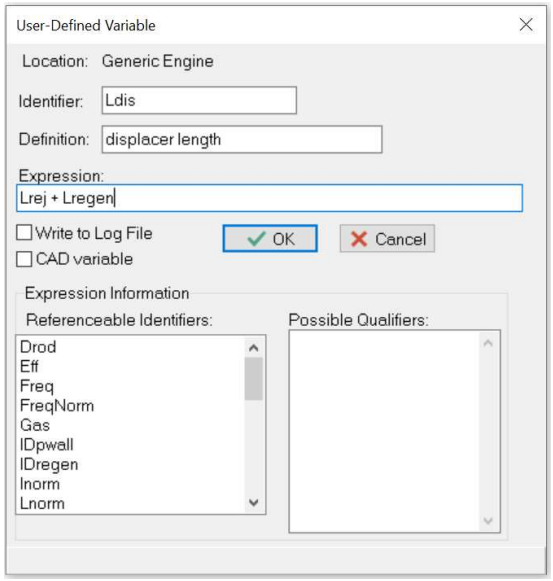
which define its density and cubic-spline interpolation pairs used for its conductivity and specific heat. Values are displayed in the current dimensional units set in the Model Class | Options dialog.

You can reference the individual properties of enumerated variables in user-defined algebraic expressions. You can also change those properties or add new materials to the enumerated list using one of the property-editing utilities supplied with the Sage software distribution. More information on these topics can be found in chapter 28.

Chapter 9

Entering Algebraic Expressions

Algebraic expressions are the common element of constraints, objective functions and user-defined variables. You enter them as ordinary strings of text within Windows edit controls. Then, at the appropriate time, Sage reads and interprets (parses) them and converts them to meaningful calculations. A typical dialog box in which you are asked to enter an expression is the one for user-defined variables which looks like this:



In all cases, the expression you enter is the basis for Sage to calculate some new value in terms of other variable values in the model. But there are rules of

syntax to follow.

Programmers will have an advantage here because the expressions understood by Sage follow the common rules of syntax for most programming languages — particularly Pascal. For example, if `A`, `B`, and `C` are valid variable identifier names, then the following are all valid expressions:

```
A + B - C
A + 2*B
((A+B)/C + 3.0E-2) / 5.0
```

Of course, most variable names are longer than a single character.

Sage does not get around to actually parsing your expression until you tell it to, by selecting the Process|Solve, Process|Parse Solution, Process|Optimize or Process|Parse Optimization menu commands. This allows you to reference identifiers not yet created and otherwise get away with anything in your expression. If, after you have entered your expression and closed its dialog box, you want Sage to check it for errors, just select Process|Parse Solution in the case of a user-defined variable or Process|Parse Optimization in the case of a constraint or objective function. These commands are equivalent to the *compile* step in a conventional computer language. If you are pretty sure your expressions are correct you can omit this step and proceed directly to Process|Solve or Process|Optimize. In this case, compiling, if successful, will be immediately followed by execution.

9.1 Ground Rules

Sage expressions are not case sensitive. That is, upper and lower case letters are equivalent within identifiers or constants. (Although there is only one valid letter for constants: `E` or its lower-case equivalent `e` used in an exponential format like `1.0E-2`).

Generally speaking, Sage ignores blank spaces, whether leading, trailing or embedded within your expression. You are welcome to use them, though, to separate variable identifiers from numerical constants or operators, thereby making your expressions more readable. However, blank spaces do connote separation. So you must not embed them within identifiers or constants, otherwise Sage will misunderstand your meaning.

9.2 Identifiers

The identifiers used in expressions are the variable names already in use in the model or those you create yourself for user-defined variables. All must begin with a letter, followed by one or more letters or digits. The following are all valid identifiers

```
x
Y1
Aardvark
PowerOutput
```

Some variable types have sub-fields that are accessed by appending a period directly after an identifier name, followed by the appropriate qualifier. For example, if `Aardvark` is a phasor-type variable, then

```
Aardvark.amp
Aardvark.arg
```

are valid identifiers. `Aardvark` alone, although valid as an identifier, will generate an error at compile time because it cannot be resolved into a floating-point type. See chapter 8 for the subfield qualifiers for the various types of variables.

9.3 Operators

The list of valid operators is

`+`	add
`-`	negate or subtract
`*`	multiply
`/`	divide

Parenthesis may be used in the usual way to control operator precedence, and they may be nested to any level. Lacking parentheses, operations occur in the usual pecking order. That is, multiplication and division come before addition and subtraction. For example, `(3 + 4 / 2)` evaluates to 5 not 7/2.

9.4 Constants

Numerical constants are also allowed in expressions. They are always understood as floating-point numbers, rather than integers. The following are all valid representations for the same constant:

```
234
2.34E+02
234.0
```

Constants may not begin with a period. In other words, .234 is not a valid constant. The reason for this rule is to avoid confusion with periods as subfield qualifiers in identifiers.

Constants in algebraic expressions require the period as the decimal separator character regardless of the Windows operating system regional setting for that character. In Sage algebraic expressions special characters like '.' and ',' have precise meanings depending on the context in which they occur. These meanings cannot change depending on regional settings lest algebraic expressions created under one set of regional settings be invalid when parsed under another.

9.5 Built-In Functions

Sage has a number of built-in functions you can use in expressions. These require either zero, one or two arguments following the function identifier, enclosed within parentheses. In the following table, the first argument is indicated by `x`, the second by `y` and either one may be another expression, perhaps involving a nested function. For example `Sqrt( 10 * Abs( ArcSin(Pi) ) )` is a legitimate construct.

function	meaning	restrictions
`Pi`	π (3.14159...)	
`Abs(x)`	absolute value of x	
`Sqrt(x)`	square root of x	$x \geq 0$
`Sqr(x)`	square of x	
`Exp(x)`	exponential function of x	
`Ln(x)`	natural logarithm of x	$x > 0$
`Sin(x)`	sine of x	
`ArcSin(x)`	inverse sine of x	x in $[-1, 1]$; result in $[-\pi/2, \pi/2]$
`Cos(x)`	cosine of x	
`ArcCos(x)`	inverse cosine of x	x in $[-1, 1]$; result in $[0, \pi]$
`Tan(x)`	tangent of x	
`ArcTan(x)`	inverse tangent of x	result in $[-\pi/2, \pi/2]$
`Sinh(x)`	hyperbolic sine of x	
`ArcSinh(x)`	inverse hyperbolic sine of x	
`Cosh(x)`	hyperbolic cosine of x	
`ArcCosh(x)`	inverse hyperbolic cosine of x	$x \geq 1$; result ≥ 0
`Tanh(x)`	hyperbolic tangent of x	
`ArcTanh(x)`	inverse hyperbolic tangent of x	result in $(-1, 1)$
`Max(x, y)`	larger of x and y	
`Min(x, y)`	smaller of x and y	
`Power(x, y)`	exponentiation (x^y)	$x > 0$
`Amp(x, y)`	magnitude of complex number (x, y)	
`Arg(x, y)`	phase of complex number (x, y)	result in $[-\pi, \pi]$

The arguments of the trigonometric functions are in radians. The `Power` function is mainly intended for raising positive arguments to fractional powers. For polynomial expressions using integer powers it is faster and less restrictive (can use a negative `x`) to use direct multiplication, perhaps in conjunction with the `Sqr` function — for example, `x*Sqr(x)` to represent x^3.

9.6 Model-Specific Functions

In addition to the built-in functions, any referenceable cubic-spline or Fourier-series variable in your model may also be used as a function by following its identifier name with an argument enclosed in parentheses. For example, if `Area` is a cubic-spline variable, then

```
Area(0.5)
```

is a valid expression that returns the interpolated value of `Area` at $x = 0.5$, where x is the independent variable. The argument can be any valid expression, not just a simple constant.

In general, the units of the argument are dimensional according to the units in effect for the independent-variable part of the underlying cubic-spline or Fourier-series variable. In the case of Fourier-series variables, the argument is always an angle with units of radians or degrees as specified in the Options|Model Class dialog. In the case of cubic-spline variables, the argument is often dimensionless but not always. If not dimensionless, its units are listed after the variable definition in the display window or output listing. Argument units may matter if you specify an argument as a simple constant then change the units setting that applies to the argument. For example, if `P` is a Fourier-series variable then `P(45)` will mean one thing if the units for the argument are degrees and quite another if radians.

9.7 Referenceable Variables and Qualifiers

You may not just include any old identifier name in the expression you enter. It must be the name of a referenceable identifier. That is, an identifier known to exist at the current level within the model structure and, if it is not a real or integer type variable, with potential to be converted to a floating-point value through a subfield qualifier, as noted above. For your convenience, identifiers for all currently referenceable variables are presented in a list box at the bottom left of any dialog requiring you to enter an expression. Clicking on an identifier in the list will display its possible subfield qualifiers, if any, in the box to the right (*see* the above illustration). You cannot drag and drop identifiers and qualifiers into your expression. You must type them in, with a period separating the two (if a qualifier is required). Be alert for qualifiers requiring sub-qualifiers, such as most Fourier-series qualifiers and cubic-spline qualifiers as explained in chapter 8. Sub-qualifiers are generally integer indices and you must remember to type them in yourself.

The rule about only using referenceable identifiers from the list box is not a hard and fast rule. You may reference other identifiers in your expression as well, if you want. But before you actually compile your expression you must create a user-defined variable having that name and whose scope includes the model level in which your expression resides.

9.8 Variable Scope

The default scope of a variable is the model component in which it resides and all generations of its child models. Looking at it another way, you may reference from a child model component any variable defined in a Parent model, or an ancestor any number of generations back. But you cannot reference the other way. At least not usually. The reason for this is to prevent variable name

conflicts from the point of view of a parent model component looking at two or more children with the same variable identifier.

There is a way around this restriction, though, in the case of user-defined variables. For these, the default variable scope applies at the moment of creation, but may be manually extended to include any number of generations back by clicking on the Increase button located at the bottom of the Specify|User Variables dialog box. The so-called *export level* may be reduced again by clicking on the Decrease button. This scope extending feature is very useful when you need to compute *big-picture* quantities in terms of variables localized in a number of distinct model components. You simply export the various local variables to some common level, say the root model, then reference them there in constraints, the objective function or other user-defined variables.

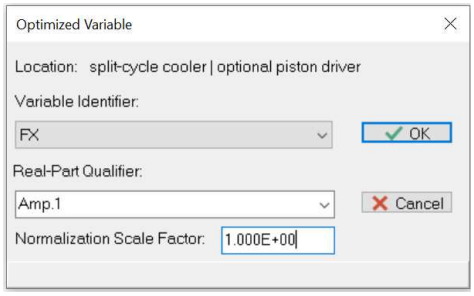
One final note: problems of self-reference will cause a runtime error as the numerical co-processor stack overflows. Self reference will occur if a variable named `A` references a variable named `B` in its defining expression while at the same time `B` references `A`. While such referencing may be allowed in computer languages that assign values in strict sequential order, this does not apply to Sage.

Chapter 10

Specifying Drivable Variables

Drivable variables are those whose values are automatically assigned by a Sage driver, such as the mapper or optimizer. Generally speaking, drivable variables must be real-valued independent inputs. However, they may also be real parts of certain complex (phasor), splines or Fourier-series variables, provided they are independent inputs. Dependent (output) variables are not drivable, for obvious reasons. Nor are constants (certain inputs) because of assumptions about constants made by the optimizer – namely, that they be constant.

Drivable variables are specified by their identifier name along with an optional subfield qualifier, similar to the way variables are referenced in algebraic expressions. But when specifying drivable variables there is no need to manually type in identifier + qualifier. You just select the two from combo-box lists within the selection dialog. For example, the selection dialog for an optimized variable (Specify|Optimized Variables) might look like this:



If there is no qualifier required for a selected identifier you will not even see the qualifier list.

The above selection process applies in most cases. However, for Fourier-series or cubic-spline variables, where the qualifier requires a subqualifier, as

explained in chapter 8, you must manually append the sub-qualifier to the qualifier. The sub-qualifier is generally an integer index which you type directly into the qualifier combo-box as an edit control. Be sure to separate qualifier from sub-qualifier with a period.

Chapter 11

Behind the Scenes

Sage model components and their variables are nothing at all like the constructs found in a typical scientific programming language like FORTRAN. They are, instead, examples of software objects which may be loosely defined as encapsulated data structures with built-in methods (functions and procedures) to manipulate their data. If this has you curious then by all means read this chapter. Otherwise you may skip it. You can run Sage perfectly well without knowing anything about its computer-science architecture.

11.1 Smart Variables

A good place to start is with the fundamental building blocks of Sage models: the variables. These are much more than named memory locations. As software objects they have some useful behaviors as well. For example, they know their names. They know how to format themselves for input and output. They know when their stored value is valid, or if it is not, how to call the proper evaluation function to make it valid. And they have a context within an overall solution scheme.

All variables are created either constant, independent, dependent or implicit. A *constant* is a variable whose value you may set via menu command at any time but remains fixed thereafter. An example of a constant would be a real-valued normalization quantity. An *independent* variable is like a constant except its value may also be set by an external driver, such as the mapper or optimizer. Collectively, constants and independent variables form the inputs of a model component. A *dependent* variable is one whose value depends on other variables, possibly other dependent variables, but eventually on non-dependent variables. Sage models contain mechanisms to invalidate dependent variables when the variables upon which they depend change. After invalidation, a dependent variable re-evaluates itself the next time it is needed. It does this by calling a function, whose address is stored locally within the variable, and waiting for the return value. An executing evaluation function may, in turn, call

upon other variables, some of which may turn out to be other invalid dependents. This process may go on for a while, cascading throughout a vast network of variables, until eventually the first evaluation function returns. In a sense, dependent variables struggling to keep themselves valid is what Sage models are all about. Finally, an *implicit* variable is one whose value is set by a nonlinear equation solver on the basis of zeroing a real-valued function whose address is, again, stored locally within the variable. Like independent variables, implicit variables are real valued. The implicit-variable evaluation function is much like a dependent-variable evaluation function, calling upon other variable values as needed, possibly giving more grief to the dependent variables who are constantly having to re-evaluate themselves. Collectively, the dependent and implicit variables form the outputs of a model component, although some are invisible to the user.

11.2 Smart Models

As you may have surmised by now, models in Sage are just collections of smart variables organized to exhibit some required behavior. But specific behaviors pertaining to physical phenomena occur in the high-level cerebral cortex of a model. Deep in the reptilian brain stem, common to all models, are a number of fundamental behaviors.

At the lowest level of awareness, models maintain a number of data structures which help them to manage their variables and any child models they might have. Thus they are able on demand to load and store themselves, their variables and their children to and from a disk file. They know how to invalidate any of their dependent variables currently valid and which models are affected by their non-dependent variables so they may ask them, in turn, to invalidate their dependent variables when required. And they are able to interface with Sage's I/O routines, its mapping and optimization drivers and its nonlinear solver in order to count, collect together or otherwise deal with variables that are either inputs, outputs, mapped, optimized or solved.

11.3 Grids

At a slightly higher level of awareness, some models know how, when asked, to deal with the various computational grids common to numerical analysis. There is a split in the model evolutionary tree here with one branch knowing about grids and the other not. The knowing branch is further divided according to the type of grid it understands. Presently supported are one-dimensional spatial grids, periodic one-dimensional time rings and two-dimensional space-time grids, denoted respectively by G_x, G_t and G_{xt}. All grids maintain nodes of real-valued variables (smart ones) at discrete spatial positions or times and support the common operations on grids such as numerical differencing, integrating and averaging. A periodic time ring is a grid where the nodes may be thought of as

equally spaced on the perimeter of a circle, so that there is no actual first or last node, except as a computer-storage convenience. Time rings are appropriate for modeling time-periodic phenomena.

11.3.1 Spatial Grids

Spatial grids are compatible with staggered-grid, also known as control-volume, solution methods. That is, they may be thought of as containing an integer number of control volumes or cells, with nodes falling at the boundaries and centers of each cell, so the total number of nodes is always odd. They do spatial differencing according to the central formula

$$\frac{\partial f}{\partial x}(i) = \frac{f(i+1) - f(i-1)}{2\,dx} \tag{11.1}$$

where i is the spatial index. For integration, spatial grids use a variation of Euler's rule, sampling only every other spatial index, at cell midpoints, thereby guaranteeing a global conservation property with staggered-grid solution schemes. This follows because when one integrates a term $\frac{\partial f}{\partial x}$ across a computational domain there is a pair-wise cancellation of f values, except at the two endpoints. In physical terms the global conservation property may be stated as "what flows out of one computational cell enters the next". The reason higher order differentiation is not used is that it would destroy this global conservation property, or complicate it greatly.

Spatial grids interpolate between neighboring solved values using Lagrange polynomial interpolation. Staggered-grid solutions generally alternate between solved and interpolated values. In the case of interpolation there is no global conservation property to worry about so, in principle, the order of interpolation can be as high as required. The current limit is set to cubic (third order) because of concerns that higher order will only cause numerical trouble due to the tendency of higher-order polynomials to produce wildly oscillating values between tabulated values. The order of interpolation may be set in the Options|Model Class dialog as either linear or cubic for those models using grids.

When possible, the neighboring solved values used in interpolation are distributed equally on either side of the interpolation point with the interpolation point at the center. For example, linear interpolation amounts to the averaging the values of the surrounding two solved variables.

$$f(i) = \frac{f(i-1) + f(i+1)}{2} \tag{11.2}$$

Cubic interpolation involves the weighted average of four neighboring values.

$$f(i) = -0.0625f(i-3) + 0.5625f(i-1) + 0.5625f(i+1) - 0.0625f(i+3) \tag{11.3}$$

When neighboring values fall outside the computational domain they are shifted toward one side or the other, as required. For example, linear interpolation at

the negative and positive domain endpoints become

$$f(i) = \frac{3}{2}f(i+1) - \frac{1}{2}f(i+3) \tag{11.4}$$

$$f(i) = \frac{3}{2}f(i-1) - \frac{1}{2}f(i-3) \tag{11.5}$$

unless the computational domain contains only one cell, in which case nodes $i+3$ or $i-3$ lie outside the computational domain and the interpolated value is just the cell-center value.

In general, spatial interpolation is carried out according to the formula

$$f(x_i) = \sum_{k=1}^{N} A_k f(x_p) \tag{11.6}$$

where N is an even number corresponding to the number of points in the set of neighboring solved points x_p used for the interpolation. The neighboring points x_p are a set of consecutive staggered locations in the neighborhood of the point of interpolation x_i, with index p given by

$$p(k) = (i-1) + 2(k - k_i) \tag{11.7}$$

k_i is the offset (0 to N) of the point of interpolation relative to the x_p. $k_i = 0$ corresponds to the x_p beginning at x_{i+1} (i.e. interpolating at the negative endpoint), $k_i = N/2$ corresponds to the x_p distributed equally on either side of x_i (i.e. central interpolation) and $k_i = N$ corresponds to the x_p ending at x_{i-1} (i.e. interpolating at the positive endpoint). The coefficients A_k are tabulated depending on N and k_i using a variation of Neville's recursive algorithm, as follows. Start with

$$F_{l,0} = \delta_l^k \qquad l = 1, \ldots, N \tag{11.8}$$

where δ_l^k is the Kronecker delta function (0 if $l \neq k$, 1 if $l = k$). Then compute

$$F_{l,m} = \frac{D_l F_{l+1,m-1} - D_{l+m} F_{l,m-1}}{2m} \qquad \begin{matrix} m = 1, \ldots, N-1 \\ l = 1, \ldots, N-m \end{matrix} \tag{11.9}$$

where the D_l are the offsets of the l^{th} p index relative to i.

$$D_l = 2(k_i - l) + 1 \tag{11.10}$$

and finally

$$A_k = F_{1,N-1} \tag{11.11}$$

This requires some explanation. Neville's recursive algorithm (*see* section 3.1 of [50]) obtains the unique $N-1$ order polynomial approximation at an interpolation point x_i to the function with values f_p at the points x_p. If the point x_i and the x_p always remain the same, the interpolation may be represented in the form of equation (11.6) where the A_k values need to be calculated only once and then saved for subsequent use. The way to calculate the A_k's is to trace

the contribution of each f_p component through the interpolation process. This is easily done by applying the Neville algorithm with all f_p values zero except for $f_p = 1$ for p corresponding to the index k. This is exactly what the above algorithm does by interpolating the set of basis functions δ_l^k. The one confusing point is that for purposes of the algorithm the tabulated x_p and f_p points are indexed sequentially using what is denoted above by the l index, as $x_{p(l)}$ and $f_{p(l)}$ for $l = 1, \ldots N$.

11.3.2 Time Grids

Time grids, or rings in Sage parlance, assume a periodic solution. They do time differencing using a scheme with its roots in the three-point backward differencing method attributed to Richtmyer and Morton [52]. Letting j denote the time index, this method is

$$\frac{\partial f}{\partial t}(j) = \frac{3f(j) - 4f(j-1) + f(j-2)}{2\,dt} \tag{11.12}$$

While this is a starting point for the method in Sage, it is not the actual method used. The method was originally improved by re-deriving it to "get the right answer for" or annihilate periodic sinusoidal functions rather than the first few terms of a polynomial, as the method was originally conceived. This was merely a matter of replacing the integer coefficients in the above formula with coefficients β_0, β_1, β_2 solved to annihilate test functions $f = 1$, $f = \sin$ and $f = \cos$. The result was a three-point backward differencing formula that served Sage well for several years.

But the problem was that although the method gave exact results for time-derivatives of purely sinusoidal functions, it introduced progressively increasing errors for higher harmonics, depending on the number of nodes in the grid. So the method was extended to annihilate as many harmonics as possible, by increasing the number of backward sampling points, according to the formula

$$\frac{\partial f}{\partial t}(j) = \sum_{k=0}^{2M} \beta_k f(j-k)) \tag{11.13}$$

where M is the order of the highest harmonic annihilated — in practice limited to $(N-1)\mathrm{div}\,2$, where N is the number of nodes in the grid. ("div" is the divide operator that rounds down to the next lowest integer) With this value of M, all available harmonics are annihilated when N is odd and all but the highest when N is even.

By requiring that formula (11.13) hold for the first M Fourier-series basis functions, it is straight-forward to show that the required β_k coefficients must

solve the following linear equation system

$$\begin{bmatrix} \sin\Delta\tau & \sin 2\Delta\tau & \dots & \sin 2M\Delta\tau \\ 1-\cos\Delta\tau & 1-\cos 2\Delta\tau & \dots & 1-\cos 2M\Delta\tau \\ \sin 2\Delta\tau & \sin 4\Delta\tau & \dots & \sin 4M\Delta\tau \\ 1-\cos 2\Delta\tau & 1-\cos 4\Delta\tau & \dots & 1-\cos 4M\Delta\tau \\ & \vdots & & \\ \sin M\Delta\tau & \sin 2M\Delta\tau & \dots & \sin 2M^2\Delta\tau \\ 1-\cos M\Delta\tau & 1-\cos 2M\Delta\tau & \dots & 1-\cos 2M^2\Delta\tau \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \\ \vdots \\ \beta_{2M-1} \\ \beta_{2M} \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ -2 \\ 0 \\ \vdots \\ -M \\ 0 \end{bmatrix} \tag{11.14}$$

with $\Delta\tau$ the dimensionless time step

$$\Delta\tau = \frac{2\pi}{N} \tag{11.15}$$

and the zeroth coefficient computed explicitly as

$$\beta_0 = -\sum_{k=1}^{2M} \beta_k \tag{11.16}$$

once those for $k \geq 1$ are available. This implicit formulation is no problem computationally. It just means that Sage must invoke a linear solver object to compute the β_k's, rather than evaluate some, perhaps, complicated expressions involving sine and cosine functions. This is required only once upon creation of the computational grid.

For integration, time grids use Euler's rule without modification — sampling and giving equal weight to every time index. This is the only possibility, since logically time grids have no beginning or end indices, or any other basis, to distinguish one node from another for weighting purposes.

11.4 Connecting Things Together

So just what it does it mean to connect two model components together across a boundary? And how is it that information passes between the two components? Whenever you make a connection there is actually a third invisible object created, a boundary connector. This object is a close relative of a model component in that it is a collection of smart variables organized for a specific behavior. The behavior in this case is to provide a common boundary-value variable (or variables) to be used by the adjoining models as part of their solution and to receive from those models, in exchange, sufficient information to establish the validity of the boundary variable. This information generally takes the form of some quantity that must be conserved across the connection.

Take steady heat flow for example. We might imagine two thin heat-conducting rods connected in series between an isothermal source and an isothermal sink. Each rod is a separate model component and the connection in the middle is the current point of interest. The physical principle governing heat flow is this:

Heat flows across the connection until the rod endpoint temperatures are equal. Simple. The Sage equivalent goes like this: Heat flow is an implicit variable managed by a boundary-connector object in the middle. Each rod maintains its own independent solution, producing an axial temperature distribution as a function of connector heat flow. When the user connects the two rod ends together, each rod receives a pointer to the connector object which it can call upon to read the current value of heat flow. Meanwhile, the connector object receives pointers to the appropriate endpoint-temperature variables within both rod models. It evaluates the difference of these two temperatures as the to-be-zeroed function component associated its implicit heat flow variable. A nonlinear solver object in the background sees all this as just another equation to be solved in its equation system, and iterates the value of heat flow until temperature continuity is achieved. This is essentially how most Sage connector objects work.

As a matter of notational convenience, we could denote the previous heat-flow boundary-connector example as $Q(T)$ where Q represents the implicit heat-flow variable in the connector and T represents endpoint temperatures in the adjoining rods, continuity of which determines Q. Using this notation we can quickly summarize the basic types of boundary connectors available in Sage:

$F(\Delta X)$ — Force determined by positional displacement continuity.

$P(\Delta V)$ — Pressure determined by volumetric displacement continuity.

$T(\theta)$ — Torque determined by rotation angle continuity.

$Q(T)$ — Heat flow determined by temperature continuity.

$\rho(P)$ — Density determined by pressure continuity.

$\dot{m}(s)$ — Gas mass flow rate determined by state-variable continuity. Actually, $\dot{m}$ symbolizes combined mass flow rate, mass-density jump and energy-density jump, while s symbolizes, momentum flow, energy flow and a momentum equation.

$I(V)$ — Electrical current determined by voltage continuity.

$\phi(\Psi)$ — magnetic flux determined by magnetic potential continuity.

$R(B)$ — Thermal radiation flow determined by radiosity continuity (combined emitted plus reflected radiation per unit area of surfaces exchanging radiation).

Not very many connector types, really. However, as with models, connectors are further subdivided according to the numerical representation of the solved variables within themselves and in the models to be joined. For each basic type there may be variations for steady-valued solutions, complex or phasor solutions and the various types of grid solutions. But the basic types of boundary connectors are those listed.

11.5 Solving

When you elect to solve the model hierarchy you have specified, you are really calling into play another software object: a nonlinear solver. The purpose of this object is to orchestrate the iterative solution of a large system of nonlinear equations — one equation (setting the evaluation function to zero) for each implicit variable in the model hierarchy. Although often numbering in the hundreds, these implicit variables are generally invisible to the user. They come about automatically, as computational grid variables and the like.

The nonlinear solver does its job by solving a sequence of linear approximations to the nonlinear equations. The coefficients of the linearized equations result from numerical partial derivatives of the evaluation functions taken with respect to the model's implicit variables. The result is a sparse matrix, solved with a special sparse-matrix solver. The solution becomes a search direction along which to seek the nonlinear solution. This process repeats until the nonlinear model equations are all satisfied within some prescribed tolerance.

Mathematically, each iteration takes the form of solving the equation

$$\boldsymbol{J}\Delta V = -F \tag{11.17}$$

for the step ΔV, where $\boldsymbol{J}$ is the Jacobian (partial-derivative) matrix and F is the to-be-zeroed function vector. This is Newton's method. While Newton's method works well for *nearly* linear system functions, it can fail to converge for nonlinear ones, especially when the starting value for V is far from the final solution. In particular, discontinuities of implicit-variable initial values across connections have been observed to be a chief cause of non-convergence in Sage. The implicit function components corresponding to these discontinuities are of the form $F = V_+ - V_-$, where V_+ and V_- are the implicit variable values on either side of the connection. Newton's method tries to zero the discontinuity in one step, which tends to destabilize the solution. It is not too difficult to avoid this problem by relaxing variable discontinuities more slowly, which is what Sage's nonlinear solver does.

The idea is to replace F on the right-hand side of (11.17) by ΔF, the desired change in F. For most components, ΔF_j is just $-F_j$, meaning that F_j is allowed to step all the way to zero. But for key components (those of the form $F = V_+ - V_-$), ΔF_j is some smaller amount — small enough to avoid destabilizing the solution. The maximum allowable ΔF_j is something that is left to smart variables to decide.

11.6 Optimizing

Likewise, when you elect to optimize your model, you are calling into play yet another software object: an optimization driver. The nonlinear optimizer in Sage employs a variation of Powell's sequential-quadratic-programming method [49] which locally approximates the nonlinear optimization problem by a succession of quadratic sub-problems (quadratic objective function and linear constraints)

each of which is readily solved. The idea is that by doing this often enough and searching along the direction from where you are to where the quadratic minimizer lies, you will eventually converge to the actual nonlinear minimizer — or rather *a* minimizer if there happens to be more than one. Powell's algorithm builds up its quadratic programming problems as it steps along by accumulating second derivative information about the objective function and constraints. It then turns to a separate quadratic optimizer for the sub-problem solution — in the Sage implementation, a convex method suited to Powell's method, reported by Goldfarb and Idnani [27]. Complicated as this may seem, it really does work.

11.7 Delphi Spoken Here

Sage is a Delphi application (Embarcadero) running under Microsoft Windows. So why not C++, some might ask. Partly for continuity's sake, since the solving and optimization engines of Sage were originally written in Pascal (the underlying language of Delphi). But also, Delphi's visual component library has taken much of the drudgery out of programming the Windows interface, a considerable part of Sage. And the inherent structure and readability of the underlying Pascal language seem to make for more reliable, maintainable code.

At any rate, the object-oriented philosophy pervades Sage. Almost everything is a class object of one sort or another. This makes some things easier than before, some things harder. Easier are creating and maintaining reliable models for complicated phenomena. A new model can inherit its basic behaviors from an earlier model then add new behaviors as needed. Thus, it is straight-forward to create from a generic gas domain, three variations each with its own turbulence-transition model. Debugging is easier too, because it is a snap to isolate and test model components one at a time. Harder is realizing global connectivity from a bunch of separate objects. Seemingly little things like summing masses of different components or imposing dimensional stack-up constraints can be vexing problems in the object-oriented world. As programmed, objects are self contained entities that know nothing of the larger context of the model they find themselves in. Sage's answer to this is to provide such things as connector objects, user defined variables and recastable inputs which were expensive in programmer time to conceive of and implement, but allow the user to play an active role in defining high-level model structure.

Part III

SCFusion Model Classes

Chapter 12

Overview and Tutorial

The SCFusion model class (stirling-cycle fusion) allows you to model a wide variety of steady or time-periodic machines based on the stirling or related thermodynamic cycles. A SCFusion model is built up from component parts of your choosing representing thermal solids, gas domains, canisters, heat exchangers, piston-cylinder pairs, and so forth. The parts function as a whole by virtue of their interconnections — heat and gas flows through appropriate boundaries, forces acting on appropriate attachment points and pressures on appropriate area faces. Add to this the possibility for user-defined variables and custom optimization specifications and you can see that the possible combinations are endless.

Sample Models The SCFusion installation includes a large number sample files to get you started (in the Apps\SCFusion\Samples sub-directory under the installation directory). These are accessible from Sage with the Help|Sample Models menu item. There may well be a sample file very close to the machine you are interested in. All you have to do is save the sample model with another name (File|Save As) and edit it to your liking. Each sample file also comes with documentation in a companion file in Adobe PDF format with the same name but file extension pdf. These are installed in the same directory as the sample files.

Tutorial A good way to get acquainted with Sage is to start with the simple spring-mass-damper resonant system model found in the _Tutorial subfolder. Open a model with Sage, its companion documentation file with a PDF reader, and follow along.

Chapter 13

Boundary Connections

SCFusion model components communicate with each other using the following boundary connections. You may only connect together like connectors of opposite sign (opposite-facing arrows).

13.1 Force Connections

F_{phsr} or F_{Gt}

These represent either phasor or time ring forces acting on points of attachment. The points of attachment will share the same motion when connected together. Force connections are used primarily for connecting springs and dampers to moving parts.

13.2 Pressure Connections

P_{phsr} or P_{Gt}

These represent either phasor or time ring pressure variations acting on area faces. The area faces share the same volume displacement when connected together. They are used primarily for connecting pistons and the like to gas domains.

13.3 Heat Flow Connections

Q_{stdy}, Q_{Gt}, Q_{Gx} or Q_{Gxt}

These represent either steady, time-grid, spatial grid, or space-time grid heat flows acting on thermal boundaries. Boundaries share the same temperature when connected together. Steady heat flows are useful for time-averaged parasitic conduction losses. Spatial grid heat flows are useful for steady but dis-

tributed heat flows, such as occurs two dimensional fins. Space-time heat flows are for connecting thermal solids to gas domains.

13.4 Gas Flow Connections

$\dot{m}_{\mathsf{Gt}}$

These represent the flow of gas from one gas domain inlet into another. Two inlets conserve mass flow rate, energy and momentum when connected together.

13.5 Density Connections

ρ_{stdy}

This represents the common mass density between a gas domain and a pressure reservoir. The two share the same mean pressure when connected together. Density connections are used to connect the SCFusion working gas to a fixed-pressure source in order to establish charge pressure. This connection type is generally used only once per SCFusion model instance, but its use is critically important.

13.6 Electrical Current Connections

I_{Gt}

These represent the flow of electrical current from one electrical component into another. Electrical boundaries share the same voltage when connected together.

13.7 Magnetic Flux Connections

ϕ_{Gt}

These represent the flow of magnetic flux from one magnetic component into another. Magnetic boundaries share the same magnetic potential when connected together.

Chapter 14

Entropy Generation

Many SCfusion model components involve irreversible entropy-generating processes and keep tabs of entropy generation in terms available energy output variables. Available energy or availability for short is just entropy multiplied by the negative of ambient temperature T_0, which is really just input Tnorm in the SCFusion root model component. A loss in availability equates to a decrease (in an engine) or increase (in a cooler) of PV power.

14.1 Lost Available Energy

The concept of availability arises from the net mechanical work W_r available from a reversible heat engine with a net inflow of heat Q (positive) at a temperature T and outflow Q_0 (negative) at a sink temperature T_0:

$$W_r = Q + Q_0 = Q(1 - T_0/T) \tag{14.1}$$

In contrast, an irreversible-cycle might produce work $W_i < W_r$ obtained from the same input Q. According to the first law of thermodynamics, and in terms of heat rejected Q_0^*

$$W_i = Q + Q_0^* \tag{14.2}$$

where Q_0^* is no longer equal to $-QT_0/T$. The lost available energy is just

$$W_r - W_i = -QT_0/T - Q_0^* = -T_0(Q/T + Q_0^*/T_0) \tag{14.3}$$

Evidently, this is just $-T_0$ times the net entropy increase of the surrounding universe as a result of the heat flows into and out-of the system. This leads us to generally characterize all internal entropy generations in the SCFusion model class in terms of losses in *availability* according to the definition

$$\text{Availability Loss} = -T_0 \times \text{Entropy Generation} \tag{14.4}$$

The value of the availability-loss concept is that it allows us to think about entropy generation in terms of the more concrete notion of lost mechanical work.

It is important to note that availability losses vary with the sink temperature T_0, which is given by input variable Tnorm in the SCFusion root model component. Therefore, availability losses really equate to lost PV powers only when all points of heat rejection are indeed at the temperature Tnorm. See reference [5] for a standard treatment of available-energy accounting.

14.2 Second-Law Balance

Entropy generation (or available energy loss) may be measured in one of two ways: by entropy flow across the external boundaries of a SCFusion model instance or by entropy generated by internal processes. The previous example accounts for entropy generation in the first way. The surrounding universe suffers a loss of available energy due to heat flow through the boundaries of a SCFusion machine. When we get specific about individual SCFusion model components we will also be able to account for entropy generation in the second way — by tallying up the individual entropy generations in all internal processes.

In principle, the two methods of accounting should give the same answer. But often they do not. This is because there is no conservation-of-entropy-generation principle built into the SCFusion model components, as there is for conservation of energy. Discrepancies arise, usually attributable to numerical artifacts in the computational solution. These include finite-differencing truncation errors, for both spatial and temporal partial derivatives, as well as interpolation errors needed to obtain solution variables at off-solution grid points. So model components that tally available energy losses generally have an available-energy discrepancy output, as well, that monitors the difference between external-boundary and internal-process accounting.

It is tempting to look at the relative magnitude of the availability accounting discrepancy as a measure of the solution accuracy. Large discrepancies may be produced by difficult to model features such as heat exchangers with highly nonlinear axial temperature distributions, large pressure ratios, etc. In such cases it is usually helpful to refine the computational grid — that is, add more time nodes (NTnode) or more control volumes (NCell).

Chapter 15

SCFusion Root Component

The highest level SCFusion model component, and the only model component you see when you create a new model instance, is the root model. Its purpose is to define a few global variables and some normalization constants as a framework for all its children which you create, or someone created before, by selecting from the component palette. Its variables are:

NTnode : (dimensionless) The number of time nodes in the computational grids of all underlying model components having such grids. NTnode is important because it dramatically affects the solution time and memory requirements, not to mention solution accuracy. Treat it gingerly by making only gradual, small changes. Changing this variable re-creates all time grids in the model and initializes solution values by interpolating between solved values of the previous grid.

You can get an idea if the solution grid is too coarse (NTnode too small) by inspecting the various Fourier series output variables produced by individual model components. If the highest harmonic is not small compared to the lower harmonics then you might want to increase NTnode to improve solution accuracy. You should also increase it to an odd number because NTnode must be odd to resolve the mean value plus both components of all harmonics available in the grid. When NTnode is even the highest harmonic is not fully determined. This leads to finite differencing errors in addition to the inaccuracy produced by not resolving all the significant harmonics in the solution.

It is a good idea to always set NTnode to an odd value, although even values are allowed by Sage. When NTnode is odd the highest harmonic available in the grid is (NTnode - 1) / 2.

Lnorm : (real, m) Length scale normalization constant. The value of this input may make the difference between a solution that converges and one that doesn't. As a rule of thumb, Lnorm should be, roughly, the cube root of the largest swept volume amplitude in the underlying SCFusion model. Some

fiddling around may be required to achieve optimal solution convergence rate. Keep in mind that a small change may have a big effect, since many quantities are normalized by the cube of Lnorm.

FreqNorm : (real, Hz) Frequency scale normalization constant. This one is not as critical as Lnorm. Generally, set it to the actual operating frequency Freq. It is a separate constant because Freq may change during optimization.

Pnorm : (real, Pa) Pressure scale normalization constant. Again, not as critical as Lnorm, but important because it establishes the initial pressure in all underlying gas domains. It should be close to the charge pressure Pcharge in the pressure-source component. Otherwise, the pressure solutions in gas domains may go unstable.

Tnorm : (real, K) Temperature scale normalization constant. Usually the ambient temperature, or about 300 K. Keep in mind that it scales available energy outputs in certain model components (see chapter 14).

Qnorm : (real, W) Heat flow normalization constant. Generally, set it to the magnitude of the total heat rejection expected for the SCFusion model. Increasing Qnorm can sometimes help a solution converge when the convergence error is hovering near the target error and the problem is due to surface heat flux stiffness (small temperature fluctuation producing large heat flux fluctuation) as can happen, for example, with a particularly effective regenerator matrix.

Freq : (real, Hz) The actual operating frequency for the SCFusion machine. Although it is set as an input, it may be optimized to satisfy an objective function or constraints.

Omega : (real, radians/s) A convenience output that converts Freq to angular frequency by multiplying by 2π.

Gas : (enumerated) The working gas type (see below).

15.1 Working Gas

The gas variable is deceptively simple. It is an enumerated-type variable that selects the working gas from a list of choices. Each available gas is itself a software object with sufficient data encapsulated within it to know its important properties.

The values of the defining properties appear after the enumerated identifier in display windows and the output listing, provided the Display Options selection is "Full Detail" in the Sage Options dialog (Options | Sage menu item. The gas choices come from a default database file gasSCF.dta, or a data base file customized by you. (*see* chapter 28)

An instance of a gas object has methods (subroutines) that can be called upon to return various properties, as functions of temperature, or various state variables, as functions of other state variables. The underlying gas-domain model components call upon these methods when needed. Sometimes, this requires only cubic-spline interpolation using the appropriate set of data pairs. Sometimes, it gets a little more complicated.

15.1.1 Ideal Gas Physics

As the name implies, ideal-gas objects presume the ideal gas equation of state

$$Pv = RT \tag{15.1}$$

The defining properties for an ideal gas are:

T0 : (K) Representative temperature T_0.

Rgas : (J/(kg K)) Gas constant R.

Specific heat : (J/(kg K)) A cubic spline variable with temperature vs specific heat $(T, c_p(T))$ data pairs covering a broad range of temperatures.

Viscosity : (kg/(m s)) A cubic spline variable with temperature vs viscosity $(T, \mu(T))$ data pairs.

Conductivity : (W/(m K)) A cubic spline variable with temperature vs conductivity $(T, k(T))$ data pairs.

Note that the cubic-spline interpolated values for c_p, μ and k do not include any pressure dependence. The default ideal gas objects are based on atmospheric-pressure data. These properties are only weakly dependent on pressure provided the fluid state is well above the critical temperature or well below the critical pressure.

Available property methods are presented below in functional notation. For example, $c_p(T)$ is the heading for the method that returns c_p as a function of T.

Specific Heat Ratio γ

This is actually a data field, not a method, initialized as

$$\gamma = \frac{1}{1 - R/c_p} \tag{15.2}$$

where c_p is the zero-pressure limit at reference temperature T_0. This is the only reason for the existence of T_0 as a data field. It is important that γ be a constant, rather than a function of T, because internal energy and entropy both depend on γ ($c_v = R/(\gamma - 1)$) as does sonic velocity.

Specific Heat $c_p(T)$

Zero-pressure c_p limit, that is. This is just a matter of cubic-spline interpolation from the specific-heat data pairs.

Prandtl Number $P_r(T)$

Calculated as a function of temperature as $P_r = c_p\mu/k$.

Viscosity $\mu(T)$

Molecular viscosity interpolated from viscosity-temperature data pairs.

Conductivity $k(T)$

Molecular conductivity interpolated from conductivity-temperature data pairs.

Sonic Velocity $u_s(T)$

Returns the isentropic, small amplitude, phase velocity of an acoustical wave

$$u_s = \sqrt{\gamma RT} \tag{15.3}$$

Specific Volume $v(T, P)$

Returns specific volume $1/\rho$, defined as

$$v = \frac{RT}{P} \tag{15.4}$$

Compressibility $Z(\rho, T)$

Returns compressibility, defined as

$$z = \frac{P}{\rho RT} \tag{15.5}$$

which equals one, by definition, for an ideal gas.

Temperature $T(\rho, \rho e, u)$

Returns temperature as function of mass density, volume-specific energy density and velocity

$$T = \frac{\gamma - 1}{R}\left((\rho e)/\rho - u^2/2\right) \tag{15.6}$$

Pressure $P(\rho, T)$

Returns pressure as function of mass density and temperature

$$P = \rho RT \tag{15.7}$$

Energy Density $\rho e(\rho, T, u)$

Returns volume-specific energy density as function of mass density, temperature and velocity

$$\rho e = \rho \left(R/(\gamma - 1)T + u^2/2 \right) \tag{15.8}$$

where $R/(\gamma - 1) = c_v$.

Entropy $s(\rho, T)$

Returns mass-specific entropy as function of mass density and temperature

$$s = R \left(\ln(T)/(\gamma - 1) + \ln(1/\rho) \right) \tag{15.9}$$

Equation of State Error $\tilde{Z}(\rho, T)$

Returns zero because the ideal equation of state relative error is zero by assumption.

15.1.2 Tabular Gas Physics

Tabular gas objects are instances of Sage's TBspline3Gas class, designed for use when the working fluid state may fall in or near the two-phase state within the model scope (low-temperature J-T, or GM crycoolers). The "BSpline" in the name refers to bicubic-spline interpolation, which is a way of fitting a piecewise-defined cubic polynomial to two-dimensional data (specific volume v and temperature T being the two dimension) so that first and second derivatives are continuous throughout the interpolation domain (*see* chapter 3 of reference [50]).

The TBspline3Gas class descends from preceding classes TBspline2Gas and TBsplineGas introduced in earlier versions of Sage. These preceding classes are still supported for file backward compatibility but are no longer recommended of Sage version 13. See chapter 29.

TBSpline3 Gases define specific heat c_p, viscosity μ and conductivity k using (v, T) interpolation tables rather than T-only interpolation pairs. Previous ideal-gas cubic-spline data-pairs (T, Cp), (T, μ) and (T, k) are still present but are now all understood as zero-pressure limits. They are automatically generated on input from the high-v column of the corresponding data table without any additional user input required (in the SCFProp.exe property management software). Previously only Cp was a zero-pressure limit, the other two properties were tabulated for a representative pressure. The only reason c_p is included as a (v, T) table is for the calculation of Prandtl number used in certain heat-transfer formulations.

In support of modeling within the two-phase region TBSpline3Gases contain two new cubic spline variables $\rho_d(T)$ and $\rho_b(T)$, representing the densities of the vapor at the dew point and liquid at the bubble point (*see* below).

A TBSpline3Gas is defined by the following values:

T0 : (K) Representative temperature T_0. A vestige inherited from the original ideal gas class, used only as the temperature for calculating the zero-pressure ratio of specific heats $\gamma = c_p/c_v$.

Z(v, T) : (dimensionless) Rectangular table of $Z(v_j, T_k)$ values where $Z = Pv/RT$ is compressibility, v is specific volume and T is temperature. The first (left) column contains the T_k values and the first (top) row the v_j values. The corresponding Z values lie in the body of the table.

ε(v, T) : (J/kg) Rectangular table of mass-specific internal energy values $\varepsilon(v_j, T_k)$, structurally similar to the $Z(v_j, T_k)$ table.

Cp(v, T) : (J/(kg K)) Rectangular table of constant-pressure specific heat values $c_p(v_j, T_k)$, structurally similar to the $Z(v_j, T_k)$ table.

Mu(v, T) : (kg/(m s)) Rectangular table of viscosity values $\mu(v_j, T_k)$, structurally similar to the $Z(v_j, T_k)$ table.

K(v, T) : (W/(m K)) Rectangular table of thermal conductivity values $k(v_j, T_k)$, structurally similar to the $Z(v_j, T_k)$ table.

ρ_d(T) : (kg/m^3) Cubic spline data pairs of dew-point density values $(T_k, \rho_d(T_k))$, with temperature values covering the lower range of the other tables up to critical temperature T_c. Usage within the TBSpline3Gas class requires that the final data pair be (T_c, ρ_c), where ρ_c is the critical density. The dew-point density $\rho_d(T)$ is the vapor density where liquid condensate first appears for vapor compressed at a fixed temperature T.

ρ_b(T) : (kg/m^3) Cubic spline data pairs of bubble-point density values $(T_k, \rho_b(T_k))$, with temperature data requirements same as for the $\rho_d(T)$ data pairs. The bubble-point density $\rho_b(T)$ is the liquid density where vapor bubbles first appear for liquid expanded at fixed temperature T).

15.1.3 Auxiliary Property Tables

The original TBSplineGas class derived auxiliary cubic-spline data tables necessary for evaluating the complete list of referenceable properties below. These data tables were volume-specific internal energy $\varepsilon(v_j, T_k)$ and mass-specific entropy $s(v_j, T_k)$, both derived from the state table $Z(v_j, T_k)$. TBSpline3Gas now derives only the entropy table this way because numerical errors in path integrations led to discrepancies in the derived internal energy data compared to REFPROP values. Be that as it may, the methods for deriving both internal energy and entropy from Z data are documented below for reference.

Internal Energy According to reference [56] (equation (232), p. 285), the differential of mass-specific internal energy is

$$d\varepsilon = c_v dT + \left[T \left(\frac{\partial P}{\partial T} \right)_v - P \right] dv \tag{15.10}$$

In terms of compressibility Z, one can substitute $R\rho TZ$ for P and $R\rho\left[Z + T\left(\frac{\partial Z}{\partial T}\right)_v\right]$ for $\left(\frac{\partial P}{\partial T}\right)_v$ so that the internal energy differential becomes

$$d\varepsilon = c_v dT + \left[R\rho T^2 \left(\frac{\partial Z}{\partial T}\right)_v\right] dv \tag{15.11}$$

The approach TBspline2Gas used to build a table of ε values was as follows:

First build the v_{max} row of the table by temporarily assigning the corner point $\varepsilon(v_{max}, T_{min})$ the value zero, then calculating the remaining points in the row by integrating (15.11) with respect to T only, from one point to the next. Assuming ideal-gas conditions at v_{max}, it is valid to substitute the zero-pressure limit $c_{v0}(T)$ for $c_v(T)$, then $c_{p0}(T) - R$ for $c_{v0}(T)$, since $c_{v0}(T)$ is not directly tabulated. $c_{p0}(T)$, on the other hand (the zero-pressure limit of c_p), is available as a cubic-spline data object.

Then, in a similar point-by-point process, complete the columns of the table by integrating equation (15.11) with respect to v only, starting at each of the (v_{max}, T_k) points already calculated and using bicubic-spline routines operating on the Z table to calculate the value of $\left(\frac{\partial Z}{\partial T}\right)_v$.

Entropy The differential of mass-specific entropy is

$$ds = \frac{d\varepsilon}{T} + \frac{P}{T} dv \tag{15.12}$$

again according to reference [56] (equation (103), p. 102). By substituting $\left(\frac{\partial \varepsilon}{\partial T}\right)_v dT + \left(\frac{\partial \varepsilon}{\partial v}\right)_T dv$ for $d\varepsilon$ this becomes

$$ds = c_v \frac{dT}{T} + \left[\left(\frac{\partial \varepsilon}{\partial v}\right)_T + P\right] \frac{dv}{T} \tag{15.13}$$

As before, the v_{max} row of the table is produced by integrating equation (15.13) with respect to T only, point-by-point, starting with a temporary value of zero for $s(v_{max}, T_{min})$, and substituting $c_{p0}(T) - R$ for $c_v(T)$. Then the columns of the table are completed by integrating equation (15.13) with respect to v only, starting at each of the (v_{max}, T_k) points just calculated.

The process so far does not establish an appropriate constant of integration. Usually this is done to give a particular reference value for entropy at some state point (v_0, T_0). In principle, this is not a problem since the equations in Sage's gas domains are affected only by changes in entropy, not absolute values. But it may be more of a problem with human perception. So an integration-constant offset is added to the entire table so that the minimum entropy value is some convenient value, such as gas constant R.

Dependant Properties All of the gas properties Sage requires can be calculated in terms of the above $Z(v,T)$, $\varepsilon(v,T)$ and $s(v,T)$ tables. The property methods that perform these calculations have the same names as those for ideal gases, except the internal calculation details are entirely different. Interpolation

replaces algebraic calculation and the specific volume v and mass-specific internal energy ε used in the state-table formulation are converted back and forth into Sage variables density ρ and volume-specific total energy ρe.

Specific Volume $v(T, P)$

Finding $v(T, P)$ is a matter of using Newton's method to iteratively solve

$$F(v) = P_{in} - P(v, T) = 0 \tag{15.14}$$

Where P_{in} is the given pressure input and $P(v, T)$ is the pressure interpolated from the table of Z values. Required in the process is $\frac{\partial P}{\partial v}$, which is supplied by bicubic spline routines operating on the Z table.

Newton's method requires a good starting value for v, otherwise it may not converge. This starting value comes from a binary search in the Z table to find the v-indices that bracket P_{in} (the indices j_{lo} and $j_{hi} = j_{lo} + 1$ so that $P(v_{j_{lo}}, T) \geq P_{in} \geq P(v_{j_{hi}}, T)$) followed by a linear interpolation between the bracketing values. The bracketing technique is reliable because P always decreases with increasing v for all gases. Pressure may remain constant during the liquid to vapor transition but it never increases.

Compressibility $Z(\rho, T)$

This is directly available by interpolation in the Z table as $Z(1/\rho, T)$.

Temperature $T(\rho, \rho e, u)$

Sage uses temperature in the solution process so this is a critical method. The first step is to convert volume-specific total energy ρe to mass-specific internal energy ε using

$$\varepsilon = (\rho e)/\rho - u^2/2 \tag{15.15}$$

then finding $T(\rho, \varepsilon)$ is once again a matter of using Newton's method, this time to iteratively solve

$$F(T) = \varepsilon_{in} - \varepsilon(1/\rho, T) \tag{15.16}$$

Where ε_{in} is the given energy input and $\varepsilon(1/\rho, T)$ is the energy interpolated from the table of ε values. Required in the process is $\frac{\partial \varepsilon}{\partial T}$, which is supplied by bicubic spline routines operating on the ε table.

The initial guess at T comes from a binary search in the ε table to find the T-indices that bracket ε_{in} (similar to above) followed by a linear interpolation between the bracketing values. The bracketing technique is reliable because ε always increases with increasing T for all gases.

Pressure $P(\rho, T)$

This is directly available by interpolation in the Z table as

$$P = Z(1/\rho, T)R\rho T \tag{15.17}$$

Energy Density $\rho e(\rho, T, u)$

This is directly available by interpolation in the ε table as

$$\rho e = \rho(\varepsilon(1/\rho, T) + u^2/2) \tag{15.18}$$

Entropy $s(\rho, T)$

This is directly available by interpolation in the s table.

Equation of State Error $\tilde{Z}(\rho, T)$

This could be based on some measure of error in the actual tabulated values of $Z(v_j, T_k)$ as well as some estimate of the cubic spline interpolation error. But it is not worth the bother since $\tilde{Z}$ is required only for an output variable in the Sage listing. So $\tilde{Z}$ is presumed zero until further notice.

15.1.4 Critical Point Meaning

For low-temperature cooler modeling and in the TBSpline3Gas framework the boundary of the two-phase region is of fundamental importance. That boundary is defined by the bubble-point and dew-point curves which when plotted on a T-v diagram meet at the top at a so-called *critical-condensation* point, or *criconden* point for short. For a pure fluid the criconden point is the same as the thermodynamic *critical* point defined by the isotherm T_c in a P-v plot where there is an inflection point, i.e. where $(dP/dv)_{T_c} = 0$ and $(d^2P/dv^2)_{T_c} = 0$. But for a mixture the criconden point may differ significantly from the thermodynamic critical point as illustrated in figure 15.1 for a 50-50 mixture of methane and butane.

For the above reason all critical values in the context of a TBSpline3Gas refer to the values at the criconden point. In particular the critical temperature T_c is the temperature known to chemical engineers as the *cricodentherm*.

15.1.5 Vapor Quality

It is sometimes useful to know the vapor quality (vapor mass fraction) for fluid in the two-phase region. For a given quantity of fluid containing vapor mass m_v and liquid mass m_l, the quality is defined as

$$X \equiv \frac{m_v}{m_v + m_l} \tag{15.19}$$

The total volume occupied by the two states is the sum $m_v/\rho_v + m_l/\rho_l$, where ρ_v and ρ_l are the vapor and liquid densities. This same total volume may also be represented in terms of a bulk density ρ as $(m_v + m_l)/\rho$. Equating these two representations of volume, dividing through by $m_v + m_l$ and using the above

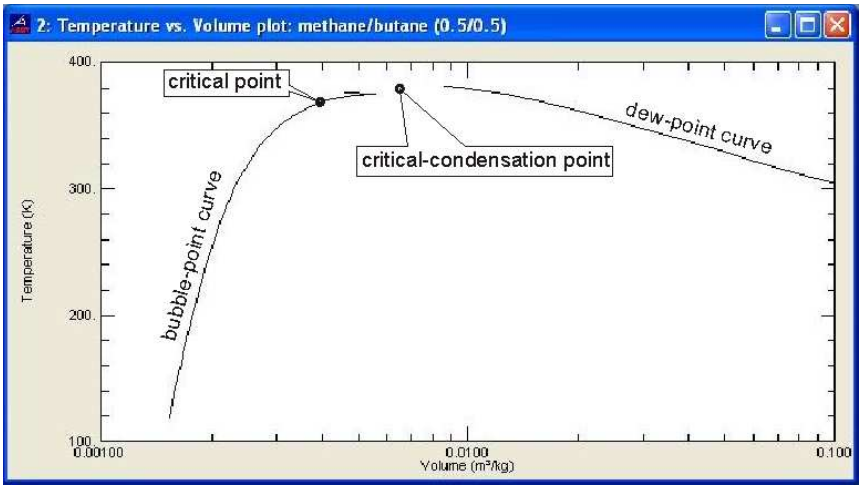


Figure 15.1: REFPROP produced T-v plot for a 50-50 mixture of methane and butane showing the critical-condensation point separate from the thermodynamic critical point. The critical-condensation point theoretically lies at the intersection of the bubble and dew-point curves although the two curves do not quite meet at the top in REFPROP.

definition of quality gives an equivalent definition of quality in terms of the three densities

$$X \equiv \frac{1/\rho - 1/\rho_l}{1/\rho_v - 1/\rho_l} \tag{15.20}$$

However, TBSpline3Gases do not separately tabulate the vapor and liquid densities ρ_v and ρ_l (as a function of v, T) so they define instead a *pseudo* quality in terms of dew-point and bubble-point densities ρ_d and ρ_b as

$$Q \equiv \frac{1/\rho - 1/\rho_b}{1/\rho_d - 1/\rho_b} \equiv \frac{v - v_b}{v_d - v_b} \tag{15.21}$$

For pure fluids Q and X are the same thing because the equilibrium pressure remains constant along an isotherm so ρ_v and ρ_l are always the same as ρ_d and ρ_b. In mixtures the pressure varies so Q and X are not quite the same. But Q is a reasonable approximation to the true vapor quality for many purposes and approaches the correct values of $Q = 0$ as $\rho \to \rho_b$ and $Q = 1$ as $\rho \to \rho_d$.

15.1.6 Transport Property Encoding in Two-Phase Region

In the two-phase region the values of transport properties c_p, μ and k in the interpolation tables are not the bulk values (mass weighted liquid/vapor average) as they are for the thermal properties Z and ε. Rather the TBSpline3Gas class

stores separate liquid and vapor phase values in a special format so that they may be recovered individually. TBSpline3Gases override the inherited transport property evaluation functions to decode the vapor values within the two-phase region and define new functions to decode the liquid values. In doing so the transport properties for the separate liquid and gas phases are available for heat transfer and flow resistance calculations.

Essentially, the property values for the liquid phase are stored between the bubble-point and critical specific volumes v_b and v_c and values for the vapor phase between the critical and dew-point specific volumes v_c and v_d, for each temperature T_k. To be more precise it is useful to normalize the specific volume range between v_b and v_d to the range 0 to 1 using the function $Q(v)$ defined by equation 15.21. The value F stored in the two-phase region of the property table is either the liquid value F_L or vapor value F_V according to the following formula in which F, F_L and F_V are understood to be functions of Q:

$$F(Q) = \begin{cases} F_L\left(\frac{Q}{Q_c}\right) & \text{if } Q < Q_c \\ F_V\left(\frac{Q-Q_c}{1-Q_c}\right) & \text{if } Q > Q_c \end{cases} \tag{15.22}$$

where Q_c is the pseudo-quality evaluated at the critical specific volume

$$Q_c \equiv \frac{v_c - v_b}{v_d - v_b} \tag{15.23}$$

For any sub-critical temperature v_c is always somewhere between v_b and v_d so Q_c always lies between 0 and 1. As tabulated above the property value F is continuous across the bubble line ($F(0) = F_L(0)$) and also across the dew line ($F(1) = F_V(1)$) but is discontinuous at $Q = Q_c$ ($F_L(1) \neq F_V(0)$).[1]

TBSpline3Gases decode the separate liquid and vapor phase properties for any x in the range 0 to 1 as

$$F_L(x) = F(xQ_c) \tag{15.24}$$

$$F_V(x) = F(Q_c + [1 - Q_c]x) \tag{15.25}$$

Except that in the F_L evaluation x is truncated to no more than $(1-\epsilon)$, where ϵ is some small positive number, to avoid liquid/vapor property crosstalk when interpolating near the discontinuity at $x = 1$. In the F_V evaluation x is truncated to no less than ϵ for the same reason near the discontinuity at $x = 0$.[2] Currently $\epsilon = 0.05$ which requires a reasonably close spacing of the v_j data points around the critical density v_c (controlled by configuration constant VptsInMixed in the RefpropToSage utility below).

[1] As Q ranges from $0 \to Q_c$, Q/Q_c ranges from $0 \to 1$ and as Q ranges from $Q_c \to 1$, $(Q - Q_c)/(1 - Q_c)$ also ranges from $0 \to 1$.

[2] As x ranges from $0 \to 1$, xQ_c ranges from $0 \to Q_c$ and $Q_c + [1 - Q_c]x$ ranges from $Q_c \to 1$, compatible with the piecewise definition of F.

Chapter 16

Moving Parts

Moving parts represent reciprocating masses, motion-constrained masses, various types of springs and dampers, etc. Prior to Sage version 13 most moving-part components came in both phasor and time-ring varieties. As of version 13 the phasor versions have been deprecated. They can still be found in the *Deprecated* tab of the component palette but using phasor components is no longer recommended. Experience has shown that sooner or later you will want to connect one to a time-ring component and be forced to replace the component with a time-ring version. In the early days of Sage it seemed that there might be a time penalty for using time-grid components but this is now negligible.

Whether phasor or time-ring, the motion is assumed periodic in time. In the phasor case, motion and forces are assumed sinusoidal so they can be represented by their complex amplitudes (see chapter 8), while in the time-ring case, model variables are solved on a periodic time-grid — a so-called time ring. The first harmonic inputs or outputs of time-ring components are identical to complex amplitudes.

16.1 Boundary Connections

Moving parts communicate with each other in one of two ways. Either via forces acting on points of attachment or pressures acting on area faces. After model solution, two attachments points connected together will share the same motion, whereas two area faces connected together will share the same displaced volume, much as two pistons connected by an incompressible hydraulic circuit would. Connected faces need not have the same area. In either case there is a positive or negative orientation to the connection. Positive-facing connectors show up as labeled arrows on the right edge of model components. Negative-facing connectors show up on the left. Only opposite-facing connectors may be connected together. This is Sage's equivalent of Newton's law, which says (sort of): a force may only mate with an equal but opposite reaction. Although the real world admits forces directed arbitrarily in three-dimensional space,

SCFusion models confine forces to lie along a one-dimensional axis.

In the edit form, force connectors are arrows labeled F_{Gt}. Pressure connectors are arrows labeled P_{Gt}. You are only allowed to connect a force to a force and a pressure to a pressure.

16.2 Moving-Part Attachments

For inter-connection purposes moving parts require child model components representing points of attachment and area faces. Some moving parts have built-in connection child models. Other moving parts allow you to create as many as you need from the model-component palette, choosing among the following options:

Icon	Purpose
	time-ring negative-facing force attachment
	time-ring positive-facing force attachment
	time-ring negative-facing area attachment
	time-ring positive-facing area attachment

When you drop one of these into the edit form, it is born with a connection arrow which you can then move up one or more levels for connection there to another moving part.

Generally speaking you will want to create exactly one connector child component for each moving part you want to connect to. But you must not make duplicate connections between the same moving parts. Doing so will cause a singular equation system at solving time.

Using child model components to customize boundary connections is a typical Sage convention for all model classes. It allows you to create exactly as many connections as your model components require without presuming any minimum or maximum amount.

16.3 Moving-Part Variables

Moving parts also have a number of output variable found in all descendants:

F : (Fourier series, N) Net boundary force acting on the component resulting from all connections.

W : (Fourier series, W) Power delivered by all boundary forces. Net force times velocity. The mean value is equivalent to Wnet in the TPhsrMov model component.

16.4 Generic Spring

TGtSpr

Generic springs are moving-part descendants that introduce a new input variable:

K : (real, N/m) Spring stiffness.

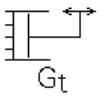
The mathematics behind the scene solves displacement x from net applied boundary force F and stiffness K using the equation

$$F = Kx \tag{16.1}$$

F and x are discrete real values on a grid.

You may think of Sage springs as having one end fixed to ground and the other subject to any number of attachment points or faces for connection to other model components — generally to reciprocating-mass components. Generic springs have no mass.

16.5 Generic Damper



TGtDmp

Generic dampers are moving part descendants that introduce a new input variable:

D : (real, N/(m/s)) Damping coefficient.

This time, the mathematics behind the scenes solves displacement x from net applied boundary force F and damping coefficient D using the equation

$$F = D\dot{x} \tag{16.2}$$

where $\dot{x}$ is velocity. Moving parts maintain a computational grid containing, not just displacement x, but velocity $\dot{x}$ and acceleration $\ddot{x}$ as well, as individual solution variables. In the grid these are labeled x, x_d and x_{dd}. The above equation determines x_d explicitly, time-differentiating x_d determines x_{dd} explicitly and the defining condition $\dot{x} = x_d$ determines x implicitly.

Like springs, you may think of Sage dampers as having one end fixed to ground and the other subject to any number of attachment points or faces for connection to other model components. You will usually be connecting them to reciprocating mass components. They, too, are massless.

16.6 Reciprocating Mass

TGtRcp

Reciprocating masses, or reciprocators for short, solve displacement x from the net resultant force F and mass M using Newton's equation of motion

$$F = M\ddot{x} \tag{16.3}$$

where $\ddot{x}$ is acceleration. F includes an internally-specified body forcing function in addition to applied boundary forces. Solving for x in the grid of solution variables x, x_d and x_{dd} goes like this: The above equation determines x_{dd} explicitly, the defining condition $\dot{x_d} = x_{dd}$ determines x_d implicitly and defining condition $\dot{x} = x_d$ determines x implicitly.

Think of Sage reciprocators as masses that move back and forth according to Newton's law of motion. Mean velocity is zero as a requirement of periodic motion, but mean position may be nonzero, depending on possible position-dependent boundary forces or pressures.

Reciprocator introduce new variables:

Mass : (real, kg) Reciprocating mass.

FF : (Fourier series, N) Forcing function.

FX : (Fourier series, m) Displacement from mean position.

You can optimize any of the drivable real-parts of the Fourier-series forcing function if you choose (*see* chapter 10).

16.7 Constrained Piston

TGtPis

This model components descend from the reciprocator components previously discussed. Except it switches the role of forcing function and displacement, so that now displacement is the input and required forcing function, to achieve that displacement, is the solved quantity. Generally, you will only want to connect a constrained motion reciprocator to model components like springs and dampers that can accommodate the irresistible displacement provided by these components. You should never directly connect together two constrained motion reciprocators.

Mathematically, a constrained piston solves required forcing function F_f from net applied boundary force F_b, displacement x and mass M, using Newton's equation of motion

$$F_f = M\ddot{x} - F_b \tag{16.4}$$

where $\ddot{x}$ is acceleration.

The constrained piston introduces new variables:

FX : (Fourier series, m) Displacement from mean position.

FF : (Fourier series, N) Required forcing function.

You can optimize any of the drivable real-parts of displacement if you choose (*see* chapter 10).

16.8 Relative Moving Parts

The preceding springs and dampers are referenced to ground (fixed inertial frame) at the end opposite the connection. So, for example, there is no way to connect one between two reciprocating masses. Relative springs and dampers solve this problem. Relative springs and dampers descend from the common abstract class of relative moving parts.

Relative moving parts contain two position coordinates instead of one — the coordinates of the positive and negative boundaries. Before, all connections were thought of as made to the same boundary, which was represented by a single coordinate. Now, positive connections are thought of as made to the positive boundary and negative connections to the negative boundary.

Relative moving parts are born with exactly two force connectors, with no provision for additional connectors. You may connect these built-in connectors to opposite force connectors in any of the previous moving parts (such as reciprocating masses). In any event, you must connect them to something, lest a singular solution result.

All relative moving parts have the variables:

Fneg : (Fourier series, N) Force acting on the component negative boundary.

Fpos : (Fourier series, N) Force acting on the component positive boundary. Fpos is equal and opposite Fneg for each component of the Fourier series.

W : (Fourier series, W) Power delivered by the boundary forces.

Sage assumes all relative moving parts are massless. As such, the dynamic force balance may be written in terms of the negative and positive boundary forces as

$$F_{neg} + F_{pos} = 0 \tag{16.5}$$

Sage uses this condition to implicitly solve the negative coordinate x_{neg} for all relative moving parts.

16.9 Relative Springs

Relative springs, just like generic (absolute) springs, are massless springs defined in terms of the input variable:

TGtRelSpr

K : (real, N/m) Spring stiffness.

Sage solves the negative coordinate x_{neg} as above and the positive coordinate x_{pos} from the applied boundary force F_{pos} and stiffness K using the equation

$$F_{pos} = K(x_{pos} - x_{neg}) \tag{16.6}$$

16.10 Relative Dampers

TGtRelDmp

Relative dampers, just like generic (absolute) dampers, are massless dampers defined in terms of the input variable:

D : (real, N/(m/s)) Damping coefficient.

This time, Sage solves the positive coordinate x_{pos} from the applied boundary force F_{pos} and damping coefficient D using the equation

$$F_{pos} = D(\dot{x}_{pos} - \dot{x}_{neg}) \tag{16.7}$$

16.11 Rotary Mechanisms

The components in this section model the various rotary mechanisms often used to drive the pistons and displacers of stirling-cycle machines. There are actually two types of rotary components, one representing various kinematic linkages and the other representing the flywheel.

16.11.1 Flywheel

TGtFlywheel

A flywheel is used to drive one or more of the kinematic linkage components below. Its child-model palette allows you to create any number of steady-torque attachments, each born with a positive-facing torque connector designed to attach to the negative-facing torque connector of a kinematic linkage. In physical terms a flywheel behaves like a constant torque drive or motor with angular momentum. It supplies exactly the torque required to balance the rotational energy input or output from any torque connections while its rotation speed fluctuates in response to those torque connections. The input and outputs for a flywheel are:

IMoment : (real, kg m^2) Moment of inertia I, including the moments of inertia of any rotating parts in attached kinematic linkages. For a disk flywheel of uniform thickness $I = 1/2MR^2$, where M is the mass and R is the disk radius. For a flywheel where all of the mass is concentrated in the outer rim $I = MR^2$.

FTorque : (Fourier series, N m) Total Torque T applied to the flywheel by all torque connections as a function of time.

FOmega : (Fourier series, radians/s) Rotation angular velocity $\dot{\theta}$ as a function of time.

FW : (Fourier series, W) Power delivered to the flywheel from all torque connections.

Rotational Realism The rotational angular velocity $\dot{\theta}$ fluctuates according to the applied torque and moment of inertia I, which is an input. Except that the mean value of angular velocity is constrained so that the rotational period is always consistent with the root model Frequency input. In other words the crank mechanism always rotates Frequency times per second although the rotational speed fluctuates somewhat as it does in an actual machine.

If the flywheel moment of inertia (input Imoment) is too small the flywheel rotation may stall as in an actual machine. To avoid numerical problems during design work it is best to start with a high moment of inertia and reduce it later on if necessary.

Torque Attachments There is only one type of torque-attachment available within the flywheel component:

Icon	Purpose
	time-ring positive-facing torque attachment

Create as many as you need. Then move the connection arrows up one level and connect them to the kinematic linkages in your model.

16.11.2 Kinematic Linkages

Kinematic linkages implement the math required to convert a rotary motion into a back and forth reciprocating motion. The rotary motion comes from attachment to a flywheel component via a torque connection. The torque connection locks together the rotation of the flywheel and linkage. The back and forth motion is coupled to one or more reciprocating masses that model such things as the piston of a compressor model or piston and displacer of a stirling-cycle model. Attaching a kinematic linkage to a reciprocating mass with a force connection cause the kinematic linkage to *drive* the reciprocating mass according to the linear motion of its specific linkage geometry.

Inputs and outputs common to all kinematic linkages are:

Rcrank : (real, m) Crank-throw radius r.

Phase : (real, radians) Initial crank-angle ρ at time zero. Time zero is when the attached flywheel rotation angle θ is zero.

FTorque : (Fourier series, N m) Torque T applied by the linkage geometry to the crank pin as a function of time.

Mandatory Flywheel Connection All kinematic linkages contain built-in negative-facing torque connectors for mandatory connection to a flywheel component (section 16.11.1) which defines the rotational angular momentum coupled to the mechanism. Without a flywheel connection a kinematic linkage will be unable to balance the torque on the crank pin produced by the linear

forces acting on the linkage geometry and the solve process will fail. The angular momentum is defined in a separate component so more than one mechanism can share the same flywheel and combine together to produce a single rotation angle solution. In the physical world this simulates having two or more kinematic linkages geared together and coupled to the same flywheel.

Reciprocating Motion All kinematic linkage components produce a linkage linear motion x as a function of crank angle and define the torque T applied to the crank pin as a function of the applied force to the reciprocating linkage. Crank angle is the flywheel rotational angle θ plus the phase offset ρ provided by input Phase. When calculating torque the reciprocating mass of the linkage and its rotational moment of inertia are presumed to be zero. Any actual reciprocating mass must be included in the attached reciprocating mass and any rotational moment of inertia in the attached flywheel.

Beware Aliasing Errors Higher harmonics in the reciprocating linkage motion can produce significant solution errors if the time grid is too coarse — e.g. root-model input NTnode is too small — and even lead to energy conservation errors in attached reciprocator or spring components if NTnode is even.

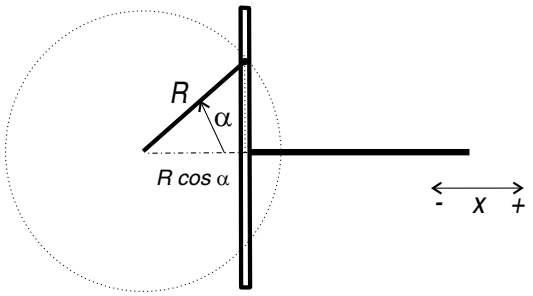
Higher harmonics are produced by the nonlinear relationship between crank rotation and linkage reciprocating motion. These harmonics are compounded if the rotational angular velocity is itself fluctuating because the flywheel moment of inertia (input Imoment) is small. If the highest harmonic resolved by the grid is significant (e.g. amplitude of highest harmonic in FX output not small compared to other harmonics) then you might want to increase NTnode to improve solution accuracy. You should also increase it to an odd number because NTnode must be odd to resolve the mean value plus both components of all harmonics. When NTnode is even the highest harmonic is not fully determined. This leads to finite differencing errors in the time grids for reciprocators and springs which may produce time-average energy dissipation or production in violation of physical principles. When NTnode is odd there are no finite differencing errors and energy conservation is numerically exact, although there can still be solution errors if NTnode is too small.

Failure to conserve energy in springs and reciprocators is especially bad because it can mislead you about the power delivered to the kinematic linkage (output W). For example the PV power delivered by the working gas to a reciprocator driven by a kinematic linkage will show up partly in the reciprocator and partly in the kinematic linkage if there is an energy conservation error.

Scotch Yoke Linkage

The drawing below shows the kinematic linkage geometry for a Scotch yoke:

TGtYokeLnk



The linkage linear displacement is just the x-component of crank pin motion

$$x = R\cos\alpha \tag{16.8}$$

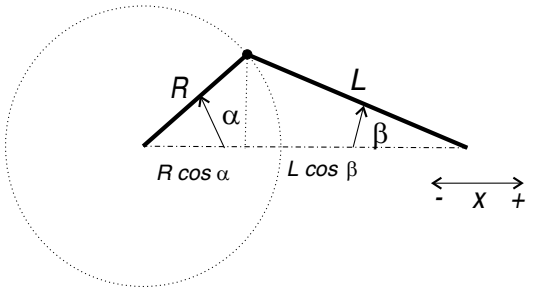
Simple Crank Linkage

A simple crank mechanism adds the input:

TGtCrankLnk

Lratio : (real, dimensionless) Connecting-rod length divided by crank-throw radius L/R. The linkage geometry requires that $L/R > 1$ ($L > R$).

The kinematic linkage geometry is shown in this drawing:



The linkage linear displacement is the sum of the x-components of crank pin motion and connecting rod displacement. Referring to the above drawing and subtracting the constant length L the linkage displacement may be written

$$x = R\cos\alpha + L\cos\beta - L \tag{16.9}$$

where angle β is:

$$\beta = \arcsin\left(\frac{R\sin\alpha}{L}\right) \tag{16.10}$$

The reason for subtracting L is so the linear displacement ranges above and below zero according to Sage's convention for moving parts. The extreme displacements are $x = R$ and $-R$ at $\alpha = 0$ and π. The displacements at $\alpha = \pi/2$ and $3\pi/2$ are both negative, not zero which has the implication that the time-average displacement is also negative. Time-average displacement is zero only in the limit of infinite connecting rod length L.

Rhombic Drive Linkage

TGtRhombic-Lnk

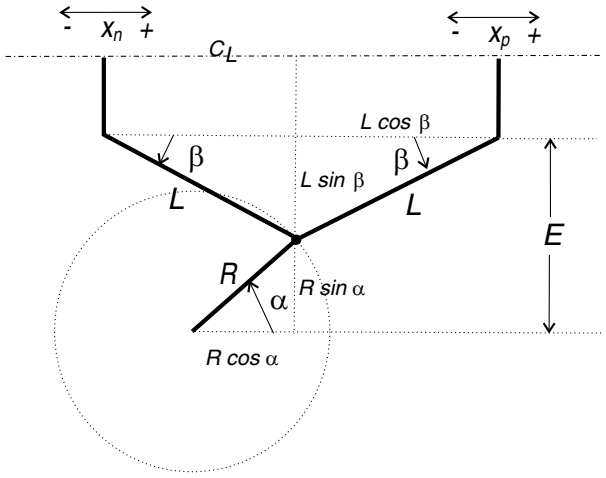
A rhombic drive mechanism has a different pedigree from previous rotating mechanisms. It descends from the relative moving parts documented above. Because of that a rhombic drive mechanism is born with positive and negative facing force connectors and does not support connections via child-model force attachments like previous rotating mechanisms.

In the physical world a rhombic drive has two counter-rotating crankshafts and four equal-length connecting rods, connected in pairs to the two crank pins (*see* drawing below). The connecting rods produce two linkage motions x_p and x_n. The Sage convention is that the x_p motion is transmitted through the positive facing force connection and the x_n motion through the negative facing force connection. The rhombic mechanism adds two inputs:

Eratio : (real, dimensionless) Rhombic eccentricity divided by crank-throw radius E/R (*see* drawing). Mathematics requires only that E/R be non-negative ($E/R \geq 0$). The drawing shows the typical case $E/R > 1$. The case $E/R = 0$ corresponds to the simple crank piston.

Lratio : (real, dimensionless) Connecting-rod length divided by crank-throw radius L/R. The linkage geometry requires that $L/R > E/R + 1$ ($L > E + R$).

The drawing below shows only the kinematic linkage geometry and applied forces for half of the mechanism, cut along the center-line (plane of symmetry) at the top of the drawing:



The linkage linear displacements are the sum of the x-component of crank pin motion plus-or-minus the connecting rod displacement. Referring to the above drawing and including a constant offset x_0 the linkage displacements may be

written

$$x_p = R\cos\alpha + L\cos\beta - x_0 \tag{16.11}$$
$$x_n = R\cos\alpha - L\cos\beta + x_0 \tag{16.12}$$

where angle β is:

$$\beta = \arcsin\left(\frac{E - R\sin\alpha}{L}\right) \tag{16.13}$$

The purpose for x_0 is so the linear displacements range equally above and below zero according to Sage's convention for moving parts. In terms of the above drawing x_0 is the average of the max and min horizontal displacements between x_p (or x_n) and the crank circle center. These max and min displacements occur when the connecting rod L is directly in-line and directly opposite the connecting rod radius R. The drawing shows the crank angle near the time of maximum x_p displacement $\sqrt{(L+R)^2 - E^2}$, which is the horizontal leg of the right triangle with hypotenuse $L+R$ and vertical leg E. Similarly the minimum x_p displacement is $\sqrt{(L-R)^2 - E^2}$. The average displacement is one-half the sum, or

$$x_0 = \frac{1}{2}\left(\sqrt{(L+R)^2 - (E)^2} + \sqrt{(L-R)^2 - (E)^2}\right) \tag{16.14}$$

16.11.3 Rotary Mechanism Theory

The physics of rotary mechanisms is relatively simple with the roles of kinematics and rotational inertia handled by distinct model components. Kinematic linkage components handle the kinematic part. They derive from the common ancestor of all time-grid moving-parts. The inherited displacement variable x represents the linear back and forth motion of the linkage solved as an explicit function of crank angle. More on crank angle later. The flywheel component handles the rotational inertia part. It implements a time grid with three state variables θ, θ_d and θ_{dd}, corresponding to the rotation angle as a function of time, angular velocity (first time derivative $\dot{\theta}$) and angular acceleration (second time derivative $\ddot{\theta}$).

Flywheel Angular Acceleration

Flywheels solve state variable θ_{dd} (angular acceleration) explicitly from the rotary equivalent of Newton's equation of motion written in the form:

$$\theta_{dd} = \frac{T_E + T_s}{I} \tag{16.15}$$

where moment of inertia I is an input and T_E is the net applied torque by external torque connections. Indirectly T_E comes from the forces acting on the reciprocating mass (or masses) attached to the kinematic linkages that are attached to the flywheel. In addition to inertial forces the reciprocating mass

transmits whatever other forces might be attached to it (e.g. pressure forces acting on piston frontal areas). T_S is the internal torque applied to the flywheel required to keep it rotating at an average angular velocity of ω (root model variable Omega).

T_S is an invisible implicit variable needed for the mathematical solution. Its purpose is to offset any time-average of the external applied torque T_E that would otherwise tend to increase or decrease the rotation speed with time. As if the output shaft is internally connected to a constant torque motor or alternator that regulates the speed. T_S is not a grid state variable but rather solved as a single implicit variable of the flywheel component itself. While the solved value of T_S is not displayed as an output variable its valued can be inferred as the negative of the mean value of the FTorque Fourier series output. More on the solution of T_S below.

Flywheel Angular Velocity

Flywheels solve state variable θ_d (angular velocity) implicitly from

$$d\theta_d/dt - \theta_{dd} = 0 \tag{16.16}$$

In other words, the time derivative of solution variable θ_d equals the solution variable θ_{dd}.

Flywheel Crank Angle

In the limit of infinite moment of inertia the flywheel rotation angle approaches $\theta = \omega t$. But because the moment of inertia is finite flywheels must solve crank angle implicitly from

$$d\theta/dt - \theta_d = 0 \tag{16.17}$$

That is, the time derivative of solution variable θ equals the solution variable θ_d. There is however a complication. Evaluating the time derivative $d\theta/dt$ in equation (16.18) does not work in Sage because $\theta(t)$ is not a periodic function. It generally increases monotonically with time so there is always a discontinuity of 2π somewhere in the grid which Sage's time differencing operator fails to deal with. The solution to this problem is to decompose $\theta(t)$ into the sum of a uniform part ωt and a fluctuating part $\tilde{\theta}$ and replace the previous equation with

$$d\tilde{\theta}/dt + \omega - \theta_d = 0 \tag{16.18}$$

Fluctuating part $\tilde{\theta}$ *is* a continuous periodic function because it results from the from higher harmonics in the angular velocity. So it causes no problems with time differencing. Sage does not actually maintain $\tilde{\theta}$ as a grid state variable but rather just evaluates the derivative $d\tilde{\theta}/dt$ as $d(\theta - \omega t)/dt$.

Implications

Satisfying equation (16.18) at all time nodes implies that the mean value of θ_d (angular velocity) is ω, as required for consistency of the root model input

Frequency. This follows by noting that the time-mean of the entire left-hand side of equation (16.18) is zero and also that the time-mean of $d\tilde{\theta}/dt$, or any time derivative for that matter, is necessarily zero in the periodic solution grid employed by Sage.

For similar reasons, satisfying equation (16.16) at all time nodes implies that the mean value of θ_{dd} is zero, which already determines implicit slack variable T_S discussed above. What then to use for the implicit function associated with T_S in the solution scheme? Sage uses the only other physical condition not imposed by the above solution method, namely that

$$\theta(0) = 0 \tag{16.19}$$

So solving T_S enforces the required initial crank angle while the rest of the solution ensures that T_S is the value required to make the time-average of θ_{dd} zero.

Kinematic Linkage Crank Angle

Kinematic linkages augment the displacement time grid inherited from their moving part ancestors (state variables x, x_d, x_{dd}) with state variables θ and θ_d, corresponding to rotation angle and angular velocity. By virtue of its torque connection, θ and θ_d in a kinematic linkage are the same as for the flywheel to which it is attached. But there needs to be a way to force the torque imposed by the flywheel connection to balance the torque produced by the reciprocating mass connection to the kinematic linkage. Kinematic linkages do that by solving θ implicitly with the torque balance equation as the implicit function.

$$T_R + T_L = 0 \tag{16.20}$$

where T_R is the torque supplied by the flywheel via the torque connection and T_L is the torque applied to the crank pin via force connections to reciprocating masses.

The kinematic linkage crank angle α is offset from the flywheel rotation angle θ by a fixed phase angle offset ρ (input Phase)

$$\alpha = \theta + \rho \tag{16.21}$$

The phase offset ρ allows the crank angle to differ from the flywheel rotation angle so two or more mechanisms connected to the same flywheel can have different phases.

Kinematic Linkage Torque

Kinematic linkages calculate the torque T applied to crank pin from the energy conservation principle

$$T\dot{\theta} = F\dot{x} \tag{16.22}$$

where $\dot{\theta}$ is state variable θ_d, F is the sum of applied boundary forces and $\dot{x}$ is state variable x_d. In English, the rate of rotational work done by the torque

acting on the crank pin equals the rate of work done by the applied forces acting on the reciprocating linkages. This principle holds because there is no mass or moment of inertia in a kinematic linkage to store energy.

This way is much simpler than calculating torque directly as a function of applied boundary force and crank angle (using force-balance diagrams and trigonometry). It also guarantees that global energy conservation holds. Any mechanical energy that enters or leaves through the reciprocating linkage necessarily equals the mechanical energy transmitted to the flywheel.

16.12 Nonlinear Spring

TGtSnl

A nonlinear spring is a generic spring where the spring stiffness K varies as a quadratic function of position x. Most actual springs are nonlinear to some extent, depending on how far extended from their rest position $x = 0$. This component allows you to model the effects of operating a spring beyond its linear range. The stiffness function is defined by four inputs:

K0 : (real, n/m) Stiffness K_0 at $x = 0$.

Xm : (real, m) Reference extension x_m.

Rp : (real, dimensionless) Stiffness ratio $R_p = K/K_0$ at $x = x_m$.

Rn : (real, dimensionless) Stiffness ratio $R_n = K/K_0$ at $x = -x_m$.

You can read these, more-or-less directly, from a plot of local spring stiffness K vs x generated either experimentally or computationally. In terms of the restoring force $F(x)$, the spring stiffness is $K(x) = -dF/dx$. A more precise approach would involve finding the best-fit parabola (not necessarily symmetric about the rest position) to the function $K(x)$ over the intended operating range and then defining K0, Xm, Rp, Rn consistent with the parabola.

For internal processing, Sage translates the above inputs into a quadratic spring stiffness in the form

$$K(x) = K_0 \left(1 + a(x/x_m) + b(x/x_m)^2\right) \tag{16.23}$$

Coefficients a and b come from substituting x_m and $-x_m$ for x in equation (16.23), which results in the equation system

$$R_p = 1 + a + b \tag{16.24}$$
$$R_n = 1 - a + b \tag{16.25}$$

with the solution

$$a = (R_p - R_n)/2 \tag{16.26}$$
$$b = (R_p + R_n)/2 - 1 \tag{16.27}$$

The restoring force provided by a nonlinear spring is

$$F(x) = -K_0 x \left(1 + \frac{a}{2}(x/x_m) + \frac{b}{3}(x/x_m)^2\right) \tag{16.28}$$

The position $x = 0$ is the neutral force position.

You can attach a nonlinear spring to a reciprocating mass for modeling the resultant motion of nonlinear spring-mass systems. Even when driven by a sinusoidal forcing function, such a system will produce a reciprocating motion with higher harmonics. And the resonant frequency will change, depending on amplitude.

16.13 Relative Nonlinear Spring

TGtRelSnl

A relative nonlinear spring is just like the above nonlinear spring, except it is connected between two moving parts instead of between a moving part and ground (fixed inertial frame). The same input variables and governing equations apply except that position coordinate x is replaced by the difference of the endpoint coordinates $x_{pos} - x_{neg}$.

16.14 Interpolated Spring

TSprFx

A descendant of the nonlinear spring component that interpolates spring force $F(x)$ directly from a cubic-spline input:

Fx : (cubic spline, dimensionless) A set of data pairs (x_i, F_i), where x_i is a position relative to the spring zero position in meters (or other units selected in the model-class options dialog) and F_i is the restoring force at that position. A restoring force means that F_i has the opposite sign as x_i. The x_i values must be listed in increasing order from the largest negative position expected during operation at the top to the largest positive position at the bottom. The number of data pairs is arbitrary and will depend on the complexity of the variation of spring force with position. The input dialog for specifying the (x_i, F_i) pairs has a View Interpolation button you can click to see the smooth curve resulting from cubic spline interpolation. It is possible to paste values from a spreadsheet directly into the pairs input dialog.

This component shares the same output variables as its ancestor nonlinear spring component. It deprecates the K0, Xm, Rp and Rn inputs because of the new formulation.

It is up to you to specify Fx according to measurements of your particular spring. If the $F(x)$ data is too nonlinear or asymmetric may have to increase the value of NTnode in the root model (number of time nodes) in order to ensure that the computational grid can resolve the $F(x)$ curve. You can gauge this by plotting the spring connector force vs time curve in the resulting model solution.

You should also keep an eye on the time-average spring power inflow (output W.Mean), which should be zero or close. Theoretically, for a continuum spring the cyclic spring work is the integral $\oint F dx$ (i.e. area under a $(x, F(x))$ curve), which must be zero because x returns to its original value over an equilibrium cycle. Because of aliasing errors it may happen when $F(x)$ is not a low-order polynomial or the resulting motion $x(t)$ is not very sinusoidal that W.Mean is non-zero. In other words the spring dissipates or creates power as a numerical artifact. Generally speaking the size of this power artifact should decrease with increasing time nodes.

16.15 Relative Interpolated Spring

TRelSprFx

A relative interpolated spring is just like the above interpolated spring, except it is connected between two moving parts instead of between a moving part and ground (fixed inertial frame). The same input variables and governing equations apply except that position coordinate x is replaced by the difference of the endpoint coordinates $x_{pos} - x_{neg}$.

16.16 Stick-Slip Damper

TGtFricDmp

A stick-slip damper approximates the frictional drag of a sliding object moving over a surface. It imposes a nearly constant drag force F_D that opposes the direction of motion. The force changes sign when the sliding velocity changes sign and is zero when the velocity is zero. In the usual engineering approximation, F_D would be the product of a coefficient of friction, depending on materials involved, and the side-force pushing the two materials together (e.g. weight or magnetic attraction). This component knows nothing about either coefficient of friction or side force. It only represents the product of the two as input F_D. The name stick-slip comes from the fact that the sliding object will not move if pushed with a force less than F_D, then suddenly begin to move when the force exceeds F_D. There are two inputs for this component:

Fdrag : (real, n) Stick-slip drag force F_D.

Vrel : (real, m/s) Reference velocity v_r. This is not a critical input but should be on the order of the sliding velocity expected in the model, for reasons explained below.

To improve solution convergence the boundary force does not change discontinuously from $-F_D$ to F_D when the component velocity $\dot{x}$ changes sign, but rather changes at a fast but linear rate between velocities $-\epsilon v_r$ and ϵv_r, where ϵ is a small number and v_r is the reference velocity input, as illustrated in figure 16.1. For clarity, figure 16.1 shows the case $\epsilon = 0.1$. The value $\epsilon = 0.001$ is hard-wired into Sage. Mathematically the stick-slip force F for any positive

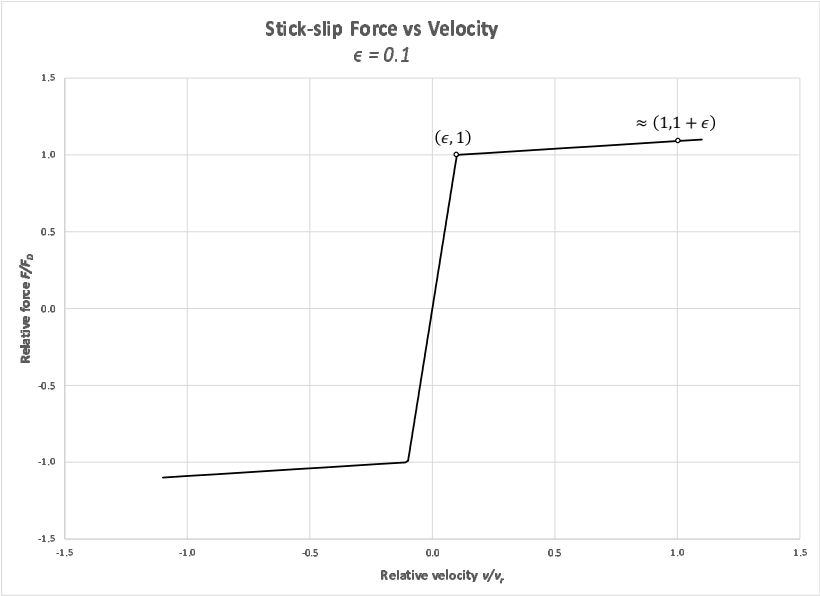


Figure 16.1: Stick-slip frictional drag as a function of velocity, as implemented by Sage. As the value of ϵ decreases the force approximates a step function. Sage uses a value $\epsilon = 0.001$.

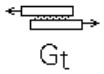
velocity $\dot{x}$ is

$$\frac{F}{F_D} = \begin{cases} \frac{1}{\epsilon}\frac{\dot{x}}{v_r} & \text{if } \frac{\dot{x}}{v_r} <= \epsilon \\ 1 + \epsilon\left(\frac{\dot{x}}{v_r} - \epsilon\right) & \text{if } \frac{\dot{x}}{v_r} > \epsilon \end{cases} \tag{16.29}$$

The force takes the value F_D at the transition velocity $\dot{x} = \epsilon v_r$ and rises to approximately $(1+\epsilon)F_D$ at the reference velocity $\dot{x} = v_r$. For a negative velocity the force is flipped. The reason why force does not remain constant above the transition velocity is because of the way Sage implements the solution. Instead of calculating force from velocity Sage actually does it the other way around, calculating the velocity $\dot{x}$ from the applied boundary force, which may be outside the range $(-F_D, F_D)$ as the solution converges. In the above formulation each value of boundary force corresponds to a unique velocity.

16.17 Relative Stick-Slip Damper

A relative stick-slip damper is just like the above stick-slip damper, except it is connected between two moving parts instead of between a moving part



TGtRelFricDmp

and ground (fixed inertial frame). The same input variables and governing equations apply except that position coordinate x is replaced by the difference of the endpoint coordinates $x_{pos} - x_{neg}$. The relative nonlinear spring descends from the relative moving parts documented above.

16.18 Motion Snubber

TGtSnubber

Something between a nonlinear spring and damper, a motion snubber imposes displacement limits on an attached reciprocating mass by generating a force that approximates an energy dissipating collision when the motion exceeds those limits. Otherwise this component produces essentially zero force on whatever it may be attached to. Simulating a collision is not an easy task for Sage because actual collision forces tend to be abrupt and dependent on the elasticity and restitution coefficient of the materials at the impact point. This component on the other hand produces a collision force that is smooth enough to resolve within Sage's coarse time-grid and distributed over the entire solution period. Because collision forces are inherently non-sinusoidal, this component only comes in a time-ring version and supports only time-ring force connections (F_{Gt}) to time-ring moving parts.

Because of the non-sinusoidal motions likely to result when using this component you may have to increase the number of time nodes in the solution grid above the default NTnode = 7 (NTnode is a root component input). You may need NTnode = 9, 11 or even higher to adequately resolve the motion. Experiment! When using this component it is a good idea inspect the actual solution grid of the snubbed component (menu: File|Save Solution Grid) because it is difficult to visualize the solved motion from the Fourier series output FX.

The motion snubber has five inputs and one output:

Mscale : (real, kg) snubbed mass scale. The approximate mass m_s of the reciprocating mass or combined reciprocating masses the snubber is connected to. This input largely determines the magnitude of the collision force. A force sufficient to stop a heavy inertial mass may be too high for a light inertial mass and generate unrealistic solution artifacts. (*see* theory below)

Xlimit : (real, m) Displacement limit x_ℓ. For motion $\boldsymbol{x}$ between $-x_\ell$ and x_ℓ collision forces are essentially zero. For $\boldsymbol{x}$ beyond the motion limits the collision force transitions smoothly over a short distance from zero to the maximum value. So the displacement limit is not a hard value and you can expect some motion beyond the limits in the final solution.

Sp : (real, dimensionless) Fraction of the baseline snubbing force implemented at $x = x_\ell$. Set to zero to eliminate collision force at positive limit.

Sn : (real, dimensionless) Fraction of the baseline snubbing force implemented at $x = -x_\ell$. Set to zero to eliminate collision force at negative limit.

Kfrac : (real, dimensionless) Spring content fraction. Determines ratio of *spring* forces to dissipation forces (F_k/F_s below) as a means to tune the resulting motion.

FX : (real, m) Displacement Fourier series output.

You can *tune* the collision force by adjusting Mscale, Kfrac and to some extent Sp and Sn. The magnitude of the collision force scales directly with the product of Mscale and Sp or Sn. The phase of the resulting motion depends somewhat on Kfrac.

Collision Theory Imagine a reciprocating mass m_s moving with sinusoidal motion $\boldsymbol{x} = \cos(\omega t)$. There is a smooth motion reversal at the positive and negative peak values which is the result of a sinusoidal applied force. If instead there is an abrupt collision force just before the peak values the motion will now contain higher harmonics — mainly odd harmonics because only they are phased correctly to oppose both positive and negative peaks of the fundamental motion. The extreme example of collision motion is a square wave for which the Fourier series decomposition is

$$x = \cos(\omega t) - \frac{1}{3}\cos(3\omega t) + \frac{1}{5}\cos(5\omega t) + \ldots \tag{16.30}$$

To produce motion with odd harmonics requires imposing a snubbing force with odd harmonics, which is the motivation for implementing a collision force as the product of a nonlinear spring and damper in the form

$$F_s = -S_b x^2 \dot{x} \tag{16.31}$$

S_b is a baseline snubbing coefficient to be determined. For $\boldsymbol{x}$ sinusoidal or nearly so F_s has high third harmonic content as shown in Figure 16.2. One might define a critical snubbing coefficient S_c as the coefficient that dissipates the peak kinetic energy of the moving mass $m_s v^2/2$ over half the cycle period. If the motion is $\boldsymbol{x} = x_\ell \cos(\omega t)$ the peak kinetic energy is $m_s {x_\ell}^2 \omega^2/2$. For that same motion the average power dissipated by the snubbing force is the time-average of $S_c(\boldsymbol{x}\dot{\boldsymbol{x}})^2$ which evaluates to $S_c {x_\ell}^4 \omega^2/4$, so the energy dissipated over half the cycle period π/ω is $S_c {x_\ell}^4 \omega\pi/4$. Equating the two dissipations and solving gives

$$S_c = \frac{2}{\pi}\frac{m_s \omega}{{x_\ell}^2} \tag{16.32}$$

The baseline snubbing coefficient S_b used in Sage's motion snubber component is somewhat higher than S_c, based on numerical experiments of actual collision simulations.

The above snubbing force is the basis for Sage's motion snubber component but there are also some practical refinements. First of all the snubbing force is not active unless the motion actually exceeds the allowable limits. Beyond the limits there is a smooth transition to full force rather than an abrupt change.

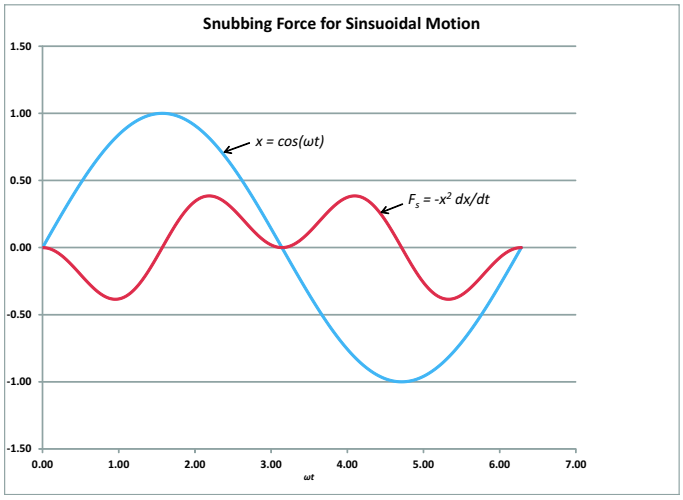


Figure 16.2: Sinusoidal piston motion $\boldsymbol{x}$ and snubbing force produced by function (16.31) with unit snubbing coefficient. The force for the positive-displacement half cycle cancels that of the negative half cycle, so the time average force and net *spring* content is zero. But the force shifts with velocity so it dissipates energy.

Abrupt changes are bad because they may cause solution instability. The force transition is implemented in terms of a weight factor W that multiplies the force. The weight factor depends on the instantaneous displacement $\boldsymbol{x}$ and cyclic peak values $x_m \pm x_1$ where x_m is the time-average displacement and x_1 is the amplitude of the first harmonic in its Fourier series expansion. The weight factor is near zero when the cyclic peak values are within limits, although not exactly zero to avoid solution indeterminacy. When instantaneous position is positive and its Fourier-series peak value exceeds the limit, the weight function increases smoothly but rapidly to the value of input Sp. When the instantaneous position is negative and its Fourier-series negative peak value exceeds the negative limit, the weight function increases smoothly but rapidly to the value of input Sn. In this way the force can simulate a single-ended collision if Sp or Sn are zero.

It is also useful to provide some degree of *spring* component to the snubbing force to prevent the motion from drifting off center if there are no external springs attached to the snubbed reciprocator and also to provide a means to tune the resulting motion so it is more realistic. So the actual force produced by the motion snubber component is the sum of a dissipative component F_s and a spring component F_k of the form

$$F = -W\left(F_s + F_k\right) = -W\left(S_b x^2 \dot{x} + K_b x\right) \tag{16.33}$$

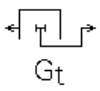
The motion snubber component does not specify K_b as an input but rather calculates it from input Kfrac, which is the ratio F_k/F_s. Ignoring fluctuating

quantities this is the ratio of spring force ($K_b x_\ell$) to snubbing force ($S_b \omega {x_\ell}^3$) or

$$\frac{F_k}{F_s} = \frac{K_b}{\omega {x_b}^2 S_b} \tag{16.34}$$

Solving for K_b, the spring stiffness is Kfrac times $\omega {x_b}^2 S_b$.

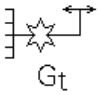
16.19 Relative Motion Snubber



TGtRelSnubber

A relative motion snubber is just like the above motion snubber, except it is connected between two moving parts instead of between a moving part and ground (fixed inertial frame). The same input variables and governing equations apply except that position coordinate $\boldsymbol{x}$ is replaced by the difference of the endpoint coordinates $\boldsymbol{x}_{pos} - \boldsymbol{x}_{neg}$.

16.20 Free Driver



TGtDrv

A free driver is similar to a time-ring generic damper except it provides a *negative* damping effect so as to simulate a driver that adds, rather than absorbs, mechanical power to whatever it is attached to. It provides a force in proportion to velocity. You might think of it as providing something like the force of a free piston in a free-piston stirling engine except without all the physics.

Like a damper, this component does not determine the phase of the force but rather adjusts to the phase of the moving object to which it is attached. The phase of that moving object must be determined by some other component of the overall model which provides a controlling phase input. So you might use a free driver to model a load, such as a linear alternator, subject to a control voltage or current that specifies the reference phase angle. The driving force depends on the input variable:

D : (real, N/(m/s)) Driving coefficient.

Sage solves displacement x from net applied boundary force F and driving coefficient D using the equation

$$F = -D\dot{x} \tag{16.35}$$

where $\dot{x}$ is velocity. Under Sage force sign conventions this always produces an equal and opposite force on the attached object that provides mechanical power input.

A free driver has one end fixed to ground and the other subject to any number of attachment points or faces for connection to other model components.

Chapter 17

Thermal Solids

Thermal solids represent the solid heat flow pathways found in SCFusion machines. Some are designed to be connected to the gas model components within heat exchangers. Some are designed to be connected only to other thermal solid components. The boundary connections to thermal solids may represent point contacts or line (surface) contacts carrying steady or time-varying heat flows. Some thermal solids are born with their boundary connectors while others get them via palette-created child model components. Ultimately your job is to connect the thermal solids in your SCFusion machine to form pathways from the working gas to a number of isothermal temperature sources or sinks.

This chapter provides detailed documentation for individual components. For usage suggestions and examples in specific model settings see the sample models in the Apps\SCFusion\Samples sub-directory under the installation directory.

17.1 Attachment Child Components

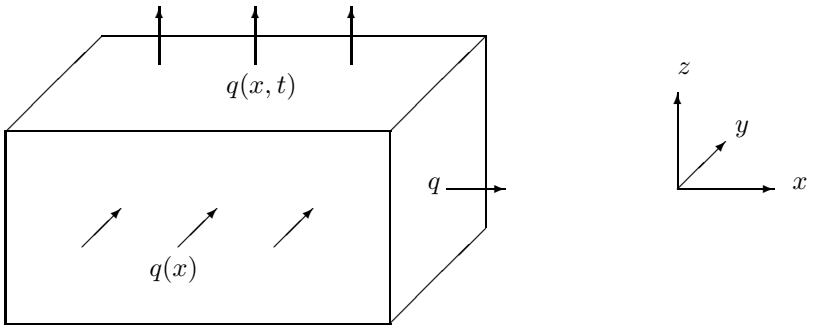
There are several types of thermal solids, with mathematical representations involving anything from single points, to space grids, to space-time grids. Some of these are born with appropriate heat-flow connectors, intended for connection to a gas component or another thermal solid component. Others have optional connections depending on which of the following child-model components you drag and drop into your edit form:

Icon	Purpose
	steady negative heat-flow end
	steady positive heat-flow end
	space-grid negative heat-flow face
	space-grid positive heat-flow face
	space-grid negative heat-flow end
	space-grid positive heat-flow end

When you drop one of these into the edit form, it is born with a connection arrow which you can then move up one level to the parent-model page for connection there to a mate, perhaps originating in another thermal solid component. The type of heat-flow connector arrow you get depends on the individual child component you create. You should have no trouble sorting out which connectors get connected where from the context of your problem and subsequent documentation.

17.2 Coordinate Conventions

All of the thermal solids attempt to adhere to a coordinate convention, both in their underlying model and their icon representation. A typical thermal solid domain is a rectangular solid that looks like this:



Generally speaking, the space-time varying $q(x,t)$ heat flow on the upper z face (with unit normal $\boldsymbol{z}$) is reserved for connection to a gas domain, the steady but spatially-distributed $q(x)$ heat flow on either y face is for connection to another a mating y face, while the steady single-point q heat flow on either x face is for connection to a mating x face. Generally, one or two, but not all, of these heat flow connections are available in any given thermal solid. Computationally, we discretize our solid in the x direction only — never the y or z directions. That is, we slice it into a stack of wafers rather than dice it into tiny cubes. Individual thermal solid model components make different assumptions about the physics within the volume elements. Sometimes there is time variation, sometimes there is not.

17.3 Entropy Generation

Most thermal solids involve the conduction of heat across finite temperature differences and therefore represent irreversible entropy-generating processes (see chapter 14).

The second law of thermodynamics applied to the external surface of our thermal solid, over a full periodic cycle is

$$\text{External Entropy Generation} = \oint_{dt} \int_{ds} \frac{\boldsymbol{n} \cdot \boldsymbol{q}}{T} \tag{17.1}$$

where $\boldsymbol{n} \cdot \boldsymbol{q}$ is the surface normal heat flux and T is the surface absolute temperature. Entropy generation defined this way refers to the increase in entropy in the universe surrounding the thermal solid domain as a result of internal irreversibilities.

The internal generation of entropy may be calculated as

$$\text{Internal Entropy Generation} = -\oint_{dt}\int_{dv}\frac{\boldsymbol{q}\cdot\nabla T}{T^2} \tag{17.2}$$

where the integrand is the local rate of entropy production due to heat flow in a temperature gradient. Since this is the only irreversible process in a thermal solid, its integral must equal the external entropy generation. We can prove this is so by applying Gauss's theorem to the external entropy generation formula, obtaining

$$\text{External Entropy Generation} = \oint_{dt}\int_{dv}\nabla\cdot\frac{\boldsymbol{q}}{T} \tag{17.3}$$

which can be simplified by substituting the vector identity $\frac{1}{T}\nabla\cdot\boldsymbol{q}+\boldsymbol{q}\cdot\nabla(1/T)$ for $\nabla\cdot\frac{\boldsymbol{q}}{T}$, then ignoring the first term because $\nabla\cdot\boldsymbol{q}\propto\frac{\partial T}{\partial t}$, which gives $\oint_{dt}\frac{1}{T}\nabla\cdot\boldsymbol{q}\propto\oint_{dT}dT/T=0$. Then substituting $-\nabla T/T^2$ for $\nabla(1/T)$ in the second term, we wind up with the above internal generation formula. This simplification holds only approximately when we substitute finite differences for exact derivatives.

Entropy generations are presented as available-energy outputs by multiplying the entropy integrals by Tnorm, the normalization temperature of the root SCFusion model component. Each model component evaluates its entropy integrals in a manner appropriate to the physics and temperature discretization embodied within it.

17.4 Thermal Properties

Certain thermal solid model components have an enumerated-type variable which selects the solid type from a list of choices. Each available solid is itself a software object with sufficient data encapsulated within it to know its important properties. These properties are.

Density : ($\mathrm{kg/m^3}$) Constant mass density ρ_s.

Conductivity : (K, W/(m K)) A cubic spline variable with temperature vs conductivity $(T, k_s(T))$ data pairs covering a broad range of temperatures.

Specific heat : (K, J/(Kg K)) A cubic spline variable with temperature vs specific-heat $(T, c_s(T))$ data pairs covering a broad range of temperatures.

The values of density and the cubic-spline interpolation pairs appear after the enumerated identifier in display windows and the output listing. The solid choices come from a default data base file solid.dta, or a data base file customized by you. (*see* chapter 28)

An instance of a solid object has methods (subroutines) that can be called upon to return conductivity k_s, specific heat c_s or thermal diffusivity α as functions of temperature T, whenever needed by the model component. This is, essentially, just a matter of cubic-spline interpolation using the appropriate set of data pairs.

17.5 Fixed Temperature Heat Sources

These components may serve as either heat sources or sinks with a fixed temperature that you specify as an input. The heat flow in any connection is supposed to be regulated by the physics of the adjoining model component which, in effect, sees a fixed temperature boundary. Connecting together two heat sources is likely to result in a singular solution.

17.5.1 Point Heat Source

This component allows any number of steady point heat-flow connections to the positive or negative x ends of adjoining thermal solids. Its variables are:

TStdyQsrc

T : (real, K) Source temperature.

QNeg : (real, W) Net heat flow through negative x end.

QPos : (real, W) Net heat flow through positive x end.

17.5.2 Time-Grid Heat Source

This component allows any number of time-grid heat-flow connections to the positive or negative x ends of adjoining thermal solids. But only thermal solids that support time varying internal temperatures. These include the thick-surface, thin-surface and rigorous-surface components of section 17.8. It differs from the above point-heater component by imposing temperature continuity at each time node instead of just time-mean temperature continuity. This can be important when the solid heat capacity is relatively low, allowing large time-varying temperatures. Its variables are:

TGtQsrc

FT : (Fourier series, K) Source temperature.

FQNeg : (Fourier series, W) Net heat flow through negative x end.

FQPos : (Fourier series, W) Net heat flow through positive x end.

17.5.3 Line Heat Source

This component allows any number of steady distributed heat-flow connections to the positive or negative y faces of adjoining thermal solids. Its variables are:

TGxQsrc

QyNeg : (real, W) Net (x integrated) heat flow through negative y face.

QyPos : (real, W) Net heat flow through positive y face.

Since a line source is always created as a child component to a parent, it always gets its temperature distribution $T(x)$ from that parent, in the form of a cubic spline variable.

17.5.4 Independent Line Heat Source

TGxQsrcInd

Similar to a line heat source except sets the temperature distribution according to an independent input rather than the temperature distribution inherited from the parent component:

Tsrc : (cubic spline, K) Temperature distribution $T(x)$ where $x = 0$ is the negative endpoint and $x = 1$ is the positive endpoint.

This component allows you to anchor the temperatures of several heat exchanger components simultaneously by defining a temperature input at a higher model level and recasting Tsrc in terms of that input. The Tinit distribution of the above line heat source is a constant used for normalization purposes and cannot be recast.

17.5.5 Isothermal Surface

TGxtQsrc

This component allows a single time-varying distributed heat-flow connection to the negative z face of a gas domain. It functions as an isothermal heat exchanger surface. Its variables are:

QwNet : (real, W) Net (t averaged and x integrated) heat flow through positive z face.

In principle, the source temperature could have x and t variation, but it doesn't. Temperature is constant with time. Only x variation is allowed much like the preceding component. Also, like a line source, this component always gets its temperature distribution $T(x)$ from a cubic spline constant in a parent model component. Heat flow generally varies both in x and t on a space-time grid according to the heat flow to or from the attached gas domain.

17.5.6 Independent Isothermal Surface

TGxtQsrcInd

Similar to an isothermal surface except sets the temperature distribution according to an independent input rather than the temperature distribution inherited from the parent component:

Tsrc : (cubic spline, K) Temperature distribution $T(x)$ where $x = 0$ is the negative endpoint and $x = 1$ is the positive endpoint.

This component allows you to anchor the temperatures of several heat exchanger components simultaneously by defining a temperature input at a higher model level and recasting Tsrc in terms of that input. The Tinit distribution of the above isothermal surface is a constant used for normalization purposes and cannot be recast.

17.6 Fixed Heat Flow Sources

These components are similar to the fixed-temperature heat source components of section 17.5 except the roles of temperature and heat flow are reversed — heat flow is input and temperature is solved. They can be used to model electrical resistance heating (cryocooler loads or engine heaters) or other situations where heat flow is a more appropriate boundary condition than temperature. Beware though that using heat flow rather than temperature as a boundary condition may destabilize your model. It may lead to diverging-temperature non-converging solutions if your model cannot accommodate the specified heat flows. Especially when solving from initial conditions.

17.6.1 Point Heater

Similar to the above point heat source except specifying heat production rather than temperature as the independent input. Its variables are:

TStdyHeater

Qhtr : (real, W) Heat production.

T : (real, K) Solved temperature.

QNeg : (real, W) Net heat flow through negative x end.

QPos : (real, W) Net heat flow through positive x end.

On solving, the temperature T adjusts until the heat flow to external model components QPos – QNeg equals heat production Qhtr.

17.6.2 Time-Grid Heater

Similar to the above time-grid heat source except specifying heat production rather than temperature as the independent input. Its variables are:

TGtHeater

FQhtr : (Fourier series, W) Heat production.

FT : (Fourier series, K) Solved temperature.

FQNeg : (Fourier series, W) Net heat flow through negative x end.

FQPos : (Fourier series, W) Net heat flow through positive x end.

On solving, the temperature FT adjusts until the heat flow to external model components FQPos – FQNeg equals heat production FQhtr.

17.6.3 Line Heater

Similar to the above line heat source except specifying heat production rather than temperature as the independent input. Its variables are:

TGxHeater

Qhtr : (cubic spline, W) Heat production distribution $Q(x)$ where $x = 0$ is the negative endpoint and $x = 1$ is the positive endpoint. The units are W (watts per dimensionless length) rather than W/m (watts per actual length). The average value of Qhtr is the total heat production.

QyNeg : (real, W) Net (x integrated) heat flow through negative y face.

QyPos : (real, W) Net heat flow through positive y face.

TNeg : (real, K) Temperature T at negative x end.

TPos : (real, K) Temperature T at positive x end.

TMean : (real, K) Spatial averaged (x average) temperature T.

A line heater is always created as a child component to a parent and gets its initial temperature distribution $T(x)$ from that parent. On solving, $T(x)$ adjusts until the heat flow to external model components QyPos – QyNeg equals heat the average value of the heat production distribution Qhtr

17.6.4 Surface Heater

TGxtHeater

Similar to the above line heat source except specifying heat production rather than temperature as the independent input. Its variables are:

Qhtr : (cubic spline, W) Heat production distribution $Q(x)$ (*see* Line Heater above).

QwNet : (real, W) Net (t averaged and x integrated) heat flow through positive z face.

TNeg : (real, K) Temperature T at negative x end.

TPos : (real, K) Temperature T at positive x end.

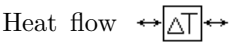
TMean : (real, K) Spatial averaged (x average) temperature T.

This component also gets its initial temperature distribution $T(x)$ from a parent component. On solving, $T(x)$ adjusts until the heat flow to the connected gas domain QwNet equals heat the average value of the heat production distribution Qhtr. The solved temperature $T(x)$ is constant with time and regulates only the time-average heat flow to the gas domain. There is generally a time dependent heat flow to the gas domain but it is regulated by the time-varying physics within the gas domain.

17.7 Heat Conductors

These are intended as parasitic conduction paths or intermediate paths from primary surfaces in contact with a gas component to heat sources or sinks. A regenerator pressure wall or a heat exchanger fin would be good examples.

17.7.1 Bar Conductor



This component models a simple linear solid conduction path with built-in steady heat-flow connectors at the positive and negative x ends. Heat flow is governed by

$$Q = \frac{A}{L}\int k_s dT \tag{17.4}$$

TStdyQcnd

where area A and length L are specified directly and conductivity k_s is a temperature-dependent property of the solid material. The integration limits are two hidden internal variables T_p and T_n, at the positive and negative x ends. The integration is accomplished without a spatial grid using a cubic-spline integration method available for solid properties. Displayed variables are:

A : (real, m^2) Cross section area A.

L : (real, m) conduction length L.

Solid : (enumerated) Solid material.

TsNeg : (real, K) Temperature T_n at negative x end.

TsPos : (real, K) Temperature T_p at positive x end.

QNeg : (real, W) Net heat flow Q_n through negative x end.

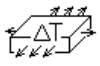
QPos : (real, W) Net heat flow Q_p through positive x end.

AEQ : (real, W) Available energy loss to heat flow.

QNeg and QPos are necessarily equal after solution, but they are listed as separate outputs because, logically, they are.

There is a variation of this component where variables A, L and Solid do not appear in the display. They are instead inherited from the parent model component. This type of conductor is typically used as a child component to a parent model component whose main purpose is not heat conduction but in whom heat conduction needs to be tagged on as an independent parasitic phenomenon. For example, a conductive-surface component within a canister (*see* chapter 19). All this is completely transparent to the user so there should be no confusion.

17.7.2 Distributed Conductor



This component models a rudimentary two-dimensional solid conduction path with any number of steady point heat-flow connections to the positive or negative x ends of adjoining thermal solids and any number of steady distributed connections to the positive or negative y faces. A typical use would be for an intermediate conduction path between a fixed-temperature heat source (sink) and one of the subsequent heat-exchanger surface components. A thermal busbar, if you will.

TGxQcnd

The thermal solid domain now has an axial center-line temperature distribution $T_s(x)$, discretized on a spatial grid, beginning at the negative x end and ending at the positive x end. It also has two additional discretized temperature distributions $T_n(x)$ and $T_p(x)$, parallel to $T(x)$, centered in the negative and positive y faces. Axial (x directed) heat flux is given by

$$q_x = k_s \frac{\partial T_s}{\partial x} \tag{17.5}$$

with $\frac{\partial T_s}{\partial x}$ being replaced by the finite-difference equivalent. Transverse (y directed) heat flux at the negative and positive y faces are given by

$$q_n = k_s \frac{T_n(x) - T_s(x)}{D/2} \tag{17.6}$$

and

$$q_p = k_s \frac{T_s(x) - T_p(x)}{D/2} \tag{17.7}$$

where D is the solid depth in the y direction. The governing equation for the interrelationship among q_x, q_n and q_p is the heat-flow continuity equation $\nabla \cdot \boldsymbol{q} = 0$ which may be written in the form

$$\frac{\partial q_x}{\partial x} + \frac{q_p - q_n}{D} = 0 \tag{17.8}$$

Actual heat flow per unit length in the y direction is either $q_n W$ or $q_p W$, where W is the solid thickness in the z direction. Actual heat flow in the x direction is $q_x A_s$ where $A_s = WD$ is the cross-sectional area.

Distributed conductor variables are:

W : (real, m) Solid thickness in z direction. Increasing this dimension gives more y heat flow for a given temperature drop.

D : (real, m) Solid depth D in y direction. Increasing this dimension gives less y heat flow for a given temperature drop.

Solid : (enumerated) Solid material.

Mass : (real, Kg) Solid mass, provided for use in constraints in case it is part of a reciprocating mass system.

QyNeg : (real, W) Net (x integrated) heat flow through negative y face.

QyPos : (real, W) Net heat flow through positive y face.

QxNeg : (real, W) Net heat flow through negative x end.

QxPos : (real, W) Net heat flow through positive x end.

AEQy : (real, W) Available energy loss to y directed heat flow, according to internal generation formula (17.2).

AEQx : (real, W) Available energy loss to x directed heat flow, according to internal generation formula (17.2).

AEdiscr : (real, W) Available energy discrepancy of above two losses compared to external generation as calculated by (17.1). (see section 17.3)

TsNeg : (real, K) Temperature T_s at negative x end.

TsPos : (real, K) Temperature T_s at positive x end.

The initial temperature distribution and length of a distributed conductor come from the parent model component.

There is a variation of this component which inherits its conduction-path cross-section area A_s and solid properties (Solid variable) from a parent component. Cross sectional area typically comes from a variable like Asolid in a canister component (*see* chapter 19) which may be either real or cubic-spline valued. Solid z-thickness W then becomes a dependent variable according to

$$W = A_s/D \tag{17.9}$$

W is either real or cubic-spline valued depending on the type of A_s in the parent. D remains an independent input, although its value is irrelevant (provided non-zero) if the component is used only to model axial heat flow (no y-face heat-flow connections). Otherwise, D may be used to control the y temperature difference for distributed heat flow, as described above.

17.7.3 Conductive Surface

TGxtQcnd

This component is similar to the previous distributed conductor except it also allows a single time-varying distributed heat-flow connection from its positive z face to the negative z face of a gas domain. In other words, the positive z face models the wetted surface of a heat exchanger. The interior models a conduction pathway to the ultimate source or sink, which may be connected through either x end or y face. A typical use would be for a rectangular heat-exchanger fin, although it is adaptable to any geometry. Like a distributed conductor, a conductive surface inherits its initial temperature distribution and length from a parent model component.

A conductive surface also inherits cross-section area A_s and wetted perimeter S_x from its parent component, typically from variables Asec and Pwet in a heat-exchanger component (*see* chapter 20), which may be either real or cubic-spline valued. This information is translated into $2n$ rectangular solid domains of z-thickness W and y depth D as shown in figure 17.1. W is a dependent variable calculated as

$$W = A_s/S_x \tag{17.10}$$

W is always a reasonable approximation of the parent heat exchanger wall thickness. D is an independent input variable. The implied number of rectangular

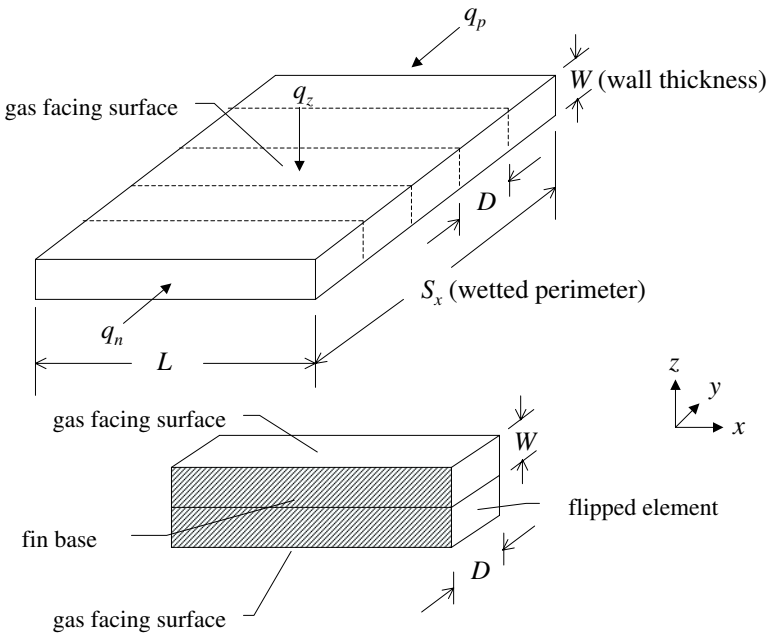


Figure 17.1: For illustration purposes, a conductive surface morphed into heat exchanger fins. The top picture corresponds to the basic thermal solid domain of thickness W broken into $2n$ pieces of width D (D is independent input variable). In the bottom picture the individual pieces are re-arranged into n heat exchanger fins, each of height D and thickness $2W$. Only one fin is shown. The same one-dimensional solution grid applies to all the fins together.

domains is just the quotient of A_s and WD or

$$2n = \frac{S_x}{D} \tag{17.11}$$

In the event the conductive surface represents rectangular fins n corresponds to the fin number. The assumption is that these rectangular domains act in parallel. The y-depth D is up to you. As you reduce D you are making y-directed heat flow easier by effectively breaking the total wetted perimeter of the heat-exchanger wall into more parallel segments, as illustrated in figure 17.1. Although this may seem complicated, most of the complexity is managed internally, Your only responsibility is to enter the correct value for D, the conduction-segment y-length, based on your understanding of the heat exchanger geometry.

In effect the heat flows a distance $W/2$ from z face to fin center through a conductor of area LS_x (wetted surface exposed to gas) then turns direction and flows to either y face (or both if both connected) over distance $D/2$ through a conductor of area $L2nW = LA_s/D$ (total fin cross section). For the special case where you set $D = W$ (wall thickness) the conductor area in the y direction works out to LS_x (same as z direction) and the total conduction length is the wall thickness W, in effect modeling heat flow directly through the heat exchanger wall. Then the z surface represents the inside tube surface and the connected y face (either one but not both) represents the outside tube surface (*see* sample model *HeatExchangers-ThermalConductors* in the Apps\SCFusion\Samples\ElementsThermoModels sub-directory under the installation directory).

Although the gas-surface heat flux may have a time-varying component, only the steady component contributes to the interior solid solution. In other words, the solid filters out any AC heat flow components, passing only the DC component to the source or sink. This is appropriate where the net steady component of gas-surface heat transfer dominates the time-varying component — as in a heat rejector or acceptor. Otherwise, in a regenerator for example, one of the quasi-adiabatic surface components might be more appropriate.

In the conductive surface the thermal solid domain has an axial temperature distribution $T_w(x)$ centered in the positive z face (serving all $2n$ fin faces of fin morphing in figure 17.1), in addition to $T_s(x)$, $T_n(x)$ and $T_p(x)$ as previously described for the distributed conductor. Gas surface heat flux at the positive z face is given by

$$\langle q_w \rangle = k_s \frac{T_s(x) - T_w(x)}{W/3} \tag{17.12}$$

where W is the solid thickness in the z direction and $\langle\rangle$ symbolizes the time-average operator. The equivalent conduction thickness is $W/3$ rather than $W/2$ because in a continuous-temperature solution the z-directed heat flux would decrease linearly from the positive z-face to zero at the negative z-face, which is insulated, branching off in the x or y directions within the solid. Under those conditions the temperature is quadratic in the z direction and $W/3$ turns out

to be the equivalent conduction length that gives the correct surface heat flux, assuming that T_s is the section-average of the continuous temperature solution.

The equations for axial and transverse heat fluxes q_x, q_n and q_p are the same as those given previously for the distributed conductor, except that q_x includes a tortuosity factor as explained below. The governing equation for the interrelationship among heat fluxes is again the heat-flow continuity equation $\nabla \cdot \boldsymbol{q} = 0$ which may now be written in the form

$$\frac{\partial q_x}{\partial x} + \frac{q_p - q_n}{D} + \frac{\langle q_w \rangle}{W} = 0 \tag{17.13}$$

The y-directed heat fluxes q_n and q_p are always based on the total z thickness A_s/D. The z-directed heat flux q_w is always based on the total wetted perimeter S_x. So, even if the parent geometry is not rectangular fins, the model continues to make sense because variables A_s and S_x are always defined.

Because a conductive surface can represent the solid surface of a matrix-type heat exchanger the axial heat flow q_x is discounted by a possible tortuosity factor $f_s < 1$ (*see* section 20.0.1). Axial heat flow q_x is calculated as

$$q_x = -k_s A_e \frac{\partial T_s}{\partial x} \tag{17.14}$$

where k_s is solid conductivity and A_e is the effective solid conduction area. For axially-uniform regenerator matrices, such as wrapped foil, A_e is just the mean solid area A_s. For axially-irregular matrices like stacked screens, A_e is the mean solid area reduced by a tortuosity factor f_s to account for the small contact areas between wires (*see* section 20.0.1). Sage does not apply the tortuosity factor to thermal conduction in the z or y directions because the first generally pertains to conduction normal to the surface of individual matrix particles and the second pertains to bulk conduction out of the matrix perpendicular to the flow direction, which may be along a continuous path (e.g. stacked screens in plane of screens). If it is necessary to model some tortuosity in the y direction you can always increase the D input.

Conductive surface variables are:

D : (real, m) Individual fin depth D. Often thought of as the fin conduction length in a typical sectional view. In general, the conduction path length (of one or more parallel segments) from the wetted surface of the heat-exchanger wall to the point(s) where heat is added or removed at the outer perimeter.

Solid : (enumerated) Solid material.

Mass : (real, Kg) Total solid mass, provided for use in constraints in case it is part of a reciprocating mass system.

W : (real, m) Effective heat-exchanger wall thickness. Half the thickness of a symmetrically-heated fin. In general, A_s/S_x, where cross-section A_s and wetted perimeter S_x come from a parent model component. Spatial average value if A_s and S_x are cubic-spline valued.

Tortuosity : (real, dimensionless) Mean (x average) solid axial conduction multiplier f_s (*see* section 20.0.1). $f_s A_s$ is the effective area A_e for thermal-solid axial conduction (*see* equation (17.14)). So named because of the tortuous solid conduction path in porous materials.

QwNet : (real, W) Net (t averaged and x integrated) heat flow through positive z gas-facing surface.

QyNeg : (real, W) Net (x integrated) heat flow through negative y face.

QyPos : (real, W) Net heat flow through positive y face.

QxNeg : (real, W) Net heat flow through negative x end.

QxPos : (real, W) Net heat flow through positive x end.

AEQw : (real, W) Available energy loss to z directed gas-surface heat flow, according to internal generation formula (17.2).

AEQy : (real, W) Available energy loss to y directed heat flow, according to internal generation formula (17.2).

AEQx : (real, W) Available energy loss to x directed heat flow, according to internal generation formula (17.2).

AEdiscr : (real, W) Available energy discrepancy of above three losses compared to external generation as calculated by (17.1). (see section 17.3)

TsNeg : (real, K) Temperature T_s at negative x end.

TsPos : (real, K) Temperature T_s at positive x end.

17.8 Quasi-Adiabatic Surfaces

These thermal solids all presume the positive z surface is subject to time-varying sinusoidal heat flux, with zero or near zero mean, usually as the primary surface in contact with the gas within a regenerator or variable-volume space. Accordingly they are born with a surface heat-flow connector, intended for connection to a gas model component. The presumed zero-mean heat flow is where the term quasi-adiabatic comes from.

Aside from that they are in many respects similar to the previous conductive surface component. You may connect either x-end or y-face to other thermal solids or heat sources via optional steady point or distributed heat-flow connections. But only the DC (steady) component of heat-flow passes through such connections. You may also connect either x-end via optional time-grid heat flow connections, which pass heat flow as a time grid. Time-grid connections impose temperature continuity at each time node, rather than just time-mean temperature continuity, which can be useful if the solid heat capacity is relatively low, allowing large time-varying temperatures. Heat-flow connections are made via

optional negative or positive heat-flow ends or faces available on the Thermal Attachments page of the child-model creation palette.

As with a conductive surface the actual solid geometry, which may be an extremely complicated porous structure, is shoe-horned into an equivalent rectangular solid based on the cross-section area A_s and wetted perimeter S_x inherited from the parent component. Independent input variable D defines the effective depth for any y directed heat flows you may wish to model.

The thermal solid domain has an axial center-line temperature distribution $T_s(x,t)$ and a temperature distribution $T_w(x,t)$ centered in the positive z face. Both are discretized on a space-time grid, beginning at the negative x end and ending at the positive x end. In support of y-directed heat flows there are also two time-independent temperature distributions $T_n(x)$ and $T_p(x)$ centered in the negative and positive y-faces, similar to those of a distributed conductor. These are discretized on a separate spatial grid.

The governing equation for the thermal solid is now the time-dependent energy equation $\frac{\partial E_s}{\partial t} + \nabla \cdot \boldsymbol{q} = 0$ where E_s is the volume-specific solid energy. If we section-average this equation through the entire z depth of the solid we may write it in the form

$$\rho_s A_s \frac{\partial e_s}{\partial t} + \frac{\partial q_x}{\partial x} + Q_w + Q_p - Q_n = 0 \tag{17.15}$$

where

A_s	=	mean solid cross section
e_s	=	mass-specific solid energy
q_x	=	solid-mode axial heat flow
Q_w	=	z surface heat transfer per unit length
Q_p	=	positive y face heat flow per unit length
Q_n	=	negative y face heat flow per unit length
ρ_s	=	solid density

The purpose of the solid energy equation is to determine T_s as an implicit variable. Axial heat flow q_x is based on the effective solid cross-section area A_e (mean area A_s reduced by a tortuosity factor f_s) the same as for a conductive surface and is given by equation (17.14). The steady transverse (y directed) heat flows per unit length at the negative and positive y faces are given by

$$Q_n = (A_s/D)\, q_n = (A_s/D)\, k_s \frac{T_n(x) - \langle T_s \rangle (x)}{D/2} \tag{17.16}$$

and

$$Q_p = (A_s/D)\, q_p = (A_s/D)\, k_s \frac{\langle T_s \rangle (x) - T_p(x)}{D/2} \tag{17.17}$$

where D is the solid depth in the y direction, A_s/D is the total solid width in the z direction (*see* section 17.7.3) and $\langle\rangle$ symbolizes the time average operator. Note that for the same temperature difference the y directed heat flux varies inversely with D^2, according to the above equations. So it important to set the value of D carefully when modeling y-directed heat flows.

For surface heat transfer Q_w Sage uses a formulation involving the temperature difference $T_s - T_w$, where T_w is understood as the surface temperature. To obtain our formulation we must digress briefly into the exact temperature solution in a solid layer subject to sinusoidal heat flux — more specifically, a solid slab of thickness d, insulated at the face $z = 0$ and subject to heat flux $q_z = e^{i\omega t}$ at face $z = d$.

This is approximately what's going on in a regenerator matrix, with the face $z = d$ corresponding to the gas-solid interface. Thickness d would represent foil half-thickness, in the case of a foil matrix. In general, d is taken as the solid volume divided by wetted surface. This is also approximately what's going on in the walls of a variable-volume space, so long as heat flows to and from the wall with zero mean heat transfer, as is often the case.

In a complex formulation, the exact temperature field solving this problem is

$$\boldsymbol{T} = -\frac{\delta}{k_s(1+i)} \frac{\cosh((1+i)z/\delta)}{\sinh((1+i)d/\delta)} e^{i\omega t} \tag{17.18}$$

where

$$\begin{aligned} \delta &= \sqrt{(2\alpha)/\omega}\text{; thermal penetration depth} \\ \alpha &= k_s/(\rho_s c_s)\text{; thermal diffusivity} \end{aligned}$$

In particular, the complex temperature at the gas-solid interface ($z = d$) is

$$\boldsymbol{T}_w = \boldsymbol{T}(d) = -\frac{\delta}{k_s(1+i)\tanh((1+i)d/\delta)} e^{i\omega t} \tag{17.19}$$

with complex heat flow per unit length

$$\boldsymbol{Q}_w = -k_s S_x \left(\frac{\partial \boldsymbol{T}}{\partial z}\right)_w = S_x e^{i\omega t} \tag{17.20}$$

where $S_x = A_s/d$ is wetted perimeter. The section-average temperature is

$$\boldsymbol{T}_s = \frac{i\delta^2}{2k_s d} e^{i\omega t} \tag{17.21}$$

which is also the solution of our mean-parameter solid energy equation (17.15) with $\frac{\partial q_x}{\partial x} = 0$ and $\boldsymbol{Q}_w = (A_s/d)e^{i\omega t}$.

Writing the complex temperature difference $\boldsymbol{T}_s - \boldsymbol{T}_w$ as $\Delta\boldsymbol{T}$, complex heat flux may be formulated as

$$\boldsymbol{Q}_w = \frac{k_s}{d} S_x \boldsymbol{N}_s \Delta\boldsymbol{T} \tag{17.22}$$

where $\boldsymbol{N}_s$ is a dimensionless number not unlike the Nusselt number of gas heat transfer. Solving equation (17.22) for $\boldsymbol{N}_s$ and substituting the above solutions (17.19), (17.20) and (17.21) for $\boldsymbol{T}_w$, $\boldsymbol{Q}_w$ and $\boldsymbol{T}_s$ gives

$$\boldsymbol{N}_s = \frac{2i(d/\delta)^2}{\frac{(1+i)d/\delta}{\tanh((1+i)d/\delta)} - 1} \tag{17.23}$$

Interesting are the limiting cases for $d/\delta \to \infty$ (thick wall, high frequency, low conductivity) and for $d/\delta \to 0$ (thin wall, low frequency, high conductivity). These are

$$\boldsymbol{N}_s = \begin{cases} (1+i)d/\delta & \text{if } d/\delta \to \infty \\ 3 & \text{if } d/\delta \to 0 \end{cases} \tag{17.24}$$

For thick solids, where thermal activity is confined to a layer roughly δ thick, surface heat flux is phase-shifted 45 degrees ahead of temperature difference $\Delta\boldsymbol{T}$. For thin solids, where the entire solid is thermally active, there is no phase shift.

In Sage models the surface heat flux is not necessarily sinusoidal. Depending on the way the component is connected it may, for example, contain a steady component, like the conductive surface of section 17.7.3. To account for this possibility the physical heat flux is formulated as

$$Q_w = 3\frac{k_s}{d}S_x(\Delta T - \Re\Delta\boldsymbol{T}) + \frac{k_s}{d}S_x\Re(\boldsymbol{N}_s\Delta\boldsymbol{T}) \tag{17.25}$$

The first term on the right-hand side represents the heat flux governed by the steady heat conduction equation for an effective solid thickness of $d/3$, or equivalently by the $\boldsymbol{N}_s$ value for the thin-wall limit of equation (17.24). The second term represent the real part of the complex heat transfer according to equation (17.22).

In this formulation the solved temperature difference $\Delta T(t) = T_s(t) - T_w(t)$ is broken into a sinusoidal temperature variation $\Re\Delta\boldsymbol{T}$ plus a residual component $\Delta T - \Re\Delta\boldsymbol{T}$ comprising the mean value plus higher harmonics. The sinusoidal temperature variation is taken to be the real part of the complex temperature variation calculated from the first harmonic cosine and sine coefficients a_1 and b_1 of the Fourier-series expansion of temperature difference outlined in section 8.5.2.

It is not Q_w that is solved from ΔT in equation (17.25) but, rather, the other way around. Sage solves ΔT implicitly, considering heat flux Q_w as given.

There are quasi-adiabatic model components for $\boldsymbol{N}_s$ computed rigorously, according to equation (17.23), as well as the thick- and thin-wall approximations according to (17.24). They all descend from a common ancestor which has the following variables:

Kmult : (real, dimensionless) Empirical multiplier for axial conduction q_x.

D : (real, m) Effective solid-conduction distance between the center line and the surface where any y face heat flow is extracted. The value of D only matters when there is a distributed heat-flow connection made at either y face.

Solid : (enumerated) Solid material.

Mass : (real, Kg) Solid mass, provided for use in constraints in case it is part of a reciprocating mass system.

Dskin : (real, m) Skin thickness d in z direction, calculated as solid volume divided by wetted perimeter (A_s/S_x).

Lambda : (real, m) Mean thermal wavelength λ in z direction, calculated as $(2\pi\sqrt{2\alpha/\omega})$.

Tortuosity : (real, dimensionless) Mean (x, t average) solid axial conduction multiplier f_s (*see* section 20.0.1). $f_s A_s$ is the effective area A_e for thermal-solid axial conduction (*see* equation (17.14)). So named because of the tortuous solid conduction path in porous materials.

FQwNet : (Fourier series, W) Net (x integrated) heat flow through positive z gas-facing surface.

QxNeg : (real, W) Net heat flow through negative x end.

QxPos : (real, W) Net heat flow through positive x end.

AEQw : (real, W) Available energy loss to z directed gas-surface heat flow, according to internal generation formula (17.2).

AEQx : (real, W) Available energy loss to x directed heat flow, according to internal generation formula (17.2).

AEdiscr : (real, W) Available energy discrepancy of above two losses compared to external generation as calculated by (17.1). (see section 17.3)

TsNeg : (real, K) Temperature T_s at negative x end.

TsPos : (real, K) Temperature T_s at positive x end.

FTsMean : (Fourier series, K) Spatial averaged (x average) temperature T_s.

Length l, cross sectional area A_s, z surface wetted perimeter S_x and initial temperature distribution T_s all come from the parent model component.

Prior to Sage version 12 (July 2021) the factor $\frac{\partial e_s}{\partial t}$ in equation (17.15) was approximated as $\langle c_s \rangle \frac{\partial T_s}{\partial t}$, where $\langle c_s \rangle$ was the solid heat capacity at the time-mean temperature $\langle T_s \rangle$. The reason for this was that evaluating $\frac{\partial e_s}{\partial t}$ directly as $c_s \frac{\partial T_s}{\partial t}$ would not have been in energy-conserving form because factor c_s is not time-constant. Numerical differencing errors would not have canceled out, resulting in imperfect energy conservation over the cycle period. The approximation $\langle c_s \rangle \frac{\partial T_s}{\partial t}$ was reasonable for solid components with relatively small cyclic temperature variations and where solid heat capacity did not vary much in that temperature range. But, neither assumption is valid for regenerator matrices operating below 10 K. So starting with version 12, $\frac{\partial e_s}{\partial t}$ is computed by evaluating $e_s(T_s)$ at each time node as $e_s(T_s) = \int_{T_0}^{T_s} c_s dT$, directly from the cubic-spline property functions that define the specific heat. This formulation embeds the effect of time-varying c_s within the time derivative and therefore

is in energy-conserving form. Integration limit T_0 is arbitrary so long as it is constant over the time cycle. It is convenient to use the time-mean temperature $T_0 = \langle T_s \rangle$ at each spatial node where the equation is applied because that minimizes the computational work required to calculate $e_s(T_s)$.

Also, the complex temperature variation was defined differently (see section 8.5.2) with the potential for errors between the relationship between Qw and ΔT for solutions with large higher-harmonic content. This was a problem in low-temperature GM-style pulse-tube cryocooler models where solid heat capacities were very low, leading to erratic convergence and invalid solutions. The present formulation attempts to correct this problem to the extent possible in a 1-D solution.

17.8.1 Thick Surface

TGxtThkWall

This quasi-adiabatic surface presumes the skin thickness d is much larger than the thermal wavelength λ. It is intended for modeling something like a thermally massive cylinder space wall, where the time-varying heat flux is confined to a thin layer near the surface.

17.8.2 Thin Surface

TGxtThnWall

This quasi-adiabatic surface presumes the skin thickness d is much smaller than the thermal wavelength λ. It is intended as a fast alternative to the following rigorous surface for modeling something like a regenerator matrix when the wire diameter, particle size, foil thickness, or whatever, is very small. If you use this component, you should check that output variable Dskin is indeed smaller than Lambda. If not, then you should use the rigorous surface component instead.

17.8.3 Rigorous Surface

TGxtMedWall

This quasi-adiabatic surface presumes the skin thickness d is on the same order as the thermal wavelength Lambda. It is intended for modeling any sort of regenerator matrix or duct wall, although the slightly-faster thin-surface model component may be justified for *finely divided* matrices. A rigorous surface calculates the Nusselt-like number $\boldsymbol{N}_s$ from equation (17.23) which takes a bit more time than the thick- or thin-surface approximations.

Chapter 18

Gas Domains

The gas-domain model components are at the heart of the SCFusion model class. Because of their complicated mathematical structure, they are also the most time consuming to solve and prone to convergence problems. From your point of view, they appear simply as palette-created child model components within heat-exchanger geometries. There are gas-domain variations for all of the basic types of fluid flow: duct flow, porous matrix flow and flow into a variable-volume space (formed by a piston in its cylinder) through an inlet. All gas domains are born with a thermal boundary connector intended for connection to a thermal solid of your choice. Gas domains, being one dimensional, admit to flow connections with other gas domains at either their positive or negative ends. Since you may want to have flow connections at only one end, at both ends, or multiple connections at one end, flow connectors are palette-created child-model components. Variable-volume gas domains have palette-created displaced-volume connectors intended for connection to area faces of moving parts.

There remains one more gas-domain connection: the charge line. Exactly one gas domain model component must have a charge-line connection to a pressure-source component to establish the amount of working gas in the sum-total of all interconnected gas domains. This is very important. Without a pressure-source connection there will be no way for Sage to determine the amount of working gas in your system and solving will not converge. Charge-line connectors are palette created because there is no way for Sage to decide which gas component should get it.

18.1 Attachment Child Components

All gas domains have the ability to produce child model components representing charge-line connectors and flow inlets. Some can also produce volumetric displacements.

When you drop a gas domain model component from the SCFusion palette into the edit form, it starts out with only a thermal connector arrow, intended for

connection to a thermal solid component. In order to make additional connections, you must first drop in one or more of the following child model components from the palette.

Icon	Purpose
	gas charge line
	negative gas inlet
	positive gas inlet
	time-ring negative-facing volume displacement
	time-ring positive-facing volume displacement

When you drop one of these into the edit form, it is born with a connection arrow which you can then move up one level to the parent-component page of the edit form for connection there to a mate, perhaps originating in another gas domain.

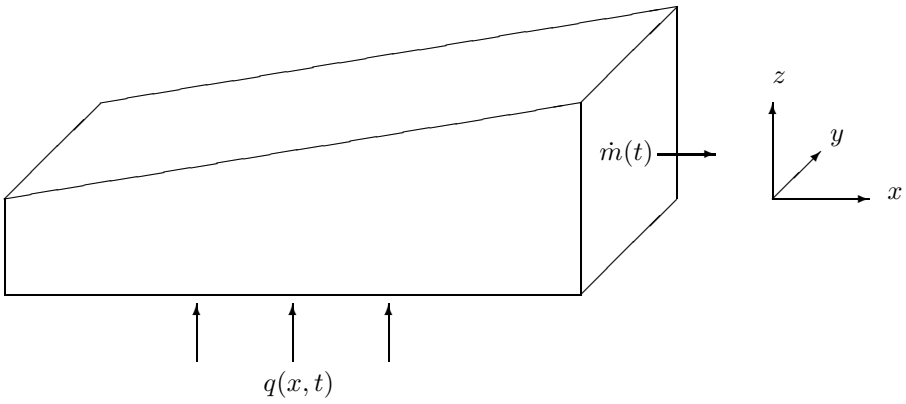
The gas charge line is for connection to a pressure-source component. It is required for exactly one gas domain component in your total interconnected gas domain, as explained above.

The two types of gas inlets are for connection to other gas domains as you see fit. Generally you will want zero or one flow inlets at each end, depending on whether your component is functioning as a dead-ended passage or through passage. In some instances you will create two (or more) flow inlets at the end of a gas component which enables it to act as a T or Y component.

The volume displacements are available only in variable-volume gas domains. They are intended for connection to area faces of moving parts like the reciprocator or constrained piston components documented in chapter 16.

18.2 Coordinate Conventions

A gas domain may be thought of as a one-dimensional rectangular region like this:



Gas flow $\dot{m}(t)$ may pass through either the positive or negative x boundaries, but nowhere else. Heat flux $q(x,t)$, generally to a thermal solid, may pass through the negative z boundary, but nowhere else. Discretization is in the x direction only, similar to a thermal solid. In principle, the cross-sectional area may vary with x, as drawn. It may even vary with time, as if the upper z surface were moving in accordance with a volume displacement.

18.3 Entropy Generation

Gas domains involve the conduction of heat across finite temperature differences as well as irreversible flow processes. Both generate entropy (see chapter 14).

The second law of thermodynamics applied to the external surface of our gas domain, over a full periodic cycle is

$$\text{External Entropy Generation} = \oint_{dt} \int_{ds} \frac{\boldsymbol{n} \cdot \boldsymbol{q}}{T} + \left[\oint_{dt} \dot{m} s \right]_{x=a}^{x=b} \tag{18.1}$$

where $\boldsymbol{n} \cdot \boldsymbol{q}$ and T are the z-surface normal heat flux and absolute temperature, and $\dot{m}$ and s are the x-end mass flow rate and mass-specific entropy. The $[\cdot]_{x=a}^{x=b}$ notation means the difference of the enclosed expression evaluated a the two endpoints. Entropy generation defined this way refers to the increase in entropy in the universe surrounding the gas domain as a result of internal irreversibilities.

There are two internal entropy generating mechanisms, heat flow and viscous friction. There are actually two distinct heat-flow mechanisms, axial conduction and film heat transfer, but at this level of generality the entropy generated by either is covered by the formula

$$\text{Conductive Entropy Generation} = - \oint_{dt} \int_{dv} \frac{\boldsymbol{q} \cdot \nabla T}{T^2} \tag{18.2}$$

where the integrand is the local rate of entropy production due to heat flow in a temperature gradient. And the entropy generation of viscous friction, in one-dimensional flow, may be calculated as

$$\text{Viscous Entropy Generation} = -\oint_{dt}\int_{dx}\frac{uAF}{T} \tag{18.3}$$

where uA is volumetric flow rate and F represents that part of the pressure gradient due to viscous friction. The numerator in the integrand is the pumping dissipation per unit length.

Entropy generations are presented as available-energy outputs by multiplying the entropy integrals by Tnorm, the normalization temperature of the root SCFusion model component. Each model component evaluates its entropy integrals in a manner appropriate to the physics and solution-variable discretization embodied within it.

18.4 Pressure Source Component

TStdyPsrc

The pressure source component establishes the charge pressure for your SCFusion model. You must have exactly one pressure source, connected to some gas domain. A pressure source acts as an infinite isobaric gas reservoir. Its only variable is

Pcharge : (real, Pa) Charge pressure.

The result of connecting a pressure source to a gas domain is that the density in the gas domain adapts itself so that t average pressure is continuous across the connection — so that the time-average pressure at the negative end of the gas domain becomes Pcharge. This is explained mathematically in section (18.6).

18.5 Gas Domain Components

There are three distinct variations of gas domains, distinguished primarily by the method by which they track the onset of turbulence. They all come with a large assortment of variables, mostly outputs, sufficient for most of your needs but not too numerous to unduly clutter your output listings. For more information you can always dump the solution grid with the File|Save Solution Grid command. The following list includes all possible variables in all types of gas domains. Some variables may be missing from individual gas domains, depending on what makes sense. Mathematical symbols, as yet undefined, will be found in the subsequent sections on theory at the end of this chapter.

Fmult : (real, dimensionless) Empirical multiplier for viscous pressure drop F.

Hmult : (real, dimensionless) Empirical multiplier for z-surface heat transfer Q_w.

Kmult : (real, dimensionless) Empirical multiplier for axial conduction q in domain interior.

KmultBnd : (real, dimensionless) Empirical multiplier for axial conduction q at domain endpoints. Default is 0 which blocks thermal conduction between gas domains. Set to 1 in both gas domains on either side of a flow connector to allow continuous thermal conduction across the flow connector.

UpwindFrac : (real, dimensionless) The relative weight factor for upwind influence in density (temperature) interpolation between control volumes in the computational grid (*see* section 18.6.7). Increasing the value will help smooth out a jagged axial temperature profile at some cost in reduced accuracy. Based on computational experiments, the default of 1.0E-2 provides sufficient stability for almost all situations with a negligible effect on accuracy. In the event of temperature instability (which you can see by plotting a file of computational solution variables), increasing the value to 1.0E-1 should stabilize almost any gas domain. Poor convergence may be a sign of temperature instability. Inputs are restricted to the range zero to one.

Klocal : (real, dimensionless) Local frictional pressure-drop loss coefficient K (see equation (18.18). Used for including entrance, exit and bend effects only in duct-type gas domains.

TbInNeg : (real, dimensionless) Incoming relative turbulence $\mathcal{T}$ at the positive end of a duct-type gas domain. The default is 1, corresponding to a sharp entrance where flow separation is expected or the upstream turbulence level is high. A value of 0 implies completely smooth incoming flow corresponding to a continuous flow area or a perfectly diffused transition and low upstream turbulence level.

TbInPos : (real, dimensionless) Like TbInNeg except at the duct-gas negative end.

FQcombust : (Fourier series, W) Combustion firing rate as a function of time. Available as an input variable only in the combustion-space gas component within a generic-cylinder component. It directly determines the combustion heating term Q_c in the gas energy equation. (*see* section 18.6.3).

PV : (Fourier series, W) Net (x integrated) PV power output ($\int_{dx} P\frac{\partial A}{\partial t}$). Available only in variable-volume gas domains where flow area A is a function of time. The mean value is the t-average PV power which most people think of when they think of PV power. A positive value indicates work done by the gas on the walls. The higher harmonics are available if you are interested.

FQwNet : (Fourier series, W) Net (x integrated) heat flow through negative z solid-facing surface (positive is from solid to gas).

HNeg : (real, W) t average stagnation enthalpy flow through negative x inlet. Stagnation enthalpy $uA(\rho e + P)$ includes all forms of energy flow — internal, PV and kinetic.

HPos : (real, W) t average stagnation enthalpy flow through positive x inlet.

PVNeg : (real, W) Approximate t average PV power flow through negative x inlet. Not to be used for strict energy balance calculations. *See* section 18.6.5.

PVPos : (real, W) Approximate t average PV power flow through positive x inlet.

QNeg : (real, W) t average axial heat flow q through negative x inlet.

QPos : (real, W) t average axial heat flow q through positive x inlet.

AEfric : (real, W) Available energy loss to viscous flow friction, according to internal generation formula (18.3).

AEQw : (real, W) Available energy loss to z directed solid-surface heat flow, according to internal generation formula (18.2).

AEQx : (real, W) Available energy loss to x directed heat flow, according to internal generation formula (18.2).

AEdiscr : (real, W) Available energy discrepancy of above losses compared to external generation as calculated by (18.1). (see section 18.3)

QwNeg : (real, W/m) t average heat flow per unit length through negative z solid-facing surface, at negative x end. Gives insight into surface heat flow distribution.

QwPos : (real, W/m) Same as QwNeg except at positive x end.

QxMean : (real, W) t and x averaged axial heat flow q.

TNeg : (real, K) t average temperature T at negative x end.

TPos : (real, K) t average temperature T at positive x end.

Vmean : (real, m^3) Gas domain volume. Provided for fixed as well as time-varying volumes, although in that case Vmean is the t average volume. *See also:* FV for variable-volume gas domains.

FTMean : (Fourier series, K) Spatial averaged (x average) temperature T.

FPMean : (Fourier series, Pa) Spatial averaged (x average) pressure P.

FPNeg : (Fourier series, Pa) Pressure P at negative boundary.

FPPos : (Fourier series, Pa) Pressure P at positive boundary.

FDP : (Fourier series, Pa) Pressure difference across domain (positive − negative x ends).

FV : (Fourier series, m^3) Domain volume for a variable-volume gas domain. The mean value (t average) and first harmonic are generally of the most interest.

FM : (Fourier series, kg) Total mass in a gas domain. The sum of all mean values (t averages) over all gas domains is the total gas mass in the system. The time derivatives of the first and higher harmonics are theoretically the same as the difference of inlet mass flow rates (FRhoUAPos - FRhoUANeg). But due to inaccuracies of finite time-differencing, this is exact only for the first harmonic and in the limit of large time nodes for higher harmonics.

FHmean : (Fourier series, W) Spatial averaged (x average) stagnation enthalpy flow $uA(\rho e + P)$.

FRhoUAmean : (Fourier series, kg/s) Spatial averaged (x average) mass flow rate ρuA.

FRhoUANeg : (Fourier series, kg/s) Mass flow rate ρuA through negative x inlet.

FRhoUAPos : (Fourier series, kg/s) Mass flow rate ρuA through positive x inlet.

MachMean : (real, dimensionless) t and x averaged Mach number.

TdMean : (real, dimensionless) Tidal amplitude divided by length L. Tidal amplitude is the oscillatory amplitude of the mean fluid displacement. It is calculated as $\langle u \rangle_1 / \omega L$ where $\langle u \rangle_1$ is the amplitude of the first harmonic (fundamental) of the Fourier series decomposition of x-average section-mean velocity u. Only available in duct or matrix gas domains.

ReMean : (real, dimensionless) t and x average Reynolds number R_e in matrix or duct gas domains or turbulent Reynolds number R_t in variable-volume gas domains.

VaMean : (real, dimensionless) t and x averaged Valensi number V_a in duct gas domains.

TbMean : (real, dimensionless or J/m^3) t and x averaged relative turbulence $\mathcal{T}$ in duct gas domains or turbulence intensity $\rho\kappa$ in variable-volume gas domains.

ZMean : (real, dimensionless) t and x averaged gas compressibility $P/(\rho RT)$ as returned by the gas property variable Gas selected in the SCFusion root component. This is always one for an ideal gas.

EOSErrMean : (real, dimensionless) t and x averaged equation of state relative error as returned by the gas property variable Gas selected in the SCFusion root component. This is always zero except for the deprecated Redlich-Kwong gas class.

Coming from the parent component are the length L, mean flow area A, wetted perimeter S_x, initial axial temperature distribution $T(x)$ in the form of a cubic spline variable and working gas (properties).

A primary purpose of a gas domain is to calculate the empirical relationships for friction factor f, Nusselt number N_u and axial conductivity ratio N_k appropriate for the flow geometry in which it finds itself (see section (18.6). Accordingly, there is a gas-domain variation for each individual type of heat exchanger geometry. But they all fit into one of the following three classes.

18.5.1 Matrix Gas Domains

TGxt...MtxGas

A *matrix* gas domain is used within a porous matrix or within uniform channels of tiny hydraulic diameter. A matrix gas domain knows about Reynolds number, defined as

$$R_e = \frac{\rho |u| d_h}{\mu} \tag{18.4}$$

where u is mean-flow velocity and d_h is hydraulic diameter defined as

$$d_h = \frac{4A}{S_x} \tag{18.5}$$

It also knows about matrix porosity, which is a variable imported from the parent component.

Wetted perimeter is the wetted surface area per unit length. The notation S_x is shorthand for $\frac{\partial S}{\partial x}$, where $S(x)$ is the cumulative wetted surface between position zero and x. For domains where S varies linearly with x, wetted perimeter is just the total wetted surface divided by domain length.

18.5.2 Duct Gas Domains

TGxt...DctGas

A *duct* gas domain is used within relatively short flow ducts (generally tubes or rectangular channels) with not so tiny hydraulic diameters. A duct gas domain knows about Reynolds number, Valensi number and turbulence intensity. Reynolds number R_e is defined as in equation (18.4). Valensi number is

$$V_a = \frac{\rho \omega d_h^2}{4\mu} \tag{18.6}$$

It is related to the ratio of hydraulic diameter to viscous penetration depth $\delta_\nu = \sqrt{2\mu/(\omega\rho)})$ according to

$$d_h/\delta_\nu = \sqrt{2V_a} \tag{18.7}$$

Turbulence intensity $\mathcal{T}$ is discussed in detail in section 18.8. Duct domains also know about aspect ratio, which is a variable imported from the parent component. And they introduce the local-loss pressure-drop coefficient K (input variable Klocal), used in frictional pressure-drop equation (18.18).

18.5.3 Variable-Volume Gas Domains

TGxtGnrCyl-Gas

A *variable-volume* gas domain is used within the generic-cylinder model component for modeling piston-cylinder spaces of SCFusion models — i.e. the volume between the pistons and the cylinders in which they ride. A variable-volume gas domain knows about turbulent Reynolds number and Valensi number. Valensi number V_a is the same as in equation (18.6) and turbulent Reynolds number R_t is defined in section 18.8. Flow area A is no longer constant. Rather, it varies with time in accordance with palette-generated volume displacements connected to the area faces of moving-part components.

18.5.4 Combustion-Space Gas Domains

TGxtCmbCyl-Gas

A *combustion-space* gas domain is alternative to the above variable-volume gas domain, also available within the generic-cylinder model component. It is intended for modeling internal-combustion spaces in a simplified way. Input variable FQcombust specifies the combustion firing rate independently of fuel or air flow rates or combustion chemistry. You are responsible for ensuring the conditions for combustion are actually met. The combustion firing rate directly determines source term Q_c in the gas energy equation (*see* section 18.6.3)).

Combustion heating is modeled as a completely reversible process that does not involve entropy generation. It is perfectly feasible to specify a negative firing rate, in which case it amounts to a reversible cooling process.

As a software object, this component descends from the above variable-volume gas domain. This means it inherits all of the physics of the variable-volume gas domain in regard to turbulent Reynolds number, etc.

18.6 Gas Domain Theory

The following equations are somewhat nonstandard in the computational fluid dynamics literature because they are designed specifically for one-dimensional internal flows with space- and time-variable flow area. Most CFD textbooks take a more general world view. So, rather than just quote the equations, we are forced to do a bit of derivation — as concisely as possible. Our starting point is the general Navier-Stokes equations in integral form, as found in Peyret and Taylor [48], except that we shall neglect body forces and relax the prevailing convention that control volume v be fixed. We shall assume that v has fixed inlet and exit boundaries, but allow moving side boundaries — time-variable flow area. The side boundaries are impermeable so that flow enters and leaves only through the inlet and exit boundaries. A rubber tube with space- and

time-variable cross section is a useful thought picture. The principle flow axis is the tube axis. In this control volume the integral gasdynamic equations are:

continuity

$$\frac{d}{dt}\int_v \rho\, dv + \int_s \rho \boldsymbol{n}\cdot\boldsymbol{V}_r\, ds = 0 \tag{18.8}$$

momentum

$$\frac{d}{dt}\int_v \rho \boldsymbol{V}\, dv + \int_s [(\boldsymbol{n}\cdot\boldsymbol{V}_r)\rho\boldsymbol{V} - \boldsymbol{n}\boldsymbol{\sigma}]\, ds = 0 \tag{18.9}$$

energy

$$\frac{d}{dt}\int_v \rho e\, dv + \int_s \boldsymbol{n}\cdot(\rho e\boldsymbol{V}_r - \boldsymbol{\sigma}\boldsymbol{V} - \boldsymbol{q})\, ds = 0 \tag{18.10}$$

where

e	=	$\varepsilon + u^2/2$; mass-specific total gas energy
$\boldsymbol{n}$	=	unit outward normal of s
$\boldsymbol{q}$	=	heat flux vector
s	=	surface of v
t	=	time
v	=	control volume
$\boldsymbol{V}$	=	Newtonian-frame flow velocity vector
$\boldsymbol{V}_r$	=	boundary-relative flow velocity vector
ε	=	mass-specific internal gas energy
ρ	=	gas density
$\boldsymbol{\sigma}$	=	stress tensor

The continuity equation states, in English, that the time rate of change of mass in the control volume equals the rate at which mass leaves through its surface boundaries. By definition, $\boldsymbol{n}\cdot\boldsymbol{V}_r$ is zero along the side surfaces, so the mass enters and leaves only through the inlet and exit boundaries.

The momentum equation is a statement of Newton's second law of motion — the time rate of change of control-volume momentum, less the momentum leaving through its surface boundaries, equals the net force acting on all surfaces. We must be a bit more careful here because momentum (velocity) must be measured in reference to a fixed Galilean frame for Newton's second law to be valid. So the "$\boldsymbol{V}$" in $\rho\boldsymbol{V}$ must continue to be absolute, not relative velocity. We will keep this distinction in mind but, fortunately, it will not affect us much because it applies only to the transverse velocity components (associated with converging, diverging or moving side boundaries) and we are necessarily ignoring transverse components in the momentum equation in a one-dimensional model.

The energy equation states that the time rate of change of control-volume internal plus kinetic energy, less that leaving through its surface boundaries equals the net heat flux through its boundaries plus the mechanical work done on its boundaries. Mechanical work, like momentum, must be measured in an inertial frame so the "$\boldsymbol{V}$" in the tensor product $\boldsymbol{\sigma}\boldsymbol{V}$ must also remain absolute, not relative.

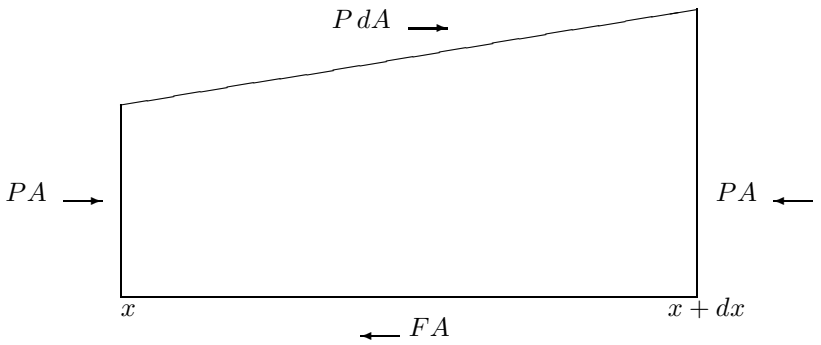


Figure 18.1: Axial surface forces $\int_s -\boldsymbol{N}\boldsymbol{\sigma}$ resolved into pressure and empirical flow-friction terms

We now convert our equations to one-dimensional differential equations in conservative form.

In all equations we replace dv in the time integrals with Adx, where A is flow area and x is the principle flow direction, take the limit as $\Delta x \to dx$ then divide through by dx. For example, the term $\frac{d}{dt}\int_v \rho\, dv$ in the momentum equation will transform to $\frac{\partial}{\partial t}(\rho A)$, and so forth.

For the surface integrals, we must consider the inlet and outlet boundaries separately from the side boundaries on a case-by-case basis.

Continuity Equation Surface integral $\int_s \rho \boldsymbol{n} \cdot \boldsymbol{V}_r\, ds$ in the continuity equation, after taking the limit as $\Delta x \to 0$ and dividing out dx, becomes $\frac{\partial}{\partial x}(\rho u A)$, where u is mean-flow velocity in the x direction.

Momentum Equation Surface integral $\int_s (\boldsymbol{n} \cdot \boldsymbol{V}_r)\rho \boldsymbol{V} ds$ in the momentum equation becomes, simply, $\frac{\partial}{\partial x}(u\rho u A)$ because there is no momentum flux through the side boundaries.

Integral $\int_s -\boldsymbol{n}\boldsymbol{\sigma} ds$, representing the forces applied to the surface, is more involved. Straight off, since this is a one-dimensional model, we will empiricise the stress tensor $\boldsymbol{\sigma}$ into thermodynamic pressure P and a viscous flow resistance term F which we may think of as the force per unit length per unit flow area due to surface shear stress. F amounts to a frictional pressure gradient. Then, pressure forces on the inlet and exit boundaries transform to $\frac{\partial}{\partial x}(PA) - P\frac{\partial A}{\partial x} = A\frac{\partial P}{\partial x}$. Viscous forces on the side boundaries transform into just $-FA$. The sketch in figure 18.1 may help make this clear:

Energy Equation Surface integral $\int_s \boldsymbol{n} \cdot (\rho e \boldsymbol{V}_r) ds$ in the energy equation becomes, simply, $\frac{\partial}{\partial x}(u\rho e A)$ because there is no energy flux through the side

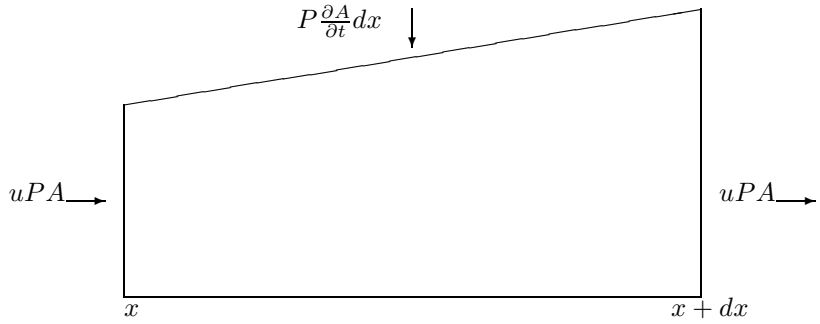


Figure 18.2: Surface work $\int_s -\boldsymbol{n} \cdot \boldsymbol{\sigma V}$ resolved into pressure terms

boundaries.

Integral $\int_s -\boldsymbol{n} \cdot \boldsymbol{\sigma V} ds$ for the inlet and exit boundaries transforms to $\frac{\partial}{\partial x}(uPA)$. No F term is involved since F is applied to the side walls. For the side walls the integral transforms to $P\frac{\partial A}{\partial t}$, the PV work done on the wall. $\frac{\partial A}{\partial t}$ derives from the transverse velocity component of the moving walls. Once again, there is no F term. The reason is because axial velocity is zero at the walls where F is applied. See figure 18.2:

Integral $\int_s -\boldsymbol{n} \cdot \boldsymbol{q} ds$ transforms to a term $\frac{\partial}{\partial x}(qA)$ for the end boundaries, where q is axial heat flux. For the negative z boundary we introduce empirical film heat transfer per unit length Q_w to represent the integral.

In terms of these simplifications our equation set becomes:

continuity

$$\frac{\partial \rho A}{\partial t} + \frac{\partial \rho u A}{\partial x} = 0 \tag{18.11}$$

momentum

$$\frac{\partial \rho u A}{\partial t} + \frac{\partial u \rho u A}{\partial x} + \frac{\partial P}{\partial x} A - FA = 0 \tag{18.12}$$

gas energy

$$\frac{\partial \rho e A}{\partial t} + P\frac{\partial A}{\partial t} + \frac{\partial}{\partial x}(u \rho e A + uPA + q) - Q_w = 0 \tag{18.13}$$

The purpose of these gasdynamic equations is to determine three implicit solution variables, which we hereby choose to be ρ, $\rho u A$ and ρe. The reason for this choice is that all are roughly continuous across area discontinuities and, except for the factor A in the second, are standard in the field. Empirical terms F, Q_w and q are explained further below.

Steady Flow Case

Sage can simulate steady-flow thermodynamic cycles like Joule-Thomson cooling or vapor-compression refrigeration by setting the number of time nodes to zero (i.e. root-model input NTnode = 0). This forces all time-derivatives in the previous equations to vanish leaving a simplified set of equations:

steady continuity

$$\frac{\partial \rho u A}{\partial x} = 0 \tag{18.14}$$

steady momentum

$$\frac{\partial u \rho u A}{\partial x} + \frac{\partial P}{\partial x} A - F A = 0 \tag{18.15}$$

or alternately

$$\frac{\partial P}{\partial x} + \frac{1}{2} \rho \frac{\partial u^2}{\partial x} = F \tag{18.16}$$

steady gas energy

$$\frac{\partial}{\partial x}(u \rho e A + u P A + q) - Q_w = 0 \tag{18.17}$$

According to the steady continuity equation mass flow rate $\rho u A$ must be constant. But velocity u can vary with position if density ρ varies, which it typically does if P or T vary. The alternate steady momentum equation results from replacing $\frac{\partial u \rho u A}{\partial x}$ with $\rho u A \frac{\partial u}{\partial x} + u \frac{\partial \rho u A}{\partial x}$ and noting that the second term vanishes according to the steady continuity equation. Factoring out A and simplifying results in a variation of Bernoulli's law, accounting for flow friction.

18.6.1 Viscous Pressure Gradient F

The term F in the momentum equation takes the place of the viscous terms in the Stokes stress tensor, which cannot be resolved directly in a one-dimensional model. The units of F are force per unit length, per unit flow area (same as $\frac{\partial P}{\partial x}$). F may be formulated in terms of Darcy friction factor f and total local loss coefficient K in a heat exchanger of hydraulic diameter d_h and length L as:

$$F = -\left(f/d_h + K/L\right) \rho u |u| / 2 \tag{18.18}$$

Friction factor may be calculated as an empirical function of instantaneous local flow conditions, according to some published correlation. Loss coefficient is generally a fixed constant, equal to the sum of local loss coefficients for sharp-edged area changes, bends, etc.

Complex Formulation for Ducts

In the case of laminar flow in duct gas domains (*see* section 18.5.2), Sage allows for a phase shift between viscous pressure gradient and velocity by formulating F in terms of a *complex* wall-shear-stress function $\boldsymbol{s} = s_r + is_i$ instead of a conventional friction factor. The following formulation applies this complex formulation to the sinusoidal component of the velocity and also accounts for the contribution of any steady (DC) velocity that may be present:

$$F = -(K/L)\rho u|u|/2 - (8\mu/d_h^2)s_{r0}(u - \Re\boldsymbol{u}) - (8\mu/d_h^2)\Re(\boldsymbol{s}\boldsymbol{u}) \tag{18.19}$$

Here, μ is viscosity and s_{r0} is the steady-flow limit for s_r. The factor $8\mu/d_h^2$ may also be written $2\rho\omega/V_a$. The first term on the right is a smeared-out local-loss per unit length carried over from the previous formulation. The last two terms amount to the superposition of the frictional pressure gradient solutions for a laminar steady-flow momentum equation and an laminar oscillating-flow momentum equation. Both equations are linear in u, so the superposition is valid.

In this formulation Sage's section-mean velocity $u(t)$ is broken into a sinusoidal variation $\Re\boldsymbol{u}$ plus a residual component $u - \Re\boldsymbol{u}$ comprising the mean value plus higher harmonics. The sinusoidal component is calculated from first-harmonic cosine and sine coefficients a_1 and b_1 of the Fourier-series expansion of section-mean velocity, according to the technique outlined in section 8.5.2.

The oscillating-flow wall-shear-stress function $\boldsymbol{s}$ is not a standard concept like friction factor. It is defined in reference [13], for incompressible laminar flow between parallel plates, as $-\boldsymbol{f}'_{wall}/\langle\boldsymbol{f}\rangle$ where $\boldsymbol{f}$ is the complex velocity profile function ($\mathbf{u}(\mathbf{y}) = \boldsymbol{f}(\mathbf{y})\mathbf{e}^{\mathbf{i}\omega\mathbf{t}}$). For other channel shapes, it can be related to the somewhat more standard $\boldsymbol{f}_\nu$ functions used in thermoacoustic theory [65] using

$$\boldsymbol{s} = \frac{iV_a}{2}\left(\frac{\boldsymbol{f}_\nu}{1-\boldsymbol{f}_\nu}\right) \tag{18.20}$$

Both $\boldsymbol{s}$ and $\boldsymbol{f}_\nu$ are functions of Valensi number, or, as thermoacoustic practitioners prefer to say, functions of the ratio of some characteristic channel dimension to viscous penetration depth. (Viscous penetration depth is $\delta = \sqrt{2\mu/(\rho\omega)}$, which is related to Valensi number by $\sqrt{2V_a} = d_h/\delta$.)

The steady-flow (low-V_a) limit s_{r0} is related to the conventional Darcy friction factor by

$$f = \frac{16s_{r0}}{R_e} \tag{18.21}$$

In other words, $16s_{r0}$ is the constant factor that appears in the numerator of the laminar friction factor correlation. This relation can be used to cross-check $\boldsymbol{s}$ calculations against the laminar steady-flow friction-factor literature.

18.6.2 Film Heat Transfer Q_w

The term Q_w in the energy equation is the heat flow per unit length, through the negative z surface, due to film heat transfer. The normal way to formulate

this is

$$Q_w = hS_x(T_w - T) = N_u(k/d_h)S_x(T_w - T) \tag{18.22}$$

where N_u is a Nusselt number, k is gas conductivity, d_h is hydraulic diameter, S_x is the wetted perimeter (wetted surface area per unit length) and $T_w - T$ is the temperature difference between the negative z surface and section-average. This is appropriate for heat transfer in phase with the film temperature difference. Matrix gas domains (*see* section 18.5.1) use this approach.

Complex Formulation for Cylinders

Variable-volume gas domains (*see* section 18.5.3), allow for a phase shift between heat transfer and film temperature difference by using a *complex* Nusselt number formulation

$$Q_w = (k/d_h)S_x\left[N_{u0}(\Delta T - \Re\Delta\boldsymbol{T}) + \Re(\boldsymbol{N}_u\Delta\boldsymbol{T})\right] \tag{18.23}$$

In this formulation the solved gas-wall temperature difference $\Delta T(t) = T_w(t) - T(t)$ is broken into a sinusoidal temperature variation $\Re\Delta\boldsymbol{T}$ plus a residual component $\Delta T - \Re\Delta\boldsymbol{T}$ comprising the mean value plus higher harmonics. The sinusoidal component is calculated from first-harmonic cosine and sine coefficients a_1 and b_1 of the Fourier-series expansion of ΔT, according to the technique outlined in section 8.5.2.

The complex-valued Nusselt number $\boldsymbol{N}_u$ (section 20.9) generally comes from some linear time-varying laminar theory [38]. In such theory the factor $(1/d_h)\boldsymbol{N}_u\Delta\boldsymbol{T}$ in equation (18.23) is just the normal temperature gradient $(\frac{\partial}{\partial y})$ at the wall for the time-varying components of the theoretical temperature solution $\Delta\boldsymbol{T}$. The factor $(1/d_h)N_{u0}(\Delta T - \Re\Delta\boldsymbol{T})$ is the normal temperature gradient of steady-flow theory applied to the residual temperature difference. Sage presumes the two temperature gradients are additive. Applying N_{u0} to the steady component of $\Delta T - \Re\Delta\boldsymbol{T}$ makes sense, but applying it to the higher-harmonic content does not. The only justification for doing so is that otherwise heat transfer would not depend on those higher harmonics, which might lead to convergence issues because of indeterminate temperature values. Complex Nusselt numbers have a significant imaginary component only for high Valensi number laminar flows, when hydraulic diameter is large compared to thermal penetration depth. In this case, the quantity $(k/d_h)S_x\Re(\boldsymbol{N}_u\Delta\boldsymbol{T})$ in equation (18.23), which represents the physical component of the complex heat flow per unit length, may be phase shifted ahead of ΔT by as much as 45 degrees. For turbulent flows or for low Valensi numbers the complex heat-flow formulation essentially reduces to the conventional real formulation (18.22).

Typical complex Nusselt number theories assume the wall temperature T_w is constant so the heat transfer formulation (18.23) is not quite right when the variation in T_w is significant. Even so, equation (18.23) is the preferred Sage formulation because it produces reasonable results in the case where the bounding wall is very thin or has very low specific heat. In that case the heat transfer Q_w is limited by the wall heat capacity, which may approach zero

(insulated surface) in extreme cases. In those cases ΔT and $\Delta \boldsymbol{T}$ both approach zero according to equation (18.23).

Prior to Sage version 12 (July 2021) the complex temperature variation was defined differently (see section 8.5.2) with the potential for errors between the relationship between Qw and ΔT for solutions with large higher-harmonic content. This was a problem in low-temperature GM-style pulse-tube cryocooler models where solid heat capacities were very low, and led to erratic convergence and invalid solutions. The previous formulation also presumed the wall temperature T_w was constant, a bad assumption in such cryocoolers. The present formulation attempts to correct both these problems to the extent possible in a 1-D solution.

Complex Formulation for Ducts

Duct gas domains (*see* section 18.5.2), attempt to improve upon the complex formulation by breaking the fluctuating complex temperature difference $\Delta \boldsymbol{T} = \boldsymbol{T}_w - \boldsymbol{T}$ into compression-driven and advection-driven components $\Delta \boldsymbol{T}_c + \Delta \boldsymbol{T}_a$, arising from pressure variations and temperature gradients carried along by the flow, the two being independent mechanisms. The reason for decomposing temperature like this is that the Nusselt numbers for the two sources of temperature variation differ, compression-driven temperature variations being roughly twice as effective at driving wall heat transfer as advection-driven temperature variations in the high Valensi-number limit, according to first-order oscillating-flow solutions [16, 64].

Because turbulent flow tends to homogenize the duct temperature distribution, the logic of the complex formulation breaks down in turbulent flow. So duct gas domains apply the complex formulation only to laminar flow, transitioning to a steady-flow formulation for turbulent flow. For laminar flow duct gas domains formulate wall heat transfer as

$$Q_{wl} = (k/d_h) S_x \left[N_{ul} (\Delta T - \Re \Delta \boldsymbol{T}) + \Re(\boldsymbol{N}_{uc} \Delta \boldsymbol{T}_c) + \Re(\boldsymbol{N}_{ua} \Delta \boldsymbol{T}_a) \right] \tag{18.24}$$

where N_{ul} is the laminar steady-flow Nusselt number, applied to the mean temperature difference (plus higher harmonics). $\boldsymbol{N}_{uc}$ is the compression-driven complex Nusselt number and $\boldsymbol{N}_{ua}$ is the advection-driven complex Nusselt number (e.g. section 20.6), applied to the compression-driven and advection-driven complex temperature differences. For turbulent flow the formulation is simply

$$Q_{wt} = (k/d_h) S_x N_{ut} \Delta T \tag{18.25}$$

where N_{ut} is the turbulent Nusselt number.

Duct gas domains define a turbulence variable $\mathcal{T}$ that measures the relative level of turbulence, ranging from 0 to 1 (*see* section 18.8). A value of $\mathcal{T} = 0$ is laminar, $\mathcal{T} = 1$ turbulent, with values between 0 and 1 transitional. The

general formulation for all turbulence values, including transitional is

$$Q_w = \begin{cases} Q_{wl} & \text{if } \mathcal{T} \le 0 \\ (1-\mathcal{T})Q_{wl} + \mathcal{T}Q_{wt} & \text{if } 0 < \mathcal{T} < 1 \\ Q_{wt} & \text{if } \mathcal{T} \ge 1 \end{cases} \tag{18.26}$$

The trick is how to evaluate $\Delta \boldsymbol{T}_c$ and $\Delta \boldsymbol{T}_a$, given that the only temperature available directly from the Sage solution is the combined result $\Delta \boldsymbol{T}$. This goes roughly as follows:

- Start with the high V_a limits of the compression-driven and advection-driven temperature variations according to Swift's first-order acoustic solution [64].

  $$\Delta \boldsymbol{T}_c^\infty = \boldsymbol{T}_w - \boldsymbol{T}_c^\infty = -\frac{T\beta}{(\rho C_p)_m}\boldsymbol{P_1} \tag{18.27}$$

  and

  $$\Delta \boldsymbol{T}_a^\infty = \boldsymbol{T}_w - \boldsymbol{T}_a^\infty = -\frac{1}{\omega}\frac{\partial T_m}{\partial x} i \langle \boldsymbol{u} \rangle \tag{18.28}$$

  $\boldsymbol{P}_1$ is the complex pressure variation and $\langle \boldsymbol{u}_1 \rangle$ is the complex section-average velocity variation, both of which are calculate per section 8.5.2. Subscript m denotes time-mean values. $\beta = (\frac{\partial v}{\partial T})_P / v$ is the gas volumetric expansivity.

- Then calculate approximate values at the current V_a as the scaled values $\Delta \boldsymbol{T}_{c0} = \boldsymbol{s}\Delta \boldsymbol{T}_c^\infty$ and $\Delta \boldsymbol{T}_{a0} = \boldsymbol{s}\Delta \boldsymbol{T}_a^\infty$ where $\boldsymbol{s}$ is a complex scale factor derived from Swift's exact solution for temperature (section-averaged) over the entire range of V_a. According to a curve fit to Swift's solutions the following scale factor is a rough approximation for both temperature variations:

  $$\arg(\boldsymbol{s}) \approx \frac{\pi}{2}(1 - |\boldsymbol{s}|) \tag{18.29}$$

  and

  $$|\boldsymbol{s}| \approx \frac{(V_a Pr)^2 - \sqrt{2(V_a P_r)^3} + 2(V_a Pr)}{(V_a Pr)^2 + 24} \tag{18.30}$$

- In order to ensure that the sum $\Delta \boldsymbol{T}_a + \Delta \boldsymbol{T}_c$ is always the actual $\Delta \boldsymbol{T}$, calculate the final values as $\Delta \boldsymbol{T}_c = \Delta \boldsymbol{T}_{c0}/\boldsymbol{R}$ and $\Delta \boldsymbol{T}_a = \Delta \boldsymbol{T}_{a0}/\boldsymbol{R}$ where $\boldsymbol{R}$ is the complex scaling ratio

  $$\boldsymbol{R} = \frac{\Delta \boldsymbol{T}_{c0} + \Delta \boldsymbol{T}_{a0}}{\Delta \boldsymbol{T}} \tag{18.31}$$

  Except when the denominator in equation (18.31) is zero, in which case take $\Delta \boldsymbol{T}_c$ and $\Delta \boldsymbol{T}_a$ both zero.

Explicit Surface Temperature

Although it might appear that ΔT is the independent variable in the preceding heat-flow equations, from the point of view of Sage's solver it is actually Q_w that is the independent variable, implicitly solved so that T_w is continuous across the gas-solid connection at the negative z surface. Sage solves ΔT implicitly from equations (18.22), (18.23) or (18.26), considering heat flux Q_w as given.

18.6.3 Combustion Heating Q_c

In rare instances where it may be appropriate to consider an internal heating source within the gas (*see* the combustion-space gas domain of section 18.5.4), an energy source Q_c is added into energy equation (18.13), on the same footing as the above film heat transfer term, except it represents internally generated heat per unit length. This energy input is treated as a simple energy source independent of any actual combustion chemistry.

18.6.4 Gas Axial-Conduction Heat Flow q

Term q in the gas energy equation is instantaneous axial heat flow which we may formulate as

$$q = -k_e \frac{\partial T}{\partial x} A \tag{18.32}$$

where k_e is the effective gas conductivity — which may exceed molecular conductivity due to turbulent enhancement. This may be formulated in terms of an axial-conductivity enhancement ratio N_k as:

$$q = -N_k k \frac{\partial T}{\partial x} A \tag{18.33}$$

18.6.5 PV Power Flow

In SCFusion machines gas-borne enthalpy flows are important but they are not everything. It is possible to transmit thermal energy through a gas duct from a high-temperature source to a low temperature sink without doing any useful work. That is a foolish thing to do because converting heat to work or vice-versa is the essential purpose of a SCFusion machine. What distinguishes useful enthalpy flow from useless enthalpy flow is the PV power transmitted by the flow. PV power flow is not a solved variable in gas-domain computational grids. It can't be because unlike other solved variables PV power is not defined at a fixed location, but rather moves along with the flow. It can be estimated from available solved variables, but only approximately.

Consider a gas element between a stationary inlet and an imaginary impermeable boundary moving with the flow. In terms the element's pressure P and volume V, the instantaneous PV power delivered to the moving boundary is

$$P\dot{V}$$

The challenge is to estimate $P\dot{V}$ from solution variables at the stationary inlet. The volume V can be written Mv, where $M = \int(\rho uA)$ (the integral of the mass flow rate passing through the stationary inlet) and v is the gas specific volume for the volume element as a whole, which is not necessarily the specific volume at the inlet. The rate of change of Mv can be approximated in terms of the time-average specific volume at the inlet v_m and time average mass M_m as

$$(\dot{M}v) \approx \dot{M}v_m + M_m\dot{v}$$

In the first term on the right the rate of change of M is just (ρuA). In the second term on the right M_m is the time integral of ρuA over a full cycle, which is zero for oscillatory flow. If flow is steady then $\dot{v}$ is zero. In either case it is reasonable to ignore the second term. If flow is a oscillatory with a DC flow component then it is not so clear, but this is just an approximation so it is convenient to ignore the second term anyway. Without the second term the approximate instantaneous PV power flow works out to

$$P\dot{V} \approx P(\rho uA)v_m \tag{18.34}$$

The time average of the right-hand side is how Sage gas domains approximate PV power flow.

18.6.6 Equations of State

Tying the gasdynamic equations together is an equation of state which serves to formulate the dependent variables P and T as functions of independent variables ρ, ρuA and ρe — specifically as $P(\rho, T)$ and $T(\rho, \rho e, \rho uA)$. The gas-property objects defined in the SCFusion root model provide these functions (see chapter 15).

18.6.7 Solution Method

The gasdynamic equations are discretized over a grid of points (x_i, t_j) uniformly spaced throughout the domain and over the cycle period. Upon this grid, variables ρ, ρuA and ρe, are solved implicitly or interpolated explicitly according to the logic of a *staggered-grid* formulation. The essential idea is to solve ρuA at alternate nodes from ρe and ρ to avoid solution instability and produce global conservation of mass, momentum and energy. Solution staggering is with respect to the spatial direction only.

The concept of control volumes is helpful for keeping things straight in the staggered-grid scheme. Control volume boundaries can be thought of as lying on alternate spatial nodes. One can think of it this way: ρuA is solved at the control volume boundaries, while ρ and ρe are solved at the midpoints. The entire domain comprises an integral number of control volumes or, including endpoints, an odd number of nodes greater than or equal to three. Global conservation of energy and mass automatically hold for the solved variables ρ and ρe if one is careful to use first-order central differencing in the continuity

and energy equations so that, in effect, quantities leaving one control volume enter the adjacent one.

Interior Points

Those nodes lying strictly within the domain interior give rise to implicit finite-difference equations for the solved variables ρuA, ρ and ρe. Variables that are not solved at interior nodes are interpolated. In the case of ρuA and ρe, interpolation takes the form of a central polynomial interpolation of neighboring solved values. The degree of the interpolation (linear or cubic) can be specified in the Options|Sage dialog. In the case of ρ, interpolation is a weighted average of central interpolation and interpolation with the set of interpolation points shifted toward the upwind direction. This weights the interpolation toward the currently upwind direction in order to stabilize the solution. You can specify the relative fraction of upwind interpolation with input variable UpwindFrac, should the default value be too small. Without the upwind weighting, density and temperature profiles can sometimes show a static zig-zag instability in gas domains where there is not a lot of thermal diffusion to dampen the effect.

Endpoint Boundary Points

These are the nodes lying at the negative and positive domain boundaries — the flow inlets. Solved variables ρ, ρuA, ρe get special treatment at such nodes, depending on whether or not there is a flow connection to an adjoining gas domain or not. This is explained fully in the following section. Other variables are simply extrapolated from the interior.

Minimal Requirement for Resolving Pressure Drop

One idiosyncrasy of the solution method is worthy of note: Pressure-drop, whether due to flow resistance or acceleration, will be ignored in any gas domain containing only one control volume. According to the staggered-grid solution scheme of the gas domain, the momentum equation will never be invoked when only one cell is present, leaving mass flow rate determined solely by Bernoulli's law in adjacent flow connectors. This behavior is not necessarily bad. It may even be desirable in some cases. But it is something to keep in mind.

The Mean Pressure Constraint

Although it may not be immediately apparent, the solution scheme so far outlined will result in a singular system of implicit equations. In physical terms, there is no mechanism to establish the mean pressure.

Resolving this dilemma requires a close inspection of the mass-continuity equations enforced in the gas domain interior. Taken together, these equations imply that the cyclic-net mass passing through the endpoint boundaries is zero, meaning that we cannot independently set boundary mass flow rate at both endpoint boundaries for all time nodes. Looking at this in another way, if we do

specify boundary mass flow rates at all time nodes (such that cyclic-net mass through boundaries is zero) and enforce the mass-continuity equation at all but one interior cell, say (i_0, j_0), then mass continuity is automatically guaranteed for (i_0, j_0). This frees us to replace the mass continuity equation at (i_0, j_0) with a mean-pressure constraint instead — which is what we do.

The penultimate sentence in the preceding paragraph is really a mathematical theorem that follows from the conservation property of central spatial differencing in our staggered-grid scheme and the time-ring-differencing property

$$\sum_j \frac{\partial f}{\partial t}(i, j) = 0 \tag{18.35}$$

which holds no matter what. To show the mass-continuity equation indeed holds at (i_0, j_0) we first use the time ring differencing property to conclude that

$$\frac{\partial \rho A}{\partial t}(i_0, j_0) = -\sum_{j \neq j_0} \frac{\partial \rho A}{\partial t}(i_0, j) \tag{18.36}$$

we immediately replace the right-hand side using

$$-\sum_{j \neq j_0} \frac{\partial \rho A}{\partial t}(i_0, j) = \sum_{j \neq j_0} \frac{\partial \rho u A}{\partial x}(i_0, j) \tag{18.37}$$

$$= \sum_j \frac{\partial \rho u A}{\partial x}(i_0, j) - \frac{\partial \rho u A}{\partial x}(i_0, j_0) \tag{18.38}$$

the first part of the above equation follows from the presumed local mass continuity for all nodes but (i_0, j_0) and the second part is just regrouping. The first term on the right may be further resolved into

$$\sum_j \frac{\partial \rho u A}{\partial x}(i_0, j) \propto \sum_j \rho u A(i_0 + 1, j) - \sum_j \rho u A(i_0 - 1, j) \tag{18.39}$$

$$\propto \sum_j \rho u A(2M, j) - \sum_j \rho u A(0, j) \tag{18.40}$$

$$\tag{18.41}$$

which follows from central differencing formula used in the computational grid, then summing the local mass-continuity equations over j at successive columns in the grid from $i_0 + 2$ to the positive boundary $i = 2M$ and from $i_0 - 2$ to the negative boundary $i = 0$. Combining the above results we are left with

$$\frac{\partial \rho A}{\partial t}(i_0, j_0) + \frac{\partial \rho u A}{\partial x}(i_0, j_0) \propto \sum_j \rho u A(2M, j) - \sum_j \rho u A(0, j) \tag{18.42}$$

which is the mass-continuity equation at (i_0, j_0) we are after, provided the right-hand side evaluates to zero, which it will if the net mass flow through the endpoint boundaries is zero.

In a system comprising many gas domains joined by flow connectors, the above result still applies to the system as a whole. So long as zero net mass flows through the ultimate boundaries of the system, we are free to replace a local mass-continuity equation at exactly one node (i_0, j_0) with a mean-pressure constraint. In a Sage model the gas domain to which the mean-pressure constraint applies is the one attached to the pressure-source component (*see* section 18.4).

18.7 Flow Connector Theory

Flow area may change abruptly between adjacent gas domains, requiring special consideration in Sage's one-dimensional formulation. Between every two gas domains that are connected together there is a special flow connector object that manages the connection behind the scenes. There are three conditions to satisfy: conservation of mass, momentum and energy. Conservation of mass is automatic because of the choice of mass flow rate ρuA as a solution variable. Sage uses momentum continuity (Bernoulli's law) as the implicit equation for solving mass flow rate in flow connectors. This is entirely consistent with the staggered-grid treatment for ρuA in the bounding gas domains. This leaves only energy conservation. Here Flow connectors pass responsibility back to the bounding gas domains but help them out by keeping track of the enthalpy on the other side of the connector and the thermal conduction through the connector. The bottom line is that because of a Bernoulli pressure change, mass density ρ and energy density ρe may jump slightly across the connector, but energy is always conserved. The inter-domain flow connector model resides in unit FlowCnct.

18.7.1 Bernoulli's Law

Momentum flow continuity is a limiting case of the integral-form momentum equation applied to the connector volume itself when the $-$ boundary approaches the $+$ boundary in the stream tube $\mathbf{V}$ shown in figure 18.3. The only terms that survive in the limit are the net momentum flow leaving through the boundaries $[u\rho uA]_+ - [u\rho uA]_- = \rho uA(u_+ - u_-)$ and the net pressure force acting on the boundaries $P_-A_- + P_m(A_+ - A_-) - P_+A_+$, the middle term being the axial component of force on the side walls with P_m some intermediate pressure. Taking $P_m = (P_- + P_+)/2$, expanding pressure terms and simplifying yields the following momentum flow continuity condition

$$\rho uA(u_+ - u_-) + (P_+ - P_-)A_m = 0 \tag{18.43}$$

where $A_m = (A_- + A_+)/2$. This can be put in the more familiar form of Bernoulli's law by using the previous mass flow continuity condition to replace ρuA with $\rho_m u_m A_m$, where $u_m = (u_- + u_+)/2$ and ρ_m is some intermediate

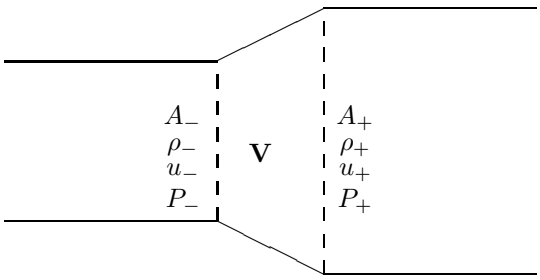


Figure 18.3: Control volume v used to evaluate momentum flow continuity; A = flow area, ρ = density, u =velocity, P = pressure.

density, leaving

$$P_+ - P_- = \rho_m u_m (u_- - u_+) = \rho_m \frac{u_-^2 - u_+^2}{2} \tag{18.44}$$

Flow connectors use this equation to implicitly determine mass flow rate $\rho u A$ through the connector, thereby ensuring that Bernoulli's law governs the pressure change across the connector.

18.7.2 Energy Continuity

Energy flow continuity is similarly derived by integrating the gas energy equation across a transition volume between adjacent components and then taking the limit as volume approaches zero. The result is

$$[(\rho e + P)uA + q]_+ - [(\rho e + P)uA + q]_- = 0 \tag{18.45}$$

A problem with this equation is that advected enthalpy $(\rho e + P)uA$ is mixed up with thermal conduction q, making them impossible to resolve individually with single equation. So flow connectors separate the energy continuity equation into two parts, enthalpy continuity and conduction continuity, to be separately enforced

$$[(\rho e + P)uA]_+ - [(\rho e + P)uA]_- = 0 \tag{18.46}$$

$$q_+ - q_- = 0 \tag{18.47}$$

Enthalpy Part

How should enthalpy continuity equation (18.46) be implemented in Sage's solution scheme? First of all it is helpful to write stagnation enthalpy flow in the form

$$H = (\rho e + P)uA = \rho u A(\rho e + P)/\rho = (\rho u A)\, h \tag{18.48}$$

where h is the mass specific enthalpy of the gas, a thermodynamic state property. In this form it is clear that h must be conserved across the flow connector. The

question is how? One problem is that there are two h's to determine, one on either side of the flow connector. Another problem is that h is not a primary solution variable. It is calculated in terms of primary solution variables ρ and ρe and dependent variable P. How can one equation (enthalpy continuity) determine all these variables? The answer is that it cannot. In the Sage solution scheme enthalpy continuity determines only the volumetric energy density ρe at the downstream endpoint. Other variables are evaluated according to this table:

variable	**upstream endpoint**	**downstream endpoint**
ρ	extrapolated	EOS
ρe	extrapolated	enthalpy continuity
P	extrapolated	extrapolated
T	extrapolated	extrapolated

The word "extrapolated" in the above table means that the variable is extrapolated from its values at interior nodes located at the centers of computations cells. Energy density ρe (which generally changes with time) is chosen as the variable used to enforce enthalpy continuity because it is a property carried with the flow and the principle of upstream causality suggests that upstream conditions determine downstream conditions, not vice-versa. In the case of pressure P it makes sense to extrapolate its value at the endpoints regardless of flow direction because pressure information propagates at sonic velocity which is *not* carried with the flow in the low subsonic flow range where all good SCFusion machines operate. In the case of mass density ρ the word "EOS" means that Sage solves it implicitly so that ρ, T and P are consistent with the fluid equation of state. To avoid a singular solution Sage cannot also evaluate T according to the equation of state so instead it extrapolates T from the interior solution. The result of all this is for incoming flow (downstream endpoint) Sage uniquely determines both ρ and ρe from the equation of state and the principle of enthalpy conservation.

Since Sage extrapolates temperature at gas domain endpoints from the interior solution there may be a slight discontinuity of temperature across flow connectors. This is more of an apparent problem than a real problem because none of the governing equations depends on T values at cell boundaries.

Prior to version 13 (October 2024) Sage implemented enthalpy conservation differently. Mass density ρ rather than energy density ρe enforced enthalpy conservation and ρ was not guaranteed to be consistent with T and P according to the equation of state. This sometimes caused trouble with ρ and T drifting out of the range of property tables. There was also a problem that came to light after relaxing the requirement that mass-specific internal energy e values always be positive, to be more consistent with REFPROP fluid data files (according to choice of energy integration constant). This means the value of ρe can be negative and possibly take on the value $-P$, making the value of enthalpy flow $H = 0$. This actually came close to happening in a test model, which would

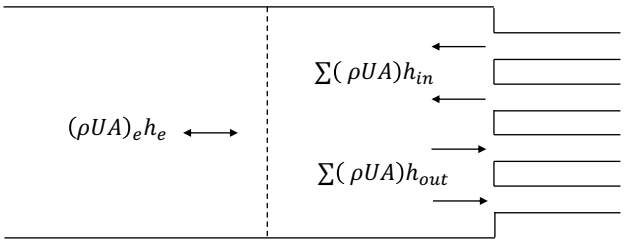


Figure 18.4: Inlets at positive end of a gas domain with enthalpy flow $(\rho UA)_e h_e$ based on internal solution values balancing $(\rho UA)_{in} h_{in}$ and $(\rho UA)_{out} h_{out}$ entering or exiting from/to external gas domains via flow connectors.

have produced a divide-by-zero error in the previous formulation. The present formulation avoids this problem by separating the ρe and P parts of enthalpy.

Complication of Multiple Connections

Gas domains support multiple connections at their endpoint boundaries. The simplest such case is a Y-branch where one domain splits into two or two domains converge to one, although more complicated connections are possible. Gas domains manage this complexity by logically breaking their endpoint boundaries into a number of distinct *flow inlets*, one for each flow connector, as illustrated in figure 18.4. The flow connectors communicate with the individual flow inlets which then communicate with the gas domain. Flow inlets are software objects encapsulated together with the gas domain.

Under this scheme, gas domains obtain their endpoint mass flow rate by polling all flow inlets to return the mass flow rate from their associated flow connectors and summing the results. Since mass is a conserved quantity and mass flow rate is a solution variable this is completely straight forward.

It is more complicated for gas domains to evaluate the correct energy density which, according to the above logic, must be based on enthalpy continuity. The problem is that a gas domain with multiple inlets can be simultaneously downstream of one flow connector while upstream of another. In other words, one inlet can see incoming flow while the other sees outgoing flow.

At this point it is convenient to denote variables at the gas domain endpoint by the subscript e and those within the individual flow inlets by subscript i.

Enthalpy conservation requires that the sum-total of stagnation enthalpy flows passing through the inlets must equal the stagnation enthalpy flow computed at the gas-domain endpoint and used in its interior solution. This latter quantity is always calculated from gas-domain endpoint variables as

$$H_e = [\rho u A]_e h_e \tag{18.49}$$

Outgoing enthalpy terms for the inlets are based on a mixed enthalpy h_m

solved according to enthalpy conservation

$$H_i = [\rho u A]_i h_m \tag{18.50}$$

Incoming enthalpy terms for the inlets are based on solution values managed by the gas domains upstream of the flow connectors from which they originate

$$H_i = [\rho u A]_i h_i \tag{18.51}$$

In terms of the above definitions the condition of enthalpy-flow continuity may be written

$$0 = H_e - \sum_i H_i = h_e(\rho u A)_e - h_m \sum_{\text{out}} (\rho u A)_i - \sum_{\text{in}} H_i \tag{18.52}$$

where designations *in* (in-flowing) and *out* (out-flowing) are relative to the local gas domain.

case in-flowing $[\rho u A]_e$ In this case the principle of upstream causality requires that the endpoint enthalpy h_e equal the mixed enthalpy h_m. The enthalpy continuity equation becomes

$$0 = h_m \left((\rho u A)_e - \sum_{\text{out}} (\rho u A)_i \right) - \sum_{\text{in}} H_i \tag{18.53}$$

According to mass flow rate continuity $(\rho u A)_e - \sum_{\text{out}} (\rho u A)_i = \sum_{\text{in}} (\rho u A)_i$ so the previous equation may be simplified to

$$0 = h_m \left(\sum_{\text{in}} (\rho u A)_i \right) - \sum_{\text{in}} H_i$$

Or

$$h_m = \sum_{\text{in}} H_i / \sum_{\text{in}} (\rho u A)_i \tag{18.54}$$

To make use of mixed enthalpy h_m within the solution it is converted to an equivalent value for ρe at the endpoint. First write h_m as

$$h_m = e_m + P/\rho \tag{18.55}$$

where e_m is the mixed energy density e_m and ρ is the endpoint density extrapolated from the interior solution. Then solve for e_m as

$$e_m = h_m - P/\rho$$

and write the value of endpoint ρe that satisfies enthalpy continuity as

$$(\rho e)_e = \rho e_m = \rho h_m - P \tag{18.56}$$

case out-flowing $[\rho uA]_e$ In this case the endpoint enthalpy h_e is extrapolated from the interior gas domain solution and the explicit solution of enthalpy conservation equation (18.52) for mixed enthalpy is

$$h_m = \left(h_e(\rho uA)_e - \sum_{\text{in}} H_i \right) / \sum_{\text{out}} (\rho uA)_i \tag{18.57}$$

The value ρe_m can be solved from h_m as before but since both local values ρ and ρe are extrapolated from the interior solution, it is not used for the local solution but only passed to flow connectors for use in downstream gas domains.

Thermal Conduction Part

Thermal conduction continuity is much easier than enthalpy continuity since it does not involve so many solution variables or change with flow direction. Flow connectors look at the conductance (conductivity times area kA) and temperature gradients of the adjoining gas domains in order to decide the appropriate value of q allowed to "pass through" the connection. That is, flow connectors explicitly calculate q which adjoining gas domains use at their endpoints.

This approach differs from the way solid domains are connected in Sage. In solid domains the connector calculates q *implicitly* according to the requirement that temperature is continuous across the connection. In gas domains this is not necessary because temperature continuity is already established by enthalpy continuity and the equation of state in the adjoining gas domains. Not only is it not necessary but if temperature continuity were separately imposed by the thermal-conduction continuity then it would completely disrupt the solution.

However, explicitly calculating thermal conduction is not completely trivial. One problem is that the adjoining gas domains may have wildly different conductances. To avoid disrupting the solution in the low-conductance domain, flow connectors calculate q based on the smaller of the two conductances. A second problem is calculating a representative temperature gradient. Sages uses a finite-difference approximation based on the temperatures T_+, a distance dx_+ beyond the endpoint of the positive domain, and T_-, a distance dx_- before the endpoint of the negative domain. dx_+ and dx_- are the node spacings in the two gas-domain grids. The final formulation for calculating q is

$$q = -\min(kA_-, kA_+) \left(\frac{T_+ - T_-}{dx_+ + dx_-} \right) \tag{18.58}$$

Doing it this way has several important properties:

- Satisfies the requirement that $q = 0$ if either $kA = 0$
- Continuous if the smaller kA changes
- Preserves the temperature gradient for the case where the temperature gradients of the two gas domains are the same.

Another problem is the complication of multiple connections. To deal with this, gas domains use the same logical breakdown into flow inlets as previously discussed for enthalpy conservation. Each inlet is allocated an equal fraction of the total gas domain conductance which it passes to the flow connector. For example, for a gas domain with N endpoint connections, each inlet is allocated conductance kA/N. The gas domains then obtain their endpoint conduction by polling all inlets to return the q's from their associated connectors and summing the result. This is similar to the way they obtain endpoint mass flow rate and works for the same reason: thermal conduction is a conserved quantity.

Conduction blocked by default In evaluating kA_- or kA_+ values at gas domain endpoints Sage factors in the empirical multiplier input KmultBnd. The default value for KmultBnd is zero, which has the effect of blocking thermal conduction through the endpoints. One reason for this is that it speeds up the solution process by limiting the scope of solution-variable dependencies between gas domains. Another reason is that thermal conduction is generally negligible between most gas components in a SCFusion model. When you want to model conduction between gas domains be sure to set KmultBnd equal to 1 in both domains.

18.8 Turbulence Models

Knowing the state of turbulence is critical for computing the empirical relationships for friction factor f, Nusselt number N_u and axial conductivity ratio N_k used in the various gas-domain model components, of which there are three basic types: matrix, duct and variable-volume.

18.8.1 Matrix Gas Domains

Matrix gas domains presume quasi-steady flow, meaning that local Reynolds number R_e completely characterizes the turbulence state. Entrance effects and time-constants for boundary-layer growth are neglected. In most porous materials, the issue of turbulence is moot anyway since relatively simple formulations for friction factor f, Nusselt number N_u and enhanced axial conductivity ratio N_k typically apply over the entire Reynolds number range of interest. In non-porous matrix materials, though, (notably wrapped foils) the state of turbulence does matter because formulations for f, N_u, and N_k tend to be completely different for laminar vs turbulent flow.

When turbulence matters, matrix gas domains correlate dimensionless groups as a weighted average of the laminar and turbulent cases. For example friction factor is computed from a laminar value f_l and a turbulent value f_t as

$$f = (1 - W)f_l + W f_t \tag{18.59}$$

The idea is that weight function W varies smoothly between 0 and 1 as flow goes from laminar to turbulent. Smooth is better than abrupt for numerical

convergence reasons.

The weight function is formulated in terms of a normalized and zero-shifted Reynolds number R_e^* defined as

$$R_e^* \equiv \frac{R_e - R_l}{R_t - R_l} \tag{18.60}$$

Here $R_l = 2300$ and $R_t = 10,000$ are the critical below-which-is-laminar and above-which-is-turbulent Reynolds numbers, conventional in the steady flow literature. Note that R_e^* varies from 0 to 1 in the transitional Reynolds-number range. The turbulence weight function is now defined as

$$W(R_e^*) = \begin{cases} 0 & \text{if } R_e^* \le 0 \\ 2R_e^{*2} & \text{if } 0 < R_e^* \le 1/2 \\ 1 - 2(1 - R_e^*)^2 & \text{if } 1/2 < R_e^* \le 1 \\ 1 & \text{if } 1 < R_e^* \end{cases} \tag{18.61}$$

This formulation is first-derivative continuous everywhere, which should be smooth enough for Sage.

18.8.2 Duct Gas Domains

In duct gas domains we no longer have the luxury of neglecting entrance effects or time constants for boundary-layer growth. In addition to Reynolds number R_e, the state of turbulence depends on Valensi number V_a, elapsed time since flow reversal and convective triggering, which is turbulence triggered by a slug of turbulent fluid that enters through one of the inlets and is carried to the point of observation by the mean flow.

Modeling turbulence in such ducts is a tall order for a one-dimensional model. Our goal is keep things as simple as possible, relying heavily on empirical observation. Researchers Terry Simon, Joerg Seume and others at the University of Minnesota [58], [60], [59], [61] have observed that transition to turbulence is delayed in proportion to $\sqrt{V_a}$ for large Valensi numbers and that flow is always laminar according to:

$$\text{always laminar if } R_e/R_l < \max(\sqrt{V_a/V_c}, 1) \tag{18.62}$$

where $R_l \approx 2300$ is a critical Reynolds number and $V_c \approx 10$ is a critical Valensi number, below which flow is essentially similar to steady flow. Above the *always laminar* condition, the Minnesota researchers observed turbulence transition due mainly to convective triggering, with turbulence persisting until the time of flow reversal, but generally not beyond, leaving the velocity profile uniformly zero across the tube cross-section at the time of flow reversal. The Minnesota observations were generally confined to mean-flow tidal displacements greater than the tube length. University of Michigan and MIT researchers R. Akhavan, R. D. Kamm and A.H. Shapiro, [3], support these conclusions and make additional observations pertaining to the spontaneous transition to turbulence of fluid in the duct interior, not affected by disturbances propagating from the inlets. They

observed that turbulence was generally delayed during most of the acceleration phase of the cycle but exploded suddenly near the end of this phase (just before or just after the time of peak velocity), with turbulent flow persisting throughout the deceleration phase, until the time of flow reversal. They saw no turbulence for peak $R_e/\sqrt{V_a}$ less than about 700 – 800, or so, consistent with the above *always laminar* condition for $V_a > V_c$. And when turbulence was present, they, like the Minnesotans, observed that the velocity profile was uniformly zeroed out at the time of flow reversal. Then during the following acceleration phase the velocity profile was very close to the exact solution for start-up from rest of laminar flow in a tube, until the time of turbulence transition. They attributed the delay in turbulence transition to the stabilizing effects of fluid acceleration, although the viewpoint here (and of the Minnesota researchers) is that it just a matter of the time required for the boundary layer to grow to an unstable thickness.

To attempt to capture all these effects, in duct gas domains there is a special turbulence intensity variable $\mathcal{T}$ added to the set of state variables and solved implicitly from its own partial differential equation. This turbulence variable $\mathcal{T}$ functions much like the weight function W in the matrix gas-domain model. Again using friction factor as an example, we compute local instantaneous friction factor from the weighted average of a laminar value f_l and a turbulent value f_t

$$f = (1 - \mathcal{T})f_l + \mathcal{T} f_t \tag{18.63}$$

Turbulence intensity $\mathcal{T}$ is to vary smoothly between 0 and 1 as flow goes from laminar to turbulent.

We may think of $\mathcal{T}$ as a normalized version of $\rho\kappa$, where κ is the time-averaged turbulence kinetic energy per unit mass

$$\kappa = 1/2 \left\langle u'^2 + v'^2 + w'^2 \right\rangle \tag{18.64}$$

More precisely, we might think of $\mathcal{T}$ as roughly proportional to $\rho\kappa/(\rho_0 u^2) \propto \kappa/u^2$, where $u(x,t)$ is the mean-flow velocity. We can estimate the constant of proportionality by noting that for the fully turbulent state ($\mathcal{T} = 1$), the RMS velocity fluctuation ($\sqrt{2\kappa}$) is about $0.08u$, according to [61]. So a reasonable estimate for $\mathcal{T}$ would be

$$\mathcal{T} \approx \frac{2\kappa}{(0.08u)^2} \tag{18.65}$$

Because of the variable nature of the normalization quantity u^2, $\mathcal{T}$ is not, strictly speaking, a tangible quantity carried along with the flow (like $\rho\kappa$ itself is). Nonetheless, it is convenient to overlook this and presume that $\mathcal{T}$ is governed by a one dimensional differential equation like our other gasdynamic equations

$$\frac{\partial \mathcal{T} A}{\partial t} + \underbrace{\frac{\partial \mathcal{T} u A}{\partial x}}_{\text{convection}} = \underbrace{G(R_e, V_a)}_{\text{generation}} - \underbrace{D(R_e, V_a)}_{\text{decay}} \tag{18.66}$$

The only problems are deciding on inlet boundary conditions and decay and generation terms to impart the desired properties to the solution $\mathcal{T}$ — namely,

that $\mathcal{T}$ be 0 when flow is to be laminar, 1 when flow is be turbulent and somewhere in between during transition. The convection term automatically takes care of convective triggering.

Inlet Boundary Conditions

The University of Minnesota work suggests that incoming fluid is always turbulent. This turbulence is either the result of high upstream turbulence level or continuously generated by separation of incoming flow at the inlet. Denoting the incoming turbulence level by $\mathcal{T}_{in}$, it appears then that a value $\mathcal{T}_{in} = 1$ is reasonable, although the value is adjustable to any value between 0 and 1 with input variables TbInNeg and TbInPos, which define the incoming turbulence level at the two ends of the duct. For outgoing flow, it makes physical sense to extrapolate $\mathcal{T}$ at the end boundary from interior values, denoted by $\mathcal{E}(\mathcal{T})$. There is no problem with $\mathcal{T}$ discontinuity at flow reversal since only the product $\mathcal{T}uA$ (always zero at flow reversal) affects the interior solution. The value of $\mathcal{T}$ at either inlet boundary is computed as

$$\mathcal{T} = \begin{cases} \mathcal{T}_{in} & \text{inflow} \\ \mathcal{E}(\mathcal{T}) & \text{outflow} \end{cases} \tag{18.67}$$

Transition

The universal observation seems to be that in turbulent oscillating flow the steady velocity profile zeros out uniformly across the duct cross-section at flow reversal. Essentially, each half cycle begins anew from rest. Simon et. al. argue in [61] that under these conditions transition should occur when the momentum-thickness Reynolds number

$$R_\delta = \frac{u\delta}{\nu} \tag{18.68}$$

reaches a critical value. u is the instantaneous mean flow velocity, $\nu = \mu/\rho$ is kinematic viscosity and δ is the so-called momentum thickness, roughly the boundary-layer thickness, but defined more precisely in Schlichting ([55], equation 7.38, p. 141) for a flat plate as

$$\delta = \int_{y=0}^{\infty} \frac{u}{U_\infty}\left(1 - \frac{u}{U_\infty}\right) dy \tag{18.69}$$

Here, y is the normal distance from the wall and $u(y)$ is velocity. Reference [2] shows that for undisturbed flow, R_δ may range between 1200 and 3000 from the start to end of transition, while the range of transition drops to R_δ between 200 and 400 with as little as 4% free-stream turbulence level.

The boundary-layer growth for a flat plate accelerated sinusoidally from rest can be seen in the velocity profiles shown in figure 18.5. In the figure the fluid is initially at rest with the wall suddenly started moving with velocity $u(0,t) = \sin\omega t$ at $\omega t = 0$. This is equivalent to the present problem of interest, which is a fixed wall with free-stream velocity suddenly started moving with velocity

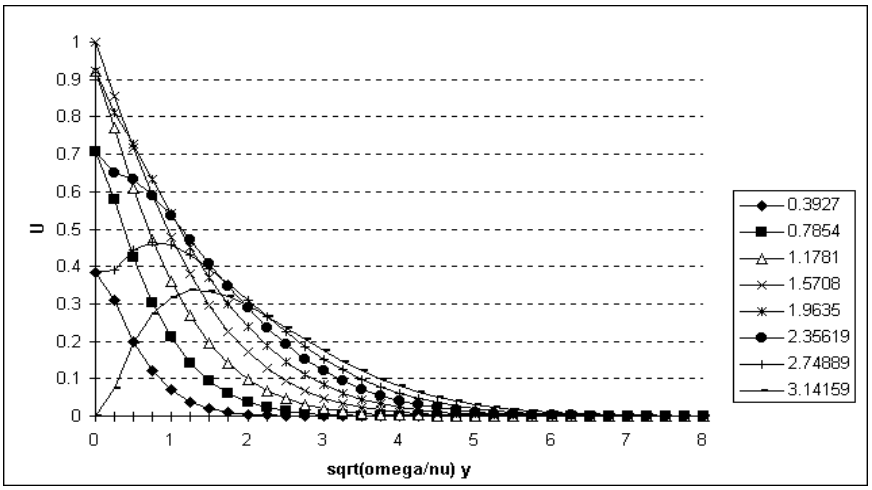


Figure 18.5: Velocity profiles at equally-spaced times $0 \leq \omega t \leq \pi$ for viscous flow governed by $\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial y^2}$. Fluid initially at rest ($u(y,t) = 0$, for $t < 0$) with wall boundary condition $u(0,t) = \sin \omega t$ for $0 \leq \omega t \leq \pi$. From a numerical solution based on the Crank-Nicolson implicit method.

$u(\infty, t) = -\sin \omega t$ at $\omega t = 0$ (driven by a sinusoidal pressure gradient $\frac{\partial P}{\partial x}$), by the addition of $-\sin \omega t$ to the velocity solution everywhere. The advantage of the velocity profiles shown is that they better show the viscous boundary layer thickness.

The momentum thickness for the case of flow accelerated sinusoidally from rest is shown in figure 18.6. Except for a brief time at the beginning of the flow and lasting until well after the time of peak velocity, the momentum thickness grows linearly with time in close approximation to

$$\delta = \sqrt{\nu/\omega}(0.15 + 0.225\,\omega t) \text{ if } 0.3 \leq \omega t \leq 2.3 \tag{18.70}$$

So for sinusoidal flow starting from rest the momentum-thickness Reynolds number, and hence the point of turbulence transition, may be easily calculated. Transition so predicted is probably valid for $0 \leq \omega t \leq 2.3$, because in this range the velocity profiles qualitatively resemble those of suddenly-accelerated flow ($u(\infty, t) = u_0$ for $\omega t \geq 0$), the conditions for which the correlation between momentum-thickness Reynolds number and turbulence transition were originally established. Beyond $\omega t = 2.3$, though, the velocity profile begins to show a pronounced maximum at some distance from the wall which, due to a mathematical technicality, has the effect of producing diminishing, then negative, momentum thicknesses even though the actual viscous-layer thickness continues to grow, as can be seen in figure 18.5. So the momentum-thickness Reynolds number cannot be used as a predictor of transition during the later stages of the flow half-cycle.

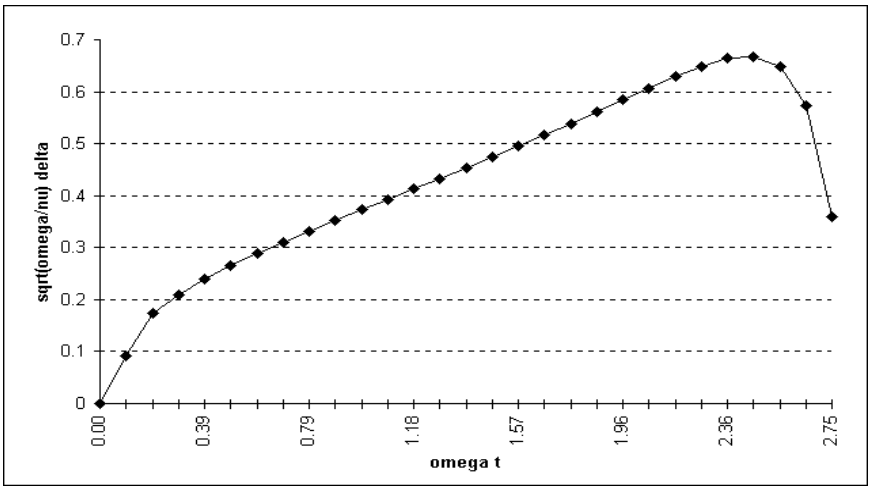


Figure 18.6: Growth of dimensionless momentum thickness $\sqrt{\omega/\nu}\delta$ with time ωt for flow accelerated sinusoidally from rest. Calculated according to formal definition (18.69) with integration performed numerically based on a finite difference solution of the velocity profile.

In order to discover critical value of the momentum-thickness Reynolds number, first write it in terms of the usual hydraulic-diameter Reynolds number as

$$R_\delta = Re\,\delta/d_h \tag{18.71}$$

Then replace factor δ/d_h by substituting approximation (18.70) for δ and expand into dimensionless variables, leaving

$$R_\delta \approx \frac{0.075 + 0.112\,\omega(t-t_0)}{\sqrt{V_a}} R_e \tag{18.72}$$

To avoid confusion, the elapsed time since flow reversal is written above and hereafter as $(t-t_0)$. And to save time, the numerator of the coefficient of R_e is hereafter written

$$F(t-t_0) \equiv 0.075 + 0.112\,\omega(t-t_0) \tag{18.73}$$

At this point, there is no need to work with R_δ directly at all. Taking the critical value of R_δ as an unknown constant $R_{\delta c}$, the turn-on time for turbulence generation may be stated

$$\text{laminar until } R_e > R_{\delta c}\frac{\sqrt{V_a}}{F(t-t_0)} \tag{18.74}$$

Even though approximation $F(t-t_0)$ is not valid for $\omega(t-t_0) > 2.3$, it will do no harm because transition will always occur before this, if at all, before or slightly after the time of peak velocity at $\omega(t-t_0) = \pi/2$.

The constant $R_{\delta c}$ may be evaluated according to the observed always-laminar condition (18.62) for high Valensi numbers ($V_a > V_c$), which states that for a cycle with peak Reynolds number R_{em} below about $730\sqrt{V_a}$, the flow remains laminar for the whole cycle, whereas for any higher R_{em}, flow becomes at least partly turbulent. On the other hand, substituting $R_{em}\sin(\omega(t-t_0))$ for R_e in momentum-thickness condition (18.74), the implication is that flow remains laminar until $R_{em}(\sin(\omega(t-t_0))F(t-t_0))$ exceeds $R_{\delta c}\sqrt{V_a}$. So, the largest R_{em} for which flow remains always laminar may be evaluated by setting $\omega(t-t_0) \approx 1.937$, where $\sin(\omega(t-t_0))F(t-t_0) \approx 0.274$ attains its maximum value. It follows that $730\sqrt{V_a}$ must equal $R_{\delta c}\sqrt{V_a}/0.274$, so that

$$R_{\delta c} \approx 200 \tag{18.75}$$

which is in reasonable agreement with the value reported in [2] for disturbed initial flow.

Transition criterion (18.74) applies only when Valensi number is above some lower limit because for $\sqrt{V_a}/(F(t-t_0))$ too low, the growth of the momentum-thickness will be limited by the duct dimensions. Below this, turbulence onset may be characterized by the well-known steady-flow criterion; namely, that flow is laminar until R_e rises above R_l. Just when this lower limit of $\sqrt{V_a}/(F(t-t_0))$ occurs may be taken as a constant c_1, conveniently determined as the value that makes the low- and high-Valensi number branches continuous in the following universal condition for the time of turbulence onset:

$$\text{if } \frac{\sqrt{V_a}}{F(t-t_0)} \geq c_1 \quad \text{then laminar until} \quad R_e = R_{\delta c}\frac{\sqrt{V_a}}{F(t-t_0)} \tag{18.76}$$

$$\text{if } \frac{\sqrt{V_a}}{F(t-t_0)} < c_1 \quad \text{then laminar until} \quad R_e > R_l \tag{18.77}$$

The value of c_1 is evidently $R_l/R_{\delta c} \approx 2300/200$ or about 11.5. So the universal turbulence onset condition may be collapsed into the single condition

$$\text{laminar until } R_e > R_t \tag{18.78}$$

where R_t is the critical Reynolds number

$$R_t = 200\max(\frac{\sqrt{V_a}}{F(t-t_0)}, 11.5) \tag{18.79}$$

Required in evaluating R_t within a computational solution is the current value of $\omega(t-t_0)$, the dimensionless time since the previous flow reversal. As of Sage version 13 the times of flow reversal (two per cycle period) are based on the sinusoidal first-harmonic approximation of fluid mass flow rate (see section 8.5.2) and to some extent the time-average mass flow rate. The elapsed dimensionless time since flow reversal is the current dimensionless time ($2\pi j/N$, for time-grid index j of NTnode nodes) minus the dimensionless time for the most recent flow reversal. If there is a non-zero time-average mass flow rate it shifts the flow reversal times (*see* below).

Criterion (18.78) establishes the time when turbulence generation begins. But what is the rate of turbulence growth, during the transition period? To answer that it is first necessary to consider turbulence decay.

Decay

The simplest and arguably most physically realistic formulation is to have the decay term active whenever turbulence is active — whenever $\mathcal{T} > 0$. For calculating the actual rate of turbulence decay we can turn to a two-dimensional κ transport equation such as the one recommended in Launder and Spaulding ([37], p. 77). Simplified to its bare essentials by neglecting the diffusion and production terms, taking the Lagrangian point of view where $D\kappa/Dt = d\kappa/dt$ and dividing out density, the resulting equation is

$$\frac{d\kappa}{dt} \propto -\frac{1}{\ell}\kappa^{\frac{3}{2}} \tag{18.80}$$

where the constant of proportionality is to be determined from pragmatic reasoning, as usual, and ℓ is a length scale proportional to the Prandtl mixing length ℓ_m. (Prandtl mixing length is the length that makes turbulent shear stress come out right in the expression $\tau = \rho\ell_m^2 \frac{\partial u}{\partial y}^2$.) Assuming that mixing length scales with hydraulic diameter we have

$$\frac{d\kappa}{dt} \propto -\frac{1}{d_h}\kappa^{\frac{3}{2}} \tag{18.81}$$

We can write this in terms of $\mathcal{T}$ by substituting $0.0032u^2\mathcal{T}$ for κ (from equation (18.65)) and taking normalization velocity u to be the absolute velocity $|u|$. The turbulence decay rate then becomes

$$\frac{d\mathcal{T}}{dt} \approx -c_d\frac{|u|}{d_h}\mathcal{T}^{\frac{3}{2}} \tag{18.82}$$

The value of c_d is based on experimental data — specifically case b of [59], for oscillatory flow with peak Reynolds number $R_{em} = 47,000$ and Valensi number $V_a = 29$ In dimensionless terms the factor u_m/d_h may be written $\omega R_{em}/(4V_a) \approx 400\omega$). Under these conditions, a value of $c_d \approx 0.05$ will produce an initial decay rate $\frac{\partial\tau}{\partial\omega t}$ at $\tau = 1$ of about -20, which is the desired value. The final decay term in equation (18.66) is then

$$D = \begin{cases} 0.05A\frac{|u|}{d_h}\mathcal{T}^{\frac{3}{2}} & \text{if } \mathcal{T} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{18.83}$$

Of some concern is how long it takes incoming turbulence to decay to zero from the worst-case inlet value of $\mathcal{T}_{in} = 1$, for flow with low Reynolds number ($R_e < R_l$). A long delay would be unrealistic. The time for the mean flow to travel from the inlet to $x = d_h$ is $d_h/|u|$. The change in turbulence over this time is $\Delta\mathcal{T} \approx \frac{d\mathcal{T}}{dt}\frac{d_h}{|u|} <= -0.05$, from equation (18.82) with $c_d = 0.05$. It follows that relaminarization occurs within about 20 pipe diameter of the inlet. This seems reasonable.

Generation

Turbulence grows if the generation term is larger than the decay term and vice-versa. Theoretical turbulence modelers (*see* Patankar [47]) suggest that this rate should be proportional to $\frac{\partial u}{\partial r}$, which scales as $|u|/d_h$, or in dimensionless terms $\omega R_e/(4V_a)$. A pragmatic approach can be used to decide the constant of proportionality. It should be large enough so that, for relatively high values of $|u|/d_h$ $(= \omega R_e/(4V_a))$ turbulence will be generated quickly, but not too quickly lest the solution become unstable or too difficult to properly resolve in the computational grid. More importantly, the growth rate should approach the decay rate as $\mathcal{T} \to 1$ (full turbulence) to limit turbulence to $\mathcal{T} = 1$. The following generation equation does that:

$$G = \begin{cases} 0.05A\frac{|u|}{d_h}(2-\mathcal{T})\mathcal{S} & \text{if } R_e > R_t \text{ and } \mathcal{T} < 2 \\ 0 & \text{otherwise} \end{cases} \tag{18.84}$$

The "2" in $2-\mathcal{T}$ is arbitrary. With that choice the leading factor 0.05 produces the same G value at $\mathcal{T} = 1$ as the decay term D.

Smoothing function $\mathcal{S}$, defined to be $W((R_e/R_t - 1)/4)$, where W is the weight function (18.61), allows the generation term come in smoothly, starting at $R_e = R_t$. Generation is half active ($\mathcal{S} = 0.5$) at $R_e = 3R_t$ and fully active ($\mathcal{S} = 1$) at $R_e \geq 5R_t$. The factor 4 was calibrated to experimental observations of turbulence transition reported in [59].

When growth is fully active the initial turbulence growth rate $\frac{\partial \mathcal{T}}{\partial \omega t}$ at $\mathcal{T} = 0$ is around $0.025R_t/Va$. According to stirling engine practice in the 1980's published in the survey [57], the average value of Valensi number for heat exchangers is $V_a \approx 100$, at which value the transition Reynolds number of equation (18.79) is $R_t \approx 8 \times 10^3$ (for $\omega(t - t_0) = \pi/2$). That puts the average dimensionless initial rate of turbulence growth on the order of 2 (similar to the maximum rate of change of the 2nd harmonic), with the rate increasing as Reynolds number increases.

Numerical Boundaries

With the above formulations for decay and generation, turbulence differential equation (18.66) can lead to turbulence values above out of the desired range $[0, 1]$ in some cases. To avoid this problem Sage adds an extra term L to the right-hand side of equation (18.66) that serves to prevent turbulence from drifting too far out of range.

$$L = \begin{cases} 0.05A\frac{u_m}{d_h}\mathcal{T}^2 & \text{if } \mathcal{T} < 0 \\ -0.05A\frac{u_m}{d_h}(\mathcal{T}-1)^2 & \text{if } \mathcal{T} > 1 \\ 0 & \text{otherwise} \end{cases} \tag{18.85}$$

u_m is the peak absolute velocity which for sinusoidal flow with amplitude u_1 and time-average offset u_0 is

$$u_m = u_1 + |u_0| \tag{18.86}$$

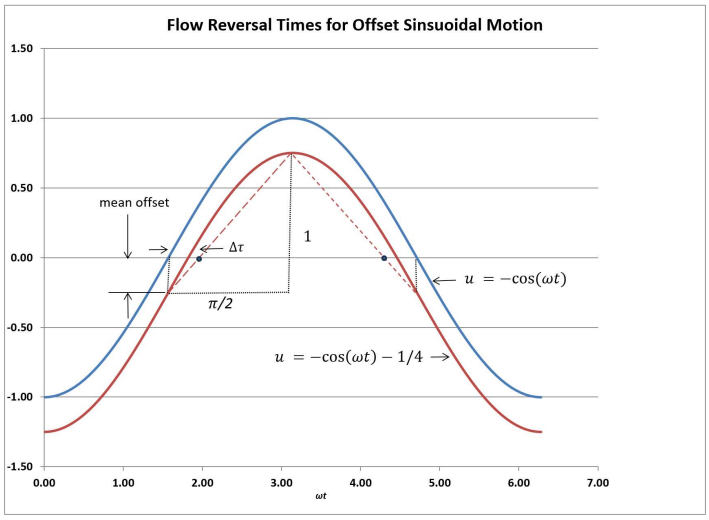


Figure 18.7: Flow reversal times (at zero velocity) for offset sinusoidal flow compared to pure oscillatory flow. Dots along dashed line segments show approximate flow reversal times.

Steady or Non-Reversing Flow

In closed-loop flow simulations it is possible to model steady flow by driving the flow with a mass-flow pump component (chapter 22) with a non-zero mean value, or pulsatile flow with both steady and oscillating components by non-zero higher harmonics. As far as turbulence modeling goes the critical distinction between oscillatory flow and quasi-steady flow (steady or pulsatile) is whether or not the flow reverses direction for part of the cycle. If it does, Sage presumes the flow is oscillatory and governed by the above turbulence model. If it does not, Sage presumes the flow is quasi-steady and governed by the above turbulence model with one simple revision. Namely the critical Reynolds number is presumed to be the steady-flow value $R_t = 2300$. This is the limiting case of equation (18.79) when Valensi number $V_a \to 0$ and the dimensionless time since flow reversal $\omega(t - t_0) \to \infty$. The generation and decay terms are the same as before.

Sage decides whether flow is oscillatory or quasi-steady based on the relative sizes of the time-average velocity u_0 and the sinusoidal first-harmonic approximation u_1. Whenever $|u_0| > u_1$ Sage presumes the flow is quasi-steady, otherwise oscillatory.

Figure 18.7 shows the case of sinusoidal oscillatory flow without and with a small time-average offset. For the offset case the flow reversal dimensionless times are shifted by amounts $\Delta\tau$. Since high precision is not required the sinusoidal variation between mean and peak velocity are approximated as line segments with slopes $\pm 2/\pi$ (dashed lines of figure). The shifts of the zero-

crossings for an offset u_0 are approximately

$$\Delta\tau \approx \pm\pi/2\,\frac{u_0}{u_1} \tag{18.87}$$

This gives the correct values at $u_0/u_1 = 0$ and 1.

18.8.3 Variable Volume Gas Domains

In variable-volume gas domains, incoming flow separation produces turbulence which mixes throughout the cylinder space uniformly and dominates all other sources of turbulence. At least, that is the assumption.

In addition to the main $x-t$ solution grid, variable volume gas domains have a special t-only grid that maintains volume V and turbulence kinetic energy $\rho\kappa$ as state variables. Volume is just

$$V(t) = A(t)L \tag{18.88}$$

κ is again the time-averaged turbulence kinetic energy per unit mass defined by equation (18.64).

According to Gedeon [14] and [15], correlations for Nusselt number N_u and enhanced axial conduction ratio N_k may be formulated directly in terms of the turbulent Reynolds number R_t defined as

$$R_t = \rho\sqrt{\kappa}d_h/\mu \tag{18.89}$$

Volume-specific turbulence kinetic energy is just $\rho\kappa$. Because of the uniform mixing assumption there is no spatial variation in $\rho\kappa$. Therefore, a time ring (special time grid for periodic solutions where the first index is logically identified as the successor to the last index) rather than a space-time grid suffices for its storage and an ordinary time-differential equation suffices for its solution

$$\frac{\partial\rho\kappa V}{\partial t} + \underbrace{[\rho u A\kappa]_{\text{in}}^{\text{out}}}_{\text{inflow}} + \underbrace{D}_{\text{decay}} = 0 \tag{18.90}$$

Compared to the duct turbulence equation, we have replaced the convection term by the net turbulence kinetic energy entering through the flow inlet(s) and neglected the generation term.

Inlet Boundary Conditions

We assume that for incoming flow, the inlet mean-flow kinetic energy is converted entirely to turbulence kinetic energy — that is, $\kappa_i = 1/2u_i^2$, where the i subscript denotes an inlet value. Velocity u_i is just the mean-flow velocity in the upstream gas duct. For outgoing flow we naturally assume κ_i takes the interior value κ. We do not need to worry about the discontinuity in κ_i at the time of flow reversal because we are interested only in the product $\rho u A\kappa_i$,

which is always zero at flow reversal. For inlets connected to multiple gas ducts, we calculate κ_i at each connection according to the previous equation. Then we sum $\rho u A \kappa_i$ over all connectors to obtain the net flow of turbulence kinetic energy through the inlet.

Decay

For turbulence decay we assume a similar form to the decay rate for duct flow, as given by equation (18.81), except we keep decay active at all times during the cycle. The following decay rate contains a calibrated leading factor that was fit to data for cylinder heat transfer with inflow-produced turbulence, as reported in [6].

$$D = 5.8 \frac{\sqrt{\kappa}}{d_h} \rho \kappa V \tag{18.91}$$

18.9 Flow Reversers


TGtRevrsNeg


TGtRevrsPos

These components serve as special "gas domains" that connect together two negative-facing or positive-facing gas inlets using built in flow-connectors, both directed positively or negatively as the case may be. They provide a simple means to reverse flow direction which otherwise would require an ad-hoc dead-ended volume component. Essentially, these components just pass information across flow connectors so that the two gas inlets at the far ends of the connectors effectively see each other. Except that velocity is reversed in the process.

Flow reversers make it possible for the temperature gradients to be in the same direction in the two gas ducts exchanging heat with each other within a parallel container or multi-length container. For example, see the *6KJTLoop* sample model in the Apps\SCFusion\Samples\Joule-Thomson sub-directory under the installation directory.

The only input variable for flow reversers is the temperature

Tinit : (real, K) Initial temperature. Used only for initializing or re-initializing solution variables ρ and ρe. As such, its value does not affect the final converged solution, but it may affect whether the solution converges at all. It should be set to a value consistent with the initial temperatures of adjacent components.

There are no output variables.

18.9.1 Theory

The job of a flow reverser is not completely trivial because it must adhere to the solution conventions imposed by Sage's flow connector scheme (*see* chapter 18). Basically there needs to be a way to force incoming mass flow rate ($\rho u A$) to equal outgoing mass flow rate. So flow reversers must implement at-least one implicit solution grid variable for this purpose. Having done that they must take additional steps to ensure energy continuity (because local solution values are

now decoupled from the incoming values). In the end, flow reversers implement a simplified version of the solution scheme used in flow restrictors (*see* chapter 22). They contain a time grid with state variables ρ, ρe, and P, governed by equations of mass continuity and energy continuity.

ρ

Mass density ρ is computed implicitly from the zero-volume mass-continuity equation

$$(\rho u A)_1 + (\rho u A)_2 = 0 \tag{18.92}$$

where $(\rho u A)_1$ and $(\rho u A)_2$ are the values imported from flow connectors 1 and 2. This forces mass flow rate in two flow connectors to be equal but opposite. Implicit functions used for solving ρe also help to determine ρ.

ρe

Basically, energy density ρe is computed implicitly from the energy continuity equation written in the form

$$\frac{\rho e + P}{\rho} = h_i \tag{18.93}$$

where ρe and P on the left are internal state variables while h_i on the right is the mass-specific enthalpy evaluated using variables imported from the component across the *incoming* flow connection. For an ideal gas h reduces to $c_p T$ in the zero-velocity limit where kinetic energy may be neglected.

For numerical reasons it is convenient to separate ρe into two variables ρe_1 and ρe_2. ρe_1 is exported to flow connector 1 and ρe_2 to flow connector 2. Each is guaranteed valid at the right time — ρe_1 whenever flow is incoming from connector 2 and vice-versa for ρe_2. The reason for this formulation is because the implicit function determining either can be formulated without a discontinuity at flow reversal, as would otherwise be the case. That is, the equation for determining ρe_1 is always that local mass-specific enthalpy $\frac{\rho e_1 + P_1}{\rho}$ equals the h_i value imported across connector 2. this enforces energy continuity when flow is incoming from connector 2 and otherwise does no harm. A similar equation determines ρe_2. The meaning of P_1 and P_2 is similar to the meaning of ρe_1 and ρe_2, and is explained below.

It might be tempting to use these continuity conditions instead to calculate ρe_1 and ρe_2 explicitly. This will not work because of a circular reference when accessing P, which itself depends on ρe.

P

Pressure is also broken into two components P_1 and P_2, similar to the treatment for ρe. P_1 is exported to the flow connector 1 and P_2 to flow connector 2. Both are calculated explicitly from the equation of state $P(\rho, T)$. This requires temperature which is calculated from the zero-velocity equation of state. For

evaluating P_1 the temperature is calculated as $T(\rho, \rho e_1, 0)$. For evaluating P_2 it is calculated as $T(\rho, \rho e_2, 0)$. (The 0's in the argument lists are velocity) Pressures is required in the solution grid so that the flow connectors can properly enforce Bernoulli's law across the connections.

Why zero velocity? Basically, velocity is irrelevant to mass or energy conservation. The only thing velocity determines is the pressure shift across the flow connectors, according to Bernoulli's law. But any pressure shift across the incoming-flow connector, produced by the velocity in the flow reverser, is canceled by an opposite pressure shift across the outgoing-flow connector. So flow reverser velocity does not matter and it is convenient to assume the simplest thing: zero velocity.

Chapter 19

Canisters

Canisters are the containers into which you normally put a matrix-type heat-exchanger. But you can fill them with more than that. You can drop in a radiation transport path for modeling radiant heat transport from one end to the other, which makes sense if the contents are sufficiently transparent. Or, you can drop in a bar or distributed conductor for modeling axial conduction in the canister walls. Some canisters even allow you to nest other canisters within them, which can be useful for modeling pistons or displacers within cylinders (*see* chapter 24). All of these attributes arise as toolbox-created child components, at your discretion. The main purpose of canisters is to pass on critical geometry-dependent information to these child components when you create them. The child components within canisters and they way they are connected to the outside world are what really matters.

All canisters are high level components that appear in the toolbox of the root-level SCFusion model component. They are born without connectors of any sort. To get connectors at this level you must move them up from the various child components you create within. Variables common to all canisters are:

NCell : (dimensionless) The number of spatial nodes in the computational grids of all underlying model components having such grids. Changing this variable causes these computational grids to re-initialize themselves. NCell will affect the solution time and memory requirements, not to mention solution accuracy. Treat it gingerly by making only gradual, small changes.

Length : (real, m) Canister length L in x direction.

Avoid : (real or cubic spline, m^2) Canister void cross sectional (x-normal) area A_c.

Asolid : (real or cubic spline, m^2) Canister solid (wall) cross sectional area A_s.

Mass : (real, kg) Canister mass m, provided for use in constraints in case it is part of a reciprocating mass system.

Solid : (enumerated) Solid (wall) material.

Tinit : (cubic spline, W) Axial temperature distribution $T(x)$ where $x = 0$ is the negative endpoint and $x = 1$ is the positive endpoint.

The working gas type (properties) comes from the parent model component. All canister types calculate mass as

$$m = \rho_s \int_{dx} A_s \tag{19.1}$$

where ρ_s is the solid density. Dependent variables Avoid and Asolid are calculated differently for each of the descendant canister types below, in terms of geometrical inputs. When they do not vary with x (axial coordinate) they are single-valued real variables. When x variation is allowed they are cubic splines.

19.1 Tubular Canister

TTubCan

A tubular canister is a right circular cylinder, just as its icon suggests. Its additional variables are:

Din : (real, m) Canister internal diameter d_i.

Wcan : (real, m) Wall thickness w.

It calculates void cross-section area as

$$A_c = \frac{\pi}{4} d_i^2 \tag{19.2}$$

and solid cross-section area as

$$A_s = \frac{\pi}{4}(d_o^2 - d_i^2) \tag{19.3}$$

where $d_o = d_i + 2w$ is outer diameter.

There is a close descendant of a tubular canister known as a *nested* tubular canister. It shares the same icon as a tubular canister but is intended for nesting within another canister so that variables NCell, Length, and Tinit come from the parent canister and do not appear in its variable list. Its inner diameter Din becomes a dependent variable based on the assumption that its outer diameter equals the inner diameter of the parent canister. In other words, a nested canister always fits snugly within its parent canister.

19.2 Annular Canister

TAnnCan

An annular canister is the volume enclosed by two concentric right circular cylinders. Its additional variables are:

Din : (real, m) Inner-wall internal diameter d_i.

Dout : (real, m) Outer-wall outside diameter d_o.

Win : (real, m) Inner-wall thickness w_i.

Win : (real, m) Outer-wall thickness w_o.

It calculates void cross-section area as

$$A_c = \frac{\pi}{4}\left((d_o - 2w_o)^2 - (d_i + 2w_i)^2\right) \tag{19.4}$$

and solid cross-section area as

$$A_s = \frac{\pi}{4}\left(d_o^2 - (d_o - 2w_o)^2 + (d_i + 2w_i)^2 - d_i^2\right) \tag{19.5}$$

There is a close descendant of an annular canister known as a *nested* annular canister. It shares the same icon as an annular canister but is intended for nesting within another canister so that variables NCell, Length, Dout and Tinit come from the parent canister and do not appear in its variable list. A nested canister always fits snugly within its parent canister. Its outer wall outside diameter Dout equals its parent canister inner diameter Din.

19.3 Tubular-Cone Canister


TTubxCan

A tubular-cone canister is similar to an ordinary tubular canister except its internal diameter and wall thickness are specified by cubic splines. In other words, both may vary with axial position. The variation may be a good deal more complex than linear, depending on the number of interpolation pairs you specify for the variables:

Din : (cubic spline, m) Canister internal diameter d_i.

Wcan : (cubic spline, m) Wall thickness w.

A powerful option now presents itself: Din and Wcan, may be optimized! At least any data value in any of their defining interpolation pairs may be optimized. Say, for example, you want to find the optimum diameter profile for a tubular-cone regenerator canister specified by three interpolation pairs. Then you would select as optimized variables Din.FData.1, Din.FData.2 and Din.FData.3. Likewise, you may also map either variable.

Tubular-cone canisters calculate void cross-section area A_c and solid cross-section area A_s using equations (19.2) and (19.3), similar to ordinary tube canisters, except d_i and d_o are taken as functions of axial position. What actually happens is that tubular cones override variables Avoid and Asolid as dependent cubic splines. These splines are each defined by n data pairs $(x, A(x))$ equal spaced in x within the interval $[0, 1]$, where n is the larger of the number of data pairs in the specification of Din or Wcan. This is self-evident in the display form or listing.

There is a subtlety here worth noticing: The resolution of Avoid or Asolid are only as good as the resolution of Din or Wcan. If you specify a linear diameter profile (a true cone) by the interpolation pairs $(0, 0.05)$ and $(1, 0.10)$ for Din, for example, you might think that A_c would then have a quadratic profile since it depends on d_i^2. You might be wrong. If the Avoid cubic spline is also defined by only two interpolation pairs, it will also have a linear profile. This is of some concern because the solutions of child model components within the canister (gas and solid domains) generally regard area as the fundamental quantity, not diameter. So your linear diameter profile is actually modeled as if it were a square-root profile. To increase accuracy you might want to specify Din with three, or more interpolation points, even though it may be linear.

19.4 Annular-Cone Canister

TAnnxCan

An annular-cone canister is an extension of an ordinary annular canister in the same way a tubular cone was an extension of an ordinary cone. Now all of the following inputs are specified as cubic splines:

Din : (cubic spline, m) Inner-wall internal diameter d_i.

Dout : (cubic spline, m) Outer-wall outside diameter d_o.

Win : (cubic spline, m) Inner-wall thickness w_i.

Win : (cubic spline, m) Outer-wall thickness w_o.

Any or all of these may be mapped or optimized by specifying the appropriate FData.n qualifier as discussed above.

Annular cones calculate void cross-section area A_c and solid cross-section area A_s using equations (19.4) and (19.5), similar to ordinary annular canisters, except all the d's and w's are taken as functions of axial position. This takes the form of overriding Avoid and Asolid as dependent cubic splines using the same logic as for tubular cones, discussed above. And the same cautionary remarks apply as well. Namely, that two-point linear diameter profiles may produce linear, not quadratic, area profiles.

Chapter 20

Heat Exchangers

Heat exchangers are the containers — or sub-containers if they are themselves found within a canister — into which you normally put a gas-domain and one or more thermal-solid model components. Their main purpose is to pass on critical geometry-dependent information to these child components when you create them. Stand-alone heat exchangers appear in the toolbox of the root-level SCFusion model component, while those needing containers appear in canister toolboxes. They are born without connectors of any sort. To get connectors you must move them up from the various child components you create within. Variables common to all heat exchangers are:

NCell : (dimensionless) The number of spatial nodes in the computational grids of all underlying model components having such grids. Changing this variable causes these computational grids to re-initialize themselves. Increasing NCell generally increases solution accuracy at the cost of greater solution time.

Length : (real, m) Heat exchanger length L in x direction. For a variable-volume heat exchanger this is a *rough* estimate of the mean flow-path length of a typical fluid particle — the path length from the inlet to a wall boundary. No need to be overly exact because thermal performance is not very sensitive to L in variable-volume heat exchangers.

Roughness : (dimensionless) Relative roughness of the surface in contact with the gas, defined as the average height ϵ of surface irregularities divided by the hydraulic diameter d_h (i.e. tube diameter or twice the gap for parallel plates). Required as an input only for heat-exchangers having uniform flow passages, where it is important for determining turbulent flow resistance. According to [32], the height ϵ is the average distance between low and high points on the surface, rather than the average distance from high points to the mean surface. Multiplying Machinery's Handbook [45] peak-to-mean roughness values by two suggests the following absolute roughness values for various manufacturing processes:

Process	ϵ (microns)
cold-rolling	2 to 6
drawing	2 to 6
extruding	2 to 6
drilling	3 to 13

So for example, a drawn seamless tube with 3 mm ID could be expected to have a relative roughness in the range of 0.001 to 0.002. A drilled hole of the same diameter as much as 0.004.

Aflow : (real, m^2) Mean flow cross sectional (x-normal) area A_f. Void volume divided by length.

Asec : (real, m^2) Mean solid cross sectional area A_s. (*see* equation (17.15) in chapter 17)

Pwet : (real, m) Wetted perimeter S_x. The surface area per unit length for gas-to-solid heat transfer.

Tinit : (cubic spline, W) Axial temperature distribution $T(x)$ where $x = 0$ is the negative endpoint and $x = 1$ is the positive endpoint.

Matrix type heat exchangers have the additional variable

Porosity : (real, dimensionless) Porosity β. Void volume divided by total volume. Total volume is void plus solid volume.

Matrix heat exchangers calculate mean-flow cross section area A_f as the product of porosity and canister cross-sectional area A_c.

$$A_f = \beta A_c \tag{20.1}$$

Duct and variable-volume type heat exchangers have the additional variable:

Twall : (real, m) Wall thickness t_w. Usually a non-critical variable used in determining solid cross-sectional area for some thermal-solid child component.

Every heat exchanger component comes with a unique gas-domain component in its toolbox. This domain is a descendant of either the matrix, duct or variable-volume type of gas domain and shares an icon with its ancestor object. The gas-domain is unique because it knows the specific correlations for friction factor, Nusselt number and axial conductivity ratio appropriate for the parent heat exchanger type. These correlations are documented below as part of the heat-exchanger documentation. Correlations are in terms of the following variables:

d_h	hydraulic diameter defined by equation (18.5)
β	porosity; void / total volume
ϵ	average height of surface irregularities (*see* Roughness above)
V_a	Valensi number defined by equation (18.6)
R_e	Reynolds number defined by equation (18.4)
P_r	$c_p\mu/k$; Prandtl number
P_e	R_eP_r; Peclet number
R_t	turbulent Reynolds number defined by equation (18.89)

20.0.1 Solid Tortuosity

Matrix type heat exchangers implement a tortuosity function for solid-mode axial thermal conduction. This function accounts for the possibility of a complicated solid conduction path which typically zig-zags through the matrix and bridges a number of contact points between particles or wires. The tortuosity function is inherited by the wall-surface child component within the porous matrix. The child component calls the tortuosity function from within its computational grid and uses the result to correctly model solid-mode axial conduction. The wall-surface child component displays the average tortuosity as an output (*see* sections 17.7.3 and 17.8). The discussion below is a condensed version of a more detailed account in memorandum [20].

Definitions

Static conduction down a porous matrix (total inside the canister, not including the canister) can be represented in terms of an average matrix conductivity k_{av}

$$Q_t = k_{av}A_c\frac{dT}{dx} \tag{20.2}$$

where A_c is the matrix canister frontal area. Compare this to the static conductions down gas and solid uniform bars of cross-section areas βA_c and $(1-\beta)A_c$, corresponding to the void and filled parts of the matrix in the simplest of models

$$Q_g = k_g\beta A_c\frac{dT}{dx} \tag{20.3}$$

and

$$Q_s = k_s(1-\beta)A_c\frac{dT}{dx} \tag{20.4}$$

k_g and k_s are the gas and solid conductivities and β is matrix porosity (void fraction). Generally the total matrix conduction is less than the sum $Q_g + Q_s$. Sage models the static gas conduction according to the first equation (20.3) but discounts solid conduction by a tortuosity factor to make the total conduction come out right. The tortuosity factor is defined as

$$f_s \equiv \frac{Q_t - Q_g}{Q_s} \equiv \frac{k_{av} - k_g\beta}{k_s(1-\beta)} \tag{20.5}$$

Calibrated Formulation

The difficult thing is figuring out the average matrix conductivity k_{av}. Sage starts with a basic formula for the average thermal conduction of a matrix consisting of spheres embedded in a medium of different conductivity, based on an example in Carslaw and Jaeger ([7], pp. 426–428). Victorian physicist James Clerk Maxwell (of Maxwell's equations) actually derived the formula for induced magnetization in a sphere placed in a uniform external field but the governing equations are the same for heat flow. When reduced to the form of a tortuosity factor according to the above definition (20.5) and multiplied by a calibration factor of the form $(k_s/k_g)^m$ the final tortuosity formulation is

$$f_s = \left(\frac{k_s}{k_g}\right)^{m-1} \left[\frac{3(k_s/k_g - \beta) + (2 + k_s/k_g)\beta}{3(1-\beta) + (2 + k_s/k_g)\beta}\right] \tag{20.6}$$

The above equation with $m = 0$ corresponds to the original Maxwell conduction model. For particular matrices Sage uses calibration exponents fit to NIST data in [40] (*see* below under particular matrix heat exchanger types). The Maxwell conduction model assumes high contact resistance between individual solid elements of a matrix. So it is not suitable for wrapped-foil matrices or any matrix with continuous solid elements in the axial direction.

20.0.2 Gas Axial-Conduction Enhancement

For complicated gas flows with microscopic details smaller than a computational grid can resolve Sage resorts to the mechanism of enhanced gas conduction to model at least some of the macroscopic thermal energy transport (the remainder being modeled as bulk enthalpy transport, based on the section-mean gas temperature and mass flow rate). Enhanced gas conduction is represented in terms of a dimensionless ratio

$$N_k \equiv k_e/k_g \tag{20.7}$$

where k_e is the effective gas conductivity and k_g is the molecular conductivity. Sage resorts to enhanced gas conduction for turbulent flows and also for flows through porous material. The former is typically known as *turbulent conduction* and the latter as *thermal dispersion*. Of the two, thermal dispersion is probably the more unfamiliar concept.

The limiting value of N_k for zero flow velocity is $N_{k0} = 1.0$. This is consistent with the above tortuosity formulation for solid-mode conduction and results in the correct overall matrix static conduction (gas + solid).

Thermal Dispersion

One way to think about thermal dispersion is in terms of a experiment for an analogous phenomenon in mass diffusion. Take a screen matrix filled with fluid (say water) initially at rest and with a segment near the middle (bounded by

two planes) colored differently from the surrounding fluid. If you were to swim through the matrix in the axial direction you would swim through clear water, then suddenly enter colored water, then emerge a short time later into clear water again. Now you can do two things. Thing one is to just let the experiment sit there with the fluid at rest. The colored and clear water will gradually diffuse into each other, smoothing out the sharp-edged color boundary. This is the mass-transfer analog of molecular conduction. Thing two is to start oscillating the fluid back and forth through the matrix in the axial direction. In this case the color boundary diffuses much faster than before due to the interaction of high-velocity flow channels between the wires and low-velocity wakes behind the wires. This is the analog of thermal dispersion.

20.1 Woven Screen Matrix


TScnMtx

The woven screen matrix component adds input variable:

Dwire : (real, m) Wire diameter d_w.

It calculates wetted perimeter S_x as

$$S_x = 4(1-\beta)A_c/d_w \tag{20.8}$$

where A_c is canister cross-section area. The following correlations are from NASA Contractor Report [18] which documents oscillating-flow regenerator tests made at Ohio University for the following stainless-steel woven-screen samples, over a range of peak Reynolds numbers from 1 to 3400:

200 mesh per inch wire diameter 53.3 microns (0.0021 in), porosity 0.6232

100 mesh per inch wire diameter 55.9 microns (0.0022 in), porosity 0.7810

80 mesh per inch wire diameter 94.0 microns (0.0037 in), porosity 0.7102

20.1.1 Friction Factor

The screen friction factor is a variation of the Ergun equation [43] with the constant term replace by a term proportional to Reynolds number raised to a negative but small exponent to better track observed reality at high R_e.

$$f = 129/R_e + 2.91R_e^{-0.103} \tag{20.9}$$

20.1.2 Nusselt Number

$$N_u = (1 + 0.99P_e^{0.66})\beta^{1.79} \tag{20.10}$$

20.1.3 Axial-Conduction Enhancement

$$N_k = 1.0 + 0.50 P_e^{0.66} \beta^{-2.91} \tag{20.11}$$

20.1.4 Tortuosity

Solid-mode tortuosity is calculated from equation (20.6) with calibration exponent $m = 0.165$, based on fitting to data for stainless-steel and phosphor bronze screens in [40].

20.2 Random Fiber Matrix

TFbrMtx

The random fiber matrix component adds input variable:

Dfiber : (real, m) Fiber diameter d_w.

It calculates wetted perimeter using the same formula (20.8) as for woven screens, presuming circular cross-section fibers. For non-circular fibers the wetted perimeter and the following correlations will probably be wrong. As they will if the fibers are not oriented perpendicular to the axial flow direction. The correlations are based on oscillating-flow regenerator tests and data-reduction procedures reported in NASA Contractor Report [18] along with more recent tests for higher porosity matrices reported in memorandum [21]. The correlations are derived from porosity-dependence curve fits to the best-fit modeling parameters for the following round-wire test samples:

2 mil Brunswick inconel, wire diameter 52.5 micron, porosity 0.688

1 mil Brunswick stainless steel, wire diameter 27.4 micron, porosity 0.820

12 micron Bekaert stainless steel, wire diameter 13.4 micron, porosity 0.897

30 micron Bekaert stainless steel, wire diameter 31.0 micron, porosities 0.85, 0.90, 0.93, 0.96 (samples cut from same sheet of material and pressed to different porosities)

24 micron Bekaert Fecralloy, wire diameter 24.3 micron, porosity 0.909

Wire diameters are typically mean effective values based on electron microscope image measurements. The two Brunswick samples were tested in 1992–03 as documented in NASA Contractor report [18]. The 12 micron Bekaert sample was tested under DOE funding in 2003. "Bekaert" refers to Bekaert Corporation, the company who made the random fibers and continues to do so as of 2008. (Brunswick Corporation no longer is in the random fiber business.) The 30 and 24 micron Bekaert samples were tested in 2006–2008 under funding provided by the NASA Headquarters Science Mission Directorate (via a NASA Glenn Research Center Grant to Cleveland State University). The peak

Reynolds numbers for the combined test ranged from about 1 to 3500. In the correlations below porosity dependence is correlated in terms of

$$\alpha \equiv \frac{\beta}{1-\beta}$$

The correlations have been most recently updated in November 2008 (Sage version 6) based on re-tests of high porosity samples as documented in [22] and [23].

20.2.1 Friction Factor

$$f = a_1/R_e + a_2 R_e^{a_3} \tag{20.12}$$

where

$$\begin{aligned} a_1 &= 25.7\alpha + 79.8 \\ a_2 &= 0.146\alpha + 3.76 \\ a_3 &= -0.00283\alpha - 0.0748 \end{aligned}$$

20.2.2 Nusselt Number

$$N_u = 1 + b_1 P_e^{b_2} \tag{20.13}$$

where

$$\begin{aligned} b_1 &= 0.186\alpha \\ b_2 &= 0.55 \end{aligned}$$

20.2.3 Axial-Conduction Enhancement

$$N_k = 1.0 + b_3 P_e^{b_2} \tag{20.14}$$

where

$$\begin{aligned} b_2 &= \text{same as above} \\ b_3 &= 1.0 \end{aligned}$$

20.2.4 Tortuosity

Solid-mode tortuosity is calculated the same as for screens, namely from equation (20.6) with calibration exponent $m = 0.165$. There is no data to back this up but the wires in random-fibers matrices are typically oriented perpendicular to the axial flow direction, similar to screens.

20.3 Packed Sphere Matrix

TSphMtx

The packed sphere matrix component adds input variable:

Dsphere : (real, m) Sphere or particle diameter d_s.

It calculates wetted perimeter S_x as

$$S_x = 6(1-\beta)A_c/d_s \tag{20.15}$$

where A_c is canister cross-section area, presuming spherical particles. For non-spherical particles the wetted perimeter and the following correlations will probably be wrong.

The following correlations are based on unpublished oscillating-flow regenerator tests using essentially the same hardware and procedures as documented in NASA Contractor Report [18]. The material tested was 173 micron diameter spherical lead particles (173 micron mean diameter ± 14 micron standard deviation) packed to porosities in the range 0.38 to 0.43.

20.3.1 Friction Factor

$$f = \left(157/R_e + 5.15R_e^{-0.137}\right)\left(\beta/0.39\right)^{3.48} \tag{20.16}$$

20.3.2 Nusselt Number

$$N_u = 1 + 0.48P_e^{0.65} \tag{20.17}$$

20.3.3 Axial-Conduction Enhancement

$$N_k = 1.0 + 3.00P_e^{0.65} \tag{20.18}$$

20.3.4 Tortuosity

Solid-mode tortuosity is calculated from equation (20.6) with calibration exponent $m = 0.145$, based on fitting to data for lead, stainless-steel and copper spheres in [40].

20.4 Wrapped Foil Matrix

TFoiMtx

The wrapped foil matrix component adds input variables:

Gap : (real, m) Gap g between foil layers.

Thk : (real, m) Foil thickness b.

It calculates wetted perimeter S_x as

$$S_x = 2A_f/g \tag{20.19}$$

where A_f is flow cross-section area according to equation (20.1). It calculates porosity as

$$\beta = \frac{1}{1 + b/g} \tag{20.20}$$

20.4.1 Friction Factor

The laminar case is based on exact theory for fully-developed flow and the turbulent case is the Altshul approximation to the Colebrook formula with a shape correction factor of 1.1, as recommended in [32] (pp. 8, 22).

- Case laminar

$$f = 96R_e^{-1} \tag{20.21}$$

- Case turbulent

$$f = 0.121\left(\epsilon/d_h + 68/R_e\right)^{0.25} \tag{20.22}$$

20.4.2 Nusselt Number

The laminar case is based on exact theory for fully-developed flow, uniform heat flux, and the turbulent case is from figure 7.4 p. 146 of reference [35].

- Case laminar

$$N_u = 8.23 \tag{20.23}$$

- Case turbulent

$$N_u = 0.025R_e^{0.79}P_r^{0.33} \tag{20.24}$$

20.4.3 Axial-Conduction Enhancement

Same as equations (20.42) and (20.43) for tubular ducts since the flow microstructure is likely to be similar.

20.4.4 Tortuosity

Solid-mode tortuosity is taken as $f_s = 1$ (no tortuosity at all) which corresponds to an uninterrupted solid conduction path.

20.5 Generic Matrix

TGnrMtx

A generic matrix component allows you to specify your own coefficients and exponents within generic formulations for friction factor, Nusselt number, axial conduction enhancement and tortuosity. The generic matrix component adds input variables:

Dhyd : (real, m) Hydraulic diameter d_h as in equation (18.5).

FsC1 : (real, dimensionless) c_1 in tortuosity formulation (20.29).

FsC2 : (real, dimensionless) c_2 in tortuosity factor formulation (20.29).

FdM : (real, dimensionless) m in tortuosity formulation (20.29).

It calculates wetted perimeter S_x as

$$S_x = 4A_f/d_h \tag{20.25}$$

where A_f is mean flow area βA_c (porosity $\times$ canister area). Its associated gas domain adds input variables:

FdC1 : (real, dimensionless) c_1 in friction factor formulation (20.26).

FdC2 : (real, dimensionless) c_2 in friction factor formulation (20.26).

FdC3 : (real, dimensionless) c_3 in friction factor formulation (20.26).

FdM : (real, dimensionless) m in friction factor formulation (20.26).

NuC1 : (real, dimensionless) c_1 in Nusselt number formulation (20.27).

NuC2 : (real, dimensionless) c_2 in Nusselt number formulation (20.27).

NuM : (real, dimensionless) m in Nusselt number formulation (20.27).

NuN : (real, dimensionless) n in Nusselt number formulation (20.27).

KrC1 : (real, dimensionless) c_1 in axial conduction enhancement formulation (20.28).

KrC2 : (real, dimensionless) c_2 in axial conduction enhancement formulation (20.28).

KrM : (real, dimensionless) m in axial conduction enhancement formulation (20.28).

KrN : (real, dimensionless) n in axial conduction enhancement formulation (20.28).

20.5.1 Friction Factor

The generic matrix friction factor is

$$f = c_1 + c_2 R_e^m + c_3/R_e \tag{20.26}$$

where c_1, c_2, c_3 and m are input variables.

20.5.2 Nusselt Number

The generic matrix Nusselt number is

$$N_u = c_1 + c_2 R_e^m P_r^n \tag{20.27}$$

where c_1, c_2, m, n are input variables.

20.5.3 Axial-Conduction Enhancement

The generic matrix axial conductivity enhancement ratio is

$$N_k = c_1 + c_2 R_e^m P_r^n \tag{20.28}$$

where c_1, c_2, m, n are input variables.

20.5.4 Tortuosity

The generic matrix tortuosity factor is

$$f_s = c_1 + c_2 \left(\frac{k_s}{k_g}\right)^{m-1} \left[\frac{3(k_s/k_g - \beta) + (2 + k_s/k_g)\beta}{3(1-\beta) + (2 + k_s/k_g)\beta}\right] \tag{20.29}$$

where c_1, c_2 and m are input variables. This reduces to the Maxwell tortuosity factor of equation (20.6) for $c_1 = 0$ and $c_2 = 1$. If that is not appropriate you can specify specify a constant tortuosity with $c_1 > 0$ and $c_2 = 0$.

20.6 Tube Bundle

The tubular heat exchanger adds input variables:

TTubDct

Dtube : (real, m) Tube internal diameter d_i.

Ntube : (real, dimensionless) Tube number n. This can take on non-integer values so that it can be optimized.

It calculates flow area as

$$A_f = n\frac{\pi}{4} d_i^2 \tag{20.30}$$

It calculates solid cross-section area in terms of outside diameter d_o ($d_o = d_i + 2t_w$ where t_w is wall thickness) as

$$A_s = n\frac{\pi}{4}(d_o^2 - d_i^2) \tag{20.31}$$

And it calculates wetted perimeter as

$$S_x = n\pi d_i \tag{20.32}$$

20.6.1 Friction Factor

The laminar case is formulated in terms of the real and imaginary components of the oscillating-flow wall-shear-stress function as discussed in section (18.6.1) with $\boldsymbol{s}$ derived from the thermoacoustic function $\boldsymbol{f}_\nu$ for circular tubes in reference [65]. The approximations below are defined piecewise from low-V_a and high-V_a asymptotic limits, with the Valensi number dividing one branch from the other chosen to make the two branches continuous. These approximations are much easier to compute than the exact functions (complex Bessel functions) and are accurate to within a worst-case relative error on the order of 25%. The low V_a laminar equivalent friction factor is $f = 64/R_e$, according to equation (18.21). The turbulent case is the Altshul approximation to the Colebrook formula as recommended in [32] (p. 8).

- Case laminar

$$s_r = \begin{cases} 4 & \text{if } V_a \leq 32 \\ \sqrt{V_a/2} & \text{if } V_a > 32 \end{cases} \tag{20.33}$$

$$s_i = \begin{cases} V_a/6 & \text{if } V_a \leq 18 \\ \sqrt{V_a/2} & \text{if } V_a > 18 \end{cases} \tag{20.34}$$

- Case turbulent

$$f = 0.11\,(\epsilon/d_h + 68/R_e)^{0.25} \tag{20.35}$$

20.6.2 Nusselt Number

The laminar case is broken into separate parts N_{u0}, $\boldsymbol{N}_{uc}$ and $\boldsymbol{N}_{ua}$ — real and complex Nusselt numbers that apply respectively to the steady, compression-driven and advection-driven components of the gas-to-wall temperature difference, as discussed in section (18.6.2). These are based on piecewise-continuous blended averages of low and high Valensi-number solutions presented in [16] and also derived from [64]. The real-valued turbulent case is based on reference [29] and there is no distinction made for the various components of temperature difference.

- Case laminar

$$N_{u0} = 6.0 \tag{20.36}$$

$$\Re(\boldsymbol{N}_{uc}) = \begin{cases} 6.0 & \text{if } \sqrt{2V_aP_r} < 6.0 \\ \sqrt{2V_aP_r} & \text{otherwise} \end{cases} \tag{20.37}$$

$$\Im(\boldsymbol{N}_{uc}) = \begin{cases} \frac{1}{5}V_aP_r & \text{if } V_aP_r < 5\sqrt{2V_aP_r} \\ \sqrt{2V_aP_r} & \text{otherwise} \end{cases} \tag{20.38}$$

$$\Re(\boldsymbol{N}_{ua}) = \begin{cases} 4.2 & \text{if } \sqrt{2V_aP_r} < 8.4 \\ \frac{1}{2}\sqrt{2V_aP_r} & \text{otherwise} \end{cases} \tag{20.39}$$

$$\Im(\boldsymbol{N}_{ua}) = \begin{cases} \frac{1}{10}V_aP_r & \text{if } V_aP_r < 5\sqrt{2V_aP_r} \\ \frac{1}{2}\sqrt{2V_aP_r} & \text{otherwise} \end{cases} \tag{20.40}$$

- Case turbulent

$$N_u = 0.036\, R_e^{0.8}(L/d_h)^{-0.055}P_r^{0.33} \tag{20.41}$$

20.6.3 Axial-Conduction Enhancement

The laminar case assumes no enhancement beyond molecular conduction. The turbulent case is based on a theoretical analysis that goes roughly as follows: In turbulent flow the total fluid shear stress can be represented as $\tau = (\mu + \mu_t)\frac{\partial u}{\partial y}$ where μ is the molecular viscosity, μ_t is the so-called turbulent viscosity and y is the coordinate normal to the wall. For turbulent tube flow, μ_t can be determined from the Blasius friction factor ($f = 0.316R_e^{-0.25}$) and the known velocity profile. Then, using a *Reynolds-analogy* type of argument, the enhanced (turbulent) axial conductivity ratio (N_k) can be equated with $P_r\mu_t/\mu$. After the details have been worked out the resulting correlation is

- Case laminar

$$N_k = 1 \tag{20.42}$$

- Case turbulent

$$N_k = 0.022R_e^{0.75}P_r \tag{20.43}$$

20.7 Rectangular Channels

The rectangular channel heat exchanger adds input variables:

TRecDct

Wchan : (real, m) Channel inner width w_i.

Hchan : (real, m) Channel inner height h_i.

Nchan : (real, dimensionless) Channel number n. This can take on non-integer values so that it can be optimized.

It calculates flow area as

$$A_f = nw_i h_i \tag{20.44}$$

It calculates solid cross-section area in terms of outer channel width w_o and height h_o ($w_o = w_i + 2b$, $h_o = h_i + 2t_w$, where t_w is wall thickness) as

$$A_s = n(w_o h_o - w_i h_i) \tag{20.45}$$

And it calculates wetted perimeter as

$$S_x = 2n(w_i + h_i) \tag{20.46}$$

Aspect ratio a is defined to be the smaller of w_i/h_i or h_i/w_i. In terms of a, there are two geometrical parameters used in the following correlations

$$\begin{aligned} b &= 1.47 - 1.48a + 0.92a^2 \\ c &= 0.438 + 0.562(1-a)^3 \end{aligned}$$

20.7.1 Friction Factor

The laminar case is similar to the case for tubes, with the factor b representing the continuum between parallel plates and rectangular channels. The value of b is about 1.5 (1.47 actually) for an aspect ratio of zero (parallel plates) and 0.91 for an aspect ratio of one (square channels). The approximations below reduce to the above *tube* approximations for $b = 1.0$ and to the approximations for parallel plates for $b = 1.5$ (except for a 10% error in the low-V_a branch for S_i). The turbulent case is the Altshul equation, the same as for tubes.

- Case laminar

$$s_r = \begin{cases} 4b & \text{if } V_a \le 32b^2 \\ \sqrt{V_a/2} & \text{if } V_a > 32b^2 \end{cases} \tag{20.47}$$

$$s_i = \begin{cases} V_a/(6b) & \text{if } V_a \le 18b^2 \\ \sqrt{V_a/2} & \text{if } V_a > 18b^2 \end{cases} \tag{20.48}$$

- Case turbulent

$$f = 0.11\left(\epsilon/d_h + 68/R_e\right)^{0.25} \tag{20.49}$$

Factor b is the result of a curve-fitting process for laminar friction factors, traceable to Terry Heames and associates at the Argonne National Laboratory [29]. The fact that b is not quite 1.0 for square channels is, perhaps, a reasonable accounting of the differences between square channels and circular tubes. The fact that b is not exactly 1.5 in the parallel-plate limit is likely a minor curve-fitting error.

20.7.2 Nusselt Number

The laminar case is similar to that for tubes with different low-Valensi limits. The turbulent case comes from reference [29].

- Case laminar

$$N_{u0} = 10.0\,c \tag{20.50}$$

$$\Re(\boldsymbol{N}_{uc}) = \begin{cases} 10.0\,c & \text{if } \sqrt{2V_aP_r} < 10.0\,c \\ \sqrt{2V_aP_r} & \text{otherwise} \end{cases} \tag{20.51}$$

$$\Im(\boldsymbol{N}_{uc}) = \begin{cases} \frac{1}{5}V_aP_r & \text{if } V_aP_r < 5\sqrt{2V_aP_r} \\ \sqrt{2V_aP_r} & \text{otherwise} \end{cases} \tag{20.52}$$

$$\Re(\boldsymbol{N}_{ua}) = \begin{cases} 8.1\,c & \text{if } \sqrt{2V_aP_r} < 16.2\,c \\ \frac{1}{2}\sqrt{2V_aP_r} & \text{otherwise} \end{cases} \tag{20.53}$$

$$\Im(\boldsymbol{N}_{ua}) = \begin{cases} \frac{1}{10}V_aP_r & \text{if } V_aP_r < 5\sqrt{2V_aP_r} \\ \frac{1}{2}\sqrt{2V_aP_r} & \text{otherwise} \end{cases} \tag{20.54}$$

- Case turbulent

$$N_u = 0.035R_e^{0.75}P_r^{0.33} \tag{20.55}$$

In the laminar case, 10.0 and 8.1 are the recommended steady-flow compression-driven and advection-driven values for parallel plates in [16] and scale factor c is the above function of aspect ratio, which was obtained by curve fitting to data in Kays and London [35] (fig 6-1, p. 120) for laminar rectangular-tube Nusselt numbers, expressed as a fraction of the parallel-plate value, under constant heat-flux boundary conditions.

20.7.3 Axial-Conduction Enhancement

Same as equations (20.42) and (20.43) for tubular ducts since the flow microstructure is likely to be similar.

20.8 Rectangular Fins


TFinDct

The rectangular fin heat exchanger is a minor variation of the rectangular duct heat exchanger that uses the same type of gas domain but alters the solid variables passed to any child thermal-solid components. It presumes the rectangular passages are formed by alternating rectangular fins. Accordingly, it adds the input variable:

Tfin : (real, m) Fin thickness t_f.

and overrides wall thickness t_w (Twall) to be a dependent variable, calculated as $t_f/2$. That is, half the fin thickness is presumed to belong to the side-wall of the adjacent duct. It calculates solid cross-section area as the fin-only cross section

$$A_s = nt_fh_i \tag{20.56}$$

where h_i (Hchan) is the fin height. This gives any distributed-conductor child model components the appropriate cross-section area for modeling fin conduction (see chapter 17). Note that solid cross section area does not include end walls and fin height is presumed equal to channel height.

The correlations for f, N_u and N_k are the same as for rectangular channel heat exchangers.

20.9 Generic Cylinder


TGnrCyl

The generic-cylinder variable-volume component adds variables:

Swet : (real, m^2) Time-mean wetted surface S_0. Not extremely critical since it has only a secondary effect on the cycle pressure through the heat transfer between the gas and wall. But there is also a thermodynamic irreversibility associated with this heat transfer which affects overall efficiency somewhat. As a rule of thumb, a value of Swet within fifty percent is good enough. Too much accuracy is not justified because of the relatively large error band for the Nusselt number correlation.

Volume : (real, m^3) Baseline volume V_0 when all volume-displacement attachments are zero. Each volume displacement adds to the baseline volume. If all volume displacements are phasors (attached to sinusoidal moving part with zero mean) then V_0 is the same as the time-mean cylinder volume. Of primary importance since it directly effects the pressure ratio of the machine. Relative accuracy within a few percent is recommended.

Sratio : (real, m) An output variable which expresses the time-mean surface area S_0 as a fraction of the surface area of a minimal-surface right-circular cylinder with the same V_0 ($S_{min} = 5.54 V_0^{2/3}$). Useful for seeing if the Swet you have entered is reasonable compared to the Volume, or for constraining when optimizing Swet and Volume. If Sratio $\ll 1$, then your Swet is too small. If Sratio $\gg 1$, then either your Swet is too large or your volume has a lot of extra wetted surface area within it. Sratio has no effect on the numerical solution so you may safely ignore it if you want.

Generic cylinders calculate time-mean flow area as

$$A_f = V_0 / L \tag{20.57}$$

They calculate solid cross-section area for use in thermal-solid child model components as

$$A_s = t_w S_0 / L \tag{20.58}$$

where t_w is the wall thickness input and L is the length input.

The time-varying part of volume comes from volume-displacement child model components of the associated variable-volume gas domain. The assumption is that the length remains fixed but the flow area varies as a function of time to produce the required volume variation. This makes the equations simpler.

20.9.1 Length

So what is the *length* anyway? There are two ways to look at it.

The first is in terms of the underlying gas domain model which consists of a one dimensional passage of some length and cross-sectional flow area, as shown in section 18.2. So the problem is how to represent a variable volume space as a one dimensional passage? Here is one way: You specify the length as an input variable — say the average length of a streamline between the entrance to the space and its stagnation point on the opposite wall or moving piston face. Sage then computes the cross-sectional area from the above formula. Although this may seem somewhat arbitrary, it does not matter much. The length may vary by an order of magnitude or so without producing much effect on the simulation outputs. This is because nothing much in the variable-volume gas domain model depends on length or flow area. Volume and wetted surface are the main things that matter. So the recommendation is to pick a rough value of Length scaled to the machine you are modeling and then forget about it.

The second way to look at length is in terms of the underlying solid wall model. In the event the wall models an external conduction path where axial conduction is an important feature, then this length might override the gas domain length in modeling importance. In most cases, though, the solid wall is thermally isolated from everything except the gas, in which case the solid conduction length is irrelevant.

20.9.2 Friction Factor

The friction factor is set to $f = 0$ as a reasonable approximation for all likely applications.

20.9.3 Nusselt Number

This correlation is taken from personal memoranda [14, 15, 17] which derived a formulation for Nusselt number in SCFusion cylinder spaces, based on an analytic expression for wall heat flux in a gas undergoing compression and expansion between parallel walls. In place of molecular conductivity, the derivation used a turbulent conductivity, assumed to increase linearly with distance from the wall. Flow into the cylinder was the source of turbulence.

The correlation is broken into real and complex parts N_{u0}, and $\boldsymbol{N}_u$ that apply to the steady and fluctuating components of the gas-to-wall temperature difference, as discussed in section (18.6.2). The steady part is, for lack of any compelling evidence to the contrary, just the steady laminar Nusselt number for tubes scaled by the spatial-mean ratio of turbulent to molecular conductivity N_k. The complex part reflects a calibration developed in [17] of the analytic expression for fluctuating wall heat flux for cylinders with inflow produced turbulence reported in [6].

$$N_{u0} = 6.0 N_k \tag{20.59}$$

$$\Re(\boldsymbol{N}_u) = \begin{cases} \sqrt{2V_aP_r} & \text{if } R_t \le 7.7\sqrt{2V_aP_r} \\ \dfrac{0.13R_t}{\ln\left(\dfrac{0.35\,R_t}{\sqrt{2V_aP_r}}\right)} & \text{otherwise} \end{cases} \tag{20.60}$$

$$\Im(\boldsymbol{N}_u) = \frac{\tanh(0.40/\eta)}{1+3\,\eta}\Re(\boldsymbol{N}_u) \tag{20.61}$$

where

$$\eta = \sqrt{\frac{R_t}{7.85V_aP_r}} \tag{20.62}$$

A correction (in Sage v12) improved this formulation for the case when inflow produced turbulence is small (e.g. modeling a variable volume gas spring without flow connections). For that case the above value for η can fall below the theoretical value reported in [38] for heat transfer in cylinder spaces without turbulence, where the theoretical value of η is

$$\eta = \sqrt{\frac{8}{V_aP_r}} \tag{20.63}$$

To correct that problem and also to avoid numerical noise in the solution caused by noise in the R_t solution, Sage replaced the calculation of η using equation (20.62) with the non-turbulent value of equation (20.63) below the cutoff value $R_t = 62.8$. The cutoff value is the value of R_t where the right-hand sides of equations (20.62) and (20.63) are equal.

20.9.4 Axial-Conduction Enhancement

According to a variation of the Prandtl mixing length hypothesis, turbulent viscosity may be expressed in terms of turbulent Reynolds number as

$$\frac{\mu_t}{\mu} = R_t\frac{\ell}{d_h} \tag{20.64}$$

where ℓ is a mixing length scale. Based on personal memorandum [17], a mean representative value for ℓ/d_h in cylinder spaces is

$$\frac{<\ell>}{d_h} \approx 0.014 \tag{20.65}$$

Assuming that the turbulent Prandtl number is unity, the equation for enhanced (turbulent) conductivity ratio becomes simply

$$N_k = 1 + 0.014R_t \tag{20.66}$$

The leading constant covers the limiting case of $R_t = 0$.

Chapter 21

Compliance Ducts

Compliance ducts are similar to heat exchanger ducts except they are intended to transfer PV power from one end to the other along a significant temperature gradient while minimizing the thermal loss between the two ends. They are often known as *thermal buffer tubes* because of this functionality. Sometimes they are called *pulse tubes*. Compliance ducts are analogous to the heat-exchanger geometries documented in chapter 20. They occur in the toolbox of the root-level model component.

21.1 Duct Geometries

Compliance ducts are typically implemented as single tubes because of the need to minimize heat transfer with the walls and other thermal losses for a given flow area.

21.1.1 Compliance Tube

TTubComplDct

The compliance tube is a descendant of the stirling-class tube-bundle heat exchanger. The main difference is that it substitutes a compliance-duct gas domain in its toolbox (below) and adds a radiation-transport model component, in case you want to model radiation transport along the tube. Wall conduction is available in the thick-wall toolbox component already present.

From your point of view, you may think of a compliance tube as a special sort of tubular container. Its variables are the same as those the tube-bundle heat-exchanger model component. Most of the interesting stuff belongs to the compliance-duct gas domain child component which you drop in from the component palette.

21.1.2 Tapered Compliance Tube

A tapered compliance tube is similar to an ordinary compliance tube except its internal diameter and wall thickness are specified by cubic splines:

TTubxComplDct

Dtube : (cubic spline, m) Tube internal diameter.

Twall : (cubic spline, m) Wall thickness.

The main reason for using a tapered compliance tube is to reduce or eliminate streaming convection by taking advantage of the dA/dx dependence of the effective wall-velocity u_w, according to equation (21.17). The proper choice of dA/dx will cancel the other term, thereby leading to zero streaming convection. Locally, at least. There may still be some variation of u_w along the tube length but it can generally be reduced compared to the case for uniform flow area. The demonstrated reduction of streaming convection by compliance-tube tapering has been reported by Olson and Swift in reference [46].

The implementation of the tapered compliance tube is similar to that of the tubular-cone canister reported in chapter 19. Dependent variables Aflow, Asec and Pwet (flow area, solid cross-sectional area and wetted perimeter also become cubic spline variables instead of uniform values. These splines are each defined by n data pairs $(x, A(x))$ equal spaced in x within the interval $[0, 1]$, where n is the larger of the number of data pairs in the specification of Dtube or Twall. So, the resolution of the dependent variables is only as good as the resolution of Dtube or Twall. In particular, if your intent is a nonlinear variation in flow area (as in a simple conical tube) you must specify more than the endpoint diameters in Dtube, even though it varies linearly with axial position. Specifying *Dtube* interpolation pairs at the relative x coordinates 0, 0.25, 0.5 and 1.0 would probably be sufficiently.

The turbulence transition model along with the heat-transfer and flow-friction correlations are unchanged from those of uniform area ducts. This is probably incorrect to some degree, with the result that heat transfer and flow friction will probably be off a bit. This should not pose a problem for compliance-tube modeling because these terms are generally small.

21.2 Compliance Duct Gas Domains

The compliance-duct type of gas domain descends from the duct-type gas domain documented in the chapter 18. The major difference is the special treatment of the convective losses that arise in the duct in the presence of an axial temperature gradient. These losses are critical because they carry thermal energy from the warm to the cold end of the compliance duct and degrade heat lift.

The gas axial conduction term q now comprises molecular conduction q_m, turbulent conduction q_t, free convection q_f, boundary convection q_b and streaming convection q_s, all considered in detail below. We can write this in dimensionless form as

$$\frac{q}{q_m} = 1 + \frac{q_t}{q_m} + \frac{q_f}{q_m} + \frac{q_b}{q_m} + \frac{q_s}{q_m} \tag{21.1}$$

where q/q_m on the left is just the axial conductivity enhancement ratio N_k and the terms on the right may be viewed as its component parts.

The variables of a compliance-duct gas domain are mostly the same as those in any gas domain documented in 18. New variables are:

Tmult : (real, dimensionless) Empirical multiplier for turbulent conduction q_t.

Cmult : (real, dimensionless) Empirical multiplier for free conduction q_f.

TiltAngle : (real, rad) Inclination angle θ for free convection. $\theta = 0$ for vertical cold-end-down orientation. $\theta = 90$ degrees for horizontal orientation and $\theta = 180$ degrees for unstable vertical cold-end-up orientation.

Gmult : (real, dimensionless) Gravitational field multiplier for free convection. The external gravitational field relative to the value on Earth. Gmult = 1 corresponds to 9.8 m/s^2.

Bmult : (real, dimensionless) Empirical multiplier for boundary convection q_b.

Smult : (real, dimensionless) Empirical multiplier for streaming convection q_t.

QmolMean : (real, W) t and x averaged molecular conduction flow q_m.

QturbMean : (real, W) t and x averaged turbulent conduction flow q_t.

QfreeMean : (real, W) t and x averaged free convection q_f.

QoscMean : (real, W) t and x averaged boundary convection q_b.

QstrMean : (real, W) t and x averaged streaming convection q_s.

GvibMean : (real, m/s^2) x averaged vibratory stabilization field g_v of equation (21.7) that tends to align density gradients along the duct axis. In free convection g_v opposes the free convection driven by the gravitational field g.

The empirical multiplier inputs give you free reign to scale any of the enhanced axial conduction mechanisms individually. They may still be scaled all-together by the inherited duct-gas input Kmult. Outputs QmolMean, ..., QstrMean are components of the inherited output QxMean, not additional losses. Their sum equals QxMean.

As of Sage version 11 there are two new inputs TiltAngle and Cmult (*see* above), and the meaning of Gmult has been revised. In previous versions there was no tilt angle input and setting Gmult to zero was the way to turn off or scale free convection. Under version 11 Gmult always refers to the external gravitational field and Cmult serves as an overall scale factor you can use to calibrate Sage to any free convection data you may have. Gmult and TiltAngle should be set according to the operating environment based on the examples of this table:

Environment	Gmult	TiltAngle (degrees)
Cold end down – on Earth	1	0
Cold end up – on Earth	1	180
Cold end down – on Moon	0.16	0
Weightless	0	NA

The role of the compliance tube is to transfer the PV work of expansion from the cold end of the compliance tube to the warm end, thereby sucking heat out of the cold source, according to a first-law energy balance. Ideally this happens in the absence of any wall heat transfer or fluid mixing and the gas within the compliance tube oscillates back and forth with a uniform velocity profile across the tube cross section. Of course, in reality, there is wall heat transfer, fluid mixing and a non-uniform velocity profile. There may even be turbulence in ill-designed compliance tubes. All these real-world phenomena buck the ideal enthalpy flow with oppositely-directed convective losses. In any particular design instance, some may be small and some may be large, but prudent analysis demands we evaluate them all. That way none are liable to grow out of bounds during optimizations.

21.2.1 Turbulent Conduction

Turbulent conduction q_t is the same as that reported for axial-conduction enhancement in tube-bundle heat exchangers in chapter 20. Repeated here for convenience, the correlating expression is

$$\frac{q_t}{q_m} = 0.022 R_e^{0.75} P_r \tag{21.2}$$

21.2.2 Free Convection

Note: This is a revised formulation as of Sage version 11. Compared to earlier formulations it includes tilt angle dependence and the suppression effect of high frequency oscillatory flow.

A gravitational field acting upon the temperature-induced density gradient within a compliance tube can produce buoyant instability that leads to free convection cells. Free convection is only a problem in compliance tubes oriented with the cold end higher than the warm end.

For correlating free convection the key dimensionless group is the Grashof number, defined here as

$$G_r = \frac{g \rho^2 L^4}{\mu^2} \frac{T_x}{T_{avg}} \tag{21.3}$$

where g is the acceleration of gravity, ρ is mean fluid density, L is tube length, T_x is the spatial temperature gradient and T_{avg} is the spatial-average temperature. In the above definition, the coefficient of fluid thermal expansion $\beta = -1/\rho \frac{\partial \rho}{\partial T}|_P$, which normally appears in the definition of Grashof number, has been replaced by its ideal-gas value $1/T$.

Tilt Angle Dependence

For a tilted pulse tube the component of gravity in the direction of the tube axis is $-g\cos\theta$, where θ is the tilt angle measured from $\theta = 0$ at cold-end-down orientation. $-g\cos\theta$ has the value g at $\theta = 180$ degrees and drops to zero at $\theta = 90$ degrees. So as a first correction to capture the effects of pulse-tube tilt angle it is reasonable to substitute $-g\cos\theta$ for g in the Grashof number. This makes sense for a narrow tube but for a wide tube there will still be some free convection because parts of the cold end are still higher than parts of the warm end at $\theta = 90$ degrees. The angle where *all* of the cold end is lower than *all* of the warm end is $90 - \arctan(D/L)$ degrees. So it makes sense instead to replace g in the Grashof number with $-g\cos\theta^*$, where θ^* is an effective tilt angle defined as (in degrees)

$$\theta^* = \min(\theta + \arctan(d/L), 180) \tag{21.4}$$

Apart from affecting the Grashof number the tilt angle also affects the boundary conditions that shape the overall convective flow within the tube. Free convection in tilted pulse tubes is not highest at a tilt angle of 180 degrees as one might expect. Rather it is around twice as high at a tilt angle on the order of 135 degrees according to measurements by Swift and Backhaus in reference [66] and Ross & Johnson in reference [54]. This may be due to the shape of the convection cell changing from axi-symmetric toroidal to an anti-symmetric shape where flow is upward on the high side of the tube and downward on the low side, similar to flow in a ketchup bottle. Whatever the reason, Sage includes in its formulation for free convection a scaling factor $F(\theta)$ derived from a quadratic curve fit to the tilt-angle dependence measured in references [66] and [54].

$$F(\theta) = 2.12 - 1.62(\theta - 2.31)^2 \tag{21.5}$$

where θ is in radians. By design, for cold-end-up orientation $F(\pi) = 1.0$. The fit to data is not especially good so errors on the order of 50% are not out of the question.

Vibratory Stabilization

Besides the effects of tilt angle there is also an oscillatory flow mechanism that tends to align density gradients along the direction of vibration. For this reason high-frequency pulse tubes can often operate in any orientation without any free convection loss. Swift and Backhaus [66] developed a theoretical framework for this vibratory stabilization mechanism and ran experiments to validate their theory. Gedeon memorandum [26] discusses this vibratory stabilization mechanism in detail and derives a stability condition, namely, convectively stable if

$$-g\cos\theta^* < 0.74\frac{a^2\omega^2}{(L+d)}\left(\frac{|\Delta T|}{T_{avg}}\right)^{1/2} \tag{21.6}$$

where a is the section-mean fluid amplitude, ω is the angular frequency and ΔT is the end-to-end temperature difference. The leading coefficient provides

a reasonable fit to measured data. The right hand side is interpreted as a stabilizing vibratory acceleration

$$g_v \equiv 0.74 \frac{\omega^2 a^2}{(L+D)} \left(\frac{|\Delta T|}{T_{avg}} \right)^{1/2} \tag{21.7}$$

The Sage formulation assumes that the net difference $(-g \cos\theta^* - g_v)$ is what drives the free convection, so the original factor g in the Grashof number is replaced with the effective value

$$g_e = \max\left[(-g \cos\theta^* - g_v), 0\right] \tag{21.8}$$

In other words, the effective gravitational acceleration is $(-g \cos\theta^* - g_v)$, except zero (stable) if negative.

Free Convection Formulation

Sage calculates free convection q_f using a correlation by Hollands in reference [30], modified according to the above considerations. Normalized by molecular conduction the Sage correlation is:

$$\frac{q_f}{q_m} = 0.20X \left(1 - e^{1.19 \frac{(X_c - X)}{X_c}} \right) F(\theta) \tag{21.9}$$

where

$$X = R_a^{1/3} \tag{21.10}$$

and X_c is the critical value of X, to be explained shortly. Free convection is zero (molecular conduction only) for $X \leq X_c$. Rayleigh number R_a is the product of effective Grashof and Prandtl numbers

$$R_a = G_{re} P_r \tag{21.11}$$

where effective Grashof number is calculated in terms of the above effective gravitational field g_e as

$$G_{re} = \frac{g_e \rho^2 L^4}{\mu^2} \frac{T_x}{T_{avg}} \tag{21.12}$$

The critical Rayleigh number, used to calculate $X_c = {R_{ac}}^{1/3}$, is computed as

$$R_{ac} = \frac{(a_1^2 + 15.9)^3}{a_1^2} \tag{21.13}$$

where

$$a_1 = 0.75 a_0 \tag{21.14}$$

and

$$a_0 = 7.66 \frac{L}{d} \tag{21.15}$$

The "a" values represent the dimensionless *horizontal wave number* for the first-mode convection cell within the tube. The value of a_0 is the value recommended

for vertical cylinders by Edwards and Catton in reference [11]. a_1 is a correction by Hollands in [30] for adiabatic wall boundary conditions. The leading coefficient in correlation (21.9) differs from the 0.0585 value recommended by Hollands. It is the result of calibration to free convection measurements by Swift and Backhaus in reference [66].

In arguing for the form of his correlating expression, Hollands presents an informative and convincing case, worth repeating. It goes something like this: For very small L/d we have essentially convection in an infinite horizontal fluid layer heated from below. In such a geometry the top-heavy fluid layer becomes unstable at some critical Rayleigh number at which point convection cells appear, causing an enhancement to molecular-conduction heat transfer. As Rayleigh number increases, additional smaller-scale cells appear, further enhancing heat transfer. Finally, turbulence ensues at high-enough Rayleigh number. When vertical walls interrupt the horizontal layer, they suppress any convection cells smaller than the wall spacing, or tube diameter in our case. Thus the critical Rayleigh number at which the first convection cell occurs increases with reducing diameter (increasing L/d). Eventually though, at sufficiently high Rayleigh numbers, the conduction enhancement approaches that of an unobstructed horizontal layer, which is correlated by the Globe and Dropkin expression $q_f/q_m = 0.069 G_r^{1/3} P_r^{0.407}$. Hollands correlation captures both the critical Rayleigh number at which convection first appears and the high-Rayleigh-number limit.

21.2.3 Boundary Convection

Boundary convection is a sort of *shuttle* heat transfer produced by the convoluted near-wall velocity profile of oscillating flow. It is the analog of displacer shuttle heat transfer in a conventional stirling machine. At high Valensi numbers, the convoluted velocity and temperature profiles within the viscous and thermal boundary layer of oscillating flow interact to produce a convective thermal energy transport beyond that produced by the complex-valued Nusselt number model in Sage. Reference [13], which applies to laminar incompressible flow between parallel plates with an axial temperature gradient, shows that the effect is small so long as Valensi number remains below about 100. However, typical compliance tubes involve Valensi numbers significantly higher than 100.

Figure 6 of that paper shows a curve A, corresponding to the exact-solution enthalpy transport, and a curve B, corresponding to that of a solution based on complex-valued Nusselt number, like that of Sage's duct-gas model. The difference between these curves is the boundary convection q_b we are looking for. Taking the high Valensi number limits of the equations for these curves (equations (65) for $P_r = 0.7$, $\sigma = \infty$) and normalizing by molecular axial conduction q_m gives normalized boundary convection in the form

$$\frac{q_b}{q_m} = 0.159 \sqrt{V_a} (2\delta/d)^2 \tag{21.16}$$

where δ is the section-mean flow tidal amplitude and d is hydraulic diameter.

Although the above results were derived for parallel-plate flow, cast as they are in terms of hydraulic diameter, they apply to tubes as well as to narrow channels or annuli. This follows because boundary convection, in the high-Valensi-number limit, occurs entirely within a thin layer (thickness on the order $d/\sqrt{V_a}$) near the wall. As such, the phenomena is insensitive to the exact shape of the duct and amounts to a convective energy flow per unit wetted perimeter. The actual value of this energy flow per unit wetted perimeter could be obtained by multiplying either previous expression by the molecular conduction per unit area $q = -kT_x$, then by the ratio of flow area to wetted perimeter $d/4$. As expected, in the high-Valensi limit (neglecting the N_u term), the result would be seen to be independent of d or any other geometrical parameter. Assuming, in particular, it would be applicable to tubes, we could then convert it back to the original form by dividing by the area-to-perimeter ratio ($d/4$, same as for plates) and then again by q.

There remains the question of what happens in turbulent flow. Evidence suggest we might then base our boundary convection equation on an effective Valensi number where molecular viscosity is replaced by turbulent viscosity — which is generally very much higher. Thus, effective Valensi number is generally greatly reduced for turbulent flow and q_b would not be as large as predicted by equation (21.16). However, since enhanced axial conduction is likely to prohibit any reasonable compliance tube from operating in the turbulent flow regime in the first place, the issue is probably moot. We might as well ignore any turbulent correction to q_b, thereby giving added incentive for flow to remain laminar.

21.2.4 Streaming Convection

Note: This is a revised formulation as of Sage version 11. Compared to earlier formulations it includes the suppression effects of cold-end-down orientation and high frequency oscillatory flow.

Streaming convection is produced by a second-order steady flow circulation within the tube interior, superimposed on the main oscillating-flow velocity field. It has no counterpart in displacer-type stirling machines. In the high Valensi-number limit (viscous and thermal boundary layer thicknesses small compared to tube diameter) the second-order terms of an oscillating velocity field produce an effective wall-velocity, often called Rayleigh streaming, replacing the usual no slip condition. This effective wall velocity u_w may be calculated using the formulation below derived by Olson and Swift [46]. *Note: prior to Sage version 7 the formulation omitted some of the smaller terms — effectively assuming* $C_1 = C_2 = 3/4$, $C_3 = 0$.

$$u_w = \frac{|P - P_0||u|}{\gamma P_0}\left[C_1 \cos\theta + C_2 \sin\theta\right] + \frac{|u|^2}{\omega}\left[\frac{3}{4}\frac{dA/dx}{A} + C_3\frac{dT_m/dx}{T_m}\right] \quad (21.17)$$

where

$$C_1 = \frac{3}{4} + \frac{(\gamma - 1)(1 - eP_r^2)}{2P_r(1 + P_r)}$$

$$C_2 = \frac{3}{4} + \frac{(\gamma-1)(1-e)\sqrt{P_r}}{2(1+P_r)}$$
$$C_3 = \frac{(1-e)(1-\sqrt{P_r})}{4(1+P_r)(1+\sqrt{P_r})}$$

$|P - P_0|$ is the pressure-fluctuation amplitude, P_0 is time-mean pressure, $|u|$ is the section-mean velocity amplitude, θ is the phase angle between them (velocity phase minus pressure phase) and A is the flow area. γ is the ratio of specific heats, P_r the Prandtl number and e represents the temperature dependence of viscosity, assumed to be of the form

$$\mu(T) = \mu(T_0)\left(\frac{T}{T_0}\right)^e \tag{21.18}$$

Sage calculates the e exponent at each point of evaluation based on the viscosity variation around T_m using a central finite differencing of the Sage gas-state viscosity function $\mu(T)$

$$e = \frac{T_m}{\mu(T_m)}\frac{d\mu}{dT} \approx \frac{\mu(T_m + \epsilon T_m) - \mu(T_m - \epsilon T_m)}{2\epsilon\mu(T_m)}$$

The time-average fluid flow in the central part of the compliance tube (away from the wall boundary layer) must satisfy the u_w boundary condition while maintaining zero section-mean flow. Prior to v11 the Sage streaming loss formulation assumed that u_w produced a zero-shifted parabolic velocity profile in the tube interior. As of v11 this has been revised to reflect the buoyant stabilization forces of cold-end-down orientation and oscillatory mean flow that tend to overwhelm this parabolic velocity profile by aligning density gradients along the direction of gravity and vibration, respectively. The result is a relatively narrow convection cell near the wall, consisting of two streams flowing in opposite directions carrying fluid from regions of different temperature. The heat exchange between the two streams results in a transverse temperature profile, which multiplied by the velocity profile produces a net (section-average) thermal streaming convection.

The account here is a summary of the streaming convection formulation derived in Gedeon memorandum [25], which includes a comparison to published data for tapered pulse tubes.

Effective Reynolds number For correlating the streaming convection loss the dimensionless group of interest is a Reynolds number based on effective wall streaming velocity and a characteristic dimension b defined as the distance from the wall to the point where the interior streaming velocity u_s is zero — the thickness of the near-wall leg of the convection cell.

$$R_{eb} = \frac{\rho|u_w|b}{\mu} \tag{21.19}$$

Reference [25] derives the value of b by assuming a cubic form for the interior flow velocity profile u_s then solving for the parameter b by balancing the viscous pressure gradient driven by streaming at the wall against the stabilizing pressure gradients of buoyancy and the oscillatory mean flow. The result is this approximation for the characteristic convection cell thickness.

$$b \approx f_c \left[2.8 \frac{(\mu/\rho)^2 L}{P_r \frac{\Delta T}{T} \left(2g \cos\theta^* + \frac{a^2\omega^2}{L} \frac{\Delta T}{T} \right)} \right]^{1/4} \tag{21.20}$$

where f_c is a calibration coefficient, μ is fluid viscosity, ρ is fluid density, T is fluid temperature, ΔT is the end-to-end temperature difference, $g \cos\theta^*$ is the effective gravitational acceleration in the tube direction (*see* equation (21.4)), $a\omega$ is the mean-flow velocity amplitude and L is the tube length. According to [25], the value $f_c = 2.1$ fits available data.

The streaming convection loss is based on a formulation q_{s0} valid at low streaming velocities and a formulation $q_{s\infty}$ valid at high streaming velocities. The low-streaming-velocity relative streaming convection is

$$\frac{q_{s0}}{q_m} \approx 1.02 \, (R_{eb} P_r)^2 \, \frac{b}{d} \tag{21.21}$$

The high-streaming-velocity limit is

$$\frac{q_{s\infty}}{q_m} \approx 1.69 R_{eb} P_r \frac{L}{d} \tag{21.22}$$

The streaming convection at any streaming velocity is calculated as a linear combination of the low and high streaming velocity limits of the form

$$q_s = (1 - W) q_{s0} + W q_{s\infty} \tag{21.23}$$

Where W is a weight function defined in terms of the critical R_{eb} value at which $q_{s\infty} = q_{s0}$. W smoothly transitions from the value 0 for R_{eb} less than 0.8 of the critical value to 1 for R_{eb} greater than the critical value. This avoids a corner at the critical value in the streaming loss as a function of R_{eb} in order to avoid numerical solution stability problems. Equating the right-hand sides of equations (21.21) and (21.22) and solving for $R_{eb}P_r$ gives the critical R_{eb} as the value that satisfies the equation

$$R_{eb} Pr = 1.66 \frac{L}{b} \tag{21.24}$$

The above correlations are derived in Gedeon memorandum [25] by solving a laminar gas energy equation.

$$\rho u_s T_x = \frac{k}{c_p} \frac{\partial^2 T_d}{\partial y^2} \tag{21.25}$$

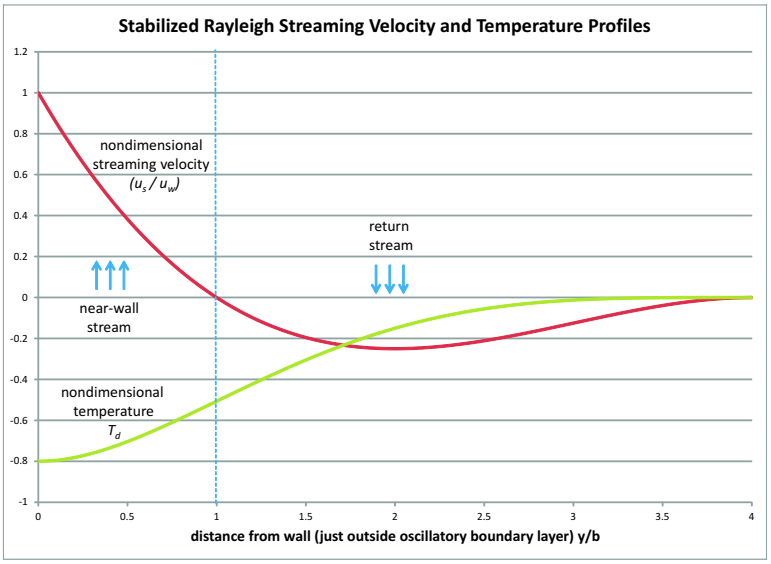


Figure 21.1: Streaming velocity u_s and temperature solution T_d extending a distance $4b$ from the wall. The near-wall stream drives the temperature one way and the return stream drives it the other way. Radial thermal diffusion tends to equalize the temperature.

which ignores the term $\rho v_s \frac{\partial T_d}{\partial y}$ on the left side associated with the energy transport produced by the transverse component of the flow velocity. This equation assumes a coordinate frame moving along with the section-mean flow where T_d represents the dissipative part of the stationary temperature solution in that reference frame. u_s is the time-average streaming velocity in the tube interior. T_x is the axial temperature gradient, assumed constant. The streaming velocity u_s drives the T_d solution. It switches sign from the near-wall to return streams and tends to increase the magnitude of the convection cell temperature difference ΔT_d in proportion to the streaming velocity. The streaming loss is the section average (denoted by $\langle\cdot\rangle$ brackets) of the streaming enthalpy flow, denoted here with zero subscript

$$q_{s0} = c_p \langle \rho u_s T_d \rangle \tag{21.26}$$

Figure 21.1 shows the nondimensional streaming velocity u_s and dissipative temperature T_d near the wall.

Pulse-Tube Escape Velocity and the Importance of Flow Straightening

Wall streaming is not the only source of forced convection in compliance tubes. A useful quantity to keep in mind is the velocity for an upward directed fluid jet starting at the cold bottom of a pulse tube to have enough kinetic energy

to make it to the warm top — the jet escape velocity. Such jets can arise from the relatively high fluid velocities in ducts or manifolds at the ends of the pulse tube or flow separations at the tube entrance if suitable care is not taken to design flow straighteners at the ends of the pulse tube (usually layers of woven wire mesh) or manifolds with very low fluid velocities.

Ignoring the stabilizing effects of the oscillatory mean flow and considering only buoyant stabilization of cold-end-down orientation, the cold jet escape velocity is in terms of gravitational field g and tube height L (length) is

$$u_c = \sqrt{(1 - \rho_h/\rho_c)gL} \tag{21.27}$$

where ρ_h and ρ_c are the fluid densities at the warm and cold ends. This is easy to derive by equating the kinetic energy per unit volume of the cold-end jet $\rho_c u_c^2/2$ with the potential energy change moving through the surrounding fluid $\int_0^L (\rho_c - \rho) g dx = (\rho_c - \rho)gL/2$ (assuming a linear density variation between ρ_c and ρ_h) then solving for u_c. Using the ideal gas relationship $\rho = P/(RT)$, the cold jet escape velocity may be written

$$u_c = \sqrt{(1 - T_c/T_h)gL} \tag{21.28}$$

Arguing similarly the escape velocity for a warm jet starting at the top to reach the bottom is

$$u_h = \sqrt{(T_h/T_c - 1)gL} \tag{21.29}$$

A cold jet that makes it to the warm end also displaces warm fluid to the cold end (neglecting heat transfer with the surrounding fluid), resulting in a round-trip enthalpy flux that may be quite significant compared to wall streaming convection.

21.3 Numerical Diffusion

Properly modeling, of even an ideal compliance tube, requires quite a few control volumes (NCell) to reduce numerical truncation errors to an acceptable level — typically about 7 or 8 in Sage. The problem is that the temperature field is mostly carried with the flow (rather than fixed to the matrix as in the case of a regenerator) which is a non-optimal state of affairs for the spatially fixed (Eulerian) solution framework of Sage.

The problem grows with increasing flow amplitude so that the error in predicted net enthalpy flux generally becomes quite significant when the tidal amplitude of the mean flow is more than about 25% of the duct length, or whenever the flow amplitude is high enough that a fluid particle at the inlet at the time of flow reversal travels more than about half way down the duct. The error tends to reduce the net refrigeration power, much as an increased gas thermal diffusion would, hence the name *numerical diffusion*.

Experience suggests that numerical diffusion is something to keep in mind but not worry about too much. It is generally not a problem during optimizations because there is usually no compelling reason for the optimizer to drive a

δ/L	NCell = 6 NTnode = 6	NCell = 8 NTnode = 8	NCell = 10 NTnode = 10	NCell = 12 NTnode = 12
0.1	0.0009	0.005	0.0004	0.0003
0.2	0.0061	0.0046	0.0036	0.0027
0.3	0.0184	0.0156	0.0127	0.0103
0.4	0.0387	0.0360	0.0315	0.0272

Table 21.1: Numerical diffusion error h/h_0 for different values of tidal amplitude ratio δ/L and grid cell numbers.

compliance duct into the region of significant numerical diffusion. So, the error tends to be self limiting. The main problem is reduced simulation accuracy when validating experimental data at large compliance-duct tidal amplitudes.

21.3.1 Quantitative Estimates

The actual amount of numerical diffusion can be measured by comparing Sage simulated results against an exact analytic solution for which enthalpy transport is zero. Such a solution is that of an ideal adiabatic compliance tube filled with non-axially-conductive gas moving back and forth with uniform sinusoidal velocity, as would be produced by two parallel-moving pistons within cylinder spaces attached to either end. If the piston cylinders are each held at different temperatures (say by means of two heat exchangers positioned between them and the compliance tube) then an equilibrium temperature profile will establish itself within the compliance tube. Provided the tidal amplitude remains low enough, this temperature profile will consist of two isothermal legs from either inlet up to the maximum penetration depth (of a fluid particle that was at the inlet at the time of last flow reversal) with a continuous linear variation in between. The linear variation is not strictly-speaking necessary but it is one valid solution. This temperature profile will merely slosh back and forth with the mean flow, the net enthalpy flow down the tube will be zero and there will be no pressure variation. This happy state of affairs continues until the tidal amplitude increases to the point where gas from one end reaches the other. At this point warm gas is suddenly cooled, or vice-versa, with the result that net enthalpy transport is no longer zero.

When modeled by Sage, the idealized compliance-tube produces a net enthalpy transport h as given by table 21.1. The results are normalized by the worst-case enthalpy transport h_0 that would occur in the case of zero tube length

$$h_0 = \frac{c_p}{\pi}\dot{m}_1(T_h - T_c)$$

where $\dot{m}_1$ is the average mass flow rate amplitude (it varies with temperature) and $(T_h - T_c)$ is the temperature difference. Theoretically h/h_0 should be zero up to the point where the tidal amplitude ratio δ/L reaches 1/2.

The results shown that h/h_0 grows quickly as tidal amplitude ratio approaches the critical value of 1/2. This is to be expected since the finite-

difference solution for the pulse-tube tends to round out the corners of a temperature profile that is theoretically approaching a step function as tidal amplitude ratio approaches 1/2. But below a tidal amplitude ratio of about 0.25, erroneous enthalpy transport h/h_0 is less than 0.01.

21.3.2 Interpretation in terms of Refrigeration Power

So how significant is an erroneous enthalpy transport of $h/h_0 = 0.01$? This is hard to say since worst-case enthalpy transport h_0 is not something we usually think about. Instead we usually think about the ideal refrigeration power of a pulse tube which is

$$h_{pv} = \frac{1}{2} P_1 \dot{V}_1$$

assuming pressure amplitude P_1 and volumetric flow rate amplitude $\dot{V}_1$ are in phase. But if we substitute $\dot{m}_1/\rho_0$ for $\dot{V}_1$, then $2P_0/(R(T_h + T_c))$ for ρ_0, we can formulate the ratio of ideal refrigeration power to worst-case enthalpy transport as

$$\frac{h_{pv}}{h_0} = \frac{\pi}{4}(1 - 1/\gamma)\frac{P_1}{P_0}\frac{(T_h + T_c)}{(T_h - T_c)}$$

So, for example, in a helium pulse tube with $(1 - 1/\gamma) = 0.4$, pressure ratio $P_1/P_0 = 0.15$ (somewhat on the high side), $T_h = 300$ and $T_c = 75$, we have $h_{pv}/h_0 = 0.079$. Therefore, an erroneous enthalpy transport of $h/h_0 = 0.01$ corresponds to $0.01/0.079 = 0.13$ of ideal refrigeration power — a significant amount. But the error quickly drops off as tidal amplitudes decreases. So, for tidal amplitude ratios on the order of 0.10, the error would only be of the order 1% of ideal refrigeration power.

Chapter 22

Flow Restrictors

Flow restrictors are composite geometry and gas-domain components that model zero-volume adiabatic flows. They were originally intended for modeling the orifi found in pulse-tube coolers but have evolved over time to represent all sorts of components that embody a relationship between mass flow rate and pressure drop, like pumps, compressors, and so forth. They are found in the toolbox of the root-level model component and are born with gas flow connectors at either x end intended for connection to gas domain model components. While you could also model flow restrictors using a more complicated heat exchanger components, the use of flow restrictors is easier and faster. Keep in mind, though, that flow restrictors are not appropriate where the volume of the component is significant or where there might be a significant amount of heat transfer to or from the gas. Variables common to all flow restrictors are:

Aflow : (real, m^2) mean-flow cross sectional area A.

Pwet : (real, m) Wetted perimeter w.

Fmult : (real, dimensionless) Empirical multiplier for viscous pressure drop.

Tinit : (real, K) Initial temperature. Used only for initializing or re-initializing solution variables ρ and ρe. As such, its value does not affect the final converged solution, but it may affect whether the solution converges at all. It should be set to a value consistent with the initial temperatures of adjacent components.

AEfric : (real, W) Available energy loss to viscous flow friction.

FT : (Fourier series, K) Gas temperature T.

FP : (Fourier series, Pa) Gas pressure P.

FDP : (Fourier series, Pa) Pressure difference across restrictor (positive − negative ends).

FH : (Fourier series, W) Stagnation enthalpy flow $uA(\rho e + P)$.

FRhoUA : (Fourier series, kg/s) Mass flow rate ρuA.

MachMean : (real, dimensionless) Time-averaged Mach number. Keep an eye on this. If it gets near unity Sage may not converge.

ReMean : (real, dimensionless) Time-average Reynolds number R_e.

When needed, flow restrictors calculate hydraulic diameter in terms of flow area A and wetted perimeter w as

$$d_h = 4A/w \tag{22.1}$$

and Reynolds number as

$$R_e = \frac{|\rho u| d_h}{\mu} \tag{22.2}$$

Flow restrictors calculate gasdynamic variables at each end with the same flow area A. So the product of density and velocity ρu is proportional to mass flow rate and is the same at either end of the flow restrictor. Flow is assumed isenthalpic which means that the temperature T is also the same, at least to the extent the gas state is ideal. But density may differ at the two ends, especially if the pressure drop across the flow restrictor is large. It is important to keep this in mind in formulating the pressure drop across the flow restrictor.

22.1 Sintered Powder Plug

TSphRstr

The sintered powder plug adds input variables:

Length : (real, m) Flow length L.

Dcan : (real, m) Diameter D_c of plug or internal diameter of canister holding plug.

Dsphere : (real, m) Powder sphere or particle diameter d_s.

Porosity : (real, m) Plug porosity β (void volume / total volume).

It calculates flow area as

$$A = \beta \frac{\pi}{4} D_c^2 \tag{22.3}$$

and wetted perimeter as

$$w = 6(1-\beta)\frac{\pi}{4} D_c^2 / d_s \tag{22.4}$$

presuming spherical particles. It calculates Darcy friction factor using the same correlation as for the stirling-class packed-sphere matrix heat exchanger geometry, namely:

$$f = (79/Re + 1.1)\beta^{-0.6} \tag{22.5}$$

and computes viscous pressure drop as

$$\Delta P = -\frac{1}{2} f \frac{L}{d_h} \rho_m u|u| = -\frac{1}{2} f \frac{L}{d_h} (\rho u)|\rho u|/\rho_m \tag{22.6}$$

Density ρ_m is the average of density at the two ends but calculating it is a bit convoluted because the solution formulation maintains only the downstream value, denoted by ρ, without a subscript. For purpose of pressure drop calculations then, Sage calculates mean density from the average density of the gas domains on either side of the flow restrictor.

Prior to Sage version 13 (October 2024) ρ_m was formulated in terms of the flow-restrictor downstream temperature T, the average pressure P_m of the gas domains connected on either side, and the gas equation of state $\rho(T, P_m)$, which was potentially unreliable for fluids in the two-phase region, where pressure is constant and specific volume indeterminate. The present method is more reliable and produces essentially the same result.

22.2 Sharp-Edged Orifice

The sharp-edged orifice adds input variables:

TOrfRstr

Dorf : (real, m) Diameter D_o of orifice.

Cd : (real, dimensionless) Discharge coefficient C_d.

Areduc : (real, dimensionless) Ratio A_o/A_h of orifice area A_o to housing area A_h. Must be > 0 and should be < 1.0. Affects the orifice pressure drop (*see* below) and also the Bernoulli pressure change between the ends of the flow restrictor and adjoining gas domains, which see gas entering or leaving the flow restrictor at the velocity in the housing rather than in the orifice itself.

A sharp-edged orifice is defined primarily by the orifice diameter but also by the housing duct in which it is contained. To avoid trouble with high velocities in the flow-restrictor gas kinetic energy term the mean flow area A is taken to be the housing area A_h (upstream and downstream flow area), calculated from inputs D_o and (A_o/A_h) as.

$$A_h = \frac{\pi}{4} D_o^2/(A_o/A_h) \tag{22.7}$$

The cross-section area of the orifice itself is calculated as

$$A_o = A_h(A_o/A_h) \tag{22.8}$$

According to the conventional formulation for positive-directed flow through a sharp-edged orifice, the mass flow rate $\dot{m}$ is governed by

$$\dot{m} = -C_d A_o \sqrt{\frac{\rho_e \Delta P}{(1 - A_o/A_h)^2}} \tag{22.9}$$

where C_d is the discharge coefficient and ρ_e is a representative density, which Sage takes as the downstream density provided pressure drop does not get too large (*see* below). Solving for ΔP, the explicit equation for pressure-drop in a sharp-edged orifice is

$$\Delta P = -\frac{2}{C_d^2}\left(1 - A_o/A_h\right)^2 \frac{\dot{m}|\dot{m}|}{2\rho_e A_o^2} \tag{22.10}$$

The final factor on the right is the velocity head $\frac{1}{2}\rho_e u_o|u_o|$ reckoned at the orifice velocity $u_o = \dot{m}/(\rho_e A_o)$. Discharge coefficient C_d accounts for the actual orifice velocity being somewhat higher than u_o due to the flow stream in the orifice being somewhat contracted below A_o. According to Idelchik ([32], p. 90), C_d ranges from about 0.84 for a thin sharp-edged orifice to 1.14 for a drilled hole in a thick plate ($L/d \geq 2$, sharp edged).

22.2.1 Low-Reynolds Linearization

In the Sage formulation a term $50/R_e$ is added to the factor $(2/C_d^2)(1-A_f/A_h)^2$ in equation (22.10) to improve solution stability when starting from zero-velocity initial values. The added term is reminiscent of the Darcy-flow part of the Ergun friction factor for porous materials. It is significant only for small R_e and serves to linearize ΔP as a function of ρu. For purposes of calculating Reynolds number the wetted perimeter is taken to be that of the housing, calculated as

$$w = \pi D_o/\sqrt{A_o/A_h} \tag{22.11}$$

22.2.2 Choked Flow

Equation (22.10) is fine for relatively low pressure drops. But when downstream pressure P drops to about half the upstream pressure P_u the velocity in the orifice reaches the speed of sound and a further reduction in downstream pressure has no affect on the flow through the orifice. The flow is said to be *choked* at that point. The critical downstream pressure P_c at which this happens is given by Schmidt ([56], p. 337), for the case of frictionless ideal-gas flow as

$$P_c = P_u\left(\frac{2}{\gamma+1}\right)^{\gamma/(\gamma-1)} \tag{22.12}$$

where γ is the ratio of specific heats. For all gases the factor on the right is near 0.5 so Sage simply takes P_c as half the upstream pressure. For the case of choked flow (i.e. downstream pressure $P_d < P_c$) Sage calculates the pressure drop from equation (22.10) evaluated at effective density ρ_e equal to the choke density ρ_c and adds to it the choked pressure drop $\pm(P_c - P_d)$, where the positive sign applies to negative directed flow and vice-versa. Sage estimates choked density ρ_c from the ideal-gas relationship

$$\rho_c = P_c/(RT) \tag{22.13}$$

Essentially, equation (22.10) takes care of the expansion loss as the jet at the orifice throat dissipates in the downstream housing and $(P_c - P)$ takes care of any additional pressure drop beyond the choke point. Estimating choked values P_c and ρ_c from ideal adiabatic relationships is not strictly correct but this component does not aim for perfection. The goal is only to approximate the physics of choked flow, where mass flow rate increases only in proportion to ρ_c under choked conditions. That is clear from equation(22.9), substituting ρ_c for ρ_e, and noting that ΔP is proportional to P_c for choked flow, which is also proportional to ρ_c.

Prior to Sage version 13 (October 2024) ρ_c was calculated from the general equation of state as $\rho(P_c, T)$, which was potentially unreliable for non-ideal fluids in the two-phase region, where pressure is constant and density indeterminate.

22.3 Asymmetric Sharp-Edged Orifice

TAsyOrfRstr

The asymmetric sharp-edged orifice is a variation of a sharp-edged orifice with a pressure recovery diffuser in one direction or the other defined by new input variable:

Recov : (real, dimensionless) Recovery area ratio R.

You might, for example, use such an orifice in a model having a closed flow loop where you need a way to cancel unwanted DC (time averaged) flow circulation. You would do this by optimizing Recov subject to a constraint like FRhoUA.mean = 0.

Asymmetric flow is implemented in terms of two areas A_+ and A_-, the former being the effective orifice flow area for positive flow and the later the effective area for negative flow. The two areas are related by the formula

$$\frac{A_+ - A_-}{A_{min}} = R \tag{22.14}$$

where R is the recovery area ratio and A_{min}, the smaller of the two areas, is the orifice flow area inherited from the parent object. In other words, for $R = 0$ there is no recovery and flow is that of the parent sharp-edged orifice. For $R > 0$ there is a recovery channel in the positive direction with exit area $A_+ = (1 + R)A_{min}$. And for $R < 0$ there is a recovery channel in the negative direction with exit area $A_- = (1 - R)A_{min}$. The succinct way to say all this is that the orifice flow area A_o is always given by

$$A_o = \begin{cases} (1 + \max(R, 0))A_{min} & \text{if } u > 0 \\ (1 - \min(R, 0))A_{min} & \text{otherwise} \end{cases} \tag{22.15}$$

At the root of the DC flow phenomenon are ill-phased density and velocity fluctuations in the gas which tend to produce DC pressure gradients. The issue

is whether or not DC pressure gradients produced by various flow resistances in the closed loop completely cancel each other. If they do not, then DC flow is the consequence. Enthalpy flow in components like the regenerator and compliance tube can be extremely sensitive to DC flow. So, one symptom of DC flow is extreme sensitivity of heat lift to small changes in input variables or grid node counts.

22.4 Check Valve

TCheckRstr

The check valve is a variation of a sharp-edged orifice that behaves like an orifice in one direction but essentially blocks flow in the other direction. New input variables are:

Aratio : (real, dimensionless) Ratio of fully-closed to fully-open flow area R. Generally a positive number $\ll 1.0$. A value of 1.0E–2 is a reasonable value. Backflow leakage increases as Aratio increases and decreases as Aratio decreases. But too small a value may cause convergence problems. The fully open area is the orifice area $\pi D_o^2/4$, where D_o is the inherited Dorf input.

Popen : (real, Pa) Pressure difference ΔP_c required to open the valve. A positive value means positive flow opens the valve. A negative value means negative flow opens the valve. At pressures above ΔP_c (or below if negative) flow resistance changes smoothly to the fully-open value at twice ΔP_c. A zero value means the valve is always open.

Instead of modulating flow by changing the actual flow area, like the asymmetric orifice, a check valve simply adjusts the flow friction to have the same effect. This avoids instabilities due to high gas flow velocities when the area suddenly goes from large to small. In the open position, check valves calculate pressure drop as the orifice pressure drop from equation (22.10), for the full open area $\pi D_o^2/4$. In the closed position, pressure drop is the full-open orifice pressure drop increased by a factor $1/R^2$, where R is the closed-to-open area ratio. So even though actual flow area does not change, the effect on pressure drop is the same as if it had.

Check valves decide whether they are fully closed, fully open or somewhere in between, based on the current pressure drop ΔP compared to the valve opening pressure ΔP_c. The valve starts to open at ΔP_c and is fully open at $2\Delta P_c$. No valve inertia is modeled. In mathematical terms, Sage calculates the valve open fraction as

$$W = \frac{\Delta P}{\Delta P_c} - 1 \tag{22.16}$$

which ranges linearly from 0 at $\Delta P = \Delta P_c$ to 1 at $\Delta P = 2\Delta P_c$. The valve flow restriction is then calculated as

$$A_r = \begin{cases} R & \text{if } W < 0 \\ 1 & \text{if } W > 1 \\ \text{smooth transition} & \text{otherwise} \end{cases} \tag{22.17}$$

Finally, the valve pressure drop compared to the full-open pressure drop ΔP_o is

$$\Delta P = \frac{\Delta P_o}{A_r^2} \tag{22.18}$$

There is a bit of circularity in the above calculations because ΔP depends on W which depends on ΔP. Sage circumvents this difficulty by calculating W with an implicit state variable Fpd substituting for ΔP. After solution convergence Fpd $= \Delta P$.

So why can't a check valve seal perfectly in the back direction? The reason has to do with solution convergence. If the seal was perfect the pressure drop across the valve would be indeterminate for the closed part of the cycle leading to convergence difficulty, especially when solving from initialized values. Even with a flow restriction of R = 1.0E–2 convergence can be problematic. It may be helpful to start with R = 1.0E–1 or larger in order to achieve convergence, then reduce R gradually.

22.5 Time-Dependent Valve

TValveRstr

The time-dependent valve is a variation of a sharp-edged orifice that behaves as if the inherited flow area (that given by input Dorf) is reduced (or increased) by a time-varying multiplier factor

$$F(t) = \max(R(t), R_{min}) \tag{22.19}$$

The time-varying factor $R(t)$ comes from the Fourier-series input variable FRestrict and the minimum allowed value R_{min} from the real input MinRestrict, which must always be positive. A time-dependent valve is intended for modeling such things as the pressurization valve in a Gifford-McMahon cooler. New input variables are:

FRestrict : (Fourier series, dimensionless) Flow area multiplier $R(t)$ above. Keep in mind that valves that open or close abruptly are difficult to resolve in the coarse computational grid employed by Sage. Better to model a more smoothly varying equivalent valve, if possible. Either that or increase the number of time nodes in the computational grid (input variable NTnode). For more information on how you might specify FRestrict to accomplish your objectives, see the section on Fourier Series in chapter 8.

MinRestrict : (real, dimensionless) Minimum flow area multiplier R_{min} above. Generally a positive number $\ll 1.0$. A value of 1.0E–2 is reasonable. Too small a value may cause convergence problems.

Like a check valve, this object does not change the actual flow area, but only adjusts the pressure drop to have the same effect, by calculating it as

$$\Delta P = \Delta P_0 / F(t)^2 \tag{22.20}$$

where ΔP_0 is the pressure drop calculated from orifice equation (22.10) as a function of current mass flow rate, at inherited flow area. This prevents instabilities due to high gas flow velocities when the area is small.

22.6 DC Flow Blocking Restrictors

TSphMembr

TOrfMembr

Another way to cancel DC flow is to use one of these flow-restrictor variations, which descend from the sintered powder plug or the sharp-edged orifice components, respectively. They automatically block DC flow by imposing on top of the pressure drop inherited from the parent component an implicitly-solved DC pressure-drop sufficient to exactly cancel any DC mass flow rate. This DC flow blocking behavior is built into the model solution and does not require an optimization constraint to solve, as in the case of the asymmetric sharp-edged orifice. The amount of DC pressure-drop required appears as output variable DPstdy.

In some cases you may want to regulate the DC flow to some desired set-point rather than canceling it entirely. You can do this by changing the value of input variable DCRhoUA from its default value of zero to the desired set-point.

The additional variables for this component are:

DCRhoUA : (real, Kg/s) set-point for DC (time-average) mass flow rate.

DPstdy : (real, Pa) pressure drop required to achieve the DC mass flow rate set-point.

Variable DPstdy is not the same as the time-mean value of the pressure-drop Fourier series FDP because FDP includes the pressure drop of the ancestor component, which may have its own DC component for reasons outlined above.

While very convenient from a modeling point of view, these components suffer from the disadvantage that they do not specify the mechanism by which DPstdy is to be generated. This is partly a good thing, because the value of DPstdy is subject to considerable numerical error. Changing the number of time or spatial nodes in the computational grid, or even the time-grid orientation with respect to the solution (by changing compressor piston phase), can dramatically affect DPstdy. This is because the time-average pressure drop is the residual of some much larger time-varying pressure drops.

Orifice designers, then, have two choices. Either they ignore DPstdy, hoping that the any DC flow effect will be too small to matter, or they deal with it. Dealing with it might involve designing-in some sort of mechanism to produce the a DC pressure drop bias of the predicted magnitude (allowing time in the test cell to calibrate it). Or it might involve eliminating DC flow by some other means — by means of an impermeable membrane, for example. In fact, it is useful to think of these components as impermeable membranes in series with an orifice.

Another disadvantage of these components is that they must only be used in situations where DC flow is possible (closed loop flow paths, not otherwise DC-flow restricted) lest a solver singularity result. In other words, if DC flow is already implied to be zero, then any DC pressure-drop imposed by these components will have no affect — and may result in a singular solution. This fact places an added burden on the model designer to recognize when DC flow is a problem and to correct it, but not over-correct it. For example, placing two DC-flow canceling components in series, is taboo.

22.7 Mass-Flow Pump

TPump

The universe of modeling options admits to another possibility — one where DC flow is desirable. To model an open thermodynamic cycle, for example. Or to investigate the effect of a prescribed amount of DC flow. This component allows you to model a mass-flow pump by recasting the mass-flow-rate Fourier series FRhoUA as an independent input variable rather than an output. What happens is the pressure drop across the component is solved implicitly in order to make the mass flow rate come out right. You can see what this pressure drop is by inspecting output variable FDP.

You can specify the mass flow rate to be anything you desire — including sinusoidal terms and higher harmonics. This means you can model much more than merely a DC flow generator. You can model the perfect flow restrictor. One that gives any desired mass flow and phasing without regard to the physical means for achieving it. And the subfields of the mass flow rate can be optimized or referenced in constraints, as discussed under the topic of Fourier-series variables in chapter 8. One warning though: If you specify a nonzero DC part of the mass flow rate, the mass-flow pump should be located within a closed loop capable of supporting DC flow, otherwise the mean pressure drop will be indeterminate and the solution will not converge — if DC flow is impossible, then there is no amount of pressure drop that will force the required value.

The flow area Aflow is another input. It is present in order to define flow velocity in the pump, but normally its value does not matter much. However, it should be large enough that the mean Mach number (output MachMean) is substantially below one.

When starting from initial values or a re-initialized solution, a mass flow pump in your model can slow down convergence, depending on the magnitude of the flow specified. The DC component of FRhouUA seems especially troublesome in this regard. If this is a problem, try starting with a small mass flow rate initially and increase it gradually over the course of several solutions.

22.8 Mass-Flow Driver

TDriver

This is a phasor version of the previous mass-flow pump. It specifies mass flow rate as a phasor input variable, having sinusoidal time variation. You can optimize the subfields of the mass flow rate, or reference them in constraints, as discussed under the topics of Complex and Phasor variables in chapter 8.

Why would you want to use a mass-flow driver since you can specify all of the Fourier-series components, not just the sinusoidal part, with the previous mass-flow pump? Because this component allows the DC part and all higher harmonics to pass unobstructed. So there is no requirement that it be placed in a closed flow loop, as is the case with the mass-flow pump. Mass-flow drivers are perfectly happy as components in either closed or dead-ended flow loops. The DC flow component cannot be over-constrained. You can even connect mass-flow drivers at both ends of a duct, for example, to achieve a desired

pressurization within the duct, provided the physics is reasonable.

The way the mass-flow driver works is by defining the desired mass flow rate as an independent phasor input variable PhsrRhoUA and implicitly solving for the phasor pressure drop that will bring this about. The phasor pressure drop is actually implemented as two implicit real variables corresponding to the real and imaginary parts. Although these variables are invisible, they give their values to the solution grid and you can monitor them in the first harmonic of output variable FDP.

As with the mass-flow pump, the flow area Aflow does not affect the pressure drop but it does affect the gas velocity. So it should be large enough that the mean Mach number (output MachMean) is substantially below one.

22.9 Flow Restrictor Theory

To be connection-compatible with the gas-domains of the stirling model class requires two things: flow restrictors must maintain a time-ring that includes state variables $P, u, \rho, \rho e$ for export to flow connectors. And they must import mass flow rate ρuA and upwind mass-specific enthalpy h from flow connectors for use in their own internal solution. ρuA must be taken as the boundary mass flow rate, at all times, and H as incoming energy, only when flow is into the restrictor domain. Other state variables may be included for convenience.

The solution grid for flow restrictors is a single time ring containing state variables $A, \rho, \rho uA, \rho e, u, T, P$. These are in the same order and have the same meanings as in the stirling-class gas domains, except there are a few omissions for state variables no longer required. Generally, the variables in the grid are understood as downwind values, except for ρe and P which are separated into negative- and positive-boundary values to allow for an entropy-generating pressure drop. Aside from that, the variables are designed to maintain mass and energy continuity across the flow restriction. Neither space nor time differencing is required anywhere. Exact meaning and method of calculation for all variables are explained in detail below.

A

Flow area A is computed explicitly from geometrical inputs appropriate to the specific type of flow restrictor. The reason for including it as a state variable is so time-dependent flow areas may be specified, as is convenient in the implementation of the time-dependent valve component.

ρ

Mass density ρ is computed implicitly from the zero-volume mass-continuity equation

$$(\rho uA)_+ - (\rho uA)_- = 0 \tag{22.21}$$

where $(\rho uA)_+$ and $(\rho uA)_-$ are the values imported from the positive and negative flow connectors. This forces mass flow rate in the adjoining flow connectors to be equal. Implicit functions used for solving ρe also help to determine ρ. The usage in these other implicit functions is consistent with ρ being the downwind value.

ρuA

Mass flow rate ρuA is computed explicitly as the average of the values in the adjoining flow connectors:

$$\rho uA = \frac{1}{2}\left[(\rho uA)_+ + (\rho uA)_-\right] \tag{22.22}$$

This fulfills the requirement that the flow restrictor use imported (ρuA)'s in its interior solution.

ρe

Downwind energy density ρe is computed implicitly by solving the energy continuity equation written in the form

$$\frac{\rho e + P}{\rho} = h_i \tag{22.23}$$

where ρe and P on the left are downwind state variables while h_i on the right is the mass-specific enthalpy evaluated using variables imported from the component across the flow connection in the upstream direction. For an ideal gas h reduces to c_pT in the zero-velocity limit where kinetic energy may be neglected.

For numerical reasons it is convenient to separate ρe into two variables ρe_- and ρe_+, thought of as lying at the negative and positive boundaries of the flow restrictor. The downwind value is ρe_+ when flow is positive and ρe_- when flow is negative. ρe_- is exported to the negative flow connector and ρe_+ to the positive flow connector. The reason this makes sense is because the implicit function determining either can be formulated without a discontinuity at flow reversal, as would otherwise be the case. That is, the equation for determining ρe_- is always that local mass-specific enthalpy $\frac{\rho e_- + P_-}{\rho}$ equals the value imported across the positive flow boundary, which is valid when flow is in the negative direction and does no harm otherwise. A similar but opposite equation determines ρe_+. The meaning of P_- and P_+ is similar to the meaning of ρe_- and ρe_+, and is explained further below.

It might be tempting to use these continuity conditions instead to calculate ρe_- and ρe_+ explicitly. This will not work because of a circular reference when accessing P, which itself depends on ρe.

u

Velocity u is computed explicitly in terms of other state variables as

$$u = \frac{(\rho u A)}{\rho A} \tag{22.24}$$

The parenthesis in the numerator indicate that $(\rho u A)$ is a single state variable. Since ρ is understood to be a downwind value so u is a downwind value.

T

Temperature T is computed explicitly in terms of other state variables from the equation of state function

$$T = T(\rho, \rho e, u) \tag{22.25}$$

The value of ρe is taken as ρe_- for negative directed flow and ρe_+ for positive directed flow. In other words, the downwind value. Since ρ and u are also downwind values the value of T is a downwind value.

P

Pressure P is a bit tricky. The first thought that comes to mind is to calculate it explicitly in terms of other state variables according to the equation of state. But then how would pressure drop be imposed on the solution?

The answer is to break down pressure into two components P_- and P_+, similar to the treatment for ρe. P_- is exported to the negative flow connector and P_+ to the positive flow connector. The idea is to calculate them explicitly, in terms of the thermodynamic pressure, calculated from the equation of state $P(\rho, T)$, plus or minus the pressure drop. For the case of P_- the equation of state is valid for the negative flow direction because of the downwind interpretation already given to ρ and T. For the positive flow direction $P(\rho, T)$ may be offset by the required pressure drop to produce the correct pressure. If this is also done similarly but oppositely for P_+ then the components across the flow connectors at either end see the correct pressure drop at all times. This can be written mathematically as:

$$P_- = \begin{cases} P(\rho, T) & \text{if } \rho u A \leq 0 \\ P(\rho, T) - \Delta P & \text{otherwise} \end{cases} \tag{22.26}$$

and

$$P_+ = \begin{cases} P(\rho, T) & \text{if } \rho u A > 0 \\ P(\rho, T) + \Delta P & \text{otherwise} \end{cases} \tag{22.27}$$

Note that $P_+ - P_-$ is always the pressure drop ΔP.

22.10 Adiabatic Compressor

TCompressor

This component descends from the above mass-flow pump (section 22.7). Like the mass-flow pump it provides a mass flow rate, usually in a closed flow-loop, with the pressure rise (or drop) across the component established by the flow characteristics of the overall flow loop. You can see what this pressure rise is by inspecting output variable FDP. As with the mass-flow pump you can uses Sage's optimizer to solve for the resistance of the flow loop to adjust FDP to a predetermined value.

Unlike the mass-flow pump the adiabatic compressor adds the compressor PV power input to the enthalpy flow stream. The ancestor mass-flow pump implemented an isenthalpic flow stream as if compressor PV power input was always balanced internally by an equal but opposite heat flow out of the pump. The present compressor requires an aftercooler or other type of heat exchanger in the flow loop to remove the heat of compression. The required PV power input is a virtual part of the model and is indicated only as an output variable (FWc). There is no actual PV power connection required to the compressor from an external Sage component.

As Expander It is also possible to run the compressor backwards (e.g. reverse the pressure change for the same mass flow rate) to model an adiabatic expander. In this case there is enthalpy removed from the flow stream by the amount of the expander PV power. In physical terms there must be something in the flow loop to boost the pressure upstream of the expander. This might be another adiabatic-compressor component acting as a true compressor upstream of the expander.

Two in Series In other words, two adiabatic compressor components may be arranged in series within a flow loop with the upstream component serving as the compressor and the downstream component serving as the expander. However, since the time-mean flow of both components is specified as input they must both be exactly equal unless there is someplace for the excess mean flow to go. Into a bypass flow path for example. Otherwise the solution will not converge.

Pressure Reference As with all Sage models, one pressure source is required to establish the mean pressure. If a model contains one compressor then that one pressure source suffices so long as the other components in the flow loop determine the pressure change across the compressor. If a model contains two compressors forming a closed flow loop then the pressure-drop across one of them is indeterminate unless there are two pressure sources, attached to the two flow segments between the two compressors. And so forth. See example file *JetEngine* in the Apps\SCFusion\Samples\JetEngines sub-directory under the installation directory.

In addition to the variables inherited from the mass-flow-pump, the adiabatic compressor components adds the new variables:

Efficiency : (real, dimensionless) Adiabatic efficiency η, an input. For a compressor the PV power input ($-W_c$) will be greater than the ideal isentropic adiabatic power input by the factor $1/\eta$. For an expander the PV power output (W_c) will be less than the ideal adiabatic power output by the factor η.

FWc : (Fourier series, W) PV power output W_c. The sign reflects the Sage convention that positive numbers indicate energy flows out of the gas. So a negative value indicates work flow into a compressor. A positive value indicates work flow out of an expander. Usually the mean value (time-mean power) is the only part of concern, but in some applications the time-varying components may also be of interest.

FHneg, FHpos : (Fourier series, W) Stagnation enthalpy flows $uA(\rho e + P)$ at the component neg and pos boundaries. These two replace output FH inherited from the mass-flow pump.

The output AEfric inherited from the flow restrictor components remains valid except instead of measuring the available energy loss due to flow resistance it measures the available energy loss (T_0 x entropy generation) due to compressor inefficiency, from whatever cause.

The formulation for adding the PV power input to the enthalpy flow stream is completely general and remains valid no matter what the mass-flow-rate looks like. The enthalpy flow increment is evaluated each node of the time grid, implicitly, based on the increase of entropy across the compressor consistent with the input efficiency η. Entropy calculations use a general formulation that does not assume ideal gas properties.

A few items documented previously under the mass-flow pump component bear repeating here:

Because the set-point mass flow rate is a Fourier series input, it is possible to produce pulsatile flows — including sinusoidal terms and higher harmonics — as well as steady flows with the adiabatic compressor component. The subfields of the mass flow rate can be optimized or referenced in constraints, as discussed under the topic of Fourier-series variables in chapter 8. If you specify a nonzero DC part of the mass flow rate, the mass-flow pump should be located within a closed loop capable of supporting DC flow, otherwise the mean pressure drop will be indeterminate and the solution will not converge — if DC flow is impossible, then there is no amount of pressure drop that will force the required value.

The flow area Aflow is another input. It is present in order to define flow velocity in the pump and normally its value does not matter much. However, it should be large enough that the mean Mach number (output MachMean) is substantially below one.

When starting from initial values or a re-initialized solution, an adiabatic compressor in your model can slow down convergence, depending on the magnitude of the flow specified. If this is a problem, try starting with a small or zero DC component of mass flow rate initially and increase it gradually over the course of several solutions. Also be sure to check that the initial temperature

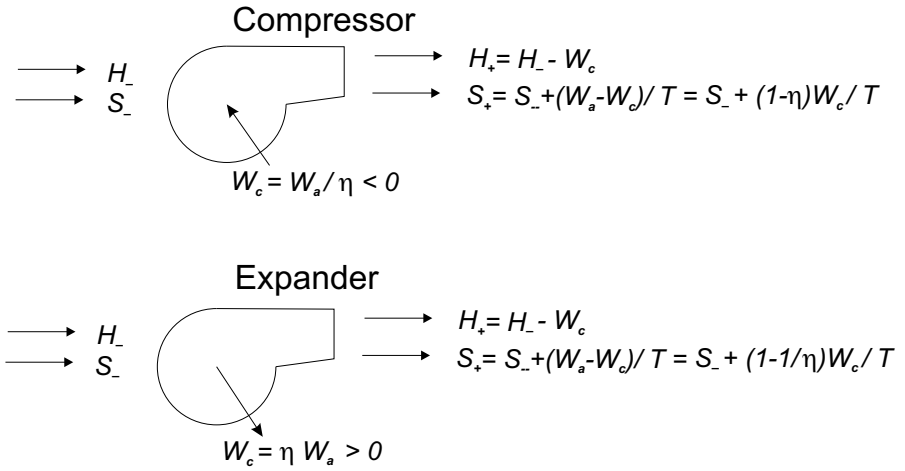


Figure 22.1: Enthalpy and entropy flows in compressor and expander modes as determined by the sign of PV power output W_c. For positive flow enthalpy flow always changes by $-W_c$ according to a first-law energy balance; entropy flow changes by the amount dQ/T where dQ is the difference between ideal and actual PV powers $W_a - W_c$. Flow is isentropic ($S_+ = S_-$) when efficiency $\eta = 1$.

value (Tinit) is close to the initial temperatures of the components to which it is connected.

22.10.1 Theory

Figure 22.1 illustrates the enthalpy flow and entropy flow relationships that define the adiabatic compressor. The flow restrictor theory outlined in chapter 8 remains the same except for two changes:

1. The governing equations determining downwind energy densities ρe_+ and ρe_- are revised to include the compressor PV work input to the enthalpy flow stream.

2. A new state variable is added to the solution grid, corresponding to the change of mass-specific enthalpy Δh across the component due to PV power output. This variable is solved implicitly according to the entropy flow change illustrated in figure 22.1

Energy densities ρe_+ and ρe_- at the negative and positive component boundaries are computed implicitly according to an energy balance equation involving incoming and outgoing enthalpy flows and compressor PV work input. For positive mass flow rate this energy balance equation takes the form

$$H_+ - W_c - H_- = 0 \tag{22.28}$$

where the H's are the enthalpy flows through the positive and negative boundaries and W_c is the PV power flow (positive for outgoing). Energy conservation requires that the PV power output equals the product of mass flow rate and mass-specific enthalpy change, or

$$W_c = -(\rho u A)\Delta h \tag{22.29}$$

Enthalpy flows may be represented in terms of state variables as

$$H = \frac{\rho u A}{\rho}(\rho e + P) \tag{22.30}$$

In the case of ρe_+, the enthalpy flow at the positive boundary H_+ is taken from local solution variables (including ρe_+ itself) and the enthalpy flow at the negative boundary H_- is imported from the component across from the flow connector at the negative boundary. The case for ρe_- is similar, except the internal and external enthalpy flows come from opposite boundaries. To avoid the energy balance equation becoming indeterminate at zero mass flow rate, the mass flow rate $\rho u A$ is divided out leaving the energy balance equation used for determining ρe_+ in the form

$$\frac{\rho e_+ + P_+}{\rho} + \Delta h - \left(\frac{\rho e + P}{\rho}\right)_- = 0 \tag{22.31}$$

The solution variable Δh ($c_p \Delta T$ for an ideal gas) is determined implicitly by the mass-specific entropy change Δs produced by compressor inefficiency.

$$\Delta s = \begin{cases} (1 - 1/\eta)\Delta h/T & \text{if } (\rho u A)\Delta h < 0 \text{ (expander)} \\ (1 - \eta)\Delta h/T & \text{otherwise (compressor)} \end{cases} \tag{22.32}$$

The governing equation for solving dh takes in the form

$$s_+ - s_- - \Delta s = 0 \tag{22.33}$$

The values of mass-specific entropies s_+ and s_- are calculated from solution variables ρ and T at the component boundaries. Downwind values of ρ and T are already part of the solution. Upwind values are imported from the components across from the flow connectors at each boundary. In the case of an ideal gas and ideal efficiency, entropy continuity leads to the well known adiabatic compressor power equation

$$W_c = c_p(\rho u A)T_{in}\left(1 - \left(\frac{P_{out}}{P_{in}}\right)^{\frac{\gamma-1}{\gamma}}\right) \tag{22.34}$$

where γ is the ratio of specific heats c_p/c_v.

22.11 Volumetric Flow Compressor

TVdotCompressor

This component models the time-average flow through a positive displacement compressor, defined by a fixed volumetric displacement produced by pistons,

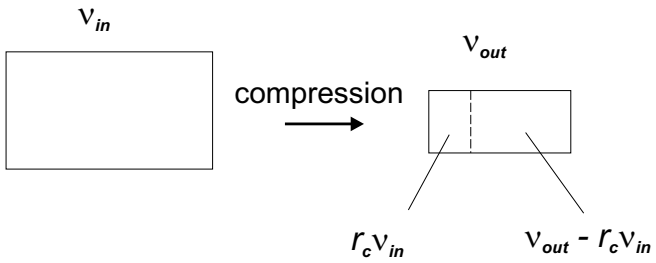


Figure 22.2: For a positive displacement compressor, the volume of a unit mass of fluid (specific volume) at the compressor inlet conditions ν_{in} and after compression ν_{out}. The fraction $r_c\nu_{in}$ (clearance volume) remains in the compressor reducing the throughput to $\nu_{out} - r_c\nu_{in}$.

rotating screws or cylinders, interleaved spiral scrolls or some such. A compressor of the type that might be connected to the rotary valve of a GM cryocooler. Except the flow at the inlet and outlet are steady, not pulsating. The time-varying compressor mechanical and flow details are not modeled, nor is there any actual gas volume within this component. So if you need to smooth out pressure spikes from time dependent valves and such, you must include buffer volumes separately.

The component descends from the previous adiabatic compressor component. The main difference compared to an adiabatic compressor component is that it specifies inlet volumetric flow rate $\dot{V}_{in}$ and clearance volume ratio r_c as inputs, rather than mass flow rate, thereby relating more directly to the specifications of an actual compressor and automatically adjusting mass flow rate as a function of upstream and downstream pressures, like a real compressor.

Vdot : (real, m^3/s) Steady volumetric flow rate $\dot{V}_{in}$ at compressor inlet. If $\dot{V}_{in}$ is positive the inlet is presumed to be the negative end of the component and the opposite if $\dot{V}_{in}$ is negative.

Rclearance : (real, dimensionless) Compressor clearance volume ratio r_c, defined as the gas volume at the end of the compression process relative to the volume at the beginning. See discussion below.

Figure 22.2 illustrates a unit mass of fluid at compressor inlet conditions (left) and later after compression (right). Initial specific volume ν_{in} at the inlet compresses down to volume ν_{out} at the exit. Of that ν_{out} an amount $r_c\nu_{in}$ remains in the compressor, where r_c is defined as the compressor relative clearance volume (Rclearance above). The remaining amount $\nu_{out} - r_c\nu_{in}$ discharges from the compressor exit.

Evidently, compared to an ideal compressor with zero clearance volume the effective volumetric flow rate of an actual compressor is reduced by the factor

$$\frac{\nu_{out} - r_c\nu_{in}}{\nu_{out}} = 1 - r_c\frac{\nu_{in}}{\nu_{out}}$$

Except check-valves and this model component prevent negative flow when this factor drops below zero. The reduction factor applies throughout the compressor so the ideal inlet volumetric flow rate $\dot{V}_{in}$ is reduced to the effective value

$$\left(1 - r_c \frac{\nu_{in}}{\nu_{out}}\right) \dot{V}_{in}$$

The net mass flow rate is just the net volumetric flow rate divided by the specific volume ($\dot{m} = \rho\dot{V} = \dot{V}/\nu$)

$$\dot{m}_{net} = \left(1 - r_c \frac{\nu_{in}}{\nu_{out}}\right) \dot{V}_{in}/\nu_{in} = (1/\nu_{in} - r_c/\nu_{out})\dot{V}_{in}$$

Net mass flow rate is a conserved quantity, the same at both ends. As $r_c \to 0$ the mass flow rate approaches that of an ideal compressor with zero clearance volume. As $r_c \to \nu_{out}/\nu_{in}$ the mass flow rate approaches zero.

In the case of compressing an ideal gas the adiabatic pressure ratio determines the specific volume ratio according to

$$\frac{\nu_{out}}{\nu_{in}} = \left(\frac{P_{in}}{P_{out}}\right)^{1/\gamma}$$

where $\gamma = c_p/c_v$ is the ratio of specific heats. So the relative clearance value must be smaller than the right-hand side for there to be any net flow.

$$r_c < \left(\frac{P_{in}}{P_{out}}\right)^{1/\gamma}$$

Another way to look at it (solving for P_{out}/P_{in}), the maximum pressure ratio is

$$\frac{P_{out}}{P_{in}} < r_c^{-\gamma}$$

22.12 Pressure-Regulated Compressor

TRegCompressor

This component descends from the previous adiabatic compressor component except that the roles of pressure rise FDP and mass flow rate FRhoUA are reversed. It specifies pressure rise as an input and produces whatever mass flow rate is required to achieve that pressure rise.

A pressure-regulated compressor is handy for those situations where the pressures at the intake and discharge of the compressor are known. One of these pressures must be established by a pressure source (pressure bottle icon). To do that just connect an appropriate gas domain near the inlet or discharge to the pressure source. Then the pressure rise FDP specified for the compressor determines the other pressure according to the convention that FDP is the difference ($P_+ - P_-$), where P_+ is the pressure attached to the positive end of the compressor (right-pointing arrow) and P_- is the pressure attached to the negative end (left-pointing arrow).

For this component to work properly it must be located in a flow loop for which mass flow rate is free to vary and that can produce an equal but opposite pressure drop for some reasonable mass flow rate.

22.13 Flow Separator

TGtSepr

This component separates a supersaturated incoming flow stream into distinct vapor-phase and liquid-phase streams. It is designed for positive flow only, with the supersaturated flow stream entering through the negative (left) boundary and the vapor and liquid streams exiting through two positive (right) boundaries. The upper right flow connector is reserved for the vapor stream and the lower right boundary is reserved for the liquid stream. The working gas type must be TBSpline3Gas (section 15.1) for this component to work properly. Otherwise there will always be zero flow directed to the liquid stream. BSpline3Gases are available in the default gas property file LowTGases.dta.

A flow separator is useful for modeling J-T coolers operating as gas liquefiers, where some of the flow stream liquefies after expansion and is no longer available to return through the recuperative counterflow heat exchanger. A flow separator may be inserted into the Sage model just after the J-T expansion orifice. Since Sage can only model continuous closed flow loops that do not abruptly end or begin, the separated vapor and liquid streams must ultimately join together again somewhere downstream of the flow separator. For example, the liquid stream might pass through a vaporizing heat exchanger then merge with the vapor stream at the warm end of the counterflow heat exchanger and re-enter the compressor intake.

The supersaturated stream condenses inside a flow separator and something must regulate the flow to the vapor and liquid exit streams consistent with the vapor mass fraction of the condensate. What does this is a pressure drop (or rise) ΔP introduced in the outgoing liquid stream, which regulates the flow leaving through the liquid line. This corresponds to the physical reality of a regulated pump of some sort. The flow through this pump is assumed to be incompressible, which means that the fluid density and internal energy do not change but there is an external pumping power input or output required equal to the pressure change ΔP multiplied by the liquid-line volumetric flow rate $((\rho UA)_{b+}/\rho_{b+})$. This pumping power appears as an output variable FWc.

A flow separator is designed to work with any incoming fluid state, not just supersaturated. In the event the incoming fluid is a pure vapor (quality $X = 1$) the flow just passes through unaltered from the incoming flow line (left) to the outgoing vapor line (upper right), with zero flow out the outgoing liquid line (lower left). Actually the liquid-line flow is never allowed to be exactly zero to prevent solution convergence problems in downstream components. In the event the incoming fluid is a pure liquid (unlikely) the flow passes through unaltered to the outgoing liquid line.

A flow separator will work with flow in the reverse direction although it is not clear what physical reality such operation corresponds to.

The input and output variables for a flow separator are:

Tinit : (real, K) Initial temperature. Used only for initializing or re-initializing solution variables ρ and ρe. As such, its value does not affect the final converged solution, but it may affect whether the solution converges at all. It should be set to a value consistent with the initial temperatures of adjacent components.

FWc : (FourierSeries, W) Pumping power done on the liquid flow stream by the control pressure drop ΔP. The sign reflects the Sage convention that positive numbers indicate energy flows out of the fluid. So a positive value indicates expansion work leaving the flow separator. A negative value indicates compression work entering the separator. No problem if small but beware the Sage optimizer relying on such things to boost efficiency.

FT : (Fourier series, K) Gas temperature T.

FP : (Fourier series, Pa) Gas pressure P.

FDP : (Fourier series, Pa) Liquid-line control pressure rise (positive) or drop (negative).

FH : (Fourier series, W) Stagnation enthalpy flow $uA(\rho e + P)$.

FRhoUA : (Fourier series, kg/s) Mass flow rate ρuA.

FX : (Fourier series, dimensionless) Quality (vapor mass fraction) of internal state.

22.13.1 Theory

Flow separator analysis starts out similar to that of the flow reverser component (section 18.9) with the implementation of internal state variables ρ (mass density) and ρe (energy density) based on conservation of mass and energy for a zero-volume component. Separating the presumed supersaturated incoming fluid stream in to liquid and vapor parts is something new and requires a new state variable X (vapor mass fraction or *quality*) and some help from the gas equation of state as to the values of mass densities ρ_v and ρ_l and internal energies e_v and e_l appropriate for saturated vapor and liquid phases at the prevailing temperature and pressure. The diagram in figure 22.3 illustrates the key state variables within a flow separator.

ρ

Mass density ρ is computed implicitly from the zero-volume mass-continuity equation

$$(\rho uA)_{a+} + (\rho uA)_{b+} - (\rho uA)_{-} = 0 \tag{22.35}$$

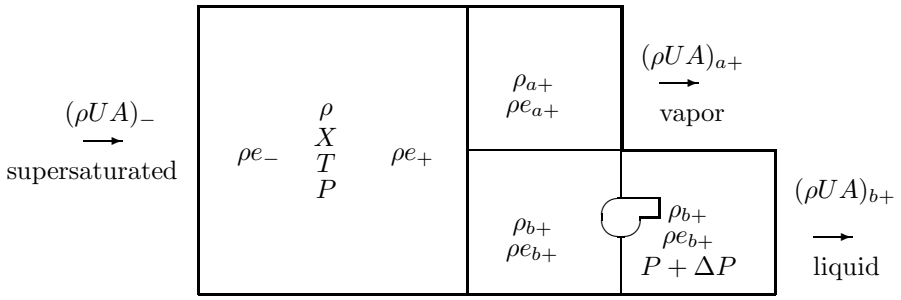


Figure 22.3: State variables for flow separator

where the ρuA's are the mass flow rates imported from flow connectors at the three flow boundaries. Subscript $a+$ denotes the positive vapor-phase exit (upper right), subscript $b+$ denotes the positive liquid-phase exit (lower right) and subscript $-$ denotes the negative inlet (left). Implicit functions used for solving ρe also help to determine ρ. The usage in these other implicit functions is consistent with ρ being consistent with the incoming flow state no matter what the flow direction.

ρe

Energy density ρe is computed implicitly from an energy continuity equation. For numerical reasons, it is broken into two distinct parts ρe_+ and ρe_-. The first, used for the presumed case of positive flow, is implicitly solved by the equation

$$\frac{\rho e_+ + P}{\rho} = h_- \tag{22.36}$$

The second, used for the unanticipated but possible case of negative flow, is implicitly solved by the equation

$$\frac{\rho e_- + P}{\rho} = X h_{a+} + (1 - X) h_{b+} \tag{22.37}$$

Separating ρe into two parts like this avoids the difficulty of using a single mixed equation that changes right-hand sides on flow reversal. Pressures P is explicitly calculated as explained below. The h's in the above equations denote the incoming mass-specific enthalpies evaluated using variables imported from the associated flow connectors. The subscripts have the same meaning as above. The quality X used as a weight factor in equation (22.37) is appropriate because the mass flow rates through connectors $a+$ and $b+$ are always held in the proportions of X to $1 - X$.

Mass and energy densities ρ and ρe_- are exported directly to the negative flow connector while ρ and ρe_+ form the basis of the mass and energy densities exported to the two positive flow connectors, but are not directly used for that purpose. They must be first broken down into vapor and liquid values ρ_{a+}, ρ_{b+}, ρe_{a+}, ρe_{b+}.

P, T

The pressure P appearing in equations (22.36) and 22.37) is calculated explicitly from the equation of state $P(\rho, T)$. This requires temperature which is calculated from the zero-velocity equation of state, as $T(\rho, \rho e_-, 0)$ or $T(\rho, \rho e_+, 0)$. (The 0 in the argument list is velocity. See section 18.9 on flow reversers for a justification of the zero-velocity assumption.) The formulation $T(\rho, \rho e_+, 0)$ is used when flow is positive and the formulation $T(\rho, \rho e_-, 0)$ in the event flow is negative. The implication, based on the way ρe_+ and ρe_- are calculated, is that P and T are consistent with the incoming flow state no matter what the flow direction. Pressures is required in the solution grid so that the flow connectors can properly enforce Bernoulli's law across the connections. P is directly exported to the negative flow connector and the positive vapor-only flow connector $a+$, but the pressure exported to the liquid-only connector $b+$ is $P + \Delta P$, where the ΔP is used to control the mass flow rate ratios for the two streams, as noted above.

X (Quality)

The vapor mass fraction X is defined in terms of vapor density ρ_v, liquid density ρ_l and bulk density ρ as (*see* section 15.1.5)

$$X \equiv \frac{1/\rho - 1/\rho_l}{1/\rho_v - 1/\rho_l} \tag{22.38}$$

In the flow separator model ρ is understood as that of the density state variable as determined by equations (22.35), (22.36), (22.37), and ρ_v and ρ_l are taken to be dew-point and bubble point values ρ_d and ρ_b, respectively, which come directly from the equation state as a functions of T. Evidently one of the things a flow separator has to do maintain mass balance is pass X mass units of vapor at density ρ_v to the vapor flow connector $a+$ and $1-X$ units of liquid at density ρ_l to the liquid flow connector $b+$. To avoid problems when ρ is not strictly between ρ_v and ρ_l flow separators actually calculate quality according to the conditional formula

$$X = \begin{cases} 1 & \text{if } \rho \le \rho_v \\ \text{from equation (15.20)} & \text{if } \rho_v < \rho < \rho_l \\ 0 & \text{if } \rho_l \le \rho \end{cases} \tag{22.39}$$

But even this is not always valid because not all equation-of-state types available in Sage models define ρ_v and ρ_l and those that do, do so only below the critical

temperature. In the event ρ_v and ρ_l are not defined, flow separators assume $X = 1$ (all vapor), and all incoming flow passes directly from the negative connector to the vapor-phase flow connector $a+$.

ΔP

Flow separators need ensure that the ratio of mass flow rates through the vapor and liquid connectors $a+$ and $b+$ are consistent with the vapor mass fraction X. Within the solution scheme of Sage the only way to control mass flow rate is via pressure, as is the case in an actual physical system. So flow separators implement pressure drop ΔP as state variable solved implicitly by the equation:

$$X(\rho u A)_{b+} = (1 - X)(\rho u A)_{a+} \tag{22.40}$$

Flow separators add this ΔP to the pressure exported to the liquid flow connector $b+$ but do not modify exported density ρ_{b+} and energy density ρe_{b+}. This is a reasonable approximation for the liquid line because of the nearly incompressible nature of the liquid state. The assumption that fluid remains in the liquid state after a pressure change is questionable in the case the pressure change reduces the pressure. Why would the liquid not then vaporize? The answer is that it will in the downstream component, as it would in reality. The physical interpretation of what happens in the flow separator is that the liquid remains in a superheated state. The degree of the pressure change required to distribute the liquid flow and its effect on the downstream state depends on the total system model.

Auxiliary Flow-Connector Variables

Variables ρ_{a+}, ρe_{a+} ρ_{b+} and ρe_{b+} appearing in figure (22.3) are the values of mass density and internal energy passed to the vapor and liquid flow connectors. They are not solved as part of the solution grid but rather just calculated when required as functions of the state variables and the gas equation of state, as follows:

$$\rho_{a+} = \begin{cases} \rho_v & \text{if } 0 < X < 1 \\ \rho & \text{otherwise} \end{cases} \tag{22.41}$$

$$\rho e_{a+} = \begin{cases} \rho_v e_v & \text{if } 0 < X < 1 \\ \rho e_+ & \text{otherwise} \end{cases} \tag{22.42}$$

$$\rho_{b+} = \begin{cases} \rho_l & \text{if } 0 < X < 1 \\ \rho & \text{otherwise} \end{cases} \tag{22.43}$$

$$\rho e_{b+} = \begin{cases} \rho_l e_l & \text{if } 0 < X < 1 \\ \rho e_+ & \text{otherwise} \end{cases} \tag{22.44}$$

The equation of state provides the equilibrium vapor and liquid mass densities and mass-specific energies ρ_v, ρ_l, e_v and e_l as functions of T. Actually, the equation of state supplies mass-specific *internal* energies ε_v and ε_l, but with the

zero-velocity assumption these are equivalent to the mass-specific total energies (internal plus kinetic) e_v and e_l.

The abrupt transitions suggested by the above equations at $X = 0$ and $X = 1$ are bad for solution convergence so Sage replaces the abrupt transitions with smooth transitions over the ranges $0 < X < \epsilon$ and $1 - \epsilon < X < 1$, where ϵ is a small number ($\epsilon = 0.1$ in the current implementation). Sage does the smoothing in such a way that the densities and specific energies ρ_v, ρ_l, e_v and e_l are always consistent with the vapor quality X. The smoothing does not affect the mass flow distribution into the vapor and liquid lines.

22.14 Evaporator-Condenser

TGtEvap

This component adds or removes heat from an incoming two-phase flow stream to completely evaporate or condense it without changing its temperature or pressure. Heat is added or removed through a built in heat-flow connection and the temperature of the connection relative to the incoming fluid temperature determines whether the fluid evaporates or condenses. If the connector temperature is higher than the fluid temperature by the amount ΔT_f (input DTfull) then it fully evaporates. If it is lower by the amount ΔT_f then it fully condenses. If somewhere between then the there is a smooth transition between evaporation and condensation with no heat transfer when the connector temperature equals the fluid temperature. The incoming flow stream must be below the critical temperature and pressure for evaporation or condensation to occur. Otherwise this component just passes the fluid without change.

In a physical sense, this component models the high heat transfer rates of boiling or condensing two-phase flow without bothering about heat exchanger details. The working gas type must be TBSpline3Gas (section 15.1.2) for the component to work properly. Otherwise the fluid will pass through without change. BSpline3Gases are available in the default gas property file LowT-Gases.dta. The component works for flow in either direction.

The need for this component arose from modeling a J-T cooler where the fluid exits the expansion orifice as a two-phase mist and the liquid component subsequently contacts a downstream warm surface that is a poor heat exchanger but suffices to evaporate the fluid through boiling heat transfer. In such a case, once the fluid has fully evaporated the heat transfer rate diminishes greatly so the vapor temperature stops increasing and remains below the warm surface temperature. This could not be modeled by simply directing the two-phase flow stream into an ordinary heat exchanger component because Sage's single-phase heat transfer formulation does not provide a high enough heat transfer coefficient to boil-off the liquid unless the heat exchanger is a very effective one (large wetted surface, small hydraulic diameter). In that case, once evaporated the fluid continues heating to near the heat-exchanger wall temperature, which is not physically realistic.

The input and output variables for an evaporator/condenser are:

Tinit : (real, K) Initial temperature. Used for initializing or re-initializing solu-

tion variables ρ and ρe and steady wall temperature T_w. As such, its value does not affect the final converged solution, but it may affect whether the solution converges at all. It should be set to a value consistent with the initial temperatures of adjacent components.

DTfull : (real, K) Temperature difference ΔT_f (positive number) at which evaporation or condensation is complete. There is a smooth transition between evaporation and condensation in the range $-\Delta T_f < (T_w - T) < \Delta T_f$

TwStdy : (real, K) Steady wall temperature T_w defined by external heat-flow connection.

QwMean : (real, W) Time average heat transfer rate to fluid from external connection. Positive for evaporation. Negative for condensation.

FT : (Fourier series, K) Gas temperature T.

FP : (Fourier series, Pa) Gas pressure P.

FH : (Fourier series, W) Downstream stagnation enthalpy flow $uA(\rho e + P)$.

FRhoUA : (Fourier series, kg/s) Mass flow rate ρuA.

FX : (Fourier series, dimensionless) Downstream fluid quality (vapor mass fraction). A measure of success in the evaporation or condensation process.

22.14.1 Theory

The evaporator-condenser component is essentially a flow restrictor (chapter 22) with zero velocity and pressure drop and a provision to add or remove heat to or from the fluid in order to affect a phase change.

The solution grid is a single time ring containing state variables ρ, ρuA, ρe, T, P, X. These have the same meanings as in the stirling-class gas domains, except X which is the fluid vapor-fraction. Generally, the variables in the grid are understood as downwind values, except for ρe which is separated into negative- and positive-boundary values. The solution enforces mass and energy continuity across the flow restriction. Neither space nor time differencing is required anywhere. Exact meaning and method of calculation for all variables are explained in detail below.

ρ

Mass density ρ is computed implicitly from the zero-volume mass-continuity equation

$$(\rho uA)_+ - (\rho uA)_- = 0 \tag{22.45}$$

where $(\rho uA)_+$ and $(\rho uA)_-$ are the values imported from the positive and negative flow connectors. This forces mass flow rate in the adjoining flow connectors to be equal. Implicit functions used for solving ρe also help to determine ρ. The usage in these other implicit functions is consistent with ρ being the downwind value.

ρuA

Mass flow rate ρuA is computed explicitly as the average of the values in the adjoining flow connectors:

$$\rho uA = \frac{1}{2}\left[(\rho uA)_+ + (\rho uA)_-\right] \tag{22.46}$$

This fulfills the requirement that the flow restrictor use imported (ρuA)'s in its interior solution.

ρe

Downwind energy density ρe is computed implicitly by solving the energy continuity equation written in the form

$$\frac{\rho e + P}{\rho} = h_i + E_w \tag{22.47}$$

where ρe and P on the left are downwind state variables while h_i and E_w on the right are the upwind mass-specific enthalpy and heat input (J/kg). Downwind mass-specific energy ρe presumes zero kinetic energy (zero velocity). Upwind h_i is evaluated using variables imported from the component across the flow connection in the upstream direction. For an ideal gas h reduces to c_pT in the zero-velocity limit where kinetic energy may be neglected.

For numerical reasons it is convenient to separate ρe into two variables ρe_- and ρe_+, thought of as lying at the negative and positive boundaries of the flow restrictor. The downwind value is ρe_+ when flow is positive and ρe_- when flow is negative. This is similar to what is done for flow restrictors. (*see* section 22)

Sage calculates phase-changing heat input E_w on non-physical principals as a function of the temperature difference $(T_w - T)$ between the external steady heat-flow connection and the fluid. If the fluid is in a two-phase state and $T_w > T$, Sage calculates E_w so as to evaporate the fluid, with full evaporation at $T_w = T + \Delta T_f$. If $T_w < T$, Sage calculates E_w so as to condense the fluid, with full condensation at $T_w = T - \Delta T_f$. If T_w is between $T + \Delta T_f$ and $T - \Delta T_f$ then E_w produces a smooth transition between evaporation and condensation.

Repeating the above in math-speak, if the fluid temperature is above the critical temperature then Sage sets $E_w = 0$. Otherwise Sage formulates E_w in terms the dimensionless temperature difference

$$\Delta\tau = \frac{T_w - T}{\Delta T_f} \tag{22.48}$$

as

$$E_w = \begin{cases} (h_v - h_i)W(\Delta\tau) & \text{if } \Delta\tau > 0 \\ (h_i - h_l)W(\Delta\tau) & \text{if } \Delta\tau < 0 \end{cases} \tag{22.49}$$

On the right there are three mass-specific enthalpy values: the incoming or upwind value h_i and the values at the vapor and liquid boundaries of the two-phase region at the same temperature, h_v and h_l. If the incoming fluid is in a

two-phase state then $h_l < h_i < h_v$. Function W is a smooth transition function that returns -1 for $\Delta\tau \leq -1$, 1 for $\Delta\tau \geq 1$ and a first-derivative continuous transition between the two for $-1 < \Delta\tau < 1$. It is generally defined by

$$W(x) = \begin{cases} 1 & \text{if } x \geq 1 \\ 1 - (x-1)^2 & \text{if } x < 1 \\ (x+1)^2 - 1 & \text{if } x \leq 0 \\ -1 & \text{if } x < -1 \end{cases} \tag{22.50}$$

T

Temperature T is computed explicitly in terms of other state variables from the zero-velocity equation of state

$$T = T(\rho, \rho e, 0) \tag{22.51}$$

The value of ρe is taken as ρe_- for negative directed flow and ρe_+ for positive directed flow. In other words, the downwind value. Since ρ is also a downwind value the value of T is a downwind value.

P

Pressure P is computed explicitly in terms of other state variables from the equation of state function

$$P = T(\rho, T) \tag{22.52}$$

X

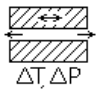
Sage calculates downstream fluid quality X (vapor fraction) from other solution variables exactly like it does for the flow-separator component, as discussed in section 22.13.

Chapter 23

Miscellaneous Parasitics

Parasitic components model some of the loss mechanisms that do not fall into the class of thermal solids or gas components.

23.1 Annulus (Shuttle/Seal/Appendix)



TAnnulus

The annulus component is intended for modeling shuttle heat transfer and clearance-seal or appendix-gap gas flow. Shuttle heat transfer is built-in while annular-gap gas flow is optional by way of toolbox-created child model components. It is actually a descendant of the basic heat-exchanger model component (chapter 20) and inherits several variables from that class. It is born as a child to a composite piston-cylinder component (chapter 24) which supplies a geometrical context in terms of two tubular canisters (piston and cylinder), including their common length, diameter, initial temperature distribution, relative displacement as a phasor variable and thermal properties. An annulus adds the variables:

XNeg : (real, dimensionless) The dimensionless x position of the annulus negative endpoint in the parent cylinder coordinate frame. $x = 0$ is the parent-cylinder negative endpoint and $x = 1$ is the positive endpoint. Default is XNeg $= 0$ for the case where the annulus negative endpoint coincides with the parent cylinder negative endpoint.

XPos : (real, dimensionless) Same as XNeg except for the positive annulus endpoint. Default is XPos $= 1$ for the case where the annulus positive endpoint coincides with the parent cylinder positive endpoint.

Gap : (real, m) Radial clearance gap g between the parent cylinders.

Length : (real, m) Annulus length. Not necessarily the same as the parent cylinder length, reflecting inputs XNeg and XPos.

TsNeg : (real, K) Temperature at annulus negative endpoint.

TsPos : (real, K) Temperature at annulus positive endpoint.

QNeg : (real, W) Shuttle heat flow at annulus negative endpoint.

QPos : (real, W) Shuttle heat flow at annulus positive endpoint.

AEQ : (real, W) Available energy loss to shuttle heat flow.

Within the annulus toolbox are tools for creating an optional foil-matrix gas-domain with one of three wall models: isothermal, conductive surface or floating. The isothermal wall is for modeling thermally-anchored seals (such as piston seals), although you must remember to account for any wall heat flow in your overall energy balance. The conductive surface is for thermally-isolated walls, such as appendix-gap walls. It inherits solid material and cross-sectional area information from the parent canisters and models the solid conduction path corresponding to their combined cross sections over that portion of their length specified by the XNeg and XPos input variables. The floating model is also an option for thermally-isolated walls, except that it does not model axial conduction. It can be useful on occasion to avoid for double-accounting for solid axial heat flows when they are separately considered in the parent canister models. However, floating walls with NCell > 2 often converge to unrealistic temperature distributions due to poor thermal anchoring.

You connect the shuttle-heat-transfer part of an annulus into the grand scheme of your SCFusion model much like you would a simple bar conductor — typically, at both ends to a temperature source (sink) objects via built-in steady-heat-flow boundary connectors. If you create a child gas domain, its flow connectors are connected to other gas domains as is appropriate in your model scheme. Two annulus instances may be ganged together in series to model a seal-appendix combination.

There are variations of this component for parallel and multi-length containers (chapter 27) which support x-distributed heat flow connections to other components in the container. Inside parallel containers NCell and Length are inherited and XNeg, Xpos fixed at 0 and 1. In multi-length containers only NCell is inherited allowing XNeg, XPos to be set independently, thereby establishing Length as a fraction of the parent Length.

23.1.1 Theory

Shuttle heat transfer is a somewhat complicated process whereby, in the presence of a temperature gradient, the moving wall of an annulus alternately picks up from and deposits heat to the stationary wall, with the net effect of transporting heat in the axial direction. This picking up and depositing of heat is governed by the time-varying heat conduction in the transverse direction across the gas-filled annular gap, between and into the two walls. The shuttle heat transfer loss q_s may be presented in the form of an enhancement to molecular gas conduction q_m within the clearance gap, although it is implemented separately from the gas

in Sage. The following formula is due to Rios [53]:

$$\frac{q_s}{q_m} = \frac{1}{2}(x/g)^2 \frac{1+\lambda}{1+\lambda^2} \tag{23.1}$$

where

g	=	radial clearance gap
k	=	molecular gas conductivity
k_s	=	wall solid conductivity
q_m	=	$-k\frac{\partial T}{\partial x}$ molecular conduction heat flux
x	=	relative displacement amplitude
λ	=	$1+(k/g)[(\delta_s/k_s)_1+(\delta_s/k_s)_2]$; the subscripts refer to the inner and outer walls
δ_s	=	$\sqrt{2k_s/(\omega\rho_s c_s)}$; thermal penetration depth
c_s	=	wall solid specific heat
ρ_s	=	wall solid density
ω	=	angular frequency

The above formula applies to the case of *thick* walls, where the wall thickness w is larger than the thermal penetration depth δ_s, for each wall. When this is not the case, shuttle heat transfer is reduced, because a thin wall cannot hold as much heat as a thick wall in the transverse heat-flow part of the shuttle mechanism. In fact a thin wall can hold only about w/δ_s as much heat. So, whenever one of the walls is thin ($w/\delta_s < 1$), Sage scales the above formula by the ratio of the smaller of w/δ_s for the two walls.

23.2 Radiation Transport Path

TQrad

The radiation transport path embodies a simple model of radiant heat transfer between two gray bodies. It is plug-compatible with a simple bar conductor and comes with two built-in steady heat flow connectors at either x end for connection to a point temperature source and sink. It is born as a child to a parent model component which supplies the radiation frontal area. A radiation transport path has variables:

Emmis : (real, dimensionless) Gray-body emissivity ϵ. A number between 0 and 1. $\epsilon = 1$ for an ideal black body. $\epsilon \approx 0$ for a silvered surface.

TsNeg : (real, K) Temperature T_n of negative x surface.

TsPos : (real, K) Temperature T_p of positive x surface.

QNeg : (real, W) Radiation heat flow at negative x surface.

QPos : (real, W) Radiation heat flow at positive x surface.

AEQ : (real, W) Available energy loss to radiation heat flow.

23.2.1 Theory

A radiation transport path models in-vacuum radiation energy transport between two gray body surfaces with same area and emissivity, presuming all the radiation leaving one surface reaches the other and vice-versa. This may be formulated as

$$q_r = \sigma \frac{\epsilon}{2-\epsilon} \left(T_n^4 - T_p^4\right) \tag{23.2}$$

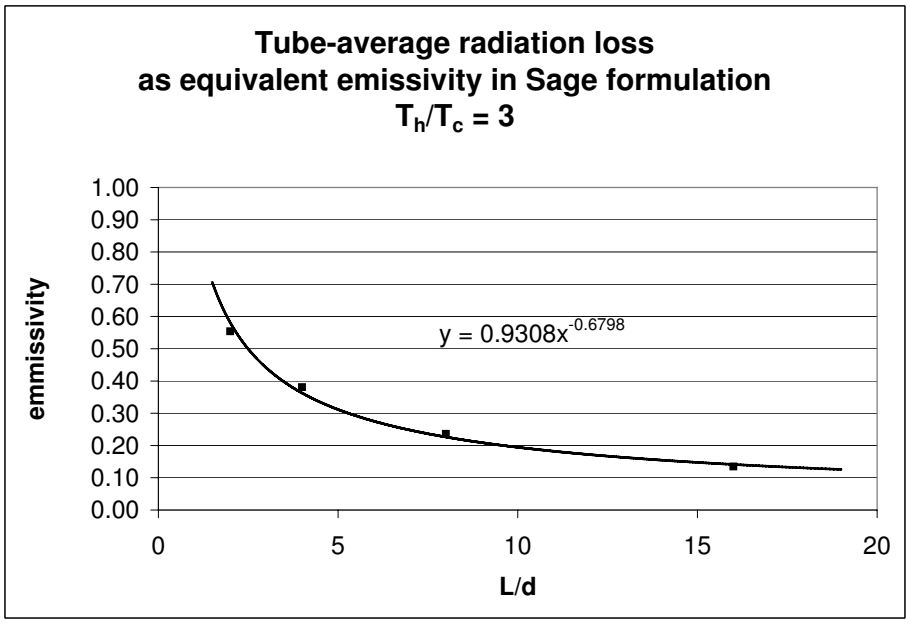
where

q_r	=	radiation heat flux
T_n	=	temperature of negative x surface
T_p	=	temperature of positive x surface
ϵ	=	gray-body emissivity
σ	=	5.669E–8 W/(m^2 K^4); Boltzmann constant

Many radiation loss situations in SCFusion machines can be modeled as radiation down a long thin tube, hot at one end and cold at the other. As a first approximation the radiation loss down the tube can be modeled as the average radiation down a uniform black-walled tube with a linear temperature variation along the wall. This can be reformulated into an equivalent emissivity for the above radiation-loss formulation. Reference [24] works out a simple correlation for equivalent emissivity ϵ (input Emmis) as a function of tube length to diameter ratio L/d, valid for $2 < L/d < 20$ and a hot-to-cold temperature ratio of 3.

$$\epsilon \simeq 0.93 \left(L/d\right)^{-0.68} \tag{23.3}$$

When plotted this equivalent emissivity looks like this:



Chapter 24

Composite Model Components

Composite model components are high level components that work like other model components except that they have built in child models that you cannot destroy. They automatically take care of some of the geometrical relationships among these built-in components that would otherwise prove bothersome to you.

24.1 Piston-Cylinder Composites

Piston-cylinder composites appear in the root-level SCFusion model toolbox. They all represent some form of piston within a cylinder, with capability to model the interaction between the two. Built-in are two nested tubular canister child components serving as the moving piston (shell) and the fixed cylinder (liner). The descendants below also build-in various types of moving-part child components.

You treat these built-in components just as if you had created them yourself. For example, you may put a regenerator matrix and gas domain within the moving shell, to serve as a regenerative displacer (chapter 19). Or you may add area attachments to the moving part (chapter 16) for connection to variable-volume gas domains (chapter 18).

Piston-cylinder composites also come with a toolbox with which you may create your own custom child model components. Available are springs and dampers for attachment to the moving part and a shuttle-heat-transfer annulus for modeling the interaction between the shell and liner, including possible gas flow.

Piston-cylinder composites are born without connectors of any sort. Any and all connectors come from the child components within and must be moved up to the top level if you wish to connect them to other root-level model components.

Variables common to all piston-cylinder composites are:

NCell : (dimensionless) The number of spatial nodes in the computational grids of all underlying model components having such grids. Changing this variable causes these computational grids to re-initialize themselves. Increasing NCell generally increases solution accuracy at the cost of greater solution time.

Length : (real, m) Shell and liner length in x direction.

Dliner : (real, m) Cylinder liner outside diameter, reflecting wall thickness of liner

Dshell : (real, m) Moving shell outside diameter.

Tinit : (cubic spline, W) Axial temperature distribution $T(x)$ where $x = 0$ is the negative endpoint and $x = 1$ is the positive endpoint.

The working gas type (properties) comes from the root model component.

There are variations of piston-cylinder composite components for parallel and multi-length containers (chapter 27) which support x-distributed heat flow connections to other components in the container. Inside parallel containers NCell, Length and Tinit are inherited from the container. In multi-length containers only NCell and Tinit are inherited allowing Length to be set independently.

24.1.1 Free-Piston and Cylinder

TGtRcpCyl

This component adds a built-in reciprocating mass intended as the basis for free-piston or free-displacer modeling. Overstroking or large time-average offsets are possible with the reciprocating mass, depending on how it connects with the larger model. One way to deal with this problem in early stages of design is to connect the reciprocator to an external constrained piston to force it to move in a prescribed manner, removing the connection later after you have implemented the proper combination of area attachments, springs, etc.

24.1.2 Constrained Piston and Cylinder

TGtPisCyl

This component adds a built-in constrained piston (or displacer) intended for first-approximation kinematic modeling or free-piston design work. The piston motion Fourier-series coefficients (mean + harmonic amplitudes and phases) are specified as input. Overstroking is no longer a problem with this option. By proper use of constraints you can set up an optimization problem to solve for the proper combination of area attachments, springs, etc., to make the piston run properly when you switch to free-piston mode.

For example, you could attach a spring to the piston and set up one area attachment to represent a drive rod. You could then optimize the spring stiffness and rod area subject to the constraints that the real and imaginary parts of the required boundary force (F.real and F.imag) are both zero. You could chose to optimize other combinations of *free-piston* variables as well, provided you wind up with two independent degrees of freedom.

24.1.3 Simple-Crank Piston and Cylinder

TGtCrankPisCyl

This component adds a built-in simple-crank kinematic piston intended for modeling a kinematically driven piston using geometrical mechanism inputs. With these inputs you have a handle on optimizing the mechanism or introducing constraints in terms of them.

24.1.4 Time-Ring Free-Piston and Free-Cylinder

TGtRcpFreeCyl

This component is similar to the free-piston and cylinder component except it adds another built-in reciprocating mass representing the moving cylinder or casing of a so-called "free cylinder" machine. The main reason for including the moving cylinder in this component is so it can pass the relative motion between the piston and cylinder to any annulus child components (*see* chapter 23) for purposes of calculating shuttle heat transfer. In free-cylinder machines the cylinder and piston can both have large amplitudes with respect to a fixed reference frame yet be moving almost in parallel so that the relative amplitude between them is small. So it is important to calculate shuttle heat transfer based on the relative piston motion, not absolute motion.

The mass of the cylinder reciprocator should include all the mass rigidly attached to the cylinder. If you want to allocate some of the mass to an external reciprocating mass you can do so provided you connect the two reciprocating masses together with a force connection, which will force them to move together with the same amplitude and phase. Generally the cylinder reciprocator will have area attachments representing volume displacements to other gas spaces in your model and be connected to ground with a spring and damper, representing the suspension and load.

Keep in mind that in a free-cylinder model, both piston and cylinder motions are understood as being relative to a fixed coordinate frame. This is different than the usual mind-set when thinking about stirling-cycle thermodynamics where the displacer amplitude, say, really refers to the amplitude of the displacer relative to the cylinder. When working with a free-cylinder model you must make all such implicit understandings explicit in the model by providing appropriate volume displacements to any affected variable-volume gas spaces via area attachments to the moving cylinder.

There is no corresponding constrained-cylinder variation of this component. If you want to constrain the cylinder motion, or the piston motion for that matter, you can just connect it to an external constrained piston component using a force connection. The two components will then move together with the same motion. During the solution process, the force between them will automatically accommodate to produce whatever force is necessary to drive the reciprocating mass. You can always break the connection later to implement "free" motion.

Chapter 25

Electromagnetic Components

The ultimate purpose of electromagnetic components is to model linear motors and alternators from first principles as part of an overall SCFusion model. Available components range from simple electrical components like resistors or capacitors to components representing moving magnets or coils in interacting with time-varying electro-magnetic fields. Linear motors constructed from these components supersede the elementary linear-motor components of section 29.3.12 offered in earlier Sage version.

Electromagnetic components are connected to each other with current and magnetic flux connections representing time-ring (periodic time grid) values at the terminals or end poles of the component. They also sometimes have mechanical force connections for attachment to moving parts (chapter 16). Current and magnetic flux connections were introduce for version 9 in 2012

25.1 Connection Child Components

Electromagnetic components have two ends corresponding to the two leads on a resistor or capacitor or the two poles of a permanent bar magnet. Some come with built-in electrical current or magnetic-flux connections at the two ends (*see* section 11.4). Others allow you to create any number of such connections at either end using terminal or pole child models, which you drag and drop into your edit form.

Icon	Purpose
	time-ring negative terminal
	time-ring positive terminal
	time-ring negative pole
	time-ring positive pole

When you drop one of these into the edit form, it is born with a connection arrow which you can then move up one level to the parent-model page for connection there to a mating connector in another component. You decide which connectors get connected where from the context of your problem and subsequent documentation.

25.2 Electromagnetic Materials

Certain electromagnetic model components have enumerated type variables (Conductor and Material) that select an electromagnetic material from a list-box of named materials. The materials are either non-magnetic electrical conductors, soft ferromagnetic materials or permanent magnets. Each has its own set of properties made available to the Sage application.

Electrical Conductor Properties There are only two properties:

Density : ($\mathrm{kg/m^3}$) mass density ρ_s

Rs(T) : (Ohm-m) cubic-spline data pairs $(T, \sigma(T))$ of resistivity σ as a function of temperature T

Soft Ferromagnetic Material Properties Include the above electrical properties in addition to:

Mur(T) : (dimensionless) cubic-spline data pairs $(T, \mu_r(T))$ of maximum relative permeability μ_r

Jsat(T) : (T) cubic-spline data pairs $(T, J_s(T))$ of saturation magnetic polarization J_s

Hcb(T) : (A/m) cubic-spline data pairs $(T, H_{cB}(T))$ of induction coercive force H_{cB}

Permanent Magnet Properties Include the above electrical properties but replace the above ferromagnetic properties with these properties particular to permanent magnets:

Kjh : (dimensionless) magnetization curve $J(H)$ shape parameter K_{jh} $(0.5 < K_{jh} < 1)$

Br(T) : (T) cubic-spline data pairs $(T, B_r(T))$ of residual magnetic flux density B_r at $H = 0$

Hcj(T) : (A/m) cubic-spline data pairs $(T, H_{cJ}(T))$ of magnetization coercive force H_{cJ}

25.3 Simple Electrical Components

Simple electrical components can model stand-alone elementary electrical circuits or serve as circuit elements that regulate voltage and current in more complicated electromagnetic models involving the electrical coil components of section 25.4.

25.3.1 Common Variables

Simple electrical components share some common output variables although not all are present, depending on the component:

FVneg : (Fourier series, V) Voltage at left (negative oriented) end V_-.

FVpos : (Fourier series, V) Voltage at right (positive oriented) end V_+.

FV : (Fourier series, V) Uniform voltage V_o.

FDeltaV : (Fourier series, V) Voltage drop $\Delta V = V_+ - V_-$.

FI : (Fourier series, A) Average of the electrical currents at the two ends $(I_- + I_+)/2$.

FWe : (Fourier series, W) Electrical power delivered to the component. The instantaneous electrical power delivered to component is $W = -I\Delta V$. Output FWe is just the Fourier series representation of W as a function of time.

Positives and Negatives In the context of electrical components the terms *negative* and *positive* can refer to two distinct things, the sign of the electrical charge or the physical orientation of the Sage component. So to minimize confusion the terms *left end* and *right end* are used to refer to the negative-oriented and positive-oriented ends of components. This terminology is consistent with the way components are displayed in the edit window.

However in mathematical symbols the left and right ends are still denoted with $_-$ and $_+$ subscripts because there are no conventional symbol for right and left. So for example, V_- refers to voltage at the left end of a component and V_+ to voltage at the right end.

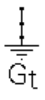
Electrical current sign convention Sage follows the unfortunate convention that electrical current represents a flow of positive charges. Together with the general Sage convention that positive flows are reckoned from the left to the right component ends, this means that a *positive* current flow corresponds to positive charges flowing in the positive direction or negative charges flowing in the negative direction. So in the case of a simple electrical resistor a negative voltage drop ΔV (voltage at right end lower than at left end) produces a positive current flow, which actually corresponds to a flow of electrons in the negative direction.

25.3.2 Connection Block

TGtEblk

This component functions like a conduction path on a printed circuit board that connects together the leads of several electrical components. Within a connection block you can drop in any number of negative or positive terminals depending on which components you are interconnecting. All leads attached to a connection block share a common voltage which is solved as part of the overall model so as to zero the net current flow, summed over all connections. There are no required inputs for a connection block.

25.3.3 Voltage References



TGtVref

This component is similar to the above connection block except you explicitly set the voltage. It functions something like a ground connection. In any electrical circuit the voltage is indeterminate unless you specify the absolute voltage (usually zero) at least one point. Otherwise adding an arbitrary constant offset V_o to all the voltages of the circuit would produce an equivalent circuit (same voltage drops and currents). This component allows you to specify reference voltages as an arbitrary periodic time grid if you should want to do that. Generally you will connect a voltage reference to one point of an electrical circuit using a positive or negative facing terminal. You may *ground* a circuit at more than one point using voltage references with different voltages to simulate ground current loops.

Additional inputs compared to the variables listed above are:

FV : (Fourier series, V) Voltage offset V_o.

A voltage reference is the only electrical component where the net current flow into or out of the component can be non-zero — where the current flow need not be the same at both ends.

25.3.4 Voltage Source

This component establishes the voltage drop ΔV across two terminals of an electrical circuit as an input. The voltages and currents in the rest of the circuit respond accordingly. A sinusoidal voltage source is a good representation of the sinusoidal power grid in that it provides whatever current necessary to produce the required voltage drop across the terminals. With the addition of higher harmonics a voltage source might represent the output of an inverter.

TGtVsrc

Additional inputs compared to the variables listed above are:

FDeltaV : (Fourier series, V) Fixed ΔV overriding above dependent variable.

25.3.5 Current Source

This component establishes the current flow I between two terminals of an electrical circuit as an input. It supplies whatever voltage drop is necessary to produce the required current. The voltages and currents in the rest of the circuit respond accordingly.

TGtIsrc

Additional inputs compared to the variables listed above are:

FI : (Fourier series, A) Fixed current I overriding above dependent variable.

25.3.6 Resistor

This component represents a simple resistor with input

R : (real, Ohms) Electrical resistance R.

TGtEres

It imposes a voltage drop between the negative and positive terminals according to Ohm's law

$$\Delta V = -IR \tag{25.1}$$

Where I is the electrical current. The minus sign is justified because a positive current flow $I > 0$ implies voltage at the left end V_- is higher than voltage at the right end V_+, so $\Delta V = V_+ - V_- < 0$.

25.3.7 Capacitor

This component represents a simple capacitor with input

C : (real, Farads) Electrical capacitance C.

TGtEcap

It is governed by this relationship between current I and voltage drop ΔV:

$$I = -C\frac{d}{dt}(\Delta V) \tag{25.2}$$

The minus sign is justified because a positive current flow $I > 0$ implies voltage increasing at the left end V_- (positive charge accumulating) and decreasing at the right end V_+ (negative charge accumulating), so $d/dt(\Delta V) < 0$.

25.3.8 Inductor

TGtEind

This components represents a simple inductor with input

L : (real, Henries) Electrical inductance L.

It is governed by this relationship between voltage drop ΔV and current I:

$$\Delta V = -L\frac{dI}{dt} \tag{25.3}$$

The minus sign is justified because a voltage higher at the left end than the right end ($\Delta V < 0$) *accelerates* current in the positive direction ($dI/dt > 0$).

25.3.9 Diode

TGtEdiode

This component represents a solid-state diode with inputs

Rfwd : (real, Ohms) limiting forward-bias *dynamic* resistance R_f (slope of voltage vs current curve). Establishes a lower bound on resistance.

Rrev : (real, Ohms) limiting reverse-bias *dynamic* resistance R_r. Establishes an upper bound on resistance.

Isat : (real, Amps) Reverse-bias saturation current I_s. This is a property of the particular diode. You can find typical values in manufacturer datasheets. The default value of 0.1 mA is probably close enough for most purposes. The sign determines the diode polarity. For negative I_s the forward-bias current direction is positive and vice-versa.

A diode imposes a voltage drop between the negative and positive terminals according to Shockley's diode equation, with linear extensions for large negative or positive currents, as explained below. The Shockley equation that covers both polarity directions is

$$I = \begin{cases} -I_s\left(e^{-\frac{\Delta V}{V_t}} - 1\right) & \text{if } I_s < 0 \\ -I_s\left(e^{\frac{\Delta V}{V_t}} - 1\right) & \text{if } I_s > 0 \end{cases} \tag{25.4}$$

Where I is the actual electrical current, I_s is the reverse-bias saturation current, $\Delta V = V_+ - V_-$ is the voltage drop and V_t is the so-called thermal voltage. I_s and

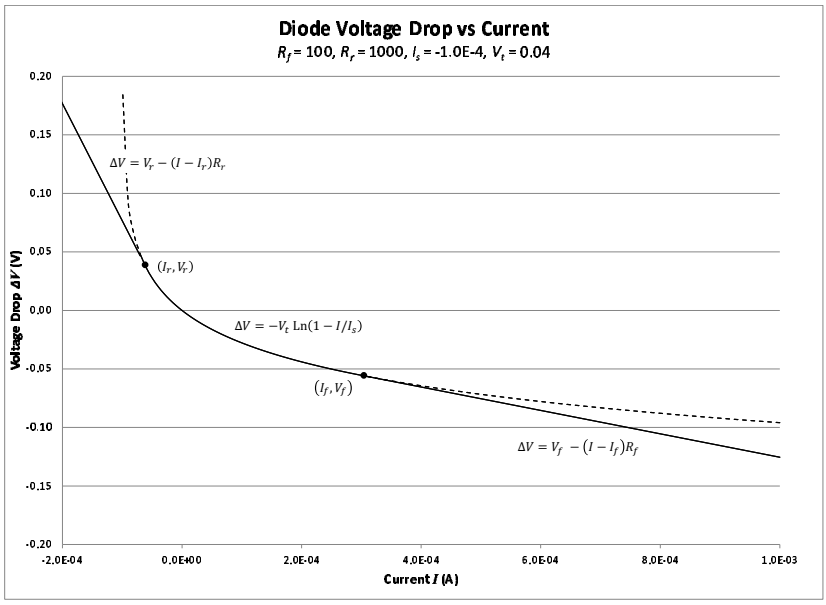


Figure 25.1: Diode voltage drop vs current from equation (25.5), with linear extensions for negative current below the saturation current I_s and high positive current. The limiting reverse-bias resistance R_r unusually low, and forward-bias resistance R_f unusually high for clarity.

V_t are properties of the diode. I_s is the input lsat. V_t is a constant embedded in Sage with the effective value 40 mV.

For the case $I_s < 0$ the current approaches I_s asymptotically from above for increasing positive ΔV (reverse-bias) and quickly increases without limit in the positive direction for negative ΔV. For the case $I_s > 0$ the current approaches I_s asymptotically from below for increasing negative ΔV (reverse-bias) and quickly decreases without limit in the negative direction for positive ΔV.

The Sage formulation requires voltage drop as a function of current, which is the inverse of equation (25.4)

$$\Delta V = \begin{cases} -V_t \ln\left(1 - \frac{I}{I_s}\right) & \text{if } I_s < 0 \\ V_t \ln\left(1 - \frac{I}{I_s}\right) & \text{if } I_s > 0 \end{cases} \tag{25.5}$$

Figure 25.1 plots this equation for the case $I_s < 0$. According to the Shockley equation, current never goes below I_s, the reverse-bias saturation current. But for numerical reasons Sage requires a voltage drop value for any current value. So

for I near I_s and below Sage uses the linear approximation $\Delta V = V_r - (I - I_r)R_r$, continuous and tangent to the Shockley equation at the point (V_r, I_r). The slope $d\Delta V/dI$ from equation (25.5) is

$$\frac{d\Delta V}{dI} = \begin{cases} -\left(\frac{V_t}{I_s - I}\right) & \text{if } I_s < 0 \\ \left(\frac{V_t}{I_s - I}\right) & \text{if } I_s > 0 \end{cases} \tag{25.6}$$

from which it is easy to find I_r by equating $-d\Delta V/dI$ with the slope R_r (input Rrev), and V_r from equation (25.5).

Similarly, it is unrealistic for the slope $d\Delta V/dI$ to approach zero for high positive currents in the forward bias direction, suggesting there is no internal resistance. So that there is always a minimum resistance Sage uses the linear approximation $\Delta V = V_f - (I - I_f)R_f$, continuous and tangent to the Shockley equation at the point (V_f, I_f). I_f and V_f are solved as before based on equating $-d\Delta V/dI$ with the slope R_f (input Rfwd), and V_f from equation (25.5).

25.3.10 Power Probe

TGtEprobe

This components represents a power flow meter. It is functionally equivalent to a short wire with zero resistance. Use it to monitor voltage, current and, more importantly, power flow at any point in an electrical circuit. There are no inputs required.

Other electrical components measure power dissipated within the component. This component overrides the FWe output to show the power *flowing* through the component, evaluated as a function of time from

$$W_e = IV \tag{25.7}$$

25.3.11 Ideal Transformer

TGtTransformer

Use this component to model an off-the-shelf transformer of adequate current rating for your application. For actually *designing* a transformer see the sample model Transformer.stl installed in a subdirectory of the Sage program directory. This ideal transformer assumes zero resistance in the windings and zero reluctance in the iron path. It just steps up voltage and steps down current (or vice-versa) according to the single input:

TurnRatio : (real, dimensionless) Ratio secondary/primary winding turn numbers N_2/N_1.

There are two built-in electrical circuit child components corresponding to the *primary winding* and *secondary winding*, each with the usual electrical-component outputs. Ideal transformers are born with four current connectors, corresponding to the ends of the primary and secondary windings.

In transformers both coils are wound around a common magnetic flux path. If subscript 1 denotes the primary winding and 2 the secondary winding then

Faraday's law (equation (25.27)) gives the voltage induced in each winding in terms of the rate of change of linked magnetic flux as

$$\Delta V_1 = N_1 \frac{d\phi}{dt} \tag{25.8}$$

$$\Delta V_2 = N_2 \frac{d\phi}{dt} \tag{25.9}$$

With zero wire resistance, ΔV_1 and ΔV_2 are the actual voltages at the transformer terminals. Eliminating the common $d\phi/dt$ gives the well known voltage step-up equation for a transformer

$$\frac{\Delta V_2}{\Delta V_1} = \frac{N_2}{N_1} \tag{25.10}$$

Regarding the magnetic flux, it is related to the magnetic potential difference around the flux path according the magnetic circuit relationship

$$\phi = \frac{\mu A}{\ell} \Delta\Psi \tag{25.11}$$

where μ is the magnetic permeability, A is the effective magnetic path cross section area and ℓ is the length. The magnetic potential difference is produced by the combined electrical current flows in the two coils according to Ampere's law (equation (25.25))

$$\Delta\Psi = N_1 I_1 + N_2 I_2 \tag{25.12}$$

From the previous two equations it follows that

$$\phi = \frac{\mu A}{\ell} \left(N_1 I_1 + N_2 I_2 \right) \tag{25.13}$$

For an ideal transformer the assumption is that $\mu A/\ell \to \infty$ while ϕ remains finite (according to (25.8) or (25.9), so that

$$N_1 I_1 + N_2 I_2 = 0 \tag{25.14}$$

or

$$\frac{I_1}{I_2} = -\frac{N_2}{N_1} \tag{25.15}$$

In other words the currents are opposite and stepped down by the same ratio that the voltages are stepped up. In an actual transformer $N_1 I_1$ and $N_2 I_2$ are not quite equal and opposite.

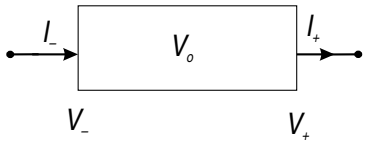
In energy flow terms, the above equations imply that

$$\Delta V_2 I_2 + \Delta V_1 I_1 = 0 \tag{25.16}$$

There is energy flow through an ideal transformer but no net energy dissipated within it.

25.3.12 Solution Method

All simple electrical components share a common computational framework which may be represented abstractly in terms of a discrete component with negative facing and positive facing ends to which voltage is applied and through which current flows, as shown in this diagram:



In some components there are two primary solution variables, voltage offset V_o and voltage drop $\Delta V = V_+ - V_-$ with voltages at the two ends evaluated explicitly as

$$V_- = V_o - \Delta V/2 \tag{25.17}$$

$$V_+ = V_o + \Delta V/2 \tag{25.18}$$

In other components there is only one primary solution variable V_o, presumed uniform throughout the component with no need to define ΔV. Electrical current I is a secondary variable evaluated as the average of the currents flowing through the negative and positive ends.

$$I = \frac{I_- + I_+}{2} \tag{25.19}$$

Generally, the current is forced to be the same at the two ends (no net charge accumulation within) so I is the uniform current flowing through the component. In the case of a capacitor the current includes the so-called displacement current of the time-varying electric field between the plates.

Current determines voltage offset Generally the condition of uniform current ($I_- = I_+$) determines the voltage offset V_o implicitly. The exception being the voltage reference components where the voltage offset is set by inputs and the current can be different at the two ends.

Component physics determines voltage drop Generally the voltage drop ΔV is evaluated explicitly as a function of current according to the physics for the particular component (*see* above for each component class). The exception being the current source component where ΔV is solved implicitly from the current input.

Current based on voltage continuity Connections between terminals provide current I solved implicitly from the condition that the voltage is the same across the connection. When there are multiple terminal connections at one or both ends of a component, the voltages are all the same. Each component

is responsible for enforcing the requirement that the sum of currents flowing through the negative boundary equals the sum of currents flowing through the positive boundary.

Solved by Sage Taken together in the Sage solution framework the implicit variables identified above suffice for Sage to solve for voltages and currents everywhere in a circuit comprising any number of simple electrical components connected together in ways that make physical sense.

25.4 Coil Components

Coil components are electromagnetic components that implement Faraday's law of induction and Ampere's law relating electric current to magnetic field. In a software sense they descend from the above simple electrical components and borrow the concept of magnetic potential and magnetic flux connections from the magnetic components to follow.

Physically, they represent coils of wire consisting of one or more turns wound on a cylindrical form (not necessarily circular). Each turn is insulated from the next and an electrical current flow through the wire from one end to the other.

Magnetic effects occurring within the coil core (inside of the cylinder on which it is wound) are communicated to other magnetic model components (e.g. ferromagnetic materials) via magnetic flux connections at the ends of the coil core cylinder — the coil end boundaries. Generally in a Sage model there will be an external closed-loop magnetic flux path originating at one coil boundary and terminating at the other. Physically this flux path represents the ferromagnetic or air path inside the coil core as well as the external path. The electrical current flowing in the coil winding generates a magnetic potential difference $\Delta\Psi$ between the two ends of the coil core (Ampere's law). A changing magnetic flux ϕ flowing through the coil core as a result of the magnetic flux connections generates a potential difference in the coil winding (Faraday's law). All Coil components share these variables:

Inputs

Dwire : (real, m) Wire diameter d_w not including insulation. If the wire cross section is not circular this should be the effective diameter for a circular wire with the same area. For example, $\sqrt{4/\pi}\,b$ for a square wire of dimension $b \times b$.

Nturns : (real, dimensionless) Number of wire turns in coil N.

Dcentroid : (real, m) Diameter of coil centroid D_c. In the case of a circular coil of rectangular or circular cross section D_c is the average of the coil inner and outer diameters. Generally, defined by $\pi D_c A_c = V_c$, where V_c is the total coil volume and A_c is the coil section area. In other words so that the coil volume equals the coil area swept over the linear distance πD_c.

Alpha : (real, dimensionless) Coil packing factor α, (conductive material volume / total volume), a number between 0 and 1. For close-packed circular wires (centers on vertices of equilateral triangular grid) the packing factor is $\pi/(2\sqrt{3}) \approx 0.91$. For rectangular packing (non-nested annular layers) the packing factor is $\pi/4 \approx 0.79$. Insulation thickness lowers the packing factor by the square of the core to insulated diameter ratio.

Tcoil : (real, K) Coil temperature T at which electrical conductivity is evaluated.

LinkMult : (real, dimensionless) An empirical multiplier F_L (range 0 to 1) used to account for any magnetic fringing flux leakage through the coil volume that subtracts from the flux passing through the inner core. The effective linked flux (average flux circumscribed by all windings) is taken as $F_L\phi$, where ϕ is the magnetic flux passing through the magnetic flux connections. (see discussion below)

Conductor : (selection list) Wire conductor material.

Outputs

Wdissip : (real, W) Coil time average electrical resistance I^2R loss.

Rcoil : (real, Ohms) Coil resistance R.

Lcoil : (real, Henries) Coil effective inductance L calculated from voltage drop ΔV and current time derivative $\dot{I}$ as $L = -\frac{\oint \Delta V \dot{I}}{\oint \dot{I}^2}$. For an ideal inductor this give the correct value as you can see by substituting the defining relationship $-L\dot{I}$ for ΔV. Essentially in this formulation L is a coefficient in an orthogonal function expansion with $\dot{I}$ the orthogonal function. The interpretation for a coil within a linear-actuator model is more complicated. The voltage drop will then have a transduction component proportional to velocity $C_f\dot{x}$ (*see* section 25.6.1). If $\dot{x}$ is not orthogonal to $\dot{I}$ (velocity not in phase with current) then Lcoil will differ from the true inductance measured when the linear actuator is stationary.

Acoil : (real, m^2) Total cross section area A_c of the coil winding normal to wires. The cross section area into which the coil is wound.

Awire : (real, m^2) Wire conductor cross section area A_w not including insulation.

Vcoil : (real, m^3) Total volume of the coil winding V_c. The volume of the entire toroidal envelope with cross section A_c enclosing the coil.

Vwire : (real, m^3) Wire conductor total volume V_w. The volume of conductive material in the winding not including insulation.

Lwire : (real, m) Wire total length L_w.

Mwire : (real, kg) Wire mass $\rho_s V_w$, where ρ_s is conductive material density.

FVneg : (Fourier series, V) Voltage at left (negative oriented) end of winding V_-.

FVpos : (Fourier series, V) Voltage at right (positive oriented) end of winding V_+.

FDeltaV : (Fourier series, V) Voltage drop $\Delta V = V_+ - V_-$.

FI : (Fourier series, A) Electrical current in winding.

FW : (Fourier series, W) Electrical power delivered to the winding.

FDeltaPsi : (Fourier series, A) Induced magnetic potential $\Delta\Psi$ from Ampere's law (25.25).

FBflux : (Fourier series, Wb) Total linked magnetic flux ϕ through coil core.

FWm : (Fourier series, W) Magnetic power delivered to component $-\dot{\phi}\Delta\Psi$.

FDeltaVem : (Fourier series, V) Electromagnetic voltage drop. The total voltage drop ΔV less the resistive component IR. Comprising the inductive component $-L\dot{I}$, related to the rate-of-change of stored electromagnetic energy with current and, for coils within linear-actuator models, the transductive component $C_f\dot{x}$, related to the rate-of-change of stored electromagnetic energy with velocity (*see* section 25.6.1).

25.4.1 Fixed Coil

TGtCoil

A fixed coil is one that does not move relative to the other magnetic components in the model. It does not consider the forces on the coil generated by the interaction of the coil magnetic field with an external magnetic field or the velocity of the coil winding through that magnetic field

Figure 25.2 shows the coil geometry. The input wire diameter d_w determines the conductive material cross section area of a single wire as

$$A_w = \frac{\pi}{4} d_w^2 \tag{25.20}$$

Additional inputs, number of turns N and coil packing factor α determine the cross section area of the total coil as

$$A_c = \frac{N A_w}{\alpha} \tag{25.21}$$

The final geometric input, centroid diameter D_c determines total coil volume as

$$V_c = \pi D_c A_c \tag{25.22}$$

The total volume of conductive material in the coil is the total volume multiplied by the packing factor

$$V_w = \alpha V_c \tag{25.23}$$

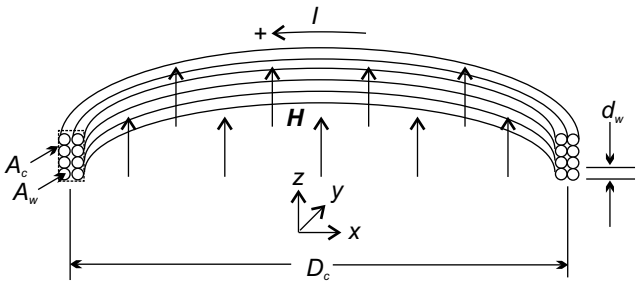


Figure 25.2: Coil geometry with arrows pointing in the direction of positive coordinate directions and current flow. According to the right-hand rule for magnetic fields a positive current produces a positive z-directed (upward)magnetic field $\boldsymbol{H}$ in the coil core.

The overall wire length is the total conductive material volume divided by the wire cross section area

$$L_w = \frac{V_w}{A_w} \tag{25.24}$$

Magnetic Field The current I flowing through the wire and the number of turns N in the coil winding determine the magnetic potential difference according to Ampere's law

$$\Delta\Psi = -NI \tag{25.25}$$

This is a well-known result that derives from the Maxwell equation relating current density J_n normal to the coil cross section and the magnetic field H in integral form:

$$\int_A J_n dA = \oint \boldsymbol{H} \cdot d\boldsymbol{s} \tag{25.26}$$

In this case integration surface A on the left side includes the currents of the individual wires in the coil cross section A_c and the path of integration on the right side can be any path that includes A_c. In particular it can be axially upward through the coil inner cylinder then diverting radially to somewhere outside the coil (where H is negligible) then downward axially, then converging radially inward to the other end of the coil. Symmetry arguments for the two radial legs and the arbitrariness of the inner axial leg leads to the conclusion that the H field is uniform in the axial direction within the coil inside cylinder and that is the only part that contributes to the loop integral. So the right side may be written as $\int \boldsymbol{H} \cdot d\boldsymbol{\ell}$ over the length ℓ of the inner cylinder. But by the definition of magnetic potential this is just $-\Delta\Psi$.

Because of the formulation in terms of magnetic potential difference $\Delta\Psi$ neither the value of the magnetic field H nor length of the coil cylinder ℓ are important individually. This explains why the coil axial length is not a required input variable.

Why can't the path of integration on the right side of the equation (25.26) include the external magnetic flux path, you ask, leading to a different conclusion. The reason is that the magnetic flux path includes its own electrical currents associated with the magnetic domains of ferromagnetic materials and these would have to be considered in the bounding surface integral of electrical current density on the left side.

Induced Voltage The rate of change of magnetic flux $d\phi/dt$ through the coil inner core and the number of turns N determine the zero-current inductive voltage difference across the coil terminals according to Faraday's law

$$\Delta V_L = N\frac{d\phi}{dt} \tag{25.27}$$

This is another well-known result that derives from the Maxwell equation relating the rate of change of magnetic flux density in the coil inner cylinder to the electrical field E loop tension in integral form:

$$\frac{d}{dt}\int_A B_n dA = -\oint \boldsymbol{E}\cdot d\boldsymbol{s} \tag{25.28}$$

In this case the integration surface A on the left side is the coil inner cylinder cross section and in terms of the magnetic flux through that cross section is just $d\phi/dt$. The path of integration on the right side applies to each turn of the winding separately. When added together for the total number of turns N the integral on the right amount to the total voltage difference across the coil terminals ΔV.

Because of the changing magnetic field produced by a time-varying current via Ampere's law there is some degree of *self induction* that tends to counter the current change by an opposing voltage change. The degree of this self induction depends on the properties of the magnetic flux path to which the coil is connected because that determines the size of the magnetic flux variation.

Flux Linkage Multiplier The above account assumes all of the magnetic flux passes through the coil core and returns by some path entirely outside the coil volume. This is usually a good approximation to reality because the magnetic flux is contained within a ferromagnetic material of very high magnetic permeability compared to the coil or surrounding air. But not always. In moving-magnet linear motor applications, for example, there is generally a gap of low magnetic permeability in the ferromagnetic flux path in the vicinity of the coil. So some fringing magnetic flux may pass through parts of the coil with the direction of flux opposite to that in the core. To account for this Sage includes the empirical multiplier input F_L (LinkMult) and modifies the induced voltage calculation of equation (25.27) to

$$\Delta V_L = F_L N\frac{d\phi}{dt} \tag{25.29}$$

to account for the fact that the linked flux is $F_L\phi$. In order to conserve energy it is also necessary to modify the magnetic potential difference calculation of equation (25.25) to

$$\Delta\psi = -F_L N I \tag{25.30}$$

This follows from an energy conservation principle. The electrical energy flow associated with the induced voltage ΔV_L is

$$\frac{dE_e}{dt} = -\Delta V_L I = -F_L N \frac{d\phi}{dt} I \tag{25.31}$$

The magnetic energy flow associated with the changing magnetic potential difference $\Delta\psi$ is, according to equation (25.35)

$$\frac{dE_m}{dt} = H\frac{dB}{dt} = -\Delta\psi\frac{d\phi}{dt} = F_L N I \frac{d\phi}{dt} \tag{25.32}$$

so that the changes of electrical and magnetic energy are equal and opposite and energy is conserved.

Resistance The electrical resistance in the wire creates a separate resistive voltage drop whenever electrical current I is flowing according to

$$\Delta V_r = -IR = -I\sigma\frac{L_w}{A_w} \tag{25.33}$$

Where σ is the electrical resistivity (Ohm-m).

Total Voltage Difference The total voltage difference across the coil terminals is the sum of the part due to time-varying magnetic flux and the component due to resistance

$$\Delta V = \Delta V_L + \Delta V_r \tag{25.34}$$

25.5 Magnetic Components

Magnetic components form the basic elements of magnetic circuits. They include air gaps, permanent magnets and ferromagnetic flux paths. All magnetic components define a scalar magnetic potential Ψ at each end of the component and a magnetic flux ϕ that is a function of the magnetic potential difference $\Delta\Psi$ across the component. The negative potential difference $-\Delta\Psi = \Psi_- - \Psi_+$ is the so-called magnetomotive force. Magnetic flux ϕ may be a function of time but is presumed uniform in space throughout the component. Each component defines a different functional relation ship $\phi(\Delta\Psi)$ between magnetic flux and magnetic potential difference. This is equivalent to a functional relationship $B(H)$ between magnetic flux density and magnetic field strength.

Energy and magnetic fields This manual does not go into the basic theory of electromagnetism. But it does have an axe to grind about the way that theory is generally presented in terms of Maxwell's equations involving a magnetic field and flux $\boldsymbol{H}$ and $\boldsymbol{B}$. Maxwell's equations, beautiful though they are, do not say anything about the physical nature of $\boldsymbol{H}$ and $\boldsymbol{B}$ and what is the difference between them. It doesn't help that the two are directly proportional in a vacuum. What are the natures of two invisible quantities that are proportional to each other?

So the Sage manual offers a mental crutch for the abstractly impaired. A mental picture that comes from taking the definition of magnetic energy as a fundamental concept along with Maxwell's equations. By taking as axiomatic this differential relationship for the change in magnetic energy per unit volume

$$dE_m \equiv \boldsymbol{H} \cdot d\boldsymbol{B} \tag{25.35}$$

assumed to apply universally in vacuum or in a solid material — no exceptions. In this form the roles of $\boldsymbol{H}$ and $\boldsymbol{B}$ are seen to be fundamentally different. In fact they are similar to the roles of force $\boldsymbol{F}$ and displacement $d\boldsymbol{x}$ in the equation for the work done on a simple mechanical spring

$$dE_s \equiv \boldsymbol{F} \cdot d\boldsymbol{x} \tag{25.36}$$

By analogy it is clear that $\boldsymbol{H}$ is something like a force and $\boldsymbol{B}$ something like a displacement. In fact $\boldsymbol{H}$ is the gradient of a magnetic potential much like $\boldsymbol{F}$ is the gradient of a mechanical potential (spring potential energy). In the case of $\boldsymbol{B}$ in is not clear exactly what is being displaced from its equilibrium value but still the idea that $\boldsymbol{B}$ is a displacement provides a useful mental image. Whatever is being displaced, it is displaced even in a vacuum. So it is some property of space itself outside our usual sensory perceptions. In these terms the fact that $\boldsymbol{B}$ and $\boldsymbol{H}$ are proportional in a vacuum may be understood to mean that $\boldsymbol{B}$ is stretching like a spring in response to an increasing $\boldsymbol{H}$ force.

Conventional terminology that $\boldsymbol{B}$ is a magnetic *flux* is misleading. It suggests $\boldsymbol{B}$ is something flowing through space rather than a displacement from an equilibrium value. It would be better to call $\boldsymbol{B}$ the magnetic displacement and if anything is to be called the magnetic flux it should be the rate of change $d\boldsymbol{B}/dt$.

25.5.1 Common Variables

For software design reasons magnetic components are subdivided into a parent component that manages the solution grid and a child *object* component with variables specific to a particular magnetic material.

Magnetic parent components share some common variables although not all are present, depending on the component:

Inputs

Lpath : (real, m) Magnetic path length ℓ.

Apath : (real, m^2) Magnetic path area A.

Outputs

Mmag : (real, kg) Mass of magnetic material.

FPsiNeg : (Fourier series, A) Magnetic potential at negative pole Ψ_-.

FPsiPos : (Fourier series, A) Magnetic potential at positive pole Ψ_+.

FH : (Fourier series, A/m) Magnetic field $H = -\Delta\Psi/\ell$.

FB : (Fourier series, C) Magnetic flux density $B = \phi/A$.

FWm : (Fourier series, W) Magnetic power delivered to the component.

From magnetic energy formulation (25.35) The instantaneous magnetic power delivered to a magnetic component of volume $v = A\ell$ is $W = \int_v H\dot{B}dv$. In terms of magnetic flux $\phi = BA$ and potential difference $-\Delta\Psi = \int Hd\ell$ this works out to $W = -\dot{\phi}\Delta\Psi$. Output FWm is just the Fourier series representation of W as a function of time.

Magnetic child objects share these common inputs:

Material : (selection list) The magnetic material selected by name from a list of ferromagnetic or permanent magnet materials, depending on the component.

Tm : (real, K) Material temperature T at which magnetic properties are evaluated.

ThkLam : (real, m) Lamination thickness a. To minimize eddy current losses magnetic materials typically comprise multiple layers or *laminations* with the layers parallel to the magnetic field direction (section 25.7.3). If there are no layers this input is still important because it affects the eddy current magnetic field and I^2R loss, both of which are modeled in Sage. Generally, it should be set to the minimum dimension of the material when viewed from the z direction parallel to the magnetic field (*see* table 25.5.1).

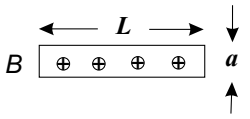
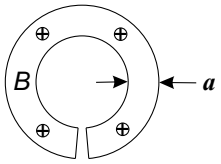
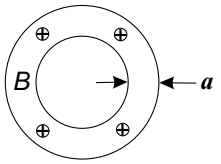
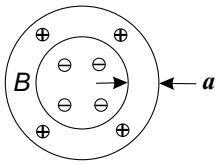
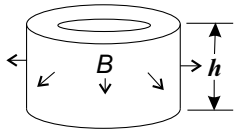
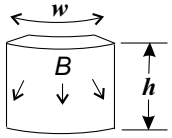
description	**shape**	ThkLam
Rectangle with $\boldsymbol{B}$ perpendicular to section		$\mathbf{min}(\boldsymbol{a}, \boldsymbol{L})$
Split ring with axial $\boldsymbol{B}$		$\boldsymbol{a}$
Closed ring with axial $\boldsymbol{B}$, no magnetic flux inside		$\boldsymbol{a}$
Closed ring with axial $\boldsymbol{B}$, return magnetic flux inside		$\mathbf{2}\boldsymbol{a}$
Closed annular ring with radial $\boldsymbol{B}$		$\boldsymbol{h}$
Annular ring with radial $\boldsymbol{B}$, ring broken into sections		$\mathbf{min}(\boldsymbol{h}, \boldsymbol{w})$

Table 25.1: Meaning of lamination thickness. For un-laminated magnetic materials the lamination thickness input ThkLam may be approximated according to this table.

25.5.2 Magnetic Connection Block

TGtMblk

A component of zero magnetic reluctance that allows you to connect two or more magnetic components together. Once connected the magnetic potentials Ψ in each component adjust so all the connected poles share a common magnetic potential and the net magnetic flux flowing into the connection block is zero ($\nabla \cdot \boldsymbol{B} = 0$). This is the magnetic analog of an electronic connection terminal except in magnetic circuits it is usually impossible to implement a magnetic flux path with negligible reluctance compared to the other circuit elements. It may be useful to think of this component as representing a physical contact between two or more magnetic poles where there is a negligible impedance of magnetic flux among them.

25.5.3 Magnetic Potential Reference

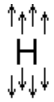
TGtMref

This component is similar to the above magnetic connection block except you explicitly set the magnetic potential Ψ. It functions something like a ground connection in an electrical circuit. In any magnetic circuit the magnetic potential is indeterminate unless you specify the absolute value (usually zero) at least one point. Otherwise adding an arbitrary constant offset Ψ_o to all the magnetic potentials of the circuit would produce an equivalent circuit (same magnetic potential drops and magnetic fluxes). Generally you will connect a magnetic potential reference to one point of a magnetic circuit using a upward or downward facing pole. In principle it is possible to *ground* a magnetic circuit at more than one point using different potentials although there is a more physical way to do that using a permanent magnet component.The one additional input compared to the inputs listed above is:

FPsi : (Fourier series, A) Magnetic potential offset Ψ_o.

Magnetic potential references are the only magnetic components where the net magnetic flux flowing into the component can be non-zero.

25.5.4 Magnetic Field Source



TGtHsrc

This component specifies the magnetic field magnitude H between two poles of a magnetic circuit as an input. It produces a magnetic potential difference $\Delta\Psi = -H\ell$, where ℓ is the magnetic path length (Lpath). The magnetic potentials and fluxes in the rest of the circuit respond accordingly.

In effect it models a coil (section 25.4) with zero magnetic reluctance where the current is automatically adjusted to produce the required magnetic field. You might use this component to approximate such a coil or to provide boundary conditions to evaluate other magnetic circuit elements.

Additional inputs compared to the variables listed above are:

FH : (Fourier series, A/m) Fixed magnetic field H overriding above dependent variable.

25.5.5 Magnetic Flux Source

TGtBsrc

This components is similar to the above magnetic field source except the roles of B and H are reversed. It specifies the magnetic flux density B between two poles of a magnetic circuit as an input. It supplies whatever magnetic potential drop is necessary to produce the required flux density. The magnetic potentials and fluxes in the rest of the circuit respond accordingly.

Additional inputs compared to the variables listed above are:

FB : (Fourier series, V) Fixed magnetic flux density B overriding above dependent variable.

25.5.6 Air Gap

TGtAirGap

This component represents simple magnetic air gap with gap length and area inputs Lpath and Apath (above). It forces a relationship between magnetic flux ϕ and magnetic potential drop $\Delta\Psi$ equivalent to the vacuum relationship

$$\boldsymbol{B} = \mu_0 \boldsymbol{H} \tag{25.37}$$

Where $\mu_0 = 4\pi 10^{-7}$ (H/m) is the permeability of free space.

The term *air gap* is traditional based on common usage in Earth's atmosphere and has nothing to do with the actual gas that may be in the gap. It would be more accurate to call this component an *empty gap*.

25.5.7 Permanent Magnet

TGtPermMag

This component provides a magnetic flux ϕ as a function of magnetic potential difference $\Delta\Psi$ according to a fixed demagnetization curve $B(H) = \mu_0 H + J(H)$ as described in section 25.7.2. In addition to Material and Tm listed above a permanent magnet has these inputs:

Jmult : (real, dimensionless) Scale factor used to change the direction or magnitude of magnetic polarization $J(H)$. A value $+1$ corresponds to a north pole at the positive end. A value -1 to a south pole at the positive end. Other values may be used to effectively weakening or strengthening the magnet relative to the Material data.

DemagLimit : (real, dimensionless) The external magnetic field beyond which significant demagnetization occurs, expressed as a fraction of H_{cB}. The default value is 1. Sage will warn you if the external field exceeds this value, otherwise it has no effect on the solution.

And adds this output:

FUtilization : (Fourier series, dimensionless) Utilization factor defined as the BH energy product along the demagnetization curve as a fraction of the maximum value. Small values for the time-mean and harmonic amplitudes suggest the magnet may be too large or too small. *See* section 25.7.2

No Hysteresis In solutions with time varying H permanent magnets do not traverse any minor hysteresis loops when traversing back and forth along demagnetization curve. This is a reasonable approximation to reality provided the external magnetic field is well below the coercive force limit H_{cJ}. It also avoids the problem of solution transients demagnetizing the magnet and leading to an unexpected solution — possibly multiple solutions depending on initial conditions and the path to the solution. In other words permanent magnets are *really permanent* in Sage.

But this is not always a good thing. If you impose a high demagnetizing field on a permanent magnet in Sage it will always recover its magnetization when the field returns to a low value. A real magnet on the other hand may be permanently demagnetized and never recover. Therefore the inputs for a permanent magnet include the value DemagLimit, the relative fraction of the coercive force H_{cB} above which the magnet undergoes significant demagnetization. The default value is 1. The Sage solver will warn you whenever $|H|/H_{cB}$ exceeds DemagLimit. If this warning occurs during the solution process as a result of transient values then you can probably ignore it. If it occurs on re-opening a model in the solved state and H_{cB} and DemagLimit are set correctly then you ignore it at your peril.

25.5.8 Soft Ferromagnetic Material

TGtSoftMag

Soft ferromagnetic materials are generally used to conduct a cyclically reversing magnetic flux with as little material as possible. So peak flux densities are generally high (near saturation). Compared to a permanent magnet the coercive force required to change the direction of magnetization is relatively low. But still there is some coercive force required, which produces an energy dissipating hysteresis loop $B(H)$ as the H field varies back and forth. Sage captures this energy dissipation by solving the Magnetization $M(H)$ as a function of both H and dH/dt formulated in terms of the magnetic properties usually available for soft iron materials. To help you keep an eye out for saturation a soft ferromagnetic material adds this output:

FJ : (Fourier series, T) Magnetic polarization $J = \mu_0 M$.

The output is magnetic polarization rather than magnetization because the saturation value J_s is more commonly available in material property tables than M_s and it is also a number on the order of 1 rather than 10^6 (in SI units).

Pseudo-Hysteresis Formulation Since magnetization M depends on the previous history as well as the current magnetic state it should, in principle, be solved as a differential equation (*see* section 25.7.1). One of the false starts of Sage development was to actually solve M as a differential equation but it was soon apparent that doing so led to unacceptable numerical errors in Sage's coarse time grid and also convergence problems. So instead of that, Sage solves M using a simplified formulation motivated by the observation that all minor

hysteresis loops fall between the two limiting curves of the outer hysteresis loop, produced by driving the magnetization to saturation in one direction then the other. Referring to figure 25.11 in section 25.7.1, the two limiting magnetization curves are roughly separated horizontally from each other by the distance $2H_c$, where Hc is the coercive force ($M = 0$ intercept. In other words, the $dH > 0$ branch is approximately $M_a(H - H_c)$ and the $dH < 0$ branch $M_a(H + H_c)$. In particular Sage formulates a shift value H_s that varies smoothly between $\pm H_c$

$$H_s = \begin{cases} \frac{dH}{d\tau}\frac{H_c}{H_1} & \text{if } H_1 > H_c \\ \frac{dH}{d\tau} & \text{otherwise} \end{cases} \tag{25.38}$$

where H_1 is the amplitude of the H and $\tau = \omega t$ is the dimensionless cycle angle. The formulation is based on the observation that for sinusoidally varying quantities the value $dH/d\tau$ takes its extreme values 90 degrees out of phase with the extreme values of H. Since the amplitude of $dH/d\tau$ is also H_1, the first branch amounts to a sinusoidal function with amplitude H_c. The second branch applies to low amplitude H solutions and scales down the shift H_s in proportion to the H amplitude. To deal with the possibility of non-sinusoidal H the value H_s is truncated to lie within the limits $\pm Hc$.

Sage estimates the amplitude H_1 using local solution information using a formulation that is valid provided the variation of H is sinusoidal or nearly so but reasonable in any case.

$$H_1 \approx \sqrt{\left(\frac{dH}{d\tau}\right)^2 + \left(\frac{d^2H}{d\tau^2}\right)^2} \tag{25.39}$$

This may make more sense if you think of $dH/d\tau = H_1 \sin\tau$ and $d^2H/d\tau^2 = H_1 \cos\tau$.

The value of M is then defined implicitly as

$$M = M_a(x_s) \tag{25.40}$$

where M_a is the function (25.96) and x_s is the shifted dimensionless magnetic field

$$x_s = \frac{H - H_s + \alpha M}{\alpha M_s} \tag{25.41}$$

Hysteresis loops for this pseudo-hysteresis approximation are illustrated in figure (25.11) of section 25.7.1 for the case of purely sinusoidal H variation. For large H amplitudes (M near saturation) the pseudo-hysteresis loops appear quite reasonable compared to those calculated with the hysteresis differential equation. For small H amplitudes the differences are more noticeable with the hysteresis loss is somewhat overestimated — the price for reliable convergence.

25.5.9 Non Magnetic Material

This component models the magnetic flux path within a non-magnetic electrically conductive material. It has the same free-space magnetic permeability as

TGtNonMag

an air gap, $\mu_0 = 4\pi 10^{-7}$ (H/m). But unlike an air gap it models eddy current losses according to the formulation of section 25.7.3. Any time-varying magnetic flux generates an opposing magnetic field, depending on material electrical conductivity and effective lamination thickness (ThkLam input).

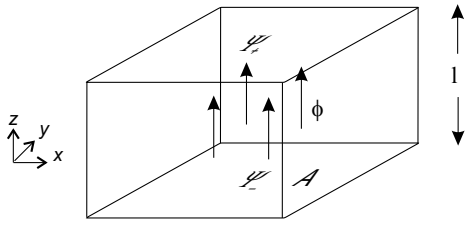
The reason for this component is that some electro-magnetic devices impose a pressure wall between internal and external parts of the magnetic flux path. For example, a linear motor design may locate the coil outside a pressure wall to minimize internal volume, avoid contamination issues and avoid the need for wire feedthroughs. Often this pressure wall must be a metal (electrically conductive) for strength purposes.

If you want to model a pressure wall made from a ferromagnetic material use the Soft Ferromagnetic Material component instead. That component also models eddy current losses but also includes the increased magnetic permeability of a ferromagnetic material and magnetic hysteresis.

For the present component the selection list for the Material input comes from the electrical materials data file. For the Soft Ferromagnetic Material component it comes from the soft ferromagnetic materials data file (*see* chapter 28).

25.5.10 Solution Method

All magnetic components share a common computational framework based on a lumped magnetic circuit approach. A magnetic component may be represented abstractly in terms of a solid of length ℓ and cross section area A with uniform magnetic potentials Ψ_+ and Ψ_- on the upward and downward facing ends. The potential difference drives a uniform magnetic flux ϕ through the component, as shown in this diagram:



Potential difference $\Delta\Psi$ and magnetic flux ϕ may vary with time.

The magnetic field within the component is presumed uniform (only a function of time) and broken into two parts, $\boldsymbol{H}+\boldsymbol{H}_{eddy}$, where $\boldsymbol{H}$ is the net magnetic field that determines the magnetic flux and magnetization and $\boldsymbol{H}_{eddy}$ is the opposing field due to eddy currents (*see* section 25.7.3 for details). The magnetic flux and potential difference are related to the magnetic fields $\boldsymbol{H}$, $\boldsymbol{H}_{eddy}$ and magnetic flux density $\boldsymbol{B}$ through these defining equations:

$$\boldsymbol{B} = \phi/A\ \hat{\boldsymbol{k}} \tag{25.42}$$

$$-\Delta\Psi = \int (\boldsymbol{H} + \boldsymbol{H}_{eddy}) \cdot d\boldsymbol{\ell} = (H + H_{eddy})\ell \tag{25.43}$$

where $\hat{\boldsymbol{k}}$ is the unit vector in along the component z axis. So the net scalar magnetic field is

$$H = -\Delta\Psi/\ell - H_{eddy} \tag{25.44}$$

Implementation Details

In some components both magnetic potential offset Ψ_o and potential drop $\Delta\Psi = \Psi_+ - \Psi_-$ are solved in the computational grid with the magnetic potentials at the two ends evaluated explicitly as

$$\Psi_- = \Psi_o - \Delta\Psi/2 \tag{25.45}$$

$$\Psi_+ = \Psi_o + \Delta\Psi/2 \tag{25.46}$$

In other components only Ψ_o is solved, presuming $\Delta\Psi = 0$. Magnetic flux ϕ is evaluated explicitly as the average of the magnetic fluxes flowing through the negative and positive ends.

$$\phi = \frac{\phi_- + \phi_+}{2} \tag{25.47}$$

Generally, the magnetic flux is forced to be the same at the two ends ($\nabla \cdot \boldsymbol{B} = 0$) so ϕ is the uniform magnetic flux flowing through the component.

Magnetic flux determines magnetic potential offset Generally the condition of uniform magnetic flux ($\phi_- = \phi_+$) determines the magnetic potential offset Ψ_o implicitly. The exception being the magnetic potential reference components where the Magnetic potential offset is set by inputs and the magnetic flux can be different at the two ends.

Component physics determines magnetic potential drop The magnetic potential drop $\Delta\Psi$ is evaluated implicitly as a function of magnetic flux ϕ according to the physics for the particular component.

The objective is to adjust $\Delta\Psi$ so that $\phi(\Delta\Psi)$ demanded by the physics equals the actual value of ϕ. Except for most components it is more convenient to formulate the relationship $B(H)$ rather than the relationship $\phi(\Delta\Psi)$, the equivalence between the two being

$$\phi(\Delta\Psi) \equiv A\ B(-\Delta\Psi/\ell - H_{eddy}) \tag{25.48}$$

Magnetic flux based on Magnetic potential continuity Connections between poles provide magnetic flux ϕ solved implicitly from the condition that the magnetic potential is the same across the connection. When there are multiple pole connections at one or both ends of a component, the magnetic potentials are all the same. Each component is responsible for enforcing the requirement that the sum of magnetic fluxes flowing through the negative boundary equals the sum of magnetic fluxes flowing through the positive boundary.

Solved by Sage Taken together in the Sage solution framework the implicit variables identified above suffice for Sage to solve for magnetic potentials and magnetic fluxes everywhere in a circuit comprising any number of magnetic components connected together in ways that make physical sense.

Validity of Scalar Potential with Eddy Currents

Technically the validity of a scalar potential formulation depends on the path-independence of $\int(\boldsymbol{H}+\boldsymbol{H}_{eddy})\cdot d\boldsymbol{s}$ between any two points in the magnetic solid, or equivalently $\oint(\boldsymbol{H}+\boldsymbol{H}_{eddy})\cdot d\boldsymbol{s}=0$ for any closed path. For the net field $\boldsymbol{H}$ this is the case because, by definition, $\boldsymbol{H}$ is the field that would be present in the absence of eddy currents and only electrical currents can disrupt the path independence of $\oint \boldsymbol{H}\cdot d\boldsymbol{s}$.[1] But this is not the case for the eddy field $\boldsymbol{H}_{eddy}$. According to section 25.7.3 it varies from zero at the outer surface of the magnetic material to maximum at the center. But it *is* the case for the section average of $\boldsymbol{H}_{eddy}$ (averaged in any x-y plane), which is what is important for a lumped parameter analysis. So there is really no difference between the eddy field $\boldsymbol{H}_{eddy}$ and the net magnetic field $\boldsymbol{H}$ when it comes to representation by a scalar magnetic potential. Both vary locally within the magnetic component and the only assumption is that the scalar magnetic potential difference between the component z ends represents the integral of the section averages of $\boldsymbol{H}+\boldsymbol{H}_{eddy}$ along the z axis.

25.6 Linear Motors, Alternators, Actuators

Electromagnetic transduction components model the conversion of electrical power to mechanical power or vice-versa via the interaction of components like permanent magnets or coils moving through an air gap between magnetic pole pieces.

25.6.1 Simple Transducer

TXducer

But first there is a simple transducer component that functions as a bridge between the electrically-challenged linear motors of section 29.3.12 and the components below that model detailed electromagnetic physics. A transducer component is born with electrical current and force connectors for connecting to other components of your model. There is only a time-ring version. It is a relative moving part, which means that each end (x_{neg} and x_{pos}) should be connected to a different moving part, one of which may be stationary.

A simple transducer throws to the dogs any electromagnetic physics that might be behind the conversion of electrical current to mechanical force and

[1] According to Maxwell's equations the integral $\oint \boldsymbol{H}\cdot d\boldsymbol{s}$ around a closed path equals the integral $\int C_n d\sigma$ over any surface bounded by that path, where C_n is the electrical current flow normal to surface element $d\sigma$.

instead simply assumes a linear relationship between the two

$$F = C_f I \tag{25.49}$$

The only physics this component implements is to produce a voltage drop across its electrical terminals according to an energy conservation principle — namely, that the mechanical power input equals the electrical power output:

$$F\frac{dx}{dt} = \Delta V I \tag{25.50}$$

Solving for ΔV and using the definition of C_f, the energy conservation principle implies that the voltage drop across the transducer terminals must be

$$\Delta V = C_f \frac{dx}{dt} \tag{25.51}$$

This voltage drop is an output of the *transduction voltage* child component built into the linear transducer component. Resistive, inductive and capacitive circuit elements are not included. They are to be implemented as separate circuit elements connected to this component by current connections.

The force given by equation (25.50) acts on the moving part attached to endpoint coordinate x_{pos} and the negative of that force on the moving part attached to x_{neg}. The x in velocity dx/dt refers to the relative position $x_{pos} - x_{neg}$. Whether a transducer functions as a motor or alternator depends on the relative phase between current I and relative velocity dx/dt and also the sign of C_f. If C_f is positive and I and dx/dt are in phase within 90 degrees then the mechanical power output is positive, otherwise it is negative.

The force or *transduction* coefficient C_f is a quadratic function of position according to the inputs:

Cf0 : (real, N/A) Force or transduction coefficient C_{f0} at $x = 0$.

Xm : (real, m) Reference extension x_m.

Rp : (real, dimensionless) Force-coefficient ratio $R_p = C_f/C_{f0}$ at $x = x_m$.

Rn : (real, dimensionless) Force-coefficient ratio $R_n = C_f/C_{f0}$ at $x = -x_m$.

These are the same as for the relative linear motor component of section 29.3.12, which this component replaces. If you already know something about the physics behind the transducer you are modeling you may be able to read Cf0, Xm, Rp, Rn more-or-less directly, from a plot of C_f vs x generated either experimentally or computationally, or more precisely from a best-fit parabola to the function $C_f(x)$ over the intended operating range.

The force produced by the motor is still the product of force coefficient and current, as in equation (25.49), but now C_f is the quadratic function of position

$$C_f(x) = C_{f0}\left[1 + a(x/x_m) + b(x/x_m)^2\right] \tag{25.52}$$

where coefficients a and b are formulated in terms of inputs as

$$a = (R_p - R_n)/2 \tag{25.53}$$
$$b = (R_p + R_n)/2 - 1 \tag{25.54}$$

This component will drive any attached reciprocating masses according to the force-coefficient inputs and electrical current flow. It is important that the components a transducer is connected to be capable of determining an absolute mean position (e.g. a constrained piston or connect to a spring to ground). Otherwise the solver will tend to drift around without converging.

25.6.2 Interpolated Transducer

TXducerCfX

The simple transducer component has proven useful over the years so Sage version 13 has added a descendant where the relationship for the force-coefficient $C_f(x)$ is more general. It is interpolated directly from a cubic-spline input:

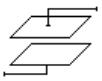
Cfx : (cubic spline, N/A) A set of data pairs (x_i, C_{f_i}), where x_i is a position relative to the transducer zero position in meters (or other units selected in the model-class options dialog) and C_{f_i} is the force coefficient at that position. The x_i values must be listed in increasing order from the largest negative position expected during operation at the top to the largest positive position at the bottom. The number of data pairs is arbitrary and will depend on the complexity of the variation of C_f with position. The input dialog for specifying the (x_i, C_{f_i}) pairs has a View Interpolation button you can click to see the smooth curve resulting from cubic spline interpolation. It is possible to paste values from a spreadsheet directly into the pairs input dialog.

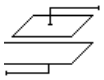
This component shares the same output variables as its ancestor linear-transducer component. It deprecates the Cf0, Xm, Rp and Rn inputs because of the new formulation.

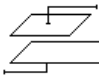
It is up to you to specify Cfx according to measurements of your particular linear motor or alternator. If the $C_f(x)$ data is too nonlinear or spiky you should increase the value of NTnode in the root model (number of time nodes) in order to ensure that the computational grid can actually resolve the $C_f(x)$ curve. You can gauge this by plotting the connector force or transduction voltage vs time curves in the resulting model solution. The voltage waveform is available in the built-in transduction-voltage subcomponent.

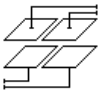
There is still linear relationship between C_f and electrical current according to equation (25.49). This means that there will be energy conservation in the conversion between mechanical and electrical power according to equation (25.50), even though there may still be some degree of numerical error in both. Numerical error will depend on the number of time nodes so it is a good idea to compare model solutions for increasing NTnode values. Generally speaking the size of numerical errors should decrease with increasing time nodes.

25.6.3 Magnetic Gaps


TGtSPGap


TGtLturnGap


TGtRturnGap


TGtTPGap

These components represent the air gap between magnetic pole pieces. Even though the pole geometry is depicted as rectangular, these components can model concentric cylindrical pole geometries that are typically found in linear motors or generators. The magnetic potential on each pole face is assumed uniform along the length and width (x and y) dimensions but variable with time. The magnetic flux emerging from or entering the upper and lower z faces of the poles may be connected to other magnetic model components via built-in magnetic flux connectors.

A *single-pole* magnetic gap (top icon in margin) represents a single opposed pair of parallel rectangular magnetic pole faces of length L (x-direction), width W (y-direction) and separated by a gap g_z (z-direction) through which magnetic flux flows. The *left-turned* and *right-turned* magnetic gaps (second and third icons in margin) are sub-types of single-pole gaps. They are intended for use with moving voice coils and deal with the special physics where the magnetic flux path of the inner (lower) pole first turns toward the left or right (x direction) for the coil extend before turning again to the z direction.

A *two-pole* magnetic gap (bottom icon in margin) represents two pairs of such pole faces separated in the x direction by a gap g_x. (See figure 25.3).

The same set of input variables serves for all magnetic gap components:

Zgap : (real, m) Total z-directed air gap g_z.

Wpole : (real, m) Common pole width W, or circumference in a cylindrical arrangement. In that case Wpole should be set to the effective pole circumference that gives the correct air-gap volume. For example, if the inner and outer pole faces are located on circles of diameters D_{in} and $D_{in} + 2g_z$ respectively then the effective pole circumference is the mid-circle circumference $W = \pi(D_{in} + g_z)$.

FringeMult : (real, dimensionless) An empirical multiplier used to scale the fringing flux outside the gap as a means to calibrate models. The 1-D Sage model employs an idealized calculation of the fringing flux based on the similarity of an electric field outside a parallel-plate capacitor (*see* section 25.6.8). Depending on the actual magnetic gap you are modeling the fringing flux may be more or less than that. The modeled fringing flux scales directly with FringeMult. It is still calculated from the idealized formulation but for an effective gap equal to FringeMult times the actual gap.

Lpole1 : (real, m) Pole 1 length L_1.

Xgap : (real, m) x-directed separation g_x between poles 1 and 2.

Lpole2 : (real, m) Pole 2 length L_2.

In the case of the two-pole magnetic gap the flux leakage between poles 1 and 2 across the gap g_x is presumed negligible, consistent with good design practices.

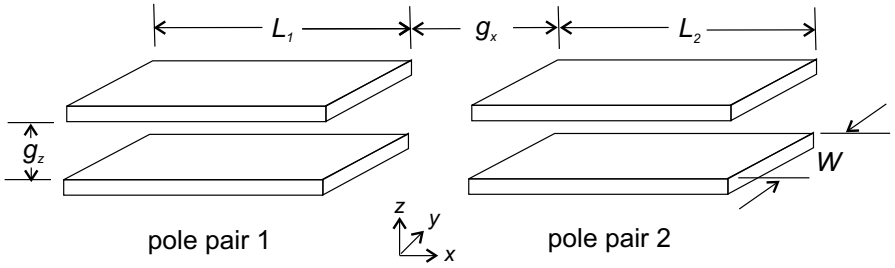


Figure 25.3: Two-pole magnetic gap geometry. The model assumes spatially-uniform, time-varying magnetic potentials on the inner faces of each pole and a z-directed magnetic flux between the poles that may vary with both x position and time.

But there is nothing in the software to prevent you from specifying a gap that is in fact too small.

Air Gap Physics Magnetic gaps defer most of the electromagnetic physics to a built-in container and its components as discussed below. But they do handle the physics for that part of the pole area not covered by the built-in container at any particular time. For this they use the same principles as the magnetic air gap discussed in section 25.5 based on the vacuum relationship between magnetic flux and potential of equation (25.37). See section 25.6.8 below for more discussion on this topic. In addition, the *left-turned* and *right-turned* gaps also deal with the variation of magnetic potential in the x direction induced by voice coils. *See* section 25.6.6.

There are two outputs associated with the poles of magnetic gaps. They are available within the *pole pair* child component(s) built into every magnetic gap component:

FBfluxAir : (Fourier series, Wb) Magnetic flux flowing through the part of the magnetic gap not covered by the moving electromagnetic container, if any, as a function of time. The uncovered part of the magnetic gap may be on one or both sides of the moving container or neither, depending on the container length and offset.

FWm : (Fourier series, W) Magnetic power inflow into the total magnetic gap calculated as $-\dot{\phi}\Delta\Psi$, where ϕ is the total pole magnetic flux and $\Delta\Psi$ is the potential difference.

In the absence of a moving container the time-average magnetic power inflow would be zero because there are no energy dissipating mechanisms in an air gap (ϕ and $\Delta\Psi$ are proportional). But with a moving container present, the magnetic energy flows to the components embedded in the container produce power flows at the poles. The magnetic energy flows to the components embedded in the container go to energy dissipations (hysteresis and eddy-current),

mechanical power flows and in the case of coils electrical energy inputs and dissipations.

For pole in left or right-turned magnetic gaps there is a third output:

FWmLinked : (Fourier series, W) Magnetic power inflow linked to the embedded voice coil or coils.

As illustrated in figure 25.5 a voice coil produces a magnetic potential difference in the *uncovered* region of the air gap between the moving container end and pole end. The magnetic flux in this region is linked through the coil because of the way the inner pole is presumed to *turn* the flux. So the magnetic power inflow to this uncovered region must be included in the total power inflow to the coil or coils embedded in the container. Therefore output FWmLinked must be included when calculating the overall energy balance within a left or right-turned magnetic gap.

25.6.4 Moving Electromagnetic Container

TEMovContainer

This component is found in every magnetic gap component as a built-in child component. Its purpose is to house and manage a series arrangement of one or more moving electromagnetic components that move along together in the x direction. For example: a coil, a moving magnet polarized in the z direction or a stack of several such magnets polarized in opposite directions as is sometimes used to modulate force versus position characteristics.

The container component defines the overall length (x-direction) of the internal component stackup and descends from the relative moving part component (section 16.8) with built-in connectors at each end. The force connector at the positive end is logically associated with the moving container and the force connection at the negative end with the outer pole pieces. The idea is that by connecting these connectors to other moving parts of your model you can simulate the relative motion of the electromagnetic container. Normally the pole pieces are fixed and the container moving but depending on how you configure your model it can be the other way around or both can be moving. Because of this flexibility the container component provides only the electromagnetic force between the inner container and outer poles leaving any inertial forces to be calculated by reciprocating mass components to which the force connectors are connected. In addition to the variables inherited from the relative moving part ancestor component moving electromagnetic containers have these variables:

Length : (real, m) Container length.

Offset : (real, m) x-directed offset of the container at its rest position. Variable x_o in figure 25.8.

Mass : (real, kg) Total mass of all electromagnetic components inside the container. An output variable.

EfluxErr : (real, W) Magnetic energy leak, an output variable. The discrepancy between the power flow into the pole gaps and the power dissipated in the moving electromagnetic container. A value of zero corresponds to perfect energy conservation. Replaces former Fscale output (*see* discussion in the *Energy Leakage* paragraph of section 25.6.8).

25.6.5 Moving Electromagnetic Components

All moving electromagnetic components are implemented in terms of a wrapper component that manages a space-time (x, t) solution grid with magnetic flux distribution variables ϕ_{x+} and ϕ_{x-} (see figure 25.10). This wrapper component defines common inputs pertaining to the component grid, z-thickness, position relative to the parent container and also a few common outputs. These common inputs and outputs are:

NCell : (dimensionless) The number of spatial cells in the computational grid. The higher the number the more accurate will be the magnetic flux distributions ϕ_{x+} and ϕ_{x-} between the outer poles and the electromagnetic component as it moves between the poles.

XnegRel : (real, dimensionless) The dimensionless x position of the component negative endpoint relative to the parent container. A number between 0 and 1, with 0 corresponding to the container negative endpoint and 1 to the container positive endpoint. The default value is 0.

XposRel : (real, dimensionless) Like XnegRel except for the component positive endpoint relative to the parent container. The default value is 1. (see below)

ZthkRel : (real, dimensionless) Component thickness as a fraction of the z-directed air gap g_z between the outer poles. A number between 0 and 1. The default value is 1. A number less than 1 corresponds to a sub-gap ΔZ between the outer poles and component on each side (see figure 25.9) with ΔZ = (1 - ZthkRel) $g_z/2$.

Length : (real, m) Dimensional component length L. An output calculated from inputs XnegRel, XposRel and the parent container length.

FFm : (Fourier series, N) Magnetic force acting on the component.

FBflux : (Fourier series, Wb) Spatial average magnetic flux through the moving component, $\int_{dx} \phi_x$, where ϕ_x is the component magnetic flux distribution.

Important When the container has more than one electromagnetic component the first must have the value XnegRel = 0 and the last XposRel = 1. subsequent components must have XnegRel equal to XposRel for the preceding component. In other words the container components must completely fill the container with no gaps or overlaps. Sage makes this assumption when allocating

the magnetic flux between the outer poles not covered by the moving container. If there is a gap then magnetic flux will be zero in that gap making it a magnetic insulator in effect with zero permeability, lower than the permeability of a vacuum. If there is an overlap then both components will see the same magnetic potential difference in the overlap as if they were connected in a parallel magnetic circuit instead of in series.

Embedded Objects The actual moving electromagnetic component embedded within the wrapper is implemented as a separate built-in child component with its own inputs, outputs and physics. For example, the *coil objects* in the moving coil components below. Such embedded components and their governing physical principles is the main topic of discussion in the following sections.

Generally speaking, moving electromagnetic components are similar to their stationary counterparts except the physics is refined to include a spatial distribution of magnetic flux density **B** depending on where a point x in the grid finds itself relative to the outer magnetic pole potentials. The quantity BW symbolized by ϕ_x is the magnetic flux per unit length (Wb/m). The total flux passing through the moving component is just $\phi = BWL = \phi_x L$.

Warning about pole spacing Even though the magnetic flux density may vary in the x direction, the flux direction itself is always presumed in the z direction (across the magnetic gap). This approximation is always reasonable for single-pole magnetic gaps but for multiple pole gaps there can be an issue in the region between one pole and the next, or near, if the pole-to-pole potential gradient (in the x direction) is not small compared to across-gap potential gradient (in the z direction). This can happen if the pole-to-pole spacing is smaller than the magnetic gap or the pole-to-pole potential difference larger than the across-gap potential difference. In that case the accuracy of the Sage solution will suffer. The problem is worse for coils or soft ferromagnetic materials where the magnetic flux direction is always proportional to the magnetic field direction. For permanent magnets — especially rare-earth magnets — it is less of an issue because the magnetic flux direction is mainly determined by the magnetization direction, which is locked into the material during fabrication.

25.6.6 Moving Coils

Moving coils implement the physics of non-moving coils discussed in section 25.4 except in the context of the flux distribution and coil orientation within the parent magnetic gap and including the mechanical force on the coil produced by the interaction with the magnetic flux. The inputs and outputs basically the same except for changes imposed by the magnetic gap geometry.

Transverse Coil

This component represents a rectangular coil of dimensions $L \times W$ moving between magnetic pole faces as illustrated in figure 25.4. The coil moves in

TCoilEMov

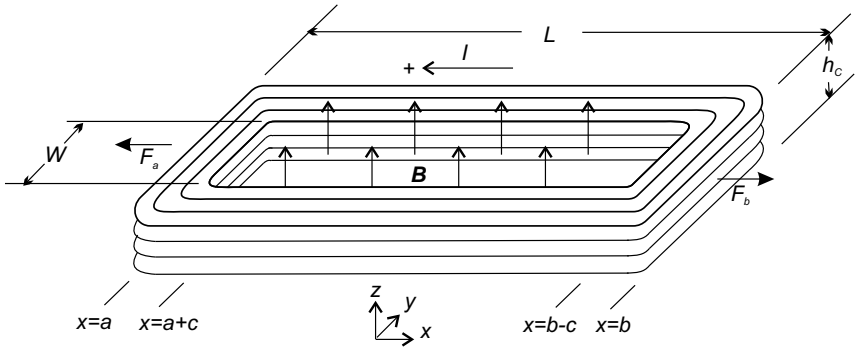


Figure 25.4: Rectangular coil normal to z-directed magnetic field. Positive current and coordinates in the direction of the arrows produce opposed forces F_a and F_b on the legs at $a < x < a + c$ and $b - c < x < b$. A variation of magnetic $\boldsymbol{B}$ field in the x direction produces a net x-directed force on the coil.

the x direction, parallel to the magnetic pole faces and perpendicular to the z-directed magnetic field between the poles. This arrangement is used for hard-drive head-positioning actuators but unlike the typical voice-coil orientation in a loudspeaker where the coil axis is parallel to the direction of motion and the pole faces.

The side legs of the coil (parallel to the direction of motion) do not interact with the magnetic field so they are presumed to lie outside the pole gap. All the magnetic field interaction is produced by the two legs normal to the direction of motion.

Sage calculates the coil volume Vcoil according to the presumed rectangular geometry of figure 25.4 assuming the coil side legs (parallel to x direction) lie just outside poles:

$$V_c = 2A_c(W + L - 2c) + \pi c^2 h_c \tag{25.55}$$

The number of turns in the coil (Nturns) remains an independent input, however there is a new output:

FillFac : (real, dimensionless) The fraction of coil length filled by the cross legs $2c/L$.

Cross leg x extent c is just the coil cross section area divided by the coil height A_c/h_c. The coil height h_c is a specified fraction of the parent magnetic gap (input ZthkRel). The Sage interface raises an exception whenever FillFac exceeds 1, indicating the coil cross legs overfill the available volume.

Magnetic Field The electrical current flowing through the coil produces a z-directed magnetic potential difference $\Delta\Psi_c$ according to Ampere's law (25.25) of section 25.4. But unlike a non-moving coil, the total potential difference $\Delta\Psi$ now includes the potential drop due to magnetic flux across the coil air gap $\Delta\Psi_a = -Bh_c/\mu_0$. So $\Delta\Psi = \Delta\Psi_c + \Delta\Psi_a$.

The potential difference also varies as a function of x. It has the full value of equation (25.25) only when x lies completely within the coil. Otherwise the magnetic potential $\oint \boldsymbol{H} \cdot d\boldsymbol{s}$ varies with the number of coil turns $n(x)$ enclosed by the integration path, which ranges from zero at the outer winding of the coil to N at the inner winding.

$$\Delta\Psi_c(x) = -n(x)I \tag{25.56}$$

The assumption is that $n(x)$ varies linearly so that $\Delta\Psi_c(x)$ also varies linearly from zero at the outer winding to the full value at the inner winding.

Induced Voltage The rate of change of magnetic flux $d\phi/dt$ linked through the coil core induces a voltage difference across the coil terminals according to Faraday's law (25.27) of section 25.4. Except now the total linked flux ϕ is the integral of a flux distribution $\int \phi_x dx$ over the distance x between windings. The linked flux varies from the extreme outer winding to the extreme inner winding. The average value determines the total coil voltage drop.

Magnetic Force Something new for a moving coil is the magnetic force between the coil and outer magnetic poles. Referring to figure 25.4 the net x-directed force on the coil is the difference between the forces F_a and F_b acting on the legs at $x = a$ and $x = b$. The force arises as the summation of Lorentz forces acting on individual charge carriers moving through the coil wires.

If the magnetic flux density $\boldsymbol{B}$ were uniform throughout the coil there would be no net force because the forces on opposite legs would exactly cancel. But if $\mathbf{B}$ varies in the x direction there is a net force in the x direction produced by the coil legs normal to the x direction. In general the differential force on any wire segment of length ds is

$$d\boldsymbol{F} = ds(\boldsymbol{I} \times \boldsymbol{B}) \tag{25.57}$$

where $\boldsymbol{I}$ is the current vector in the coil reference frame[2]. For those familiar with screw threads, the force direction is the direction of advance of a right-hand screw normal to the plane containing $\boldsymbol{I}$ and $\boldsymbol{B}$ rotating in the direction from $\boldsymbol{I}$ to $\boldsymbol{B}$. Under the assumption that $\boldsymbol{I}$ and $\boldsymbol{B}$ are always aligned with the coordinate directions, then only the magnitudes I and B matter with positive values corresponding to the directions drawn. With B a function of x and t

[2]In the coil reference frame the current flow $\boldsymbol{I}$ is the number of charges per unit length of wire drifting with velocity $\boldsymbol{u}_d$ in the wire direction, which is actually quite small for typical currents. What affect does the coil velocity have on the magnetic force? In the pole reference frame the charges are moving with velocity $\boldsymbol{u}_d$ plus the velocity of the moving wire $\boldsymbol{u}_c$. But in this frame both negative and positive charges are moving and the magnetic forces on the positive charges due to the $\boldsymbol{u}_c$ velocity component cancel the magnetic forces on the negative charges. Even if the wire is not exactly electrically neutral the net excess charge one way or the other is small compared to the total number of charge carriers. Besides, whatever net force due to $\boldsymbol{u}_c$ there may be is of no concern because it is always directed perpendicular to the direction of coil motion.

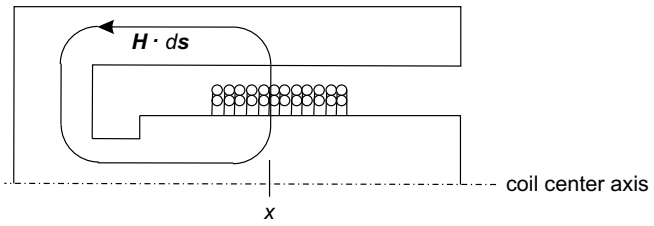


Figure 25.5: Left-directed voice coil cross section showing how the number of turns in the coil inducing a magnetic potential difference $\oint \boldsymbol{H} \cdot d\boldsymbol{s}$ varies with the position x where the path intersects the coil.

only the total x-directed force on a single winding of the coil crossing the gap at $a + x$ and $b - x$ is simply

$$F_w = I\left(\int_{b-x} Bds - \int_{a+x} Bds\right) = IW\left[B\right]_{a+x}^{b-x} = I\left[\phi_x\right]_{a+x}^{b-x} \tag{25.58}$$

where ϕ_x is just BW. Assuming there are N windings (turns) in the coil extending across the magnetic gap from a to $a + c$ on one end and $b - c$ to b on the other the total force on the coil is

$$F = NI\left(\langle\phi_x\rangle_{b-c}^{b} - \langle\phi_x\rangle_{a}^{a+c}\right) \tag{25.59}$$

where $\langle\phi_x\rangle_{x_1}^{x_2}$ denotes the average flux distribution between x_1 and x_2. This equation is fine but for reasons that will be clear later it is possible to write it in a form that does not directly involve the electrical current by replacing NI with the magnetic potential difference generated by the coil $-\Delta\Psi_c$. The result after also replacing ϕ_x with WB is

$$F = -W\Delta\Psi_c\left(\langle B\rangle_{b-c}^{b} - \langle B\rangle_{a}^{a+c}\right) \tag{25.60}$$

Voice Coils

TLcoilEMov

TRcoilEMov

These components represent circular coils moving in the direction of the coil axis in the typical arrangement used for loudspeaker voice coils. The lower pole corresponds to the inner cylindrical pole piece. The magnetic flux turns axially within the inner pole toward either left (negative x-direction) or right (positive x-direction) flux-path components to complete the external magnetic circuit. Hence there are two orientations of this component, depending on which side of the inner pole connects to the external magnetic path. Each exists only within a special magnetic gap parent, either a *left-turned* or *right-turned* magnetic gap.

Sage calculates the coil volume Vcoil assuming pole width W represents the coil centroid circumference and the coil thickness h_c is a specified fraction of the parent magnetic gap (ZthkRel).

$$V_c = L\frac{\pi}{4}\left[(W/\pi + h_c)^2 - (W/\pi - h_c)^2\right] \tag{25.61}$$

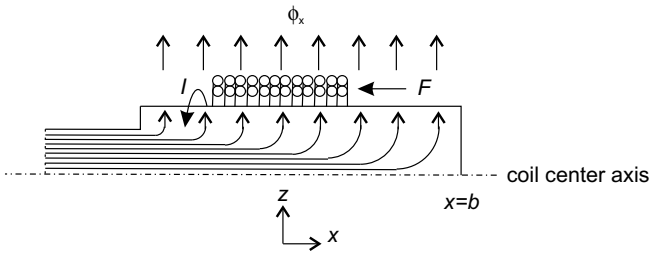


Figure 25.6: Left-directed voice coil cross section showing how the magnetic flux linked through the coil core varies with position due to the continuous upward diversion into the air gap. A positive z-directed magnetic flux distribution ϕ_x produces a negative x-directed force F on the coil for an electrical current I in the direction drawn.

The coil cross-section area A_c (normal to wires) is simply

$$A_c = Lh_c \tag{25.62}$$

Since the coil cross section is now specified by the external geometry the previous input Nturns (number of wire turns in the coil) is now a output. It is calculated according to the number of wires of individual cross section A_w that will fit into the available cross section A_c given the specified packing factor α.

$$N = \alpha \frac{A_c}{A_w} \tag{25.63}$$

Magnetic Field The electrical current flowing through a voice coil produces a magnetic potential difference per Ampere's law as discussed in section 25.4 except because the coil axis is along the x direction instead of the z direction there are differences, as illustrated for a *left-turning* inner pole in figure 25.5. The magnetic potential $\oint \boldsymbol{H} \cdot d\boldsymbol{s}$ varies with the number of coil turns $n(x)$ enclosed by the integration path, depending on the position x where the path intersects the coil. In the Sage solution the magnetic potential induced by the coil is presumed concentrated across the air gap between poles but it is not localized to the coil. The potential varies from zero a any pole point to the left of the coil to the full potential $\Delta\Psi = -NI$ at any point to the right of the coil. At points within the coil the potential varies linearly between the two. The case of a right-turning inner pole is similar except the integration path encloses the coil turns to the right of point x.

Induced Voltage The rate of change of magnetic flux $d\phi/dt$ passing through the coil core (the *linked* flux) still induces a voltage difference across the coil terminals according to Faraday's law (25.27) of section 25.4. But now the linked flux varies with position. Referring to figure 25.6 for the case of a left-turning inner pole piece, the linked flux at a point x in the coil is the total z directed

(radial) flux crossing the magnetic gap between x and the pole endpoint, or $\phi_L(x) = \int_x^b \phi_x$, where x is a point within the coil and b is the pole endpoint. The flux change that determines the voltage difference is $d\langle\phi_L\rangle/dt$ where $\langle\phi_L\rangle$ is the average linked flux over the coil length. The case of a right-turning inner pole piece is similar except the pole endpoint is reversed.

Magnetic Force Unlike a transverse coil, each turn of a voice coil creates an x directed force in the same direction. For a voice coil of circumference W moving through an external magnetic flux density $\boldsymbol{B}$ the differential force on a single turn of wire carrying current I is

$$d\boldsymbol{F} = W(\boldsymbol{I} \times \boldsymbol{B}) \tag{25.64}$$

Under the assumption that $\boldsymbol{I}$ and $\boldsymbol{B}$ are perpendicular and give rise to a force in the x direction, only the magnitudes I and B matter and the x-directed force on the i-th turn of wire for current in the direction shown in figure 25.6 is

$$F_i = -WIB \tag{25.65}$$

Noting that WB is just the flux distribution ϕ_x and that the number of wires per unit length is N/L, where N is the number of turns in the coil and L is the coil length, the differential force acting on a coil segment of length dx is

$$dF = -I\phi_x \frac{N}{L} dx \tag{25.66}$$

So the total force acting on the coil is

$$F = -\frac{NI}{L} \int_x \phi_x \tag{25.67}$$

25.6.7 Moving Magnetic Components

TPermMagEMov

Moving magnetic components implement the physics of their non-moving counterparts discussed in section 25.5 except resolve the magnetic flux and field in the x direction and override the Lpath and Apath inputs in terms of the parent magnetic gap dimensions. Fourier series outputs for magnetic potentials, fields and flux densities are omitted because they would not capture the variation of these quantities with x. For detailed information about such things you can refer directly to the solution grid.

For software design reasons the embedded objects representing magnetic components are subdivided into a parent *embedment* piece with variables common to all components and a child *object* piece with variables specific to the particular component. The graphical-interface icons for the embedment piece are all similar to icon shown in the margin for the case of a permanent magnet. The icons for the child object piece appear in the margins under the individual sections below describing particular components.

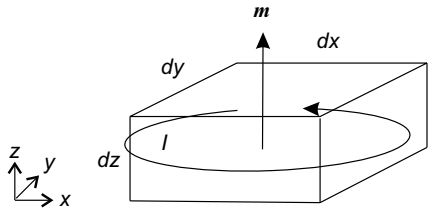


Figure 25.7: Magnetic domain represented by a single effective current loop I enclosing area $dx\,dy$ in a differential volume $dx\,dy\,dz$. The current induces a magnetic dipole moment $\boldsymbol{m}$ oriented in the z direction with magnitude $m = Idx\,dy$ and average volumetric density $-M$.

Air Gap

This component represents an empty gap that may be used as an axial spacer (parallel to magnetic gap) between other magnetic components.

TEmbdAirGap

Permanent Magnet

This component represents a rectangular magnet of dimensions $L \times W$ (length times width) moving between magnetic pole faces or a radially magnetized ring of magnets moving between cylindrical pole faces where W corresponds to the average of the inner and outer pole circumferences. The magnetization is presumed parallel to the z-axis (across magnetic gap).

TEmbdPerm-Mag

Fourier series output FUtilization representing magnet utilization factor *does* make sense for a moving magnet and is included. The same definition applies as for a stationary permanent magnet, namely the BH energy product along the demagnetization curve as a fraction of the maximum value. Except now the value is an x-spatial average.

B H Relationship The $B(H)$ functional relationship is the same as described in section 25.5 for a stationary magnet except based on an x-distribution of magnetic flux ϕ_x as described above.

Magnetic Force The total magnetic potential difference across a magnet with z thickness h_m may be broken into two terms

$$\Delta\Psi \equiv \Delta\Psi_M - h_m B/\mu_0 \tag{25.68}$$

in effect defining $\Delta\Psi_M$, the component of the potential difference driving the magnetization.[3] In terms of $\Delta\Psi_M$ and pole width W the x-component of force acting on the moving magnet is

$$F = -W \int_x \Delta\Psi_M \frac{\partial B}{\partial x} \tag{25.69}$$

[3] $B \equiv \mu_0(H + M) \Leftrightarrow B/\mu_0 \equiv M + H \Leftrightarrow h_m B/\mu_0 \equiv \Delta\Psi_M - \Delta\Psi$

This follows as a generalization of equation (25.60) derived above for transverse moving coils except applied the individual magnetic domains in a permanent magnet. For example, applied to the current loop I within the differential volume $dx\,dy\,dz$ illustrated in figure 25.7. Current I produces a magnetic potential difference $\Delta\Psi_I = -I$. So substituting $\Delta\Psi_I$ for $\Delta\Psi_c$, and dy for W in equation (25.60) gives

$$dF = -dy\,\Delta\Psi_I\,[B]_x^{x+dx} \tag{25.70}$$

Taking the limit as $dx \to 0$ gives

$$dF = -\Delta\Psi_I \frac{\partial B}{\partial x}\,dx\,dy \tag{25.71}$$

Integrating over the total magnet volume assuming uniform $\Delta\Psi_I$ and B in the y direction, the dy integral reduces to the integrand times W so that the total force is

$$F = -W \int_x \int_z \Delta\Psi_I \frac{\partial B}{\partial x} \tag{25.72}$$

Assuming B does not depend on z, $\frac{\partial B}{\partial x}$ factors out of the z integral and then $\int_z \Delta\Psi_I = \Delta\Psi_M$, resulting in equation (25.69). The same derivation also applies to the case when eddy currents are present in the magnet. Then $\Delta\Psi_M$ is the component of the magnetic potential difference driving both magnetization (magnetic domain currents) and eddy currents.

Force by Energy Conservation The force on a magnet can also be derived from energy considerations. Think of the total force on the magnet as the sum of a large number of force elements acting on the individual magnetic domains moving through the external magnetic field. Referring to figure 25.7 each domain has an associated magnetic dipole moment $\boldsymbol{m}_i$ (not necessarily in the z direction) with a volumetric moment density $-\boldsymbol{M}_i$ (i is the domain index). By the nature of permanent magnets, or any ferromagnetic material for that matter, the magnitude of each magnetic dipole remains constant and only the angle of alignment with an external magnetic field $\boldsymbol{H}$ changes.

Mechanical energy input $\boldsymbol{F_i} \cdot d\boldsymbol{x}$ resulting from force $\boldsymbol{F_i}$ acting on $\boldsymbol{M}_i$ increases the magnetic energy $\boldsymbol{H} \cdot d\boldsymbol{B}$ within the volume element, but only the part associated with the magnetization component of the magnetic field $-\boldsymbol{M}_i \cdot d\boldsymbol{B}$. More on this later but for now it follows that the x component of force acting on the magnetic domain is

$$F_i = -\boldsymbol{M}_i \cdot \frac{\partial \boldsymbol{B}}{\partial x} \tag{25.73}$$

Summing this over all the domains in the magnet amounts to an integral over the magnet volume. In the one-dimensional Sage formulation the **B** field is presumably directed along the z axis and varies only in the x direction. So in integrating $\boldsymbol{M} \cdot \frac{\partial \boldsymbol{B}}{\partial x}$ over any y-z plane, the vector $\frac{\partial \boldsymbol{B}}{\partial x}$ factors out leaving only the integral of $\boldsymbol{M}$ which may be replace by the integral of the average magnetization in the y-z plane. In the Sage formulation the average magnetization is

presumed oriented along the z axis with the scalar magnitude M, which may be replaced with the equivalent $\Delta\Psi_I/dz$. Likewise $\boldsymbol{B}$ can be replaced with its scalar magnitude B. So the integrand reduces to the same one above in equation (25.71) and likewise the total force reduces to the same equation (25.69).

Returning to the justification for only $-\boldsymbol{M}_i \cdot d\boldsymbol{B}$ balancing the mechanical energy change, equation (25.73) is equivalent to

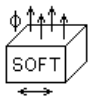
$$F_i = (\boldsymbol{H} - \boldsymbol{B}/\mu_0) \cdot \frac{\partial \boldsymbol{B}}{\partial x} \tag{25.74}$$

which follows from the definition $\boldsymbol{H} \equiv \boldsymbol{B}/\mu_0 - \boldsymbol{M}_i$. Arguing as before it follows that in the one-dimensional Sage formulation the x-component of force acting on the moving magnet may be written as

$$F = h_m W \int_x (H - B/\mu_0) \frac{\partial B}{\partial x} \tag{25.75}$$

In this form the equation reveals something about global energy conservation. In the total electromagnetic model it is necessary that $F dx$ energy changes balance $\Delta V I$ energy changes in electrical coils. For that to be the case the change in magnetic energy resulting from the two must be the same. Taking the case of an air-core solenoid for example, the change in magnetic energy in the core as a result of coil electrical work dE_c is $(H - B/\mu_0)dB$.[4] So the change in magnetic energy associated with mechanical energy input must be likewise.

Soft Ferromagnetic Material



TEmbdSoft-Mag

This component represents a rectangular piece of soft ferromagnetic material of dimensions $L \times W$ (length times width) moving between magnetic pole faces or a radially magnetized ring of material moving between cylindrical pole faces where W corresponds to the average of the inner and outer pole circumferences.

The magnetic energy and force relationships are the same as for a permanent magnet. The only difference is that the functional relationship between magnetic flux density B and magnetic field H is governed by the hysteresis formulation described in section 25.5 for a stationary soft ferromagnetic material except in terms of an x-distributed of magnetic flux ϕ_x.

25.6.8 Solution Method

The geometrical layout and solution method for the combination of a magnetic gap, moving magnetic container and electromagnetic components within are illustrated starting with figure 25.8, which shows a top-level side view (y-axis into page).

[4] This follows because the voltage drop induced in the coil is $\Delta V = NA\,dB/dt$, where N is the number of winding turns and A is the core section area. The core magnetic field is $H = NI/\ell - B/\mu_0$, where ℓ is the flux path length in the core. So $I = (H - B/\mu_o)\ell/N$ and the coil energy works out to $dE_c/dt = A\ell(H - B/\mu_0)dB/dt$

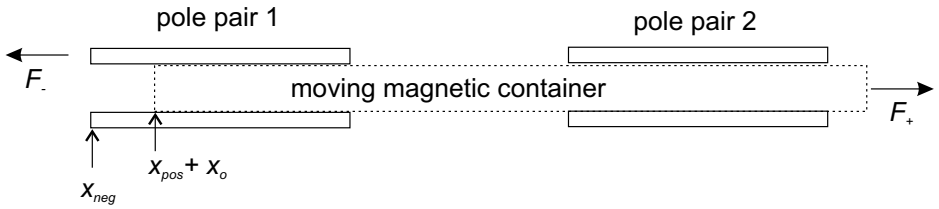


Figure 25.8: Side view of two-pole magnetic gap with moving magnetic container inside. The magnetic force between the poles and moving container produces equal and opposite forces F_+ and F_- transmitted to other moving components of the model, or vice-versa. The magnetic container is a member of the relative moving part family defining two position coordinates x_{neg} and x_{pos}. The left (negative) end of pole pair 1 is identified with x_{neg} and the left end of the moving container with $x_{pos} + x_o$ where x_o is a fixed offset.

The interesting magnetic physics begins with figure 25.9. The magnetic pole faces are presumed to have uniform magnetic potentials Ψ_{1+}, Ψ_{1-}, Ψ_{2+}, Ψ_{2-}. Magnetic flux is presumed to flow in the z direction between the pole faces, passing through the total air gap g_z or through an electromagnetic component of some thickness lying between the pole faces, in series with air gaps Δz on each side. A relatively small fringing magnetic flux also flows between the pole faces and points on the surface of the electromagnetic component lying outside the pole faces, in which case the flux passes through an additional effective air gap g_f as illustrated for the point x. Likewise for points lying to the left of pole pair 1 or to the right of pole pair 2. For such calculations the outer magnetic potential is presumed to be that of the nearest pole face, or in the case of the region between poles a smooth transition between the potentials of the pole faces on either side. Points on the faces of the electromagnetic component to the left of the mid-plane between the poles are assumed to transfer magnetic flux to pole pair 1 and those to the right to pole pair 2.

Fringe Field The effective air gap g_f for the fringe region beyond the pole faces is based on an approximation to the field solution for a magnetic dipole (formed by the pole faces) in vacuum (air). This is a reasonable assumption if something like a voice coil occupies the fringe region but not so reasonable if there is a ferromagnetic material or another pole face there. In those cases the model underestimates the fringing flux to some extent but it is hard to do better in a one-dimensional solution. For a magnetic dipole the magnetic potential variation in the z direction along the plane of symmetry equidistant from the two poles (plane midway between pole faces) varies with distance x from the pole, for $x \gg g_z$ as

$$\frac{\partial \Psi}{\partial z} \approx \frac{1}{8} \frac{g_z^3}{x^3} \left(\frac{\partial \Psi}{\partial z} \right)_{x=0} \tag{25.76}$$

where g_z is the pole separation distance. In terms of the Sage solution the effective pole separation $s = g_z + 2g_f$ in the fringe region that gives the same magnetic flux distribution for a given pole-to-pole potential difference, again for $x \gg g_z$, is

$$\frac{s}{g_z} \approx 8\frac{x^3}{g_z^3} \tag{25.77}$$

But s/g_z must approach unity for $x = 0$ and also a magnetic gap is not quite as simple as a magnetic dipole. In fact by similarity of governing equations the magnetic field between magnetic pole faces is similar to the electric field in a parallel plate capacitor. This is useful because fringe fields in capacitors has been well studied. So the approach to approximating flux fringing in Sage is to start with a formulation involving an unknown parameter a that has the correct x variation for $x \gg g_z$

$$\frac{s}{g_z} = 1 + a\frac{x^3}{g_z^3} \tag{25.78}$$

and then fit a so the resulting fringing flux is consistent with published capacitor data. The baseline formula for capacitance per unit length in terms of plate spacing g_z and width L (using Sage notation) is basically $C \propto L/g_z$. According to [42], the effective plate width increment L_f that must be added to the actual width to produce the actual capacitance using the baseline formula is accurately correlated by

$$\frac{L_f}{g_z} = \frac{1}{\pi}\left(1 + \ln\frac{2\pi L}{g_z}\right) \tag{25.79}$$

The width increment is only a week function of L/g_z so it is reasonable to assume a representative value $L/g_z = 10$ for present purposes, in which case the required length increment is

$$L_f = 1.636\, g_z \tag{25.80}$$

In other words the fringing flux amounts to a plate extension of $0.818\, g_z$ on each side. A spreadsheet numerical integral of fringing flux density (per side) based on equation (25.78) of the form

$$B = B_0\frac{1}{1 + a\frac{x^3}{g_z^3}} \tag{25.81}$$

produces the required total flux fringing flux for the value $a = 3.15$.

$$\int_0^\infty B = 0.818 B_0 g_z \tag{25.82}$$

The indefinite integral of fringing flux at any point x beyond the pole end is approximately correlated as

$$\int B\, d\xi = B_0 g_z \begin{cases} 1.13\xi - 0.443\xi^2 & \text{if } \xi \le 1 \\ 0.818 - \frac{1}{6.3(\xi+0.1)^2} & \text{if } \xi > 1 \end{cases} \tag{25.83}$$

where $\xi = x/g_z$

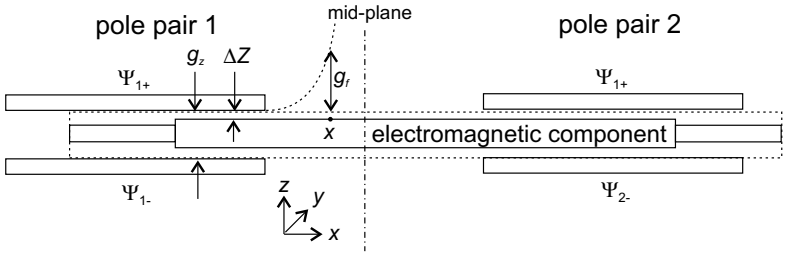


Figure 25.9: Two-pole magnetic gap showing inner electromagnetic components of different thicknesses, arranged in series to fill the container length.

Solution Logic Figure 25.10 depicts the solution variables and solution logic. At the outer level are lumped potentials Ψ_+ and Ψ_- evaluated from inner solution variables Ψ and $\Delta\Psi$ as $\Psi_+ = \Psi + \Delta\Psi/2$ and $\Psi_- = \Psi$ - $\Delta\Psi/2$. Ψ and $\Delta\Psi$ are solved much like the solution for an ordinary air gap. Potential offset Ψ is solved implicitly in terms of the total pole magnetic fluxes ϕ_+ and ϕ_- from the continuity condition

$$\phi_+ = \phi_- \tag{25.84}$$

Potential difference $\Delta\Psi$ is solved implicitly from the condition that the total magnetic flux ϕ (evaluated as $(\phi_+ + \phi_-)/2$) is the sum of the part through the air gap ϕ_{air} (to the left of the dotted line) plus the part through the embedded magnetic component ϕ_e.

$$\phi = \phi_{air} + \phi_e \tag{25.85}$$

The magnetic flux through the air gap is evaluated explicitly from the vacuum relationship

$$\phi_{air} = -\mu_0 \frac{\Delta\Psi}{g_z} A_{air} \tag{25.86}$$

Where $\mu_0 = 4\pi 1.0^{-7}$ (H/m) is the permeability of free space and A_{air} is the uncovered area of the pole (to the left of the dotted line). The embedded component magnetic flux ϕ_e is the integral of the magnetic flux distribution ϕ_x (evaluated as $(\phi_{x+} + \phi_{x-})/2$) for that part of the embedded component exchanging flux with the outer poles (the whole component for a single-pole gap or that part on one side or the other of the mid-pole position for a two pole gap).

Within the embedded magnetic component flux distribution ϕ_x is solved independently as a function of the outer potentials Ψ_+ and Ψ_-. The solution occurs in a spatial grid so as to capture the changing flux with position x while moving through the gap. The potential offset Ψ_e is solved implicitly at each point of the grid in terms of the magnetic flux distributions ϕ_{x+} and ϕ_{x-} (Wb/m) from the continuity condition

$$\phi_{x+} = \phi_{x-} \tag{25.87}$$

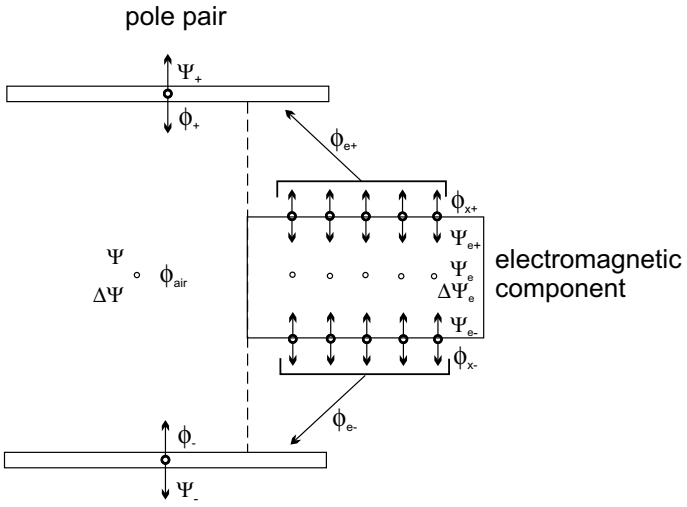


Figure 25.10: Solution schematic for electromagnetic component located at the negative end of the container between and offset from magnetic pole faces. The arrows indicate the direction of information flow, not magnetic flux direction. The gap between the poles is expanded for clarity.

At this point there is a twist in the solution logic required because there is no external connector object to enforce magnetic potential continuity. So instead of solving magnetic potential continuity from continuity of magnetic flux distribution it is the other way around. Flux distributions ϕ_{x+} and ϕ_{x-} are solved implicitly from the conditions that the outer potential difference equals the sum of the potential difference across the inner component plus that across the air gaps on either side, namely

$$\Psi_+ - \Psi_{e+} = \Delta\Psi_z \tag{25.88}$$

and

$$\Psi_- - \Psi_{e-} = -\Delta\Psi_z \tag{25.89}$$

where $\Delta\Psi_z$ is the potential difference for flux ϕ_x flowing across the air gap Δz (or $\Delta z + \Delta x$ for points outside the pole boundaries). In the above equations Ψ_+ and Ψ_- come from the previous air-gap solution so they may be taken as given and Ψ_{e+} is evaluated as $\Psi_e + \Delta\Psi_e/2$ and Ψ_{e-} as Ψ_e - $\Delta\Psi_e/2$. Potential difference $\Delta\Psi_e$ comes from the internal physics of the electromagnetic component as a function of flux distribution ϕ_x.

Energy Leakage Normally the principle of global energy conservation is built into the numerical differencing methods used for Sage's model components. The energy flowing out of one computational cell is exactly balanced by the energy entering another. But this is not so simple for electromagnetic components

moving between fixed poles. The grid of the embedded magnetic component moves relative to the poles and the boundaries of the computational cells do not always line up with the pole endpoints. The magnetic flux from the poles is apportioned into the moving grid so as to conserve magnetic flux but the magnetic potential across the grid computational cells and the magnetic force calculation are not guaranteed 100% accurate. In effect there may be a small amount of energy leakage between the outer poles and the embedded moving component.

Prior to v12.3 Sage repaired this energy leak by scaling the value of the magnetic force acting on the components inside the moving container so that the time-average electro-magnetic and mechanical power dissipated in the moving container equaled the magnetic energy flow into the pole gaps. This approach worked reasonably well, although it sometimes did not make sense — e.g. when magnetic force was uniformly zero or produced negligible power because it was in phase with displacement or the displacement was very small. Sometimes it produced slow or non-converging solutions, or scale factors far different from 1.

Starting with v12.3 Sage abandoned scaling the magnetic force as a means of producing energy conservation and replaced the scale-factor output Fscale of moving electromagnetic containers with EfluxErr, an output that just displays the discrepancy between the power flow into the pole gaps and the power dissipated in the moving container.

25.7 Magnetic Material Physics

In a vacuum the magnetic flux displacement resulting from a magnetic field H is simply

$$B \equiv \mu_0 H \tag{25.90}$$

where $\mu_0 \approx 1.257\text{E-}6$ (H/m) is the permeability of free space. In ferromagnetic materials, including permanent magnets, magnetic domains align in response to a magnetic field, increasing the magnetic flux displacement. The domain alignment is characterized by the so-called magnetization Mdefined by

$$B \equiv \mu_0 (H + M) \tag{25.91}$$

The relative permeability μ_r is the factor by which the vacuum permeability must be multiplied to account for the magnetization. It is defined by

$$\mu_r \equiv \frac{H + M}{H} \tag{25.92}$$

Domain alignment may also be characterized in terms of the magnetic polarization J defined by

$$B \equiv \mu_0 H + J \tag{25.93}$$

In this form it is clear that the residual magnetic flux B_r at $H = 0$ is the same as the residual magnetic polarization J_r.

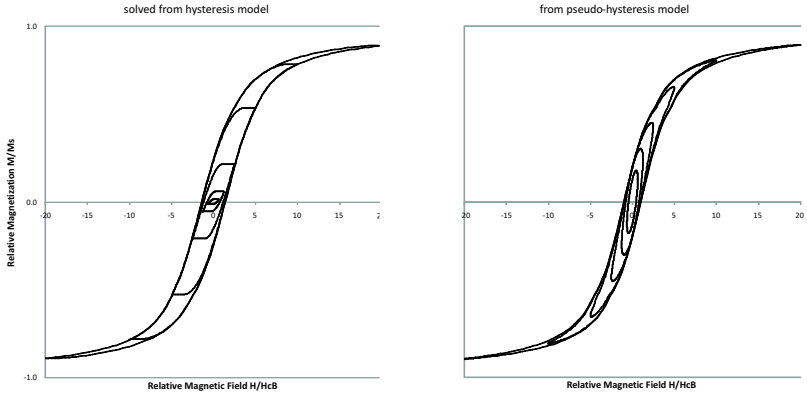


Figure 25.11: Excel spreadsheet solutions of differential equation (25.94) for H cycling back and forth between positive and negative limits over six different ranges, decreasing in powers of two. Compared to curves from pseudo-hysteresis equation (25.40) on the right. The curves calculated from the differential equation overestimate coercive force H_c slightly. Parameter $\alpha = 1.49\text{E-}4$, based on properties for non-oriented silicon steel (2.9%): $\mu_r = 8000$, $M_s = 1.59\text{E}6$ A/m, $H_c = 40$ A/m.

25.7.1 Soft Ferromagnetic Materials

Soft ferromagnetic materials are characterized by high relative magnetic permeability μ_r and relatively low coercive force (H at $M = 0$ intercept) compared to a permanent magnet. In typical usage the magnetization alternates from near saturation in one direction to near saturation in the other so that magnetic hysteresis losses may be significant.

Hysteresis Model

Jiles and Atherton [33] in 1986 laid out the mathematical underpinnings of magnetic hysteresis in the form of a differential equation. The primary variable to be determined is the magnetization M in the defining relationship $B \equiv \mu_0(H + M)$. In the Jiles and Atherton formulation M is the solution of a differential equation. The underlying differential equation for M is:

$$\frac{dM}{dH} = \begin{cases} \frac{(M_a - M) \vee 0}{H_c} & \text{if } dH > 0 \\ \frac{(M - M_a) \vee 0}{H_c} & \text{if } dH < 0 \end{cases} \tag{25.94}$$

The $\vee$ operator on the right side is the *max* operator (i.e. $\Delta M \vee 0$ means the larger of ΔM and 0). $M_a(H)$ is the so-called *anhysteretic* magnetization — the average of the upper and lower branches of the actual hysteresis curve shown in figure (25.11). H_c is the coercive force (point where the lower branch of the hysteresis curve crosses the H axis).

Compared to the formulation used in Jiles and Atherton the above equation omits a relatively small term $\alpha(M_a - M)$ in the denominator on the right side. The idea is that the solution $M(H)$ approaches asymptotically the anhysteretic magnetization $M_a(H)$ for large negative and positive values of H but typically falls somewhat above or below M_a depending on the direction H is changing. The purpose of the $\vee$ operator is to avoid a non-physical behavior in the M solution when dH changes sign. For example just before a maximum-H reversal point where dH goes from positive to negative the M value is always slightly below M_a. So just after flow reversal $M - M_a$ is negative and without the $\vee$ operator M would continuing to increase ($(M - M_a)dH > 0$). With the $\vee$ operator the M value instead remains constant until H decrease to the point where $M_a(H)$ again drops below M (or dH reverse direction again).

In solving the above differential equation there is a complication because the anhysteretic magnetization is defined in terms of an effective magnetic field H_e related to magnetic domain coupling. This effective magnetic field is

$$H_e = H + \alpha M \tag{25.95}$$

where α is the inter-domain coupling coefficient. The anhysteretic magnetization is taken to be a function that varies smoothly between saturation values $-M_s$ and M_s:

$$M_a(x) = M_s(\coth(x) - 1/x) \tag{25.96}$$

The independent variable is the dimensionless effective magnetic field

$$x \equiv \frac{H_e}{\alpha M_s} \tag{25.97}$$

This differs from the Jiles & Atherton formulation which defines x as H_e divided by another parameter. But it will turn out that the above formulation suffices for the M solution to have the correct slope near $M = 0$, which is good enough for Sage.

To speed up computation Sage approximates the function $\coth(x) - 1/x$ on the right side by the function

$$f(x) = \begin{cases} \operatorname{sign}(x) - \frac{1}{x} & \text{if } |x| > 3 \\ \frac{x}{3}\left[1 - \frac{x^2}{18}\left(1 - \frac{x^2}{27}\right)\right] & \text{otherwise} \end{cases} \tag{25.98}$$

The two branches of the approximation are continuous and smooth at $x = \pm 3$. Both original and approximate functions are shown in figure 25.12.

M intercept H_c Referring to the lower branch of the major hysteresis loop in figure 25.11), the M intercept of the magnetization solution $M(H)$ starting from a large negative H is H_c. This is already built into the governing differential equation (25.94). To see this it is helpful to simplify the $dH > 0$ branch of that equation to

$$\frac{dM}{dH} = \frac{M_a - M}{H_c} \tag{25.99}$$

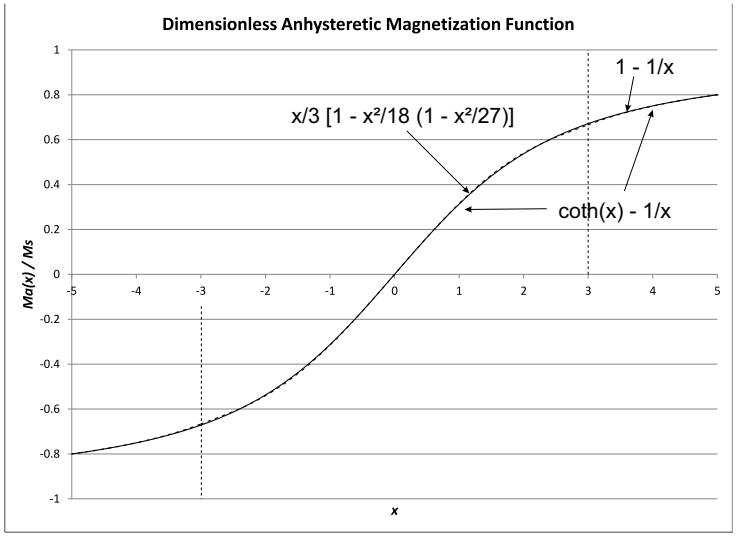


Figure 25.12: Dimensionless anhysteretic magnetization function (25.96) and piecewise approximation (25.98).

So the slope of M at H_c is $(M_a(H_c) - M(H_c))/H_c$. On the other hand the slope of the anhysteretic magnetization curve M_a (curve midway between upper and lower branches of the major hysteresis loop) at H_c is approximately $M_a(H_c)/H_c$ (since $M_a(0) = 0$). For the two slopes to be equal requires $M(H_c) = 0$. In other words that the M intercept is H_c. This conclusion applies to all useful soft ferromagnetic materials where the curves $M(H)$ and $M_a(H)$ are nearly linear and parallel in the range $(-H_c, H_c)$.

Determining α So the slope of M and M_a are the same near $M = 0$ but what determines the steepness? Parameter α does. To see this start with the definition of magnetic susceptibility $\kappa = \mu_r - 1$. It follows from equation (25.91) and the definition of relative permeability $B = \mu_o \mu_r H$ that

$$\kappa = dM_a/dH \tag{25.100}$$

Using the above approximation (25.98) (namely, $f(x) \approx x/3$ near $x = 0$), it follows that

$$\kappa = \frac{dM_a}{dH}_{H_e=0} = \frac{d}{dH}\left(\frac{1}{3\alpha}(H + \alpha M)\right) = \frac{1}{3\alpha}\left(1 + \alpha \frac{dM}{dH}\right) \tag{25.101}$$

But since the M and M_a are close it is reasonable to approximate dM/dH on the right side with $dM_a/dH = \kappa$, obtaining

$$\kappa = \frac{1}{3\alpha}(1 + \alpha\kappa) \tag{25.102}$$

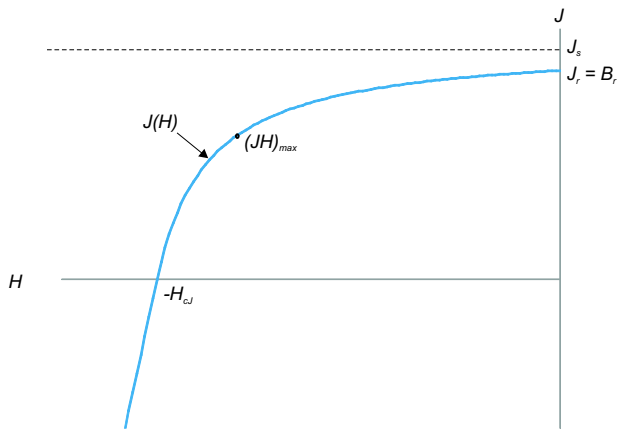


Figure 25.13: The approximate demagnetization curve given by equations (25.107) and (25.108) determined by parameters J_r, H_{cJ} and K_{jh}.

or solving for α:

$$\alpha = \frac{1}{2\kappa} \tag{25.103}$$

In other words, this is the value of inter-domain coupling coefficient α in the definition of effective magnetic field (25.95) required for the solution to have the correct magnetic susceptibility κ.

At this point parameter α is specified in terms of bulk material properties and the solution $M(H)$ to differential equation (25.94) can in principal be calculated. Except under the coarse time grid employed in Sage and the nonlinearity of $M(H)$ coupled with possible non-sinusoidal variations in H produce convergence issues and numerical artifacts in the solution. So the Sage formulation uses the pseudo-hysteresis approximation discussed in section 25.5.8.

25.7.2 Permanent Magnets

Permanent magnets are characterized by high coercive forces and are supplied in a magnetized state. Unlike soft ferromagnetic materials they are never, by design, subject to a high enough demagnetizing field to change the magnetization. Rather the magnetization varies only slightly and nearly reversibly so that hysteresis losses can be neglected. For practical purposes they are defined according to a fixed demagnetization curve $B(H)$ that can be formulated as follows.

Permanent magnets like other ferromagnetic materials have a saturation magnetic polarization J_s where all the magnetic domains align with the external magnetic field. As the external magnetic field varies from a positive high value to a negative high value the magnetic polarization follows the upper branch of a hysteresis loop which always looks something like the approximation shown in figure (25.13). The saturation and residual magnetic polarizations J_s and J_r

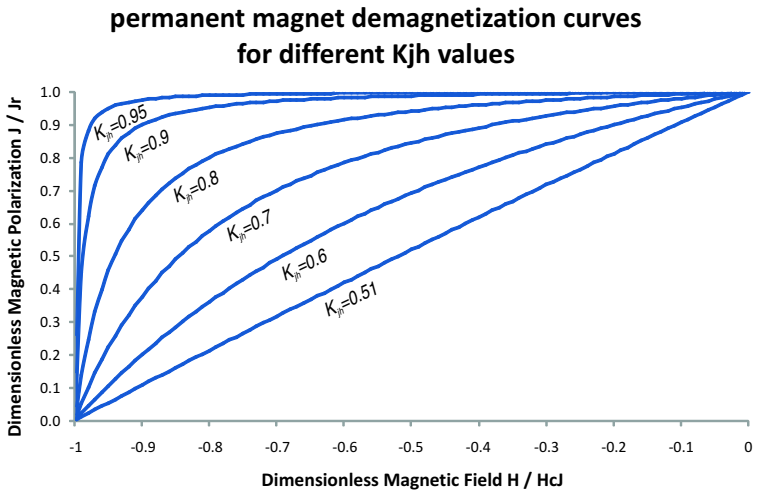


Figure 25.14: Permanent magnet demagnetization curves for different values of shape parameter K_{jh}. Curves generated using equation (25.107).

and the coercive force H_{cJ} are all labeled on the curve. The exact shape of the demagnetization curve is not important so Sage approximates it in a simple but general way. Sage formulates the part of the hysteresis curve for $J > 0$ as the simple hyperbola $J \propto -1/H$ with the origin displaced to the second quadrant. The part for $J < 0$ is a linear continuation of that function, which is physically incorrect for $H \ll H_{cJ}$ but moot since permanent magnet are demagnetized if they operate below H_{cJ}. In Sage, the linear continuation facilitates the implicit solution of H.

The shape of this function in the second quadrant (upper half of figure 25.13) is determined by how close J_r is to J_s. If J_r is near the asymptotic value J_s then there will be a pronounced knee in the curve. If J_r is small compared to J_s then the curve will be approximately linear. The $(JH)_{max}$ point (where $JdH + HdJ = 0$) occurs at the knee of the curve so the shape of the curve is also determined by the $(JH)_{max}$ value compared to the product $J_r H_{cJ}$. It is convenient to define the dimensionless shape factor K_{jh} by

$$K_{jh}^2 \equiv \frac{(JH)_{max}}{J_r H_{cJ}} \tag{25.104}$$

For a nearly linear J(H) demagnetization curve the $(JH)_{max}$ point falls near the point $(-H_{cJ}/2, J_r/2)$, so the value of K_{jh} is close to $1/2$. For a curve with a pronounced knee the $(JH)_{max}$ point falls near $(-H_{cJ}, J_r)$, so the value of K_{jh} is close to 1.

It is straight forward but a bit tedious to work out the approximate demagnetization function in terms of B_r, H_{cB} and K. It is useful to define a

constant

$$C \equiv \frac{1/K + 1}{1/K^2 - 1} \tag{25.105}$$

and then

$$D \equiv \frac{C}{C^2 - 1} \tag{25.106}$$

in terms of which the approximate demagnetization function turns out to be

$$J(H) = B_r \left[CD - \frac{D^2}{CD + \frac{H}{H_{cJ}}} \right] \text{ if } H \geq -H_{cJ} \tag{25.107}$$

For $H < H_{cJ}$ the demagnetization function is just the linear continuation

$$J(H) = B_r C^2 \left[\frac{H}{H_{cJ}} + CD - \frac{D}{C} \right] \tag{25.108}$$

The demagnetization curves resulting from different values of K are shown in figure 25.14.

Maximum Energy Product

Permanent magnets are often characterized by their so-called maximum energy product BH along the demagnetization curve. Although H is negative along the demagnetization curve and B positive so it is really $-BH$ that has a maximum.

What is the significance of the maximum energy product? Given a simple magnetic circuit consisting of a permanent magnet of length L_m in series with a linear magnetic material ($B = \mu H$) of length L_g and cross section A_g, the magnet energy product $-BH$ is proportional to the total energy stored in the magnetic material. This follows because the total energy stored in the magnetic material is $1/2 B_g H_g L_g A_g$. But B and H in the two materials are related by $B_m A_m = B_g A_g$ and $-H_m L_m = H_g L_g$, so that the total energy stored in the magnetic material is $-1/2 B_m H_m L_m A_m$ — in other words, proportional to the magnet BH product and the magnet volume.

In a magnetic circuit application the actual magnet $-BH$ product as a fraction of the maximum value can be viewed as a *magnet utilization factor*. In Sage permanent magnet components it is represented by the output FUtilization where the maximum value $(BH)_{max}$ is evaluated numerically for each permanent magnet instance because it is complicated to find an analytic expression in terms of the above magnetization formulation ($BH \equiv JH + \mu_0 H^2$). If the utilization factor is low then the permanent magnet may not being used to its full potential. It may be low because H is small (magnet driving flux through the circuit with minimal effort) in which case you might get by with a smaller magnet. Or it may be low because B is small (magnet too weak to drive flux through the circuit) in which case you may need a larger magnet. Because B and H vary with time in most applications the magnet utilization factor also varies with time so it is output as a Fourier series. Generally speaking the mean value and harmonic amplitudes should not all be small.

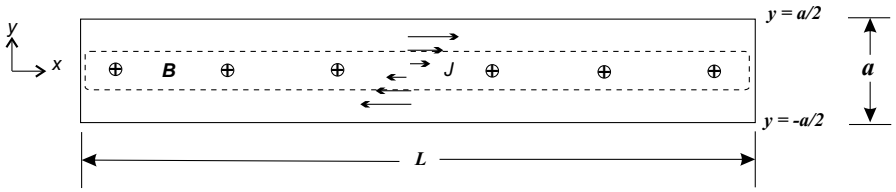
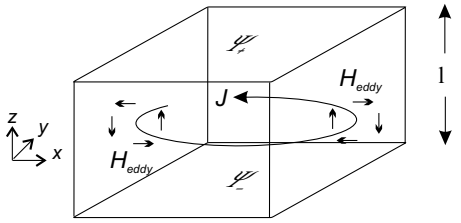


Figure 25.15: Cross section of a single lamination of magnetic material showing eddy current distribution J resulting from changing magnetic flux $d\boldsymbol{B}/dt$ directed into the page and uniform within the lamination. A typical current *streamline* follows the dotted line, which is somewhat idealized near the endpoints.

25.7.3 Eddy Currents

Magnetic materials are usually also electrical conductors and a changing magnetic flux induces an electrical current flow just like in a wire coil. Except the current flow is in the form of a current distribution within the bulk material. The current flows along the electric field induced by the changing magnetic flux in the form of simply-connected closed loops as depicted by loop $\boldsymbol{J}$ in this sketch. (*note*: The symbol J denotes current density here but is also used to denote the magnetic polarization elsewhere.)



But there are no net voltage differences within the material — the electric field produces a voltage gradient along the field direction that is canceled exactly by an opposite voltage gradient due to electrical resistance to the current flow. All other things being equal, materials with low resistivity tend to have large eddy currents in order to create sufficient large opposing voltage gradients. Except the eddy currents themselves produce magnetic fields (H_{eddy} of illustration) that oppose the changing magnetic flux so materials with low resistivity reduce the changing magnetic flux somewhat. In the case of a superconductor the eddy currents are always exactly enough to completely cancel any changing magnetic flux. So you would not want to use a superconducting material in an AC magnetic circuit.

To minimize electrical dissipation, the eddy currents are by designed confined in practical magnetic circuits to thin layers within *laminations*. The lamination layers are always parallel to the magnetic field direction, which is the z direction in Sage. Figure 25.15 shows a typical lamination with the z direction into the

page. The figure depicts the current distribution $J(y)$ flowing in any $x-y$ plane of the material. It is uniform in the x direction and symmetrical about the x axis. This is reasonable (e.g. satisfies solution boundary conditions) except for small regions near the x endpoints where the current reverses direction. The analysis below ignores these end effects.

Consider the current path indicated by the dotted line in figure 25.15. From Faraday's law the total change of magnetic flux enclosed by the current path ($d/dt \int BdA$) induces a voltage drop ΔV_L equal to the negative of the line-integral $\oint \boldsymbol{E} \cdot d\boldsymbol{s}$ along the path. Presuming B is uniform within the enclosed area of approximately $2yL$ the total inductive voltage change around the loop is

$$\Delta V_L = 2L\frac{dB}{dt}y \tag{25.109}$$

Applying Ohm's law to the same current loop and approximating the path length as $2L$ the total resistive voltage drop is

$$\Delta V_R = -2L\sigma J(y) \tag{25.110}$$

where σ is the electrical resistivity. Adding these two equations and using the fact that $\Delta V_L + \Delta V_R = 0$ in a closed loop, the result is a simple formulation for the current density in a lamination

$$J(y) = \frac{1}{\sigma}\frac{dB}{dt}y \tag{25.111}$$

The eddy current distribution produces a magnetic field that opposes the external magnetic field. Looking at the current streamlines as windings in a coil the field within the region between $(-y, y)$ results from the current streamlines outside that region, according to Ampere's law. The magnetic field varies from zero at the outer surface to maximum along the x axis. For any $y > 0$ the magnetic field is

$$H_J(y) = \int_y^{a/2} J(y)dy \tag{25.112}$$

Substituting the right side of equation (25.111) for $J(y)$ gives

$$H_J(y) = \frac{1}{\sigma}\frac{dB}{dt}\left(\frac{a^2}{4} - y^2\right) \tag{25.113}$$

Omitting some details the average magnetic field over the range $0, a/2$ is

$$H_{eddy} = \frac{a^2}{12}\frac{1}{\sigma}\frac{dB}{dt} \tag{25.114}$$

In the superconductor limit $\sigma \to 0$ the two previous equations are singular unless $dB/dt = 0$. That is interesting but Sage does not deal with that possibility and would produce a divide-by-zero error. The net magnetic field H that determines

the magnetic flux and magnetization usually associated with ferromagnetic materials (e.g. $B = \mu_0(H + M)$) is defined in terms of the external magnetic field H_{ext} and the mean eddy field as

$$H = H_{ext} - H_{eddy} \tag{25.115}$$

The electrical dissipation produced by the eddy current distribution can be calculated in one of two ways. First by analogy with a bundle of wires. Imagine the eddy currents flow in rectangular wire elements of length L and area ℓdy, where ℓ is the z thickness of the magnetic material. The loss in such a wire element is

$$dW_{eddy} = I^2 R = (J\ell dy)^2 \left(\sigma \frac{L}{\ell dy}\right) = \sigma L \ell J^2 dy \tag{25.116}$$

Substituting the right side of equation (25.111) for J and integrating over the lamination limits $(-a/2, a/2)$ gives the total eddy current instantaneous power dissipation

$$W_{eddy} = \frac{1}{12}\frac{a^3 L\ell}{\sigma}\left(\frac{dB}{dt}\right)^2 \tag{25.117}$$

or power dissipation per unit volume

$$w_{eddy} = \frac{1}{12}\frac{a^2}{\sigma}\left(\frac{dB}{dt}\right)^2 \tag{25.118}$$

Another way to calculate the magnetic power dissipation per unit volume is directly from the rate of change of magnetic energy per unit volume

$$w_{eddy} = H_{eddy}\frac{dB}{dt} \tag{25.119}$$

According to equation (25.114) both give the same result. The important thing to note is that the loss scales with the square of the lamination thickness a and inversely with the resistivity σ.

Ring Configurations

The cross section of a magnetic circuit perpendicular to the flux direction is often a circular ring. There are three cases of interest, a split ring, a closed ring with zero magnetic flux inside the ring and a closed ring with an equal but opposite magnetic return flux on the inside. In principle, such magnetic circuits can also be subdivided into laminations consisting of nested rings. For purposes of Sage modeling it is necessary to figure out how to model the eddy current losses in these rings in terms of equivalent rectangular laminations.

Case 1 split ring This case is illustrated in figure 25.16. Neglecting curvature effects ($a \ll R_m$) this is just the result of bending a rectangular lamination into a circular shape. The same analysis applies as before and the power dissipation per unit volume in terms of the ring (lamination) thickness a is again given by equation (25.118).

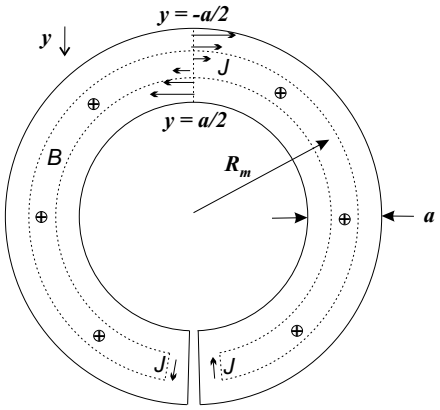


Figure 25.16: Cross section of a single lamination of magnetic material bent into a split ring shape of mean radius R_m. Neglecting curvature effects ($a \ll R_m$), the eddy current distribution J is the same as figure 25.15 for a rectangular lamination.

Case 2 closed ring, no magnetic flux inside This case is illustrated in figure 25.17. Compared to figure 25.16 the upward and downward currents at the split vanish and the current flows in concentric rings. The assumption that there is no magnetic flux inside the ring means that the average current enclosing any point inside the ring is zero so that the current flows in one direction above the mean radius R_m and the other direction below. So the current distribution is the same as the split ring case.

Case 3 closed ring, return magnetic flux inside This case is illustrated in figure 25.18. The total magnetic flux in the core is presumed equal and opposite that in the ring ($\int_{core} B_{core} = -\int_{ring} B_{ring}$). Compared to case 2 the current distribution must be shifted. A current path at the ring inner diameter now incloses the maximum magnetic flux, with the enclosed magnetic flux diminishing for progressively larger current paths to zero at the outer diameter. In terms of the local coordinate y of figure 25.18, the current distribution is zero at the outer diameter ($y = 0$) and increases linearly (ignoring curvature effects) to a maximum value at the inner diameter ($y = a$).

The equations for induced and resistive voltage drops are similar to equations (25.109) and (25.110). The section area between the outer diameter and the current path at y is approximately $2\pi R(y)y$, where $R(y) = 2\pi(R_m + a/2 - y)$ is the current path circumference. So the total inductive voltage change around the current path is

$$\Delta V_L \approx 2\pi R(y)\frac{dB}{dt}y \tag{25.120}$$

Applying Ohm's law to the same current path, the total resistive voltage drop is

$$\Delta V_R = -2\pi R(y)\sigma J(y) \tag{25.121}$$

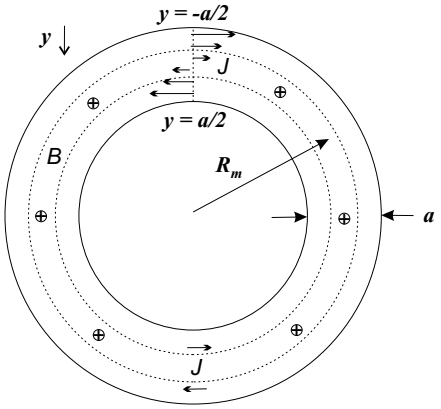


Figure 25.17: Cross section of a single lamination of magnetic material bent into a closed ring shape with no magnetic flux inside the ring. The eddy current distribution J is identical to figure 25.16.

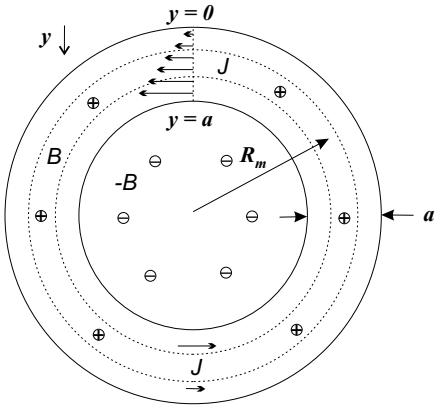


Figure 25.18: Cross section of a single lamination of magnetic material bent into a closed ring shape with an opposite return magnetic flux inside the ring. As a consequence the eddy current distribution J shifts compared to figure 25.17, varying approximately linearly from 0 at the outer diameter to maximum at the inner diameter.

Solving $\Delta V_L + \Delta V_R = 0$ leads to the same eddy current density as equation (25.111). The eddy current loss is also similar to the rectangular case except for *wire* element length L replaced by the circular path circumference $2\pi R(y)$ and integrating dW_{eddy} from $y = 0$ to $y = a$. Approximating $2\pi R(y)$ by the mean value $2\pi R_m$, gives the total eddy current power dissipation

$$W_{eddy} = \frac{1}{3}\frac{a^3 2\pi R_m \ell}{\sigma}\left(\frac{dB}{dt}\right)^2 \tag{25.122}$$

or power dissipation per unit volume

$$w_{eddy} = \frac{1}{3}\frac{a^2}{\sigma}\left(\frac{dB}{dt}\right)^2 \tag{25.123}$$

This is four times larger than cases 1 and 2, or the same as the loss for a rectangular lamination with thickness $2a$. In other words, same as a rectangular lamination with thickness twice the ring thickness.

Chapter 26

Radiation Exchange Components

The components in this chapter model thermal radiation exchange between the walls of radiation enclosures and objects inside. There are two types of radiation surface components — one for uniform temperature surfaces and one for distributed temperature surfaces. Then there are several components that represent the various possible view configurations between the surfaces — the fraction of radiation emitted from one surface reaching the other. Radiation between two surface components is modeled by connecting them to a common view configuration component. Each radiation surface is connected this way to all the other radiation surfaces in the enclosure with which it exchanges radiation — one view configuration component for each surface-to-surface view.

26.1 Sample Radiation Enclosures

Before getting into the details of specific radiation exchange model components you might want to take a look at some representative examples of radiation enclosure submodels that you can cut and paste into your own models and revise and expand accordingly. The examples are found in the Apps\SCFusion\Samples\-RadiationEnclosures sub-directory under the installation directory. Each Sage model file has a companion document in pdf format. There are examples for radiation between the ends of a cylindrical can, radiation shields (nested-spherical and stacked-disk) and radiation from a cold finger to a surrounding cylindrical enclosure.

You can use the copy-and-paste function of Sage's Edit menu (or the tool buttons) to copy model components from the sample files into your data file. You can only copy single, disconnected, model components this way (although any child components are copied too). After you copy a model component you have to first open the model you wish to paste it into. It is not possible to copy and paste between two separate Sage applications using the Windows clipboard.

26.2 Radiation Surface Components

Radiation surface components define emissivity and surface area without specifying anything about the geometrical shape of the surface. Inputs and outputs common to all radiation surface components are:

A : (real, m^2) Surface area emitting or absorbing radiation.

Emiss : (real, dimensionless) Gray-body emissivity ϵ. A number between 0 and 1. $\epsilon = 1$ for an ideal black body. $\epsilon \approx 0$ for a silvered surface. The surface is assumed to emit and reflect radiation diffusely with reflectivity equal to emissivity.

Fself : (real, W) Self view factor (F_{ii}, where i is the surface index) implied by all the radiation connections of the model. A diagnostic output. A positive value is normal for concave surfaces (e.g. the outer of two concentric spherical surfaces) where one part of the surface can *see* another part. A positive value for a flat or convex surface implies that not all the radiation leaving that surface is arriving at other surfaces of your model. That may be acceptable if Fself is small or the neglected radiation is going to surfaces at the same temperature where radiation exchange is irrelevant. A negative value implies some of the radiation leaving the surface is accounted for twice. That may be acceptable if Fself is small.

Br : (real, W/m^2) Surface radiosity B. The outgoing radiation intensity for combined emitted and reflected radiation — reflected from all other connected radiation surface components.

Rad : (real, W) Incoming radiation heat load as a result of radiation exchange with other radiation surface components.

Radiation surface components are designed to solve the surface temperature from an energy balance principle. But there are times when you will want to anchor the temperature of a radiation surface at some value. You can do this by connecting the radiation surface to another thermal solid in your model with a thermal conduction connection. In order to inherit the ability to make thermal conduction connections radiation surface components descend from the heat source components of chapter 17.

26.2.1 Lumped Radiation Surface

TStdyRsurf

Lumped radiation surfaces assume the entire surface temperature is uniform at a single temperature and define these additional outputs:

Ts : (real, K) Lumped surface temperature T.

QNeg : (real, W) Net conductive heat flow through negative-facing heat-flow connections.

QPos : (real, W) Net conductive heat flow through positive-facing heat-flow connections.

These variables are inherited from the point heat source component of chapter 17 except that Ts is overridden as a solved variable instead of an input. Temperature is solved so that the net energy flow from combined radiation and conduction is zero.

Thermal Attachments

Lumped radiation surface components include attachments for positive-facing and negative facing thermal radiation flows as well as attachments for conductive heat flows:

Icon	Purpose
	steady negative-facing radiation flow
	steady positive-facing radiation flow
	steady negative-facing conductive heat flow
	steady positive-facing conductive heat-flow

Create as many as you need. Then move the connection arrows up one level and connect them to the view-configuration components or temperature sources in your model.

Positive and negative facing radiation flows are understood as attached to the *same* radiation surface rather than a the opposite sides of a wall. Both view directions are available to facilitate the multiple connections required in a typical radiation enclosure.

26.2.2 Distributed Radiation Surface

TGxRsurf

Distributed radiation surfaces assume the surface has a temperature distribution along its length. The length of the surface comes from the parent component under which a distributed temperature radiation surface always resides. The parent component may be a canister, duct, matrix annular gap, generic cylinder or parallel container. Use of this component is illustrated in the example model RadColdFinger.ltc.

Distributed temperature radiation surfaces employ a simplified model of radiation heat transfer that neglects the variation in view factors along the surface. This is generally only appropriate when the surface length (in the direction of the temperature distribution) is small compared to the distance separating the subdivided surface from other surfaces of the radiation enclosure. (*see* section

26.4.3) for more details. Distributed temperature radiation surfaces define these additional outputs:

Te : (real, K) Effective surface temperature T.

QyNeg : (real, W) Net conductive heat flow through negative-facing heat-flow connections.

QyPos : (real, W) Net conductive heat flow through positive-facing heat-flow connections.

QyNeg and QyPos are inherited from the line heat source component of chapter 17. T_e is new. It is the effective radiation temperature defined by equation (26.30) below that gives the same radiation heat transfer as the distributed surface temperature.

Thermal Attachments

Distributed radiation surface components include attachments for positive-facing and negative facing thermal radiation flows as well as attachments for conductive heat flows:

Icon	Purpose
	steady negative-facing radiation flow
	steady positive-facing radiation flow
	space-grid negative heat-flow face
	space-grid positive heat-flow face

The radiation flow connectors are actually identical to those used for lumped-temperature surfaces. So it is possible to interchangeably connect the same view configuration components to either lumped or distributed-temperature radiation surfaces.

Meaning of Distributed Temperature

The surface temperature $T_s(x)$ is discretized on a spatial grid and solved according to an energy balance principal:

$$q_c + q_r = 0$$

where q_c is the thermal conduction heat flux (W/m) of all heat-flow face attachments and q_r is the radiation heat flux of all radiation flow attachments.

The radiation flow attachments do not actually support a distributed radiation flux to other surfaces. Rather they supply only a total heat flux over the entire surface according to its effective radiation temperature T_e. The radiation heat flux is localized internally according to equation (26.31) below.

26.3 View-Configuration Components

View configuration components represent the geometry of the two adjoining radiation surfaces as well as their relative orientation in space. There are view-configuration components for a selected number of common cases. Inputs and outputs common to all view configurations are:

FAmult : (real, dimensionless) Empirical multiplier that can be used to scale the default view area FA to account for radiation blocked by other surfaces or differences between actual surfaces and the perfect geometrical shapes assumed in view configuration theory.

FA : (real, m^2) View area $F_{np}A_n$ or $F_{pn}A_p$. The product of an internal calculation and FAmult.

BrNeg : (real, W/m^2) Radiosity B_n of negative view.

BrPos : (real, W/m^2) Radiosity B_p of positive view.

RadNeg : (real, W) Radiation flow $R_{net\_}$ negative view

RadNeg : (real, W) Radiation flow R_{net+} positive view

AEQ : (real, W) Available energy loss to radiation heat flow.

Mandatory Surface Component Connections

All view configuration components contain built-in negative-facing and positive-facing radiation connectors for mandatory connection to radiation surface components.

26.3.1 Generic Configuration

TrcGeneric

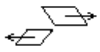
If none of the specific view configurations below are appropriate you can define you own view factor with this component. Variable FA is an input rather than output with the intent that you will recast it as an expression involving one or both of the output variables:

An : (real, m^2) Area A_n of radiation surface connected across negative boundary.

Ap : (real, m^2) Area A_p of radiation surface connected across positive boundary.

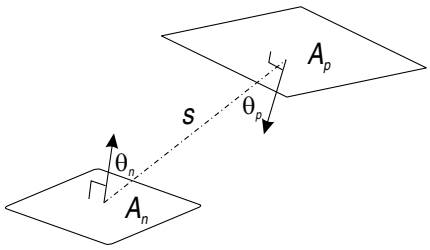
For example if half of the radiation leaving A_n reaches A_p then you would recast FA as the expression 0.5 * An. It is always possible to express the view area FA as a function of $A_n$and A_p as the view configurations below demonstrate. You may formulate any required parameters besides An and Ap as user-defined inputs. Use of this component is illustrated in the example model RadGeneric.ltc.

26.3.2 Planar Elements



TrcPlanar-Elmts

Two planar elements A_n and A_p separated by distance s with angles θ_n and θ_p between surface normals and the view direction. The surface of element A_n exchanges radiation with the surface of element A_p :



This is the differential basis for all finite area view factors. It is found in Sparrow and Cess [63] and example A-1 posted at *www.me.utexas.edu/~howell/sectionc/A-1.html*. The FA product (either $F_{np}A_n$ or $F_{pn}A_p$ is

$$FA = \frac{A_n \cos\theta_n A_p \cos\theta_p}{\pi s^2} \tag{26.1}$$

Strictly speaking this equation applies only to infinitesimal planar elements A_n and A_p but is a reasonable approximation whenever their dimensions are small compared to the separation distance s. The shape of the planar elements is irrelevant. The normal angles and separation distance are inputs for this configuration:

ThetaN : (real, radians) angle θ_n between A_n surface normal and view direction.

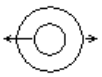
ThetaP : (real, radians) angle θ_p between A_p surface normal and view direction.

Sepr : (real, m) separation distance s between planar elements.

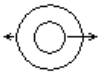
Surface areas A_n and A_p come from the radiation surfaces connected to the negative-facing and positive-facing views respectively.

26.3.3 Concentric Spheres

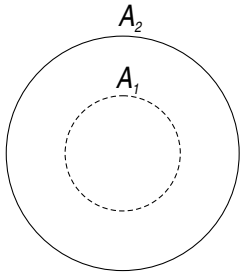
Two concentric spheres. The outer surface of an inner sphere A_1 exchanges radiation with the inner surface of the outer sphere A_2:



TrcConcSphNP



TrcConcSphPN



As reported in example 31 on p. 790 of [62] view factor $F_{12} = 1$ and view factor $F_{21} = A_1/A_2$. In other words all of the radiation leaving surface A_1 reaches A_2 (obviously) but only the fraction A_1/A_2 of the radiation leaving surface A_2 reaches A_1 (from reciprocity condition $F_{12}A_1 = F_{21}A_2$). The fact that all the radiation from the inner sphere reaches the outer is true even if the spheres are not concentric. So these view factors apply to the case of non-concentric spheres too, so long as one is completely contained in the other.

In the Sage implementation there are two separate components, a *NP* configuration and a *PN* configuration. In the *NP* configuration the inner surface is presumed to be surface A_n connected to the negative-facing view and the outer surface A_p connected to the positive facing view, as indicated in the component icon. In the *PN* configuration the roles are reversed.

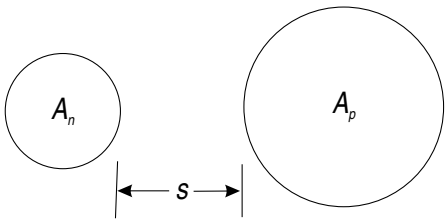
There are no input variables for concentric sphere configurations other than the inherited FAmult of all configurations. The areas A_1 and A_2 are imported from the radiation surfaces across the connection. View area output FA is calculated as $F_{12}A_1 = A_1$ (which has the same value as $F_{21}A_2$). For the two cases the FA product is

$$FA = \begin{cases} A_n & NP \text{ configuration} \\ A_p & PN \text{ configuration} \end{cases} \tag{26.2}$$

26.3.4 Separated Spheres

Two spheres separated by distance s. The surface of sphere A_n exchanges radiation with the surface of sphere A_p :

TrcSeprSph



This case corresponds to example C-137 posted at *www.me.utexas.edu/~howell/sectionc/C-137.html*. An approximate formulation for the FA product (either $F_{np}A_n$ or $F_{pn}A_p$ is

$$FA = \left(1 - \sqrt{1 - \frac{A_n}{Y}}\right)\left(1 - \sqrt{1 - \frac{A_p}{Y}}\right) Y \tag{26.3}$$

where

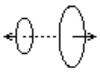
$$Y = \left(\sqrt{4\pi}s + \sqrt{A_n} + \sqrt{A_p}\right)^2$$

This approximation derived by J.D. Felske [12] applies to all sphere spacings and has a worst-case error of 5.8% when the spheres are touching ($s = 0$). The separation distance is an input for this configuration:

Sepr : (real, m) separation distance s between closest points.

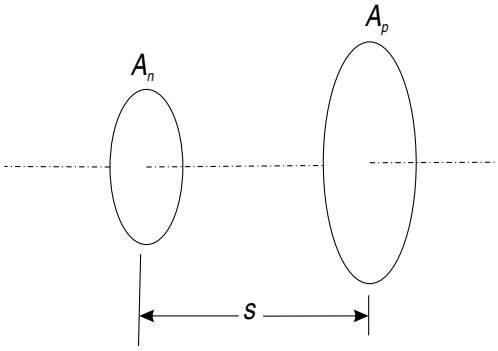
Surface areas A_n and A_p come from the radiation surfaces connected to the negative-facing and positive-facing views respectively.

26.3.5 Parallel Disks



TrcParallDsk

Two parallel coaxial disks separated by distance s. The surface of disk A_n exchanges radiation with the surface of disk A_p :



This case corresponds to example C-41 posted at *www.me.utexas.edu/~howell/sectionc/C-41.html*. The FA product (either $F_{np}A_n$ or $F_{pn}A_p$ is

$$FA = \frac{1}{2}\left(Y - \sqrt{Y^2 - 4A_nA_p}\right) \tag{26.4}$$

where

$$Y = \pi s^2 + A_n + A_p$$

The separation distance is an input for this configuration:

Sepr : (real, m) normal separation distance s between disks.

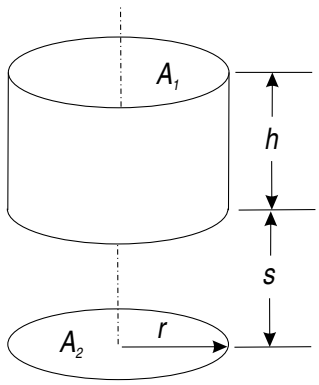
Surface areas A_n and A_p come from the radiation surfaces connected to the negative-facing and positive-facing views respectively.

26.3.6 Disk to Inside of Cylinder with Same Radius

TrcDskCylNP

TrcDskCylPN

A right circular cylinder of radius r and height h separated by distance s from a coaxial disk of the same radius. The inner cylindrical surface A_1 exchanges radiation with the disk surface A_2:



This case corresponds to example C-81 posted at *www.me.utexas.edu/~howell/sectionc/C-81.html*. The $F_{12}A_1$ product is

$$F_{12}A_1 = \frac{A_1}{4}\left[\left(1+\frac{H_2}{H_1}\right)\sqrt{4+(H_1+H_2)^2}-(H_1+2H_2) - \frac{H_2}{H_1}\sqrt{4+H_2^2}\right] \tag{26.5}$$

where H_1 and H_2 are defined as

$$H_1 = \frac{h}{r}$$
$$H_2 = \frac{s}{r}$$

In the Sage implementation there are two separate components, a *NP* configuration and a *PN* configuration. In the *NP* configuration the base is presumed to be surface A_n connected to the negative-facing view and the cylindrical surface A_p connected to the positive facing view, as indicated in the component icon. In the *PN* configuration the roles are reversed. The separation distance is an input for this configuration:

Sepr : (real, m) separation distance s between disk and cylinder.

The areas A_n and A_p imported from the radiation surfaces across the connection determine cylinder radius and height as

$$r = \begin{cases} \sqrt{A_n/\pi} & NP \text{ configuration} \\ \sqrt{A_p/\pi} & PN \text{ configuration} \end{cases}$$

and

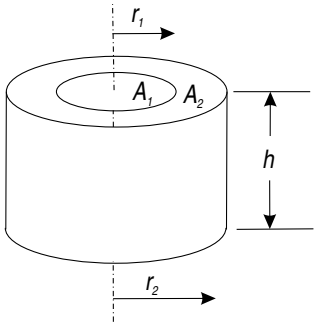
$$h = \begin{cases} A_p/(2\pi r) & NP \text{ configuration} \\ A_n/(2\pi r) & PN \text{ configuration} \end{cases}$$

26.3.7 Inside of Cylinder to Disk Within Base or Top

TrcEndCylNP

TrcEndCylPN

A disk of radius r_1 within the base or top of a right circular cylinder of radius r_2 and height h. The disk surface A_1 exchanges radiation with the inner cylindrical surface A_2:



This case corresponds to example C-80 posted at *www.me.utexas.edu/~howell/sectionc/C-80.html*. The $F_{12}A_1$ product is

$$F_{12}A_1 = \frac{A_1}{2}\left[1 - R^2 - H^2 + \sqrt{(1+R^2+H^2)^2 - 4R^2}\right] \tag{26.6}$$

where R and H are defined as

$$\begin{aligned} R &= \frac{r_2}{r_1} \\ H &= \frac{h}{r_1} \end{aligned}$$

In the Sage implementation there are two separate components, a *NP* configuration and a *PN* configuration. In the *NP* configuration the disk is presumed to be surface A_n connected to the negative-facing view and the cylindrical surface A_p connected to the positive facing view, as indicated in the component icon. In the *PN* configuration the roles are reversed. The cylinder diameter is an input for this configuration.

Dcyl : (real, m) cylinder diameter $2r_2$.

The areas A_n and A_p imported from the radiation surfaces across the connection determine disk radius and cylinder height as

$$r_1 = \begin{cases} \sqrt{A_n/\pi} & \textit{NP} \text{ configuration} \\ \sqrt{A_p/\pi} & \textit{PN} \text{ configuration} \end{cases}$$

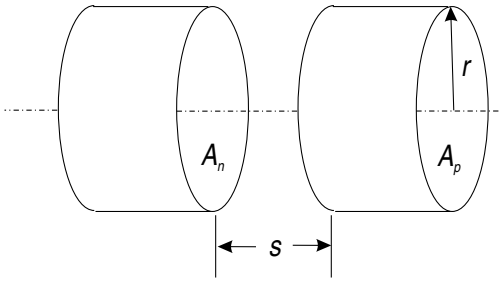
and

$$h = \begin{cases} A_p/(2\pi r_2) & \textit{NP} \text{ configuration} \\ A_n/(2\pi r_2) & \textit{PN} \text{ configuration} \end{cases}$$

26.3.8 Collinear Cylinders

Two collinear right circular cylinders of inner radius r separated from each other by a distance s. The inside surface A_n of one cylinder exchanges radiation with the inside surface A_p of the other:

TrcColinCyl



This case corresponds to example C-87 posted at *www.me.utexas.edu/~howell/sectionc/C-87.html*. The FA product (either $F_{np}A_n$ or $F_{pn}A_p$ is

$$\begin{aligned} FA &= \frac{2\pi r^2}{4}\left[\frac{2A_nA_p}{(2\pi r^2)^2} + G\left(\frac{A_n+2\pi rs}{2\pi r^2}\right) + G\left(\frac{A_p+2\pi rs}{2\pi r^2}\right)\right. \\ &\quad \left. -G\left(\frac{A_n+A_p+2\pi rs}{2\pi r^2}\right) - G\left(\frac{s}{r}\right)\right] \end{aligned} \tag{26.7}$$

where G is the function defined by

$$G(x) \equiv x\sqrt{x^2+4}$$

The cylinder diameter and separation distance are inputs for this configuration:

Dcyl : (real, m) Cylinder internal diameter ($2r$).

Sepr : (real, m) Separation gap (s).

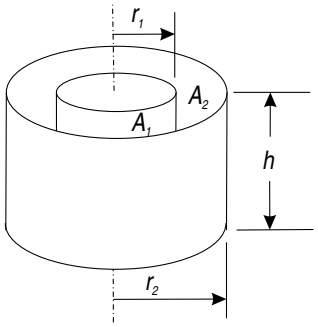
Areas A_n and A_p come from the radiation surfaces connected to the negative-facing and positive-facing views respectively.

26.3.9 Coaxial Cylinders of Same Height

TrcCoaxCylNP

TrcCoaxCylPN

Two coaxial right circular cylinders of radius r_1 and r_2 ($r_1 < r_2$) with the same height h. The inner cylindrical surface A_1 exchanges radiation with the outer cylindrical surface A_2:



This case corresponds to example C-92 posted at *www.me.utexas.edu/~howell/sectionc/C-92.html*. The $F_{12}A_1$ product is

$$\begin{aligned} FA &= 2h^2\left[\frac{1}{2}(R_2^2 - R_1^2 - 1)\arccos\left(\frac{R_1}{R_2}\right) + \pi R_1 - \frac{\pi}{2}AB\right. \\ &\quad -2R_1 \arctan\left(\sqrt{R_2^2 - R_1^2}\right) \\ &\quad \left. +\sqrt{(1+A^2)(1+B^2)}\arctan\left(\sqrt{\frac{(1+A^2)B}{(1+B^2)A}}\right)\right] \end{aligned} \tag{26.8}$$

where R_1, R_2, A and B are defined as

$$\begin{aligned} R_1 &= \frac{r_1}{h} \\ R_2 &= \frac{r_2}{h} \\ A &= R_2 + R_1 \\ B &= R_2 - R_1 \end{aligned}$$

In the Sage implementation there are two separate components, a *NP* configuration and a *PN* configuration. In the *NP* configuration the inner cylindrical surface is presumed to be surface A_n connected to the negative-facing view and the outer cylindrical surface A_p connected to the positive facing view, as indicated in the component icon. In the *PN* configuration the roles are reversed. The cylinder height is an input for this configuration.

Hcyl : (real, m) cylinder height h.

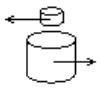
The areas A_n and A_p imported from the radiation surfaces across the connection determine the cylinder radii as

$$r_1 = \begin{cases} A_n/(2\pi h) & NP \text{ configuration} \\ A_p/(2\pi h) & PN \text{ configuration} \end{cases}$$

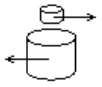
and

$$r_2 = \begin{cases} A_p/(2\pi h) & NP \text{ configuration} \\ A_n/(2\pi h) & PN \text{ configuration} \end{cases}$$

26.3.10 Offset Cylinders of Different Diameter

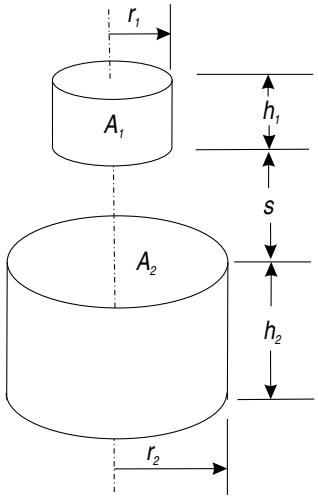


TrcOffsetCylNP



TrcOffsetCylPN

Two coaxial right circular cylinders of radius r_1 and r_2 ($r_1 < r_2$) and heights h_1 and h_2 offset by distance s. The outside surface A_1 of the smaller cylinder exchanges radiation with the inside surface A_2 of the larger cylinder:



This case corresponds to example C-98 posted at *www.me.utexas.edu/~howell/sectionc/C-98.html*. The $F_{12}A_1$ product is

$$FA = \frac{A_1}{L}\left(f_F(L+D) + f_F(Y+D) - f_F(D) - f_F(L+D)\right) \tag{26.9}$$

where D, Y, L and R are defined as

$$\begin{aligned} D &= \frac{s}{r_2} \\ Y &= \frac{h_2}{r_2} \\ L &= \frac{h_1}{r_2} \\ R &= \frac{r_1}{r_2} \end{aligned}$$

and function f_F is defined as

$$f_F(x) = \frac{f_B(x)}{8R} + \frac{1}{2\pi}\left[x \arccos\left(\frac{f_A(x)}{f_B(x)}\right)\right.$$

$$-\frac{1}{2}\sqrt{\left(\left(\frac{f_A(x)+2}{R}\right)^2-4\right)}\arccos\left(\frac{f_A(x)R}{f_B(x)}\right)$$
$$-\frac{f_A(x)}{2R}\arcsin(R)\Bigg]$$

where

$$\begin{aligned} f_A(x) &= x^2+R^2-1 \\ f_B(x) &= x^2-R^2+1 \end{aligned}$$

In the Sage implementation there are two separate components, a *NP* configuration and a *PN* configuration. In the *NP* configuration the smaller cylindrical surface is presumed to be surface A_n connected to the negative-facing view and the larger cylindrical surface A_p connected to the positive facing view, as indicated in the component icon. In the *PN* configuration the roles are reversed. The cylinder diameters and separation are inputs for this configuration.

DcylInner : (real, m) smaller cylinder external diameter $2r_1$.

DcylOuter : (real, m) larger cylinder internal diameter $2r_2$.

Sepr : (real, m) Separation gap (s).

The areas A_n and A_p imported from the radiation surfaces across the connection determine the cylinder heights as

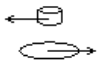
$$h_1 = \begin{cases} A_n/(2\pi r_1) & NP \text{ configuration} \\ A_p/(2\pi r_1) & PN \text{ configuration} \end{cases}$$

and

$$h_2 = \begin{cases} A_p/(2\pi r_2) & NP \text{ configuration} \\ A_n/(2\pi r_2) & PN \text{ configuration} \end{cases}$$

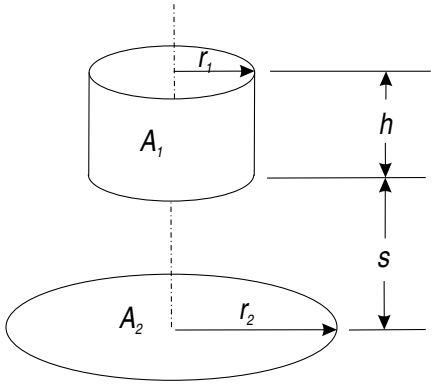
26.3.11 Outside of Cylinder to Offset Disk with Larger Radius

A right circular cylinder of radius r_1 and height h separated by distance s from a coaxial disk of larger radius r_2. The outer cylindrical surface A_1 exchanges radiation with the disk surface A_2:



TrcCylDskNP

TrcCylDskPN



The case for zero separation distance ($s = 0$) corresponds to example C-77 posted at *www.me.utexas.edu/~howell/sectionc/C-77.html*. For that case the $F_{12}A_1$ product is

$$\begin{aligned} F_{12}A_1 &= \frac{\pi r_2^2 B}{4} + r_1 h \left\{ \arccos\left(\frac{A}{B}\right) \right. \\ &\quad -\frac{1}{2H}\sqrt{\left(\frac{(A+2)^2}{R^2} - 4\right)} \arccos\left(\frac{AR}{B}\right) \\ &\quad \left. -\frac{A}{2RH} \arcsin R \right\} \end{aligned} \tag{26.10}$$

where

$$\begin{aligned} R &= \frac{r_1}{r_2} \\ H &= \frac{h}{r_2} \\ A &= H^2 + R^2 - 1 \\ B &= H^2 - R^2 + 1 \end{aligned}$$

For nonzero separation distance ($s > 0$) Sage calculates $F_{12}A_1$ as the complement to the sum of other view factors (e.g. according to equation (26.19)). Imagine the disk in the drawing above is extended to form the inside surface of a complete can by adding a disk of the same diameter resting on the top edge of the cylinder and a cylinder between the outer edges of the two disks. All the radiation leaving the inner cylinder reaches the inside surface of this can.

The view factor for the inner cylinder to the top disk is given above. The view factors for the inner cylinder to the outer cylinder (divided into two pieces of heights h and s are given in sections 26.3.9 and 26.3.10. The view factor to the lower disk in question is the remainder.

In the Sage implementation there are two separate components, a *NP* configuration and a *PN* configuration. In the *NP* configuration the cylinder is presumed to be surface A_n connected to the negative-facing view and the disk surface A_p connected to the positive facing view, as indicated in the component icon. In the *PN* configuration the roles are reversed. The cylinder diameter and disk-to-cylinder separation are inputs for this configuration.

Dcyl : (real, m) cylinder diameter $2r_1$.

Sepr : (real, m) separation distance s between cylinder and disk.

The areas A_n and A_p imported from the radiation surfaces across the connection determine disk radius and cylinder height as

$$r_2 = \begin{cases} \sqrt{A_p/\pi} & NP \text{ configuration} \\ \sqrt{A_n/\pi} & PN \text{ configuration} \end{cases}$$

and

$$h = \begin{cases} A_n/(2\pi r_1) & NP \text{ configuration} \\ A_p/(2\pi r_1) & PN \text{ configuration} \end{cases}$$

26.4 Radiation Exchange Theory

Sparrow and Cess [63] discuss the physics of radiation exchange between gray surfaces within enclosures. The formulation in this section is essentially the same except presented in a way consistent with Sage's implicit solution scheme. The formulation considers radiation exchange in an enclosure having N surfaces with areas A_i, emissivities ϵ_i and temperatures T_i. In a Sage model these variables are distributed among the radiation surface components of the model and the solution equations are likewise distributed. Key assumptions are:

absorptivity = emissivity The coefficients of emission ϵ and absorption α are equal.

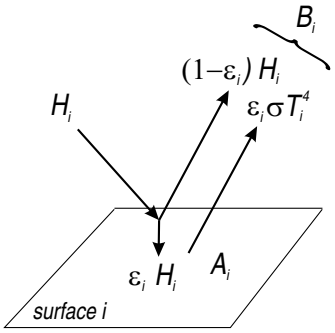
diffuse outgoing radiation The emitted and reflected radiation from each surface is diffusely distributed.

uniform incident radiation The incident radiation on any surface from another is uniformly distributed.

26.4.1 Net Surface Radiation Heat Transfer

The sketch below shows a gray surface with total incoming radiation (per unit area) H_i. The outgoing radiation is the reflected amount $(1-\epsilon_i)H_i$ plus the emitted radiation $\epsilon_i \sigma T_i^4$, where $\sigma = 5.66961\text{E–}8$ W/(m^{2}K^4) is the Stefan-Boltzmann

constant. The symbol B_i denotes the *radiosity*, the combined emitted plus reflected radiation.



The net rate of heat loss Q_{Ri} from a surface A_i is formulated in general as the difference between emitted radiation $\epsilon_i \sigma T_i^4 A_i$ and the absorbed portion of the incident radiation $\epsilon_i H_i A_i$

$$Q_{Ri} = \epsilon_i \left(\sigma T_i^4 A_i - H_i A_i \right) \tag{26.11}$$

The incident radiation is the total amount that arrives from all the surfaces in the enclosure (including radiation from A_i to itself).

$$H_i A_i = \sum_{j=1}^{N} B_j F_{ji} A_j \tag{26.12}$$

where B_j is the radiosity emitted from surface j and F_{ji} is the fraction of radiation emitted from surface j reaching surface i — the *view factor*. Radiosity will be considered more below but for now substituting equation (26.12) for $H_i A_i$ in equation (26.11) gives

$$Q_{Ri} = \epsilon_i \left(\sigma T_i^4 A_i - \sum_{j=1}^{N} B_j F_{ji} A_j \right) \tag{26.13}$$

This constitutes N of the $2N$ equations required for solving the unknowns B_i and either T_i or Q_{Ri} (depending on which is unknown for a given surface). The other N equations come from the definition of radiosity B_i as the emitted radiation per unit area plus the *reflected* part of the incident radiation per unit area

$$B_i = \epsilon_i \sigma T_i^4 + (1 - \epsilon_i) H_i \tag{26.14}$$

Multiplying through by A_i (to convert the left-hand side to total emitted radiation) and again substituting equation (26.12) for $H_i A_i$ the previous equation may be written

$$B_i A_i = \epsilon_i \sigma T_i^4 A_i + (1 - \epsilon_i) \sum_{j=1}^{N} B_j F_{ji} A_j \tag{26.15}$$

It will be helpful to isolate the T_i and B_j terms into separate equations by multiplying equation (26.13) by $(1-\epsilon_i)/\epsilon_i$ and adding to equation (26.15) to get

$$B_i A_i = \sigma T_i^4 A_i - \frac{1-\epsilon_i}{\epsilon_i} Q_{Ri} \tag{26.16}$$

then subtracting equation (26.13) from equation (26.15) to get

$$Q_{Ri} = B_i A_i - \sum_{j=1}^{N} B_j F_{ji} A_j \tag{26.17}$$

The previous equation can be put into terms of the radiosity exchange between surfaces by writing $\sum_{j=1}^{N} B_j F_{ji} A_j$ as $B_i F_{ii} A_i + \sum_{j \neq i} B_j F_{ji} A_j$ giving

$$Q_{Ri} = (1 - F_{ii}) B_i A_i - \sum_{j \neq i} B_j F_{ji} A_j \tag{26.18}$$

A full accounting of the radiation leaving surface i requires that

$$\sum_{j=1}^{N} F_{ij} = 1 \tag{26.19}$$

In English, all the radiation emitted from surface i goes to other surfaces or itself. So one may replace the term $(1 - F_{ii})$ in equation (26.18) with $\sum_{j \neq i} F_{ij}$ to obtain

$$Q_{Ri} = \sum_{j \neq i} B_i F_{ij} A_i - \sum_{j \neq i} B_j F_{ji} A_j \tag{26.20}$$

Equations (26.16) and (26.20) are the basis for the Sage solution scheme described next.

26.4.2 Sage Solution Scheme

It is convenient to write the individual terms $B_i F_{ij} A_i$ and $B_j F_{ji} A_j$ appearing in the right-hand side of equations (26.20) using single symbols

$$R_{ij} \equiv B_i F_{ij} A_i \tag{26.21}$$

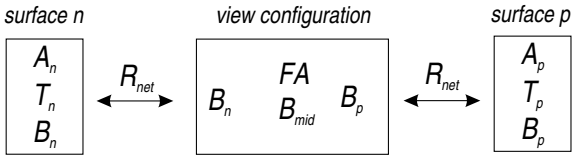
$$R_{ji} \equiv B_j F_{ji} A_j \tag{26.22}$$

R_{ij} is understood as the radiosity emitted from surface A_i that reaches surface A_j and vice-versa. In terms of R_{ij} and R_{ji} equation (26.20) becomes

$$Q_{Ri} = \sum_{j \neq i} (R_{ij} - R_{ji}) \tag{26.23}$$

The radiation terms $R_{ij} - R_{ji}$ (for $j \neq i$) are what *flow* through radiation connectors. They are solved implicitly according to the principle that the radiosity B is continuous across the connection. The following diagram represents

the Sage solution scheme for two area elements A_n and A_p connected by a view-configuration element. The symbol R_{net} denotes the net radiation flow $R_{np} - R_{pn}$.



Radiation Surface Components

In the surface components A_n and A_p are inputs and T_n, T_p, B_n, B_p are solved. T_n and T_p are solved implicitly from the component energy balance. For example, the equation for solving T_n is

$$Q_{Rn} + Q_{cond} = 0 \tag{26.24}$$

In English, the net heat loss from radiation balances the net heat loss through conductive heat flow connections. Sage calculates Q_{Rn} explicitly from equation (26.23) where the individual $R_{nj} - R_{jn}$ terms come from the radiation connections (labeled R_{net} in the above drawing) and Q_{cond} comes from any conventional conductive heat transfer connections to the surface. B_n and B_p are solved explicitly from equation (26.16).

View-Configuration Components

In the view-configuration component either $F_{np}A_n$ or $F_{pn}A_p$ (whichever is easier) is calculated explicitly from the areas A_n and A_p across the connections and other geometrical inputs. A reciprocity principle requires that $F_{np}A_n = F_{pn}A_p$. Either represents the effective view area of an imaginary window through which surfaces A_n and A_p see each other. Calculating only one simplifies things and also eliminates the possibility that $F_{np}A_n$ might not equal $F_{pn}A_p$ due to programming error.

View configurations also enforce the requirement that the net radiation flows R_{net} imported from the radiation connections at each end are equal and related to the difference in radiosities across the connections by

$$(B_n - B_p)FA = R_{net} \tag{26.25}$$

This equation comes from the definitions (26.21) and (26.22) and by substituting FA for either $F_{np}A_n$ or $F_{pn}A_p$. In this form the electrical circuit analogy for radiation heat transfer is clear. Radiation exchange R_{net} is like an electrical current driven by a voltage difference $B_n - B_p$ across a resistance $1/FA$. Or maybe more appropriately a thermal resistance analogy where R_{net} is like a heat flow driven by a temperature difference $B_n - B_p$ across a thermal resistance

$1/FA$. At any rate view configurations calculate a somewhat fictitious but useful midpoint radiosity (think temperature) B_{mid} implicitly according to

$$R_{net+} - R_{net-} = 0 \tag{26.26}$$

where R_{net+} is the radiation flow imported from the connector to surface A_p and similarly for R_{net-}. Enforcing this equation has the effect of forcing the radiation flows to be the same at both ends. Then to insure that equation (26.25) is satisfied view configurations calculate B_n and B_p explicitly according to

$$B_p = B_{mid} - \frac{1}{2}\frac{R_{net+}}{FA} = 0 \tag{26.27}$$

$$B_n = B_{mid} + \frac{1}{2}\frac{R_{net-}}{FA} = 0 \tag{26.28}$$

Self Radiosity

The radiosity emitted from a surface A_i that reaches itself ($B_i F_{ii} A_i$) does not matter to the overall energy balance of surface i but it is nonetheless important to consider. The self view factor F_{ii} could be an explicit input for surface components but it is more convenient to evaluate it indirectly from the radiation accounting principle (26.19) which implies that it must be

$$F_{ii} = 1 - \sum_{j \neq i} F_{ij} \tag{26.29}$$

In other words, F_{ii} is determined from all the view factors by which A_i sees the other surfaces of your model. Doing it this way means that if you fail to properly include all the radiation connections of a radiation enclosure the error will show up as more or less than the correct amount of self radiation. The implied self view factor F_{ii} is the output Fself for radiation surface components. Keeping an eye on it can help you diagnose problems with your radiation model. It should be zero for a flat or convex surface and greater than zero only for a concave surface. Never negative.

26.4.3 Distributed Temperature Surface Components

In principle radiation exchange with a surface having a temperature distribution rather than a uniform temperature can be modeled by subdividing that surface into small pieces where the temperature is nearly uniform on each piece. But modeling the radiation transfer grows complicated quickly because each piece typically *sees* the other surfaces of the enclosure with a different view factor. So if there are N subdivisions there are N view factors required for each of the other surfaces of the radiation enclosure. Not to mention the view factors among the subdivided pieces themselves.

But when the conditions listed below are satisfied modeling radiation exchange with a distributed-temperature surface can be greatly simplified.

Uniform emissivity The emissivity is uniform over the surface.

Convex surface No two pieces of the subdivided surface see each other.

Uniform view factors The view factors from the subdivided surface elements to other surfaces of the radiation enclosure are all the same.

These are the assumptions underlying Sage's distributed radiation surface component. A necessary condition for the validity of the last assumption is that the surface length (in the direction of the temperature distribution) is small compared to the distance separating the subdivided surface from other surfaces of the radiation enclosure. For example, the subdivided outer surface of a relatively short small diameter cylinder exchanging radiation with the inner surface of a larger coaxial cylinder.

To the extent the above assumptions are valid a subdivided surface A_e with equal area elements A_i and temperatures T_i $(i = 1..N)$ has the same radiation heat transfer characteristics as the original surface at the effective temperature T_e provided that

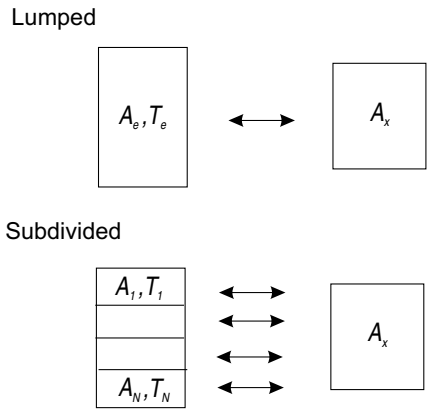
$$T_e^4 = \frac{1}{N} \sum_{i=1}^{N} T_i^4 \tag{26.30}$$

And the radiation heat transfer Q_{Ri} to an individual element A_i is related to the radiation heat transfer Q_{Re} to the lumped surface A_e by

$$Q_{Ri} = \frac{1}{N} \left(Q_{re} - \epsilon_e \sigma A_e (T_e^4 - T_i^4) \right) \tag{26.31}$$

Two-Surface Case

The above equations are most easily derived for the case where lumped surface A_e exchanges heat with only one other surface A_x, as illustrated here:



Applying equation (26.13) to the lumped case gives the radiation heat transfer to surface A_e as the sum of emitted plus incoming radiation

$$Q_{Re} = \epsilon_e \left(\sigma T_e^4 A_e - B_x F_{xe} A_x \right) \tag{26.32}$$

where the assumption that $F_{ee} = 0$ (no self view) has been used to simplify the last term on the right side. Applying equation (26.13) to the subdivided case gives the radiation heat transfer to each of the surfaces A_i as

$$Q_{Ri} = \epsilon_i \left(\sigma T_i^4 A_i - B_x F_{xi} A_x \right) \tag{26.33}$$

where this time the no-self-view assumption $F_{ji} = 0$, $j = 1..N$ has simplified the last term on the right side. The previous equation can be further simplified by requiring that A_e be subdivided into equal area pieces with $A_i = A_e/N$ and using the assumptions of uniform emissivity and view factors in the form $\epsilon_i = \epsilon_e$ and $F_{xi} = F_{xe}/N$ (radiation from A_x to A_e distributed uniformly among the A_i). Under these assumptions the previous equation becomes

$$Q_{Ri} = \frac{1}{N} \epsilon_e \left(\sigma T_i^4 A_e - B_x F_{xe} A_x \right) \tag{26.34}$$

The total radiation heat transfer to all the subdivided surfaces is found by summing equation (26.34)

$$\sum_{i=1}^{N} Q_{Ri} = \epsilon_e \left(\sigma \left[\frac{1}{N} \sum_{i=1}^{N} T_i^4 \right] A_e - B_x F_{xe} A_x \right) \tag{26.35}$$

By comparing this equation with equation (26.32) it is clear that the total radiation heat transfer to the subdivided surfaces will be the same as the radiation heat transfer for the lumped case provided the lumped temperature T_e satisfies equation (26.30). Eliminating the common term $B_x F_{xe} A_x$ between equation (26.34) and equation (26.32) gives the above formula (26.31) for the individual radiation heat transfers in terms of the lumped heat transfer.

The radiation leaving the lumped or divided surfaces and reaching surface A_x is the same in either case. The reflected radiation is the same because of the uniform emissivity assumption ($\epsilon_i = \epsilon_e$). The emitted radiation is the same (e.g. $\epsilon_e \sigma T_e^4 A_e = \epsilon_e \sum_{i=1}^{N} \sigma T_i^4 A_i$), again by choice of T_e.

General Case

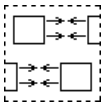
The general case where surface A_e exchanges radiation with multiple surfaces may be derived similarly by replacing surface A_x in the above analysis with M surfaces $A_k : k = (N+1)..(N+M)$ (the k indices start with $N+1$ to avoid confusion with the indices for the subdivided surfaces $A_i : i = 1..N$). The term $B_x F_{xe} A_x$ appearing in equations (26.32) and (26.34) is now replaced with $\sum_k B_k F_{xe} A_k$ which cancels as before and leads to the same conclusions about equivalent bulk temperature T_e and individual radiation heat transfers.

Chapter 27

Submodels and Containers

The components in this section allow you to group other model components together under a separate window.

27.1 Submodels


TSCFusion-Submodel

A submodel allows you to create all the child model components available in the root model, except organized one level deeper in the model tree and displayed in a separate window. This is handy for organizing large models, multi-stage models, managing display screen clutter and copy/pasting of groups of components simultaneously. You may promote connector arrows from within a submodel up to the root model level for connection there to other submodels or model components at the root level.

All of the input variables of the root model are inherited by a submodel and passed to any child model components created within the submodel. There are no new input or output variables introduced by a submodel. So whether or not a model component resides at the root level or in a submodel does not affect the model solution.

Using the cut and paste tools it is possible to move model components from the root model to a submodel and vice-versa. So long as the model component is not externally connected.

27.2 Containers

Container components are similar to submodel components except not all child components available in the root model are available in the container and new input variables are introduced.

The main reason for the existence of containers is to support heat exchanger modeling. For example, you might want to model the transverse heat flow (distributed between y-faces) between separate heat-exchanger components representing the two legs of a counterflow heat exchanger. Sage normally does not

allow this because the components might have different lengths, which would require the heat flux value (W/m) to change discontinuously across the boundary.

There are two ways to escape Sage's normal restrictions against such inter-component heat-flow connections. The first is create the components together within a *parallel container*. The second is to create them within a *multi-length container*.

Sample model *HeatExchangers-CounterflowRecuperative* in the Apps\SCFusion\-Samples\ElementsThermoModels sub-directory under the installation directory gives specific examples of interconnected equal-length and different-length heat exchangers.

27.2.1 Parallel Container

TSCFContainer

A parallel container defines a common length, number of computational cells (NCell) and initial temperature distribution (Tinit) inherited by all child components. Any canister or heat-exchanger child components created within the parallel container inherits these same values. This permits you to make transverse heat-transfer connections from the thermal solid components within the canisters or heat exchangers by promoting their connector arrows to a common level and connected them.

The only variables in parallel containers are the following inputs:

NCell : (dimensionless) The number of spatial nodes in the computational grids of all underlying model components having such grids.

Length : (real, m) Common length of all underlying model components.

Tinit : (cubic spline, W) Axial temperature distribution $T(x)$ where $x = 0$ is the negative endpoint and $x = 1$ is the positive endpoint. Underlying model components use Tinit as a fixed boundary condition or just an initial value, depending on whether the temperature of the component is fixed or solved.

Many of the model components in the root-level child-model creation palette are available in the parallel-container palette, except for moving parts and other components where the concept of length is undefined. In particular, canisters, heat exchangers, and composite (piston/cylinder) components are available. Also available are the line heat source and distributed conductor components. A line heat source may be used to provide temperature boundary conditions to other components defined within the parallel container. The temperature so provided is that defined by the Tinit input variable of the parallel container. A distributed conductor may be used to provide a solid conduction path between other components. In either case the thermal connections are made via Gx heat-flow connections of negative or positive heat flow faces available on the Thermal Attachments page of the components to be connected.

27.2.2 Multi-Length Container

TSCFMLContainer

A multi-length container permits a different approach to transverse heat-flow connections, the result of which is to allow different-length components to be connected together. You might want to do this, for example, to connect together the walls of a tube spiral-wrapped around a straight central tube, as the iconic representation suggests. A special heat-flux transformer component, available in the basic page of the multi-length container child-model creation palette makes this possible (see below). Any components created as child models of a multi-length container can be interconnected via heat-flux transformers.

A multi-length container defines a number of computational cells (NCell) and initial temperature distribution (Tinit) that are inherited by all child components. It is similar to the above parallel container except that a common length is not defined. The only variables in parallel containers are the following inputs:

NCell : (dimensionless) As above.

Tinit : (cubic spline, W) As above.

Heat-Flux Transformer

TGxQtransformer

A heat-flux transformer is the essential component available within a multi-length container for making transverse heat flow connections between components of different length. It overrides Sage's usual connection compatibility test and permits you to inter-connect thermal solids within any two heat exchangers (conductors or quasi-adiabatic surfaces) by connecting their negative or positive heat flow arrows to one of its two built-in connector arrows, via Gx or Gxt y-face heat-flow connections.

A heat-flux transformer works by changing the heat flux value (W/m) from one face to the other by the ratio of lengths for the two interconnected thermal solids. This conserves the total heat energy flow (W) across the boundary. The actual implementation does this by way of a temperature grid whose values are defined implicitly by the following energy equation.

$$Q_n L_n = Q_p L_p \tag{27.1}$$

where Q_n and Q_p denote the heat fluxes at the negative and positive y-face boundaries and L_n and L_p are the lengths of the components connected across the two boundaries.

Warning about Convergence and Energy Conservation

A heat-flux transformer is designed to be interconnected only to components created within a multi-length container. But there are no software restrictions enforcing this rule. You can, in principle, connect it between higher-level components by promoting connection arrows to a higher level in the model. But if

you do so there is no guarantee that the solution will converge or that Sage's global energy conservation principle will hold.

If the NCell value for the heat flux transformer (inherited from multi-length container) is greater than for both higher level models then the solution will not converge. This is because a heat-flux transformer does not correspond to a physical object. Its defining energy equation only matches up boundary heat flows. If there are more T's in the heat-flux transformer grid than Q's in the conjoined model grids, there will be too many implicit function elements to satisfy simultaneously, given number of Q's available to do so. (This would not be a problem if there were a real solid energy equation present because then the internal temperature solution could adapt to any boundary condition by way of appropriate internal energy flows.)

A more subtle problem occurs if the NCell values differ for the two interconnected components: Global energy conservation will not hold. Grid interpolation methods, which are used to exchange solution information across connections, produce numerical truncation errors in the case when NCell values differ. This leads to slightly differ values for the total heat flow leaving one component domain and entering the other ($\int_n Qdx \neq \int_p Qdx$). Small energy conservation errors might not seem so bad but there is always the danger that the optimizer could exploit them and find a way to extract a large amount of "free"energy in order to boost power or efficiency.

Chapter 28

Material Properties

SCFusion model components specify physical properties for the working gas, solid materials and electromagnetic materials according to the named material selected in a list box of options. Sage allows you to access the underlying properties of the selected material in user-defined expressions and provides utility programs for modifying material properties or creating new materials.

28.1 Referenceable Properties

Access to physical properties is similar to calling a function in a programming language. Sage presumes that any arguments passed to these functions are in current dimensional units (set in the Model Class | Options dialog) and provides a returned value in current dimensional units. For example, a property function may require temperature T as an argument. The value of T can be an explicit numerical constant or another referenceable variable. Numerical constants are not recommended because they do not change if you change the current dimensional units. For example, the expression Solid.Ks(300) will return the thermal conductivity at 300 K if the current units for temperature are Kelvin and 300 C if centigrade. To avoid this problem it is better to reference a temperature whose value automatically changes along with current dimensional units. For example in a thermal-solid model component, Solid.Ks(TsNeg) will return the thermal conductivity at the temperature of the negative component boundary. TsNeg is the built-in output for the temperature at the negative boundary.

Gases

For the gas properties assigned to input Gas defined at the root model level you may reference:

Gas.T0	Representative temperature (relevant only for ideal gases).
Gas.Rgas	Gas constant R.
Gas.Cp0(T)	Zero-pressure limit of specific heat c_{p0} as a function of temperature T.
Gas.mu0(T)	Zero-pressure limit of viscosity μ_0 as a function of temperature T.
Gas.K0(T)	zero-pressure limit of thermal conductivity k_0 as a function of temperature T.
Gas.Vsound(T)	Zero-pressure limit speed of sound $\sqrt{\gamma_0 RT}$as a function of temperature T, where $\gamma_0 = 1/(1 - R/c_{p0}))$.
Gas.Cp(Rho, T)	Specific heat c_p as a function of density Rho and temperature T.
Gas.mu(Rho, T)	Viscosity μ as a function of density Rho and temperature T.
Gas.K(Rho, T)	Thermal conductivity k as a function of density Rho and temperature T.
Gas.Prandtl(Rho, T)	Prandtl number $c_p\mu/k$ as a function of density Rho and temperature T.
Gas.P(Rho, T)	Pressure P as a function of density Rho and temperature T.
Gas.s(Rho, T)	Mass-specific entropy s as a function of density Rho and temperature T.
Gas.TBeta(Rho, T)	$T\beta$, where $\beta = (\frac{\partial v}{\partial T})_P/v$ is the volumetric expansivity, as a function of density Rho and temperature T.
Gas.EOSErr(Rho, T)	Relative error of equation of state as a function of density Rho and temperature T (placeholder reserved for future use).
Gas.Compr(Rho, T)	Compressibility $Pv/(RT)$ as a function of density Rho $(1/v)$ and temperature T.
Gas.v(T, P)	Specific volume $1/\rho$ as a function of temperature T and pressure P.
Gas.T(Rho, RhoE, U)	Temperature T as a function of density Rho, volume-specific energy density RhoE and flow velocity U.
Gas.RhoE(Rho, T, U)	Volume-specific energy density ρe as a function of density Rho, temperature T and flow velocity U.

The following properties are only relevant for gases defined by a tabular equation of state.

Gas.Tc	The temperature T_c above which there is no two-phase state at any pressure. For a pure fluid this is the critical temperature. For a fluid mixture it is the temperature at the critical-condensation point, known in chemical engineering parlance as the *cricondentherm*.
Gas.Rhoc	Density ρ_c corresponding to T_c. For a pure fluid this is the critical density. For a fluid mixture it is the density at the critical-condensation point.
Gas.RhoD(T)	Dew-point vapor density ρ_d as a function of temperature T. Valid only for T below Tc where two phases are possible.
Gas.RhoB(T)	Bubble-point liquid density ρ_b as a function of temperature T. Same restrictions as RhoDew.
Gas.Qual(Rho, T)	Pseudo vapor quality (mass fraction) as a function of bulk density Rho and temperature T, defined by equation (15.21) of section 15.1.5.
Gas.CpL(Rho, T)	Liquid-phase specific heat c_p as a function of density Rho and temperature T. Valid only for T below Tc and Rho above the dew-point density RhoD where the liquid phase is present. To access the vapor-phase specific heat use the inherited identifier Cp(Rho, T). It has no restrictions other than that T and Rho must be in the tabulated range. Inside the two-phase region Cp(Rho, T) returns the vapor-phase value. Outside the two-phase region it returns the appropriate single-phase value, which may be either liquid or vapor.
Gas.MuL(Rho, T)	Liquid-phase viscosity μ as a function of density Rho and temperature T. Same restrictions as CpL.
Gas.KL(Rho, T)	Liquid-phase thermal conductivity k as a function of density Rho and temperature T. Same restrictions as CpL.

Thermal Solids

For the thermal properties assigned to input Solid within various model components you may reference:

Solid.Rhos	Density ρ_s.
Solid.Ks(T)	Thermal conductivity k_s as a function of temperature T.
Solid.Cs(T)	Specific heat c_s as a function of temperature T.
Solid.Diffusivity(T)	Thermal diffusivity $k_s/(\rho_s c_s)$ as a function of temperature T.

Electrical Conductors

For the electrical properties assigned to input Material within various model components you may reference:

Density	mass density ρ_s (kg/m^3)
Rs(T)	resistivity σ (Ohm-m) as a function of temperature T

Density is a single value for each material. Rs is defined by a set of cubic spline data pairs (T, Rs(T)).

Soft ferromagnetic materials

For the soft ferromagnetic properties assigned to input Material within various model components you may reference the above electrical properties in addition to:

Mur(T)	maximum relative permeability μ_r (dimensionless)
Jsat(T)	saturation magnetic polarization J_s (T)
Hcb(T)	induction coercive force H_{cB} (A/m)

permanent magnets

For the permanent magnet properties assigned to input Material within various model components you may reference the above electrical properties in addition to these properties particular to permanent magnets:

Kjh	magnetization curve $J(H)$ shape parameter K_{jh} $(0.5 < K_{jh} < 1)$
Br(T)	residual magnetic flux density B_r (T) at $H = 0$
Hcj(T)	magnetization coercive force H_{cJ} (A/m)

28.2 Material Property File Utilities

The selection list boxes of named materials for the working gas, thermal solids and electromagnetic components come from material data files. While there are enough named materials in these files for many needs, there is the distinct possibility that you will want to specify some material not available. Anticipating this, the Sage distribution includes some utilities that allow you to revised property data for existing materials or add data for new materials.

The default property data files are located in your program directory (default C:\Program Files (x86)\Gedeon \Sage x\Apps), under the following filenames.

filename	material type
gasSCF.dta	gases
solid.dta	thermal solids
ESolid.dta	electrical materials
FMSolid.dta	ferromagnetic material
PMSolid.dta	permanent magnets

28.2.1 Changing Properties with SCFProp

You can create a custom property file using the SCFProp program distributed with the SCFusion model class. This program is labeled "property database editor" in your Window's program group. With SCFProp running, File|Open the property file you want to start from, modify the data using the self-evident program dialogs, then File|Save As under a new name. Saving under a new name is important to avoid future conflicts with an upgrade distribution file of the same name. With SCFProp it is easy to change properties of an existing material or create an entirely new one. **Note:** All property values must be entered in SI units.

To use your custom property file in Sage simply change the corresponding file name in the dialog that opens under the Options|Model Class menu command. The values you set are stored in the program initialization file (e.g. SCFusion.ini), so they will be in effect for the current session as well as future sessions.

Sage reads data from property files whenever you change the value of a gas or solid variable selected from a list. This data becomes part of your model while Sage is executing and is stored in its input data file when your model is inactive. The next time you open the model, Sage automatically compares the data of any such variables with data by the same name, if any, in the current property files. You are ordinarily not aware of this unless there is a discrepancy, in which case Sage will put up a dialog giving you the option to update your model's data according to the property files or ignore the discrepancy.

Cubic Splines

Many properties are temperature dependent and represented by a discrete set of samples. When non-sampled values are needed, they are interpolated using standard cubic spline techniques, essentially fitting a piecewise-defined cubic polynomial to data (*see* chapter 3 of reference [50]). Standard cubic splines are defined so that first and second derivatives are continuous throughout the interpolation domain. This can lead to wiggles and over-shoots at *corners*, where the slope of the data changes abruptly. To avoid this problem Sage uses so-called *constrained* cubic splines that sacrifice second-derivative continuity in order to minimize overshoot at corners [36].

Thermal Solid Properties

The source for thermal solid data was mainly reference [68] and to a lesser degree reference [34].

In critical applications you should check over the conductivity and specific heat data pairs that appear in the display windows or output listing. In many cases, solid conductivity is not very predictable below 100K. Data from different sources for the same *nominal* material can differ by orders of magnitude presumably due to small variations in alloy composition. Solid specific heats, like conductivities, vary dramatically with temperature. However, there is much less

scatter in the data as reported from several sources. Apparently, specific heat is not quite as sensitive to variations in material composition as is conductivity.

Electrical Properties

The sources for electrical resistivity data were ASM handbook [4] and Cryogenic Engineering textbook [28].

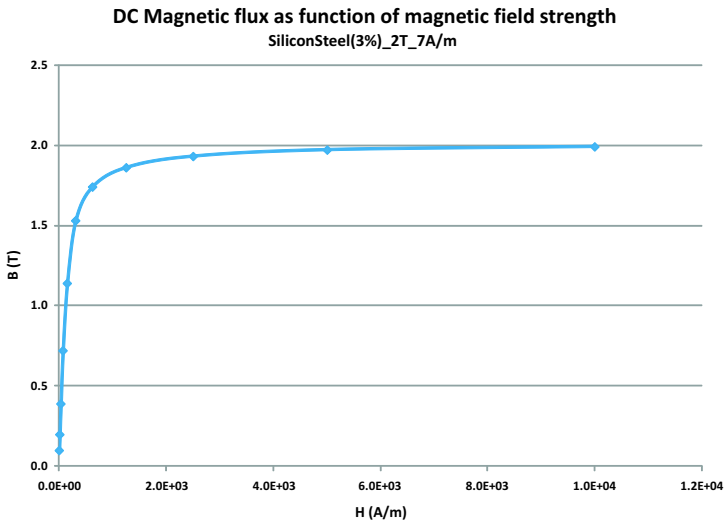
Ferromagnetic Properties

Soft Ferromagnetic Properties are generally available in the magnetic material literature or supplier data sheets, although usually not directly. It is rare to find these properties tabulated over a range of temperatures, in which case it is reasonably to assume that they all drop linearly to zero at the Curie temperature. Actually the relative permeability drops to the value one at the Curie temperature but that is essentially zero for practical purposes. Values are often reported in CGS rather than SI units, in which case the following conversion table may be useful.

$$\begin{array}{lcl} 1\ \text{T} & = & 10^4\ \text{Gauss} \\ 1\ \text{A/m} & = & 0.01257\ \text{Oersted} \end{array}$$

Often instead of saturation magnetization J_s a vendor will provide the value of the magnetic flux B at some large value of applied field H. The hope is that magnetization J is close to the saturation value J_s at the max B value reported. Note that the magnetic flux B is larger than the magnetization by the term $\mu_0 H$, which amounts to an extra 0.0126 T for every 10,000 A/m. For large applied fields this extra $\mu_0 H$ should be subtracted from the max B value when setting the value J_s.

Maximum relative permeability μ_r may be called many things and the value reported may not be the appropriate value for Sage. In Sage, the combination of μ_r and J_s determine the anhysteretic magnetization function M_a shown in figure 25.12. In the product literature if a B-H curve is also available it is a good idea to use any reported value of μ_r as an initial value and adjust it so the B-H curve produced by Sage matches the data over the entire range of H. You can do this by using a simple Sage model consisting of a soft ferromagnetic bar bounded by magnetic potential references to generate a set of B vs H data points over a range of steady (DC) H fields values. Sample model SoftFerroB-Hmap.stl shows how to generate such a map which is used to produce a plot like the one below:



Permanent Magnet Properties

Permanent Magnet Properties can be found in supplier data sheets. The $J(H)$ demagnetization curve shape parameter K_{jh} is unique to Sage and is entered according to a visual comparison of the actual demagnetization curve and the approximate curves shown in figure 25.14 of section 25.7.2.

Coercive Forces In the above properties there are two coercive forces H_{cJ} and H_{cB} both positive numbers, defined as follows. Start from a high positive magnetic field H where the polarization is near saturation. Then gradually reduce H to zero and below. H_{cB} is the absolute value of H at the point when the magnetic flux density B falls to zero. H_{cJ} is the absolute value of H at the point when the magnetic polarization J falls to zero. H_{cB} is always the smaller of the two because at $H = -H_{cB}$ the polarization is still positive at the value $J = \mu_0 H_{cB}$ (from definition (25.93) above).

For a soft ferromagnetic material H_{cJ} and H_{cB} are essentially the same. The difference between H_{cJ} and H_{cB} can be estimated by from the Taylor expansion approximation $H_{cB}/(H_{cJ} - H_{cB}) \approx dM/dH$ where $dM/dH = \mu_r - 1$ is usually a very large number for a soft ferromagnetic material. So $(H_{cJ} - H_{cB})$ is very small compared to H_{cB}.

Gas Properties

The sources for default ideal-gas data were thermophysical properties and P-v-T tabulations in references [68], [10], [69] and [44]. The value for R is simply the universal gas constant $\mathbf{R} = 8.314\text{E}3$ J/(kmol K) divided by the molecular weight of the gas. The source for tabular-gas data was NIST REFPROP software.

Entering Tables of State For TBspline3Gases the data-entry process is complicated by the need to enter five state tables — compressibility Z, internal-energy ε and transport properties c_p, μ and k. Further complicating things is the tricky nature of c_p, μ and k in the two-phase region as explained in section 15.1.2. TBSpline3Gas gases are identified by the name BSpline3 Gas in the Add menu item of the SCFProp.exe application. The data values are visible in the scroll box titled *Properties of Selected Item* at the right of the form, when you highlight a tabular equation-of-state gas type.

Entering state tables using the SCFProp application is not recommended. A better way is to use RefpropToSage.exe application as discussed below. However, for sake of completeness here is the procedure using SCFProp:

Click on the *Edit Properties* button below the scroll box, then on the appropriate button of the sub-dialog box. You should now see a file-open dialog enabling you to load a file containing the appropriate data table. You do not enter values one by one as you do for single-valued data or cubic-spline interpolation pairs. You load an entire file of values all at once. This is to allow you to create the property table(s) using a spreadsheet program like Excel or by some other automatic means that might be easier and more reliable than entering each value by hand. For symmetry sake, there is also a provision for saving the a gas's property data to a separate file, although you will probably not need to do this very often.

Property Table Data Format SCFProp expects to load a tab-delimited file of values. Excel normally produces such a format when saving as a text file (*.txt). The file must contain a rectangular table of data fields as follows:

first row a label such as "T \ v" identifying the nature of the first column and row of the table, followed by a number of specific volume v values in increasing order (important) with units (m^3/kg)

subsequent rows a temperature value T with units (K) followed by a number of property values under each v of the first row. The temperature values must be increasing in subsequent rows. The property values are up to you.

There is no limit to the number of v columns or T rows in the table. The easiest way to understand this is probably from the following, rather short, example — roughly what you might see on your Excel screen:

T \ v	0.005	0.006	0.007	0.008	0.009	0.01
3	5.8909	1.6582	9.11E-02	3.09E-02	3.47E-02	3.86E-02
3.5	5.2087	1.5932	0.26822	5.18E-02	5.82E-02	6.47E-02
4	4.7152	1.5693	0.42009	7.86E-02	8.84E-02	9.82E-02
4.5	4.3514	1.5679	0.54942	0.18123	0.12542	0.13935
5	4.0773	1.5788	0.66019	0.31901	0.20832	0.18872
5.5	3.8661	1.5964	0.75589	0.43549	0.32443	0.29551
6	3.7001	1.6174	0.8393	0.5353	0.42366	0.38945
6.5	3.5671	1.6397	0.91255	0.62182	0.50954	0.4708

You must adhere to this format closely, otherwise SCFprop may squawk at you.

Interpolation Details One way you can see if the resolution of your table is adequate is by creating a file of interpolated values by clicking the *interpolation details* button below the scroll box in the SCFProp form. When you click this button, a dialog opens allowing you to specify a range of v and T and number of points. The result is a tab-delimited file of values, similar to your original compressibility table. You can inspect this file with Excel.

Note that the interpolation values produced are the values interpolated between your data points, which may or may not include the original data points. So, for example, by interpolating a bunch of points in a small region of $v - T$ space you can see exactly how the bicubic spline routines are interpolating between your Z values in a small region of your original table.

In addition to interpolated Z and ε values, you also get interpolated values entropy s. Because of arbitrary constants of integration s values may be offset somewhat from values you might be comparing them against. Constant offsets do not matter.

28.2.2 Entering TBspline3Gas data with RefpropToSage

Windows application RefpropToSage.exe allows you to read native fluid (*.fld) or mixture (*.mix) files distributed with the NIST REFPROP software[41] and convert the data to TBSpline3Gas properties. What makes this possible is a dynamic link library refprop.dll, distributed with the REFPROP software. The application RefpropToSage.exe is part of the Low-T Cooler distribution but you will have to obtain the refprop.dll from NIST (http://www.nist.gov/srd/nist23.cfm) and save a copy in the same directory as RefpropToSage.exe or in the Windows DLL search path (Windows or Windows system directory).

Using RefpropToSage is a four step process. Figure 28.1 shows the graphical interface of the RefpropToSage conversion utility.

1. Open a REFPROP fluid or mixture file (File menu item).

2. Review and adjust the plots at the top of the form showing temperature and specific volume data values. These are the (v_j, T_k) interpolation data points at which Sage gas properties will match REFPROP values. You can make adjustments by changing the configuration constants in the edit boxes. Constants TminBuffer, TmaxBuffer, VminBuffer, VmaxBuffer adjust the spacing between the first and last v_j and T_k data values and the minumum and maximum REFPROP values. Constants TptsBelowTcrit, TptsAboveTcrit, TptsFarAboveTcrit control the number of points in three regions relative to the critical temperature, as explained below. Constants VptsBelowLiquid, VptsInMixed, VptsAboveVapor control the number of points in the liquid single phase region at the lowest temperature, the two-phase region just below the critical temperature and the vapor single-phase region at the lowest temperature, as explained below.

3. Click the Convert To Sage Gas button and inspect the plots in the interactive plotting dialog that pops up (Figure 28.2) to make sure there are

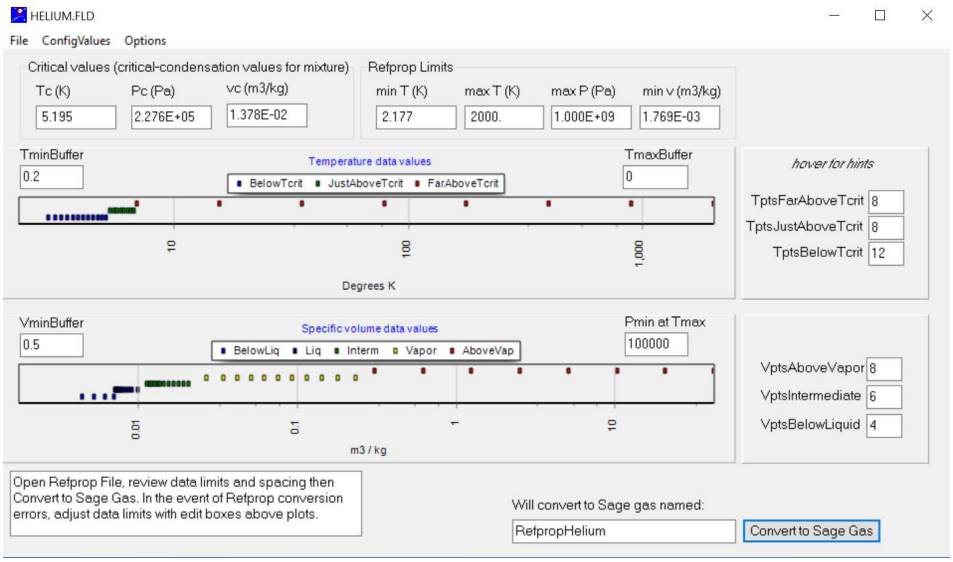


Figure 28.1: Interface for the RefpropToSage conversion utility.

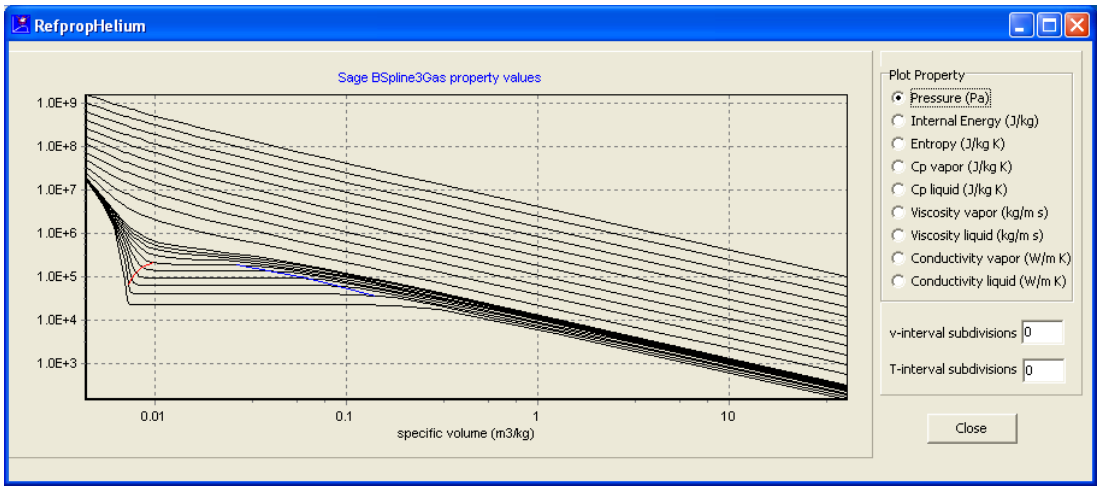


Figure 28.2: Dialog for interactively plotting gas properties after conversion to Sage format (TBSpline3Gas).

no anomalies. The Convert To Sage Gas disappears after the conversion process to declutter the form.

4. When you are satisfied with the result click the SaveBSpline3Gas.Data button in the bottom right corner to save the converted data to a Sage-format dta file. You may then use the SCFProp.exe program to append the resulting file to the gas-property data base file you use with the Low-T Cooler application (the one selected in the Options|Model Class dialog, Properties tab.

Plot Interaction For the dialog shown in Figure 28.2 mouse clicking on plot lines or T, v data points displays the numerical value of the isotherm or individual T, v value. Resize the plot by holding the left mouse button down and dragging between the upper left and lower right corners of a rectangle of interest. Undo by dragging in the opposite direction. Reposition the plot by holding the right mouse button down and dragging.

The edit boxes v-interval subdivisions and T-interval subdivisions to the right of the property values plot allow you to inspect interpolated values between tabulated data points. With zero subdivisions the plots just connect the tabulated data points. With subdivisions greater than zero the plots also show values interpolated between the data points. It is a good idea to set subdivision values to at least 1 so you can spot any spurious wiggles or other anomalies and possibly correct them by modifying the configuration constants (e.g. increasing the number of T or v data points).

Configuration constants (set in edit boxes) are automatically saved whenever you save the BSpline3Gas data to a file. The constants are saved as a companion to the original REFPROP fluid or mixture file in the form of a tab-delimited text file with the suffix Config appended to REFPROP file name and file extension .tab. For example, HeliumConfig.tab.

The ConfigValues menu item allows you to load configuration constants from any previously saved configuration file. You can also independently save a configuration constant file or restore default values.

Gas Mixtures are defined in REFPROP mixture files (.mix file extension). The REFPROP software contains a number of pre-defined mixtures and also allows you to define your own custom mixtures and save them to .mix files (Store button of the Specify Mixture Composition dialog). A REFPROP mixture file only contains information about the molar fractions of the mixture components and file names where their properties are stored. The RefpropToSage utility must be able to locate the individual fluids specified in a mixture file. It will look for them first in the same directory as the mixture file and then in one or more search paths specified in the Options menu item.

Mixture component fractions apply to the bulk fluid in a homogeneous state (e.g. all vapor or all liquid). But there is some confusion in the two-phase state where at a given temperature the fluid fractions in liquid and vapor phases

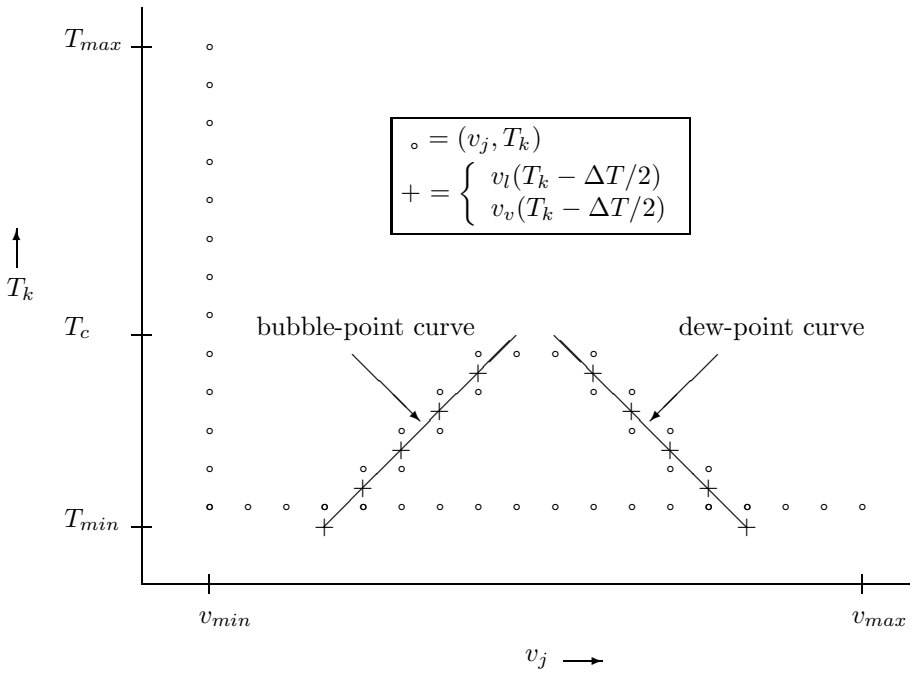


Figure 28.3: BSpline3Gas interpolation points.

generally differ from the bulk fractions. This two-stage variability limits Sage models using gas mixtures to processes where *there is no significant separation of the vapor and liquid downstream of the point where they are generated*. Sage can model vapor condensation or liquid boiling in a flow process as long as both phases flow together. Sage cannot model processes like fractional distillation where separating the vapor from the liquid is a fundamental part of the process. In fractional distillation the vapor of a two-phase equilibrium state is physically separated to a different location and condensed at a lower temperature. That physical separation resets the baseline component fractions.

Implementation Details The implementation of the RefpropToSage program is straight forward but not completely trivial. Special considerations are required for some properties in the two-phase region and sometimes at low-volume (high density) points out of REFPROP's range.

Interpolation Points To avoid problems it is necessary to choose the (v_j, T_k) interpolation points with care near the boundaries of the two-phase region. These boundaries are defined by the values of saturated vapor and liquid volumes $v_d(T)$ and $v_b(T)$ at the dew and bubble points available in REFPROP. The way this is done is illustrated somewhat abstractly in figure 28.3. v_j and T_k values are drawn in non-linear scale so the saturation curves $v_d(T)$ and $v_b(T)$ appear

as straight lines. The temperature points T_k are chosen as follows:

- First a number of points (TptsBelowTcrit) spaced by ΔT from $T_{min} + \Delta T/2$ to just below the critical temperature $T_c - \Delta T/2$. T_{min} is near the minimum value supported by REFPROP for the particular fluid.
- Followed by a number N points (TptsAboveTcrit) spaced by ΔT from $T_c + \Delta T/2$ to $T_c + (N - \frac{1}{2})\Delta T$.
- Followed by a number of points (TptsFarAboveTcrit) spaced in equal ratios from $T_c + (N + \frac{1}{2})\Delta T$ up to T_{max} (the maximum value supported by REFPROP)

The specific volume points v_j are then chosen with reference to the saturated liquid and vapor densities v_b and v_d tabulated at points offset by $\Delta T/2$ from the above T_k values, at the points $T = T_{min}, \ldots, T_c - \Delta T$:

- First a number of equal spaced points (VptsBelowLiquid) between a minimum value v_{min} and the lowest liquid volume $v_b(T_{min})$.
- Followed by the $v_b(T)$ values, $T = T_{min} + \Delta T, \ldots, T_c - \Delta T$
- Followed by a number of points (VptsIntermediate) spaced in equal ratios between just after the last point on the bubble line $v_b(T_c - \Delta T/4)$ to just below the critical volume v_c and the same number of points from just above v_c to just before the first point on the dew line $v_d(T_c - \Delta T/4)$
- Followed by the $v_d(T)$ values, $T = T_c - \Delta T, \ldots, T_{min} + \Delta T$
- Followed by a number of points (VptsAboveVapor) spaced in equal ratios from $v_d(T_{min})$ up to v_{max}, where v_{max} is the greater of twice $v_d(T_{min})$ and RT_{max}/P_{min}, for P_{min} a reasonable minimum pressure such as 1 atmosphere

Chosen in this way it is a simple matter to know which k correspond to temperatures below T_c and which j, k indices correspond to points in the two-phase region.

Saturation Pressure In theory and in REFPROP the slope of saturation pressure $(\frac{dP}{dv})_T$ in the two-phase region is much smaller than the slope in the liquid or vapor single-phase regions. For a pure fluid at a given temperature T the pressure remains constant as specific volume goes from the saturated liquid value $v_b(T)$ (liquid bubble point) to the saturated vapor value $v_d(T)$ (vapor dew point), as illustrated in figure 28.4. From the point of view of Sage convergence this is bad because the sudden condensation or evaporation of fluid that results from a small change in pressure in the saturation region might play havoc with the numerical solution.

To remedy this problem the RefpropToSage program takes some liberties with pressure in the saturation region — it adds a monotone decreasing smoothing function to the REFPROP pressure. This is illustrated in figure 28.5 which

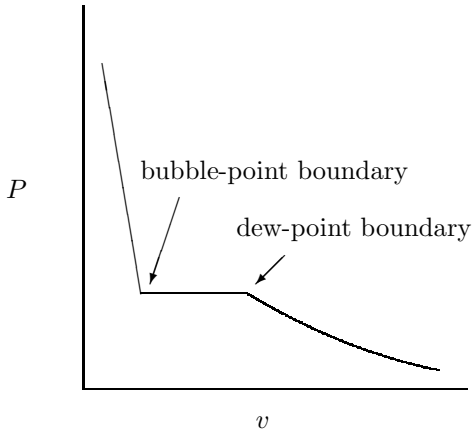


Figure 28.4: Typical pressure isotherm below T_c for a pure fluid.

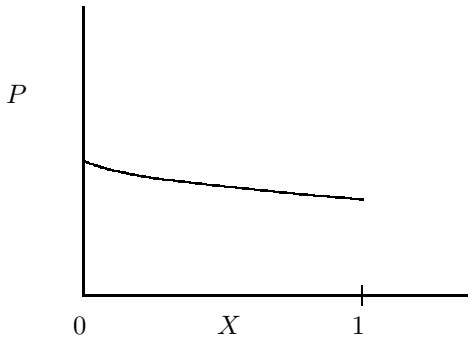


Figure 28.5: Smoothed pressure isotherm between $X = 0$ (last liquid volume v_l before the bubble point) and $X = 1$(first vapor volume v_v above the dew point). X is just the specific volume relative to the interval over which the smoothing occurs.

shows a smoothed pressure isotherm $P_s(X)$ between a dimensionless specific-volume $X = 0$ at liquid volume v_l just below the bubble point and $X = 1$ at vapor volume v_v just above the dew point. The smoothing function is the sum of a linearly decreasing pressure in the two-phase region plus a small positive step at $X = 0$ that rapidly decreases exponentially within the two-phase region. The purpose of this step is to smooth the P slope discontinuity at the bubble line just enough to ensure that $P(v)$ always strictly decreases just after the bubble line when interpolating $P(V)$ for an isotherm tabulated between the T_k data points, which otherwise might not be the case. The linear part of the pressure-smoothing function for $X > 0$ is

$$P_l(X) = (P(1) - Pd)\,(2X - 1) \tag{28.1}$$

where P_d is the dew-point pressure at $X = X_d$ (dimensionless value near 1 of dew-point volume v_d). This imposes a pressure correction that progressively decreases above the midpoint volume $X = 1/2$ and progressively increases below $X = 1/2$, with the average value zero. So the Pdv work across the two-phase region is not affected. The pressure difference $P(1) - P_d$ at dew point depends on the spacing of the volume data points which varies depending on the settings of RefpropToSage.

The exponentially decreasing step part is

$$P_e(X) = dPe^{-X/X_c} \tag{28.2}$$

where dP is the pressure step at $X = 0$, on the order the pressure difference between the T_k isotherm (current) and T_{k+1} isotherm (next), and X_c is the dimensionless spacing of the v_j values at the bubble line, a small number that determines how fast the step decays.

The final smoothed pressure is

$$P_s(X) = P(X) + P_l(X) + P_e(X) \tag{28.3}$$

where $P(X)$ is the REFPROP pressure. This pressure smoothing is embedded in the compressibility data table $Z(v_j, T_k)$ because pressure itself is not directly tabulated.

Internal Energy REFPROP calculates fluid internal energy ε in both single-phase and two-phase regions and RefpropToSage uses the REFPROP values directly. Prior to version 14, Sage shifted the REFPROP values so they were always positive, but this is no longer required. According to REFPROP documentation, "The absolute values of enthalpy, entropy, and energy at a single state point are meaningless. It is only the difference between two different state points that matter".

Transport Properties in the Two-Phase Region RefpropToSage encodes values for transport properties k, μ and c_p (conductivity, viscosity and specific heat at constant pressure) in the two-phase region according to the scheme

outlined in section 28.2.1. In the single-phase region RefpropToSage records the REFPROPtransport-property values directly.

in the interactive plotting dialog (Figure 28.2) you can plot the vapor and liquid transport properties separately. Vapor properties are defined for temperatures above the critical temperature or sub-critical temperatures with volumes above the bubble line. Liquid properties are defined only for temperatures below the critical temperature and volumes below the dew line. Because of the way the properties are stored you may see anomalies in the values of vapor properties near the bubble line or liquid properties near the dew line. You should be able to remove these anomalies by increasing the VptsInMixed configuration constant which has the effect of refining the v-grid near the critical specific volume.

Out-of-Range Single-Phase Properties In the single-phase region REFPROP defines all required properties in theory but can sometimes fail to return valid results at extremely high temperatures and low volumes (corresponding to extremely high pressure). When REFPROP fails RefpropToSage linearly extrapolates the failed property from previously calculated values at higher volumes. The assumption is that REFPROP is more likely to fail at low rather than high specific volumes so RefpropToSage tabulates properties in order of decreasing specific volume with the likelihood that values that may be needed for extrapolation will be available. RefpropToSage warns you when property extrapolation occurs. The hope is that the extrapolated values will fall at temperatures or pressures outside the range of interest.

If property extrapolation fails (no values available to extrapolate from) then the BSpline3Gas conversion will fail. Reducing the maximum tabulated temperature (increasing TmaxBuffer) or increasing the minimum tabulated specific volume (increasing VminBuffer) may solve the problem.

Mixture Difficulties Not all gas mixtures are equal when it comes to REFPROP. Property calculations in the two-phase region can fail to converge, especially if the component fluids have wildly different critical points. There may be temperatures in the two-phase region for which one or more of the component fluids is always in a supercritical single-phase state. For example, a mixture of nitrogen and water (humid air), where water can condense from vapor to mostly liquid at an elevated temperature where nitrogen remains always in the vapor phase, well above its critical temperature. On the other hand a fluid mixture can behave almost like a pure fluid. For example, an 80-20 nitrogen-oxygen mixture (dry air) where the two components condense more-or-less together.

Keep this in mind when running the RefpropToSage utility for gas mixtures. Success is not always guaranteed. You may have to play around with the configuration constants to avoid errors in the two-phase region or abandon your efforts entirely, hoping that a future release of the REFPROP DLL will solve the problem.

Chapter 29

SCFusion Deprecated Classes

Certain model classes have outlived their usefulness over the 30 year history of Sage. Often because of the evolution toward ever faster computers with more memory. Newer components replaced the functionality of earlier components and added new functionality. These model classes are still retained for backward file compatibility but gathered here to avoid cluttering the rest of the user's guide.

29.1 Deprecated Gas Classes

29.1.1 Tabular Gases

Previous tabular gas class TBSplineGas and TBSpline2Gas have been superseded by TBSpline3Gas. The original TBSplineGas derived its table of internal energy values from its input table of compressibility values and used zero-pressure limits for transport properties viscosity and thermal conductivity. TBSpline2Gas read the internal energy table as an input but continued to use zero-pressure limits for transport properties. With TBSpline3Gas viscosity and thermal conductivity are tabulated over a range of both pressures and temperature and the RefpropToSage utility was introduced for generating input property tables directly from Refprop DLL software.

29.2 Redlich-Kwong Gases

Gases based on the Redlich-Kwong equation of state predate any of the tabular gas classes. They were introduced as the first effort to add real gas behavior to Sage models in the days when ideal gases were the only option. Subsequent development of tabular gas classes (tabulated state properties) offered more

accurate gas properties with negligible impact on computational performance, essentially eliminating the need for Redlich-Kwong gases.

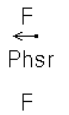
29.3 Deprecated Moving Parts

Phasor moving parts were introduced first in the early days of Sage as a natural evolution of the familiar complex-solution analysis of mechanical systems governed by linear equations. Time-ring moving parts (periodic time grid) followed later, with the ability to resolve higher harmonic components in the solution, as well as the sinusoidal component (equivalent to the phasor value). So models evolved that used both phasor and time-ring moving parts. This could lead to problems. For example, if you decided to replace a phasor piston built into a composite piston-cylinder component with a time-ring piston it was necessary to replace the entire composite component, often including a regenerator, clearance seal and appendix gap inside. Better to deprecate phasor moving parts and avoid the problem entirely, while also simplifying the Sage interface.

In the edit form, phasor force connectors are arrows labeled $\mathsf{F}_{\mathsf{phsr}}$. Pressure connectors are arrows labeled $\mathsf{P}_{\mathsf{phsr}}$. You are only allowed to connect a force to a force and a pressure to a pressure, and only when both have the same solution structure. That is, you cannot connected phasors to time rings.

29.3.1 Moving-Part Attachments

The following point and area-face attachments are deprecated in the model-component palette for phasor moving parts:

Icon	Purpose
	phasor negative-facing force attachment
	phasor positive-facing force attachment
	phasor negative-facing area attachment
	phasor positive-facing area attachment

The following volume-displacement attachments are deprecated in the model-component palette for variable-volume gas components:

Icon	Purpose
	phasor negative-facing volume displacement
	phasor positive-facing volume displacement

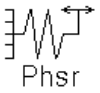
29.3.2 Moving-Part Variables

Phasor moving parts have a number of output variable found in all descendants:

F : (phasor, N) Net boundary force acting on the component resulting from all connections. A negatively directed boundary force cancels a positively directed force.

Wnet : (real, W) Time-averaged power delivered by all boundary forces. The mean value of net force times velocity.

29.3.3 Generic Spring



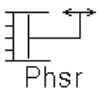
TPhsrSpr

Phasor springs solve displacement x from net applied boundary force F and stiffness input K using the equation

$$F = Kx \tag{29.1}$$

where variables F and x are complex numbers and solution for x is straightforward in terms of complex algebra.

29.3.4 Generic Damper



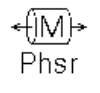
TPhsrDmp

Phasor dampers solve displacement x from net applied boundary force F and damping coefficient D using the equation

$$F = D\dot{x} \tag{29.2}$$

where $\dot{x}$ is velocity. In complex-amplitude notation $\dot{x}$ may be written $i\omega x$, from which it is easy to solve for x explicitly.

29.3.5 Reciprocating Mass



TPhsrRcp

Phasor reciprocating masses solve displacement x from the net resultant force F and mass M using Newton's equation of motion

$$F = M\ddot{x} \tag{29.3}$$

where $\ddot{x}$ is acceleration. In complex-amplitude notation$\ddot{x}$ may be written $-\omega^2 x$ so it is easy to solve for x explicitly.

Phasor reciprocators introduce new variables:

Mass : (real, kg) Reciprocating mass.

FF : (phasor, N) Forcing function. An applied body force specified within the model component itself rather than arising as a boundary force.

X : (phasor, m) Displacement from mean position.

29.3.6 Constrained Piston

TPhsrPis

Phaseor constrained pistons solve required forcing function F_f from net applied boundary force F_b, displacement x and mass M, using Newton's equation of motion

$$F_f = M\ddot{x} - F_b \tag{29.4}$$

where $\ddot{x}$ is acceleration.

The phasor constrained piston introduces new variables:

Xamp : (real, m) Displacement amplitude.

Xphase : (real, radians) Displacement phase angle.

FF : (phasor, N) Required forcing function to achieve Xamp and Xphase in light of applied boundary forces.

Displacement is given by two real variables rather than a single phasor so that amplitude or phase may be more conveniently optimized if need be. Physical displacement is

$$x = \mathsf{Xamp}\cos(\omega t + \mathsf{Xphase}) \tag{29.5}$$

29.3.7 Relative Moving Parts

All phasor relative moving parts have the variables:

Fneg : (phasor, N) Force acting on the component negative boundary.

Fpos : (phasor, N) Force acting on the component positive boundary. By default, Fpos is equal and opposite Fneg, or 180 degrees out of phase. (*see* equation (16.5))

Wnet : (real, W) Time-averaged power delivered by the boundary forces. For a relative spring, this is zero. For a relative damper this is the power absorbed by the damper.

29.3.8 Relative Springs

TPhsrRelSpr

Sage solves the negative coordinate x_{neg} as above and the positive coordinate x_{pos} from the applied boundary force F_{pos} and stiffness K using the equation

$$F_{pos} = K(x_{pos} - x_{neg}) \tag{29.6}$$

29.3.9 Relative Dampers

TPhsrRelDmp

Sage solves the positive coordinate x_{pos} from the applied boundary force F_{pos} and damping coefficient D using the equation

$$F_{pos} = D(\dot{x}_{pos} - \dot{x}_{neg}) \tag{29.7}$$

29.3.10 Simple Crank Piston

TGtCrankPis

This component has been superseded by the combination of a flywheel, simple crank linkage and reciprocating mass which together provide greater functionality.

A simple crank piston is a descendant of the time-ring constrained piston where Fourier-series displacement is now calculated as a dependent variable in terms of geometrical input variables pertaining to a simple crankshaft and connecting-rod kinematic mechanism.

Mathematically, displacement is given by

$$x = r\left(\cos\theta + \sqrt{\cos^2\theta + L^2/r^2 - 1}\right) - L \tag{29.8}$$

where r is crank-throw radius, L is connecting-rod length and θ is crank angle, defined in terms of angular frequency ω and a fixed phase angle ρ as

$$\theta = \omega t + \rho \tag{29.9}$$

In Sage, the above variables become:

Rcrank : (real, m) Crank-throw radius r.

Lratio : (real, dimensionless) Connecting-rod length divided by crank-throw radius L/r.

Phase : (real, radians) Crank-angle phase shift ρ.

Any or all of these may be optimized if need be.

29.3.11 Flexure Springs

These model components descend from the phasor spring components documented above. Except linear spring stiffness is now calculated as a dependent variable in terms of physical dimensions for spiral-arm flexure springs. And the mass of the spring is reflected in its attachment force. Displacement x is now solved from net applied boundary force F, stiffness K and effective mass m_e using the equation

$$F = Kx + m_e\ddot{x} \tag{29.10}$$

There are two phasor versions. The first version represents a stack of flexure elements located at one point. The second represents a stack of flexure elements separated into two parts by some axial distance, acting as a couple.

Flexure spring components date from the early days of Sage before the advent of CAD design software with built in FEA stress analysis and modal analysis that are much more accurate and general than Sage components. So Sage flexure springs have turned out to be an evolutionary dead end. But still, the mathematical formulation below for spring stiffness, peak stress, effective mass and resonant frequency provide some simple relationships that have proven useful for guiding CAD design and explaining inherent trade-offs between such things as resonant frequency, stress and mass that tend to get lost in the details of CAD modeling.

The theory behind Sage spiral arm flexures is an extrapolated case of helical coil spring theory. It applies, more or less, to flexures having spiral arms of uniform width and thickness. Since the analysis is only approximate, a number of calibration constants are included in the model. The idea is that you can reset their default values, if you want, based on a separate finite-element analysis of your particular geometry.

Flexure Stack

TPhsrFlx

This component is intended as a plug-compatible replacement for a generic phasor spring in a resonant-system model. You would use this component during an advanced stage of design where you wanted to actually design and optimize a spiral-arm flexure spring, rather than just work with its generic spring stiffness within a larger system. The flexure-stack component introduces new variables:

Ca, Cr, Cs, Cw, Cm : (real, dimensionless) Empirical calibration constants according to following theory.

E : (real, N/m^2) Elastic modulus E of spring material.

Rho : (real, N/m^2) Density ρ of spring material.

T : (real,m) Thickness t of a single flexure.

W : (real, m) Spiral arm width w.

D : (real, m) Active diameter D. Defined by circle where arms attach to outer rim.

M : (real, dimensionless) Number k of arms per flexure.

N : (real, dimensionless) Number n of flexures in stack.

Kr : (real, N/m) Radial stiffness K_r. Spring stiffness for deflection transverse to spring axis.

Sm : (real, N/m^2) Peak torsional stress τ_m.

wn : (real, 1/s) Resonant angular frequency ω_n. The operating frequency at or above which the analysis breaks down due to internal resonance.

Me : (real, kg) Effective mass M_e. That mass which moving with the same displacement amplitude as the flexure central attachment point produces the same reaction force.

Tn : (real, radians) Spiral arm total turning angle θ_T. The angle through which a radius vector rotates as it traverses a spiral arm from its inner to outer attachment points.

Flexure-Stack Couple

TPhsrFlxCpl

This component is a minor variation of the above flexure component (a descendant actually) that presumes the stack of flexures is divided into two groups spaced by some distance along the central axis, so as to produce a restoring moment to a tilting moment load. It, too, is a plug-compatible replacement for the generic phasor spring component. The angular stiffness within this component is intended for use in optimization constraints rather than some sort of transverse force connection to other model components. New variables introduced by the flexure-stack couple are:

Sc : (real, m) Couple separation distance S.

Ka : (real, N m / radian) Angular stiffness K_a.

You could optimize one of these subject to a constraint on the other.

Angular stiffness, defined as the restoring moment produced per unit tilt angle in radians, may be expressed in the form

$$K_a = \frac{1}{4} K_r S^2 \tag{29.11}$$

where K_r is the total radial stiffness of both groups of flexures combined and S is the distance separating the groups. A simple force diagram will convince you this is so.

Theory

The principle assumption is that spiral flexure arms behave at each point like the coils in a helical spring with the same radius of curvature. This is reasonable provided that (1) the arms actually turn through some minimum angle as they spiral out from the center (so as to produce load moments similar to those in helical springs), (2) they are longer than some minimum length (so that we may ignore the stresses due to slope and roll constraints at the end attachments) and (3) the radius of curvature does not change too abruptly.

Spiral Geometry A spiral flexure may be geometrically characterized by four independent variables:

D	active diameter
t	thickness
w	arm width
k	number of arms

If we take $r(\theta)$ to be the radius (distance from center to spiral-arm neutral axis) as a function of turning angle, then the spiral arm shape is defined by the geometrical requirement that $\Delta r/\Delta\theta = k(1+\epsilon)w/2\pi$. In English: r increases by k arm widths for each turn. Factor $(1+\epsilon)$ accounts for the gap between arms and the fact that the radial distance between edge-bounding involutes progressively deviates from perpendicular arm width as r decreases. If $dr/d\theta$ is as above then arm shape $r(\theta)$ must be

$$r(\theta) = r_0 + \frac{k(1+\epsilon)w}{2\pi}\theta \tag{29.12}$$

where r_0 is the inner radius. The total turn angle for each spiral arm is that angle θ_T for which $r(\theta_T) = D/2$. If we ignore r_0 then total turn angle is approximately

$$\theta_T \approx \frac{\pi D}{k(1+\epsilon)w} \tag{29.13}$$

In the following analysis, we will require that θ_T should always be at least $\pi/2$ or so. And we shall assume that ϵ is a design constant of minor consequence to be subsumed by calibration constants.

Axial Spring Stiffness From Wahl's classic tome of spring design (reference [70], equation 10-9, p. 129), we find the formula for axial deflection x per unit coil turning angle in a rectangular-bar helical spring under applied load F_x

$$\frac{dx}{d\theta} \approx 8.3\frac{F_x}{wt^3E}r^3 \tag{29.14}$$

This is for Poisson's ratio $= 0.3$, $t \ll w$ and $w \ll D$. Applying this locally to each point of our spiral arm gives total axial deflection

$$x = \int_0^{\theta_T}\frac{dx}{d\theta}d\theta = \frac{2\pi}{k(1+\epsilon)w}\int_0^{D/2}\frac{dx}{d\theta}dr = \frac{2\pi}{k(1+\epsilon)w}\frac{8.3F_x}{wt^3E}\int_0^{D/2} r^3 dr \tag{29.15}$$

Taking $\epsilon = 0.5$, for example, and carrying out the integration gives per-arm total axial deflection

$$x \approx 0.54\frac{D^4}{kw^2t^3E}F_x \tag{29.16}$$

and axial spring stiffness for the total flexure

$$K_a = \frac{kF_x}{x} \approx 1.8\frac{k^2w^2t^3E}{D^4} \tag{29.17}$$

Radial Spring Stiffness The calculation of radial stiffness proceeds along similar lines. Again from Wahl (equation 24-7 p. 280) we get the formula for radial deflection y per unit coil turning angle in a rectangular-bar coil spring under shear load F_y

$$\frac{dy}{d\theta} \approx 6.0 \frac{F_y}{tw^3 E} r^3 \tag{29.18}$$

Integrating this over the total turning angle and taking $\epsilon = 0.5$, exactly as before, gives per-arm total radial deflection

$$y \approx 0.39 \frac{D^4}{kw^4 tE} F_y \tag{29.19}$$

and radial spring stiffness for the total flexure

$$K_r = \frac{kF_y}{y} \approx 2.6 \frac{k^2 w^4 tE}{D^4} \tag{29.20}$$

This averages out some large fluctuations that occur each quarter turn, so it is important that we not apply it to turning angles below about $\pi/2$. It also ignores buckling instability.

Peak Stress Assuming the peak arm stress is pure torsion due to axial displacement, then according to Wahl (equation 10-7, p 128) it occurs at the outer end of the arms and is given by

$$\tau_m \approx 1.5 \frac{F_x D}{wt^2} \tag{29.21}$$

This is for $t \ll w$, $w \ll D$ and ignores any stress concentrations at the arm endpoint connections. Solving previous deflection equation (29.16) for F_x in terms of deflection x and substituting for F_x gives the equation for peak stress in terms of axial deflection

$$\tau_m \approx 2.8 \frac{kwtE}{D^3} x \tag{29.22}$$

Effective Mass Based on kinetic and potential energy arguments the spiral-arm effective reciprocating mass as a fraction of actual mass is the area-weighted displacement-squared divided by the center displacement squared, or

$$M_e/M = \frac{\int_0^1 x^2(r) 2\pi r \, dr}{\pi x_0^2} \tag{29.23}$$

The above equation presumes a spiral arm of unit outer radius and center deflection x_0. From the previous axial-deflection formula (29.16) we may take, $x(r) = x_0(1 - r^4)$ — highly nonlinear with most of the drop-off near the outer radius. Substituting into the previous equation and integrating gives the result

$$M_e/M = 0.53 \tag{29.24}$$

Actual mass M is just density ρ multiplied by the part of flexure disk area occupied by spiral arms, or $M = \rho(1/(1+\epsilon))t\pi D^2/4$. Taking $\epsilon = 0.5$, effective mass works out to

$$M_e = 0.28\rho t D^2 \tag{29.25}$$

Resonant Frequency Once we know axial spring stiffness and effective reciprocating mass, we may approximate resonant frequency as

$$\omega_n \propto \sqrt{K_a/M_e} \approx 2.5\frac{kwt}{D^3}\sqrt{E/\rho} \tag{29.26}$$

which is just the resonant frequency of the flexure with no attached mass. This is not quite correct, because the bending shape of the spiral arms loaded by uniformly distributed inertial forces (imposed by its mass distribution) is apt to differ from the shape under static deflection produced by a central loading and clamped outer rim. However, with the way these things usually go, the above formula is probably off by no more than a constant of proportionality which can be calibrated out with finite-element analysis.

Calibrated Formulae Introducing into our equations the effect of n, the number of flexures stacked together, the above formulae become:

axial stiffness

$$K_a = C_a n\frac{k^2w^2t^3E}{D^4} \tag{29.27}$$

radial stiffness

$$K_r = C_r n\frac{k^2w^4tE}{D^4} \tag{29.28}$$

peak stress

$$\tau_m = C_\tau\frac{kwtE}{D^3}x \tag{29.29}$$

effective reciprocating mass

$$M_e = C_m n\rho t D^2 \tag{29.30}$$

resonant angular frequency

$$\omega_n = C_\omega\frac{kwt}{D^3}\sqrt{E/\rho} \tag{29.31}$$

29.3.12 Linear Motors

As of Sage version 13, both phasor and time-ring linear motors are deprecated classes. They have been superseded by transducer components (section 25.6.1) which are electro-magnetic components that implement electrical current through a connection to an electrical circuit rather than as an input.

A linear motor provides a reciprocating force as a function of electrical current, which is an input. There is no detailed analysis of electromagnetic fields in physical geometries — all the physics is collapsed into a single electromagnetic force coefficient input. There are phasor and time-ring versions.

You can connect a linear motor to a reciprocating mass through a standard force connection, just like you would a spring or damper. The motor will drive the mass according to the current and force-coefficient inputs. In the time-ring case, a spring is mandatory to determine the equilibrium position of the reciprocating mass. Otherwise the solver will tend to drift around without converging. Generally, even a very weak spring will do.

Within a larger model, a linear motor can drive a reciprocating mass which, in turn, has a face area attached to the compression space gas. In the time-ring case, besides a spring attached to the reciprocating mass, there must also be a balancing area face on the other side of the reciprocating mass (opposite the compression-space attachment), attached to a buffer space. Otherwise there would be a great pressure-force imbalance on the piston which the spring would have difficulty offsetting. This inconvenience is the price for realism.

Phasor Motor

TPhsrMtr

In the phasor motor, the current and resultant force are sinusoidal. Its effects can be produced by the built-in forcing function input of a reciprocating mass, although the motor component also calculates the electrical resistance (I^2R) losses in the coil. The motor is defined by four inputs:

Icoil : (phasor, A) Electrical current I in the coil.

Cf : (real, N/A) Force coefficient C_f produced by electromagnetic interaction.

Rcoil : (real, ohm) Electrical resistance R of the coil.

Inorm : (real, A) Current scale used to normalize Icoil for internal Sage purposes.

with a new output variable (besides those inherited from the ancestral moving part component):

Wcoil : (real, W) Electrical resistance I^2R losses in the coil.

The force produced by the motor is just the product of force coefficient and current

$$F = C_f I \tag{29.32}$$

Time-Ring Motor

TGtMtr

In the time-ring motor the current is a Fourier series input and the force coefficient is a quadratic function of position. It can produce forces beyond the scope of even a Fourier series forcing function. One option would be to play around with the interaction between a nonlinear current and a nonlinear force coefficient, perhaps canceling each other out. Replacing the phasor-motor inputs Icoil and Cf, the time-ring motor has the following inputs:

FIcoil : (Fourier series, A) Electrical current I in the coil.

Cf0 : (real, N/A) Force coefficient C_{f0} at $x = 0$.

Xm : (real, m) Reference extension x_m.

Rp : (real, dimensionless) Force-coefficient ratio $R_p = C_f/C_{f0}$ at $x = x_m$.

Rn : (real, dimensionless) Force-coefficient ratio $R_n = C_f/C_{f0}$ at $x = -x_m$.

As with the analogous inputs for a nonlinear spring, you can read Cf0, Xm, Rp, Rn more-or-less directly, from a plot of C_f vs x generated either experimentally or computationally, or more precisely from a best-fit parabola to the function $C_f(x)$ over the intended operating range.

The force produced by the motor is still the product of force coefficient and current, as in equation (29.32), but now C_f is the quadratic function of position

$$C_f(x) = C_{f0}\left[1 + a(x/x_m) + b(x/x_m)^2\right] \tag{29.33}$$

where coefficients a and b are formulated in terms of inputs as

$$a = (R_p - R_n)/2 \tag{29.34}$$
$$b = (R_p + R_n)/2 - 1 \tag{29.35}$$

just as for the nonlinear spring stiffness function.

Relative Motors

TPhsrRelMtr

TGtRelMtr

The above linear motor components also come in relative-position versions — a relative phasor motor and relative time-ring motor. They are just like their absolute counterparts above, except they are connected between two moving parts instead of between a moving part and ground (fixed inertial frame). The same input variables and governing equations apply except that instead of applying a single force to one or more moving parts attached to a single position coordinate x they apply the force given by equation 29.32 to the moving part attached to endpoint coordinate x_{pos} and the negative of that force to the moving part attached to x_{neg}.

29.3.13 Motion Filters


TGtPhsr-MotionFilter


TPhsrGt-MotionFilter

There may be times when you want to connect a phasor moving part to a time-ring moving part and vice-versa. Normally that is impossible because the force connectors for the two are incompatible. Motion filters make it possible although there is some loss of physical realism as a result.

There are two classes of motion filters. They both have built-in phasor and time-ring force connections facing opposite directions but the orientations are reversed. There are no input or output variables.

A motion filter forces the displacement grid of the time-ring moving component it is attached to to have a sinusoidal displacement. To make that happen

it provides to that component whatever time-average or higher harmonic force components are required to make it so. The phasor moving component attached to the other side sees only the sinusoidal component of that force. In effect the time-mean and higher harmonic force components are filtered out and do not pass through a motion filter.

In physical terms a motion filter behaves like an infinitely heavy inertial mass when it comes to higher harmonics of force but like an infinitely light inertial mass when it comes to the fundamental sinusoidal force component. Obviously that is not physically realistic but it may be a convenient simplification provided it is a reasonable approximation to assume that the moving component connected to the phasor side moves sinusoidally.

These components are found in the *Phsr Moving Parts* component palette of the root model and also the *Springs and Dampers* component palette of phasor composite (piston-cylinder) components. This allows phasor moving parts in these locations to be connected to time-ring components in other parts of the model without the bother of replacing the phasor moving parts with time-ring counterparts.

Usage Restrictions

There are some things you cannot do with motion filters.

You cannot connect a time-ring constrained piston to a phasor reciprocator. That is because a time-ring constrained piston allows you to explicitly specify non-zero higher harmonics for the motion. But that conflicts with the sinusoidal displacement forced by a motion filter. On the other hand it is perfectly okay to connect a phasor constrained piston to a time-ring reciprocator.

If you connect a time-ring reciprocator driven by non-sinusoidal forces to a phasor reciprocator you must make sure the forces acting on the former determine its mean position. Generally that requires it to be anchored by a spring component.

29.3.14 Piston-Cylinder Composites

Phasor Free-Piston and Cylinder

TPhsrRcpCyl

This component adds a built-in phasor reciprocating mass intended as the basis for free-piston or free-displacer modeling. Choose this composite model when you want to solve for the limit-cycle sinusoidal response according to Newton's equation of motion. Since overstroking is a distinct possibility, with potential negative volumes in variable-volume gas domains, you will want to make sure you know the proper combination of area attachments, springs, etc., before choosing this model component.

Phasor Constrained Piston and Cylinder

TPhsrPisCyl

This component adds a built-in phasor constrained piston (or displacer) intended for first-approximation kinematic modeling or free-piston design work.

The sinusoidal piston motion is now specified as input. Overstroking is no longer a problem with this option. And by proper use of constraints you can set up an optimization problem to solve for the proper combination of area attachments, springs, etc., to make the piston run properly when you switch to free-piston mode.

Phasor Free-Piston and Free-Cylinder

TPhsrRcp-FreeCyl

This component is similar to the phasor free-piston component except it adds another built-in phasor reciprocating mass representing the moving cylinder or casing of a so-called "free cylinder" machine. The main reason for including the moving cylinder in this component is so it can pass the relative motion between the piston and cylinder to any annulus child components (*see* chapter 23) for purposes of calculating shuttle heat transfer. In free-cylinder machines the cylinder and piston can both have large amplitudes with respect to a fixed reference frame yet be moving almost in parallel so that the relative amplitude between them is small. So it is important to calculate shuttle heat transfer based on the relative piston motion, not absolute motion.

29.4 Deprecated Thermal Solids

29.4.1 Floating Isothermal Surface

TGxtFloat

This component was originally intended to be a faster version (in computational time) of the quasi-adiabatic surfaces of chapter 17 for use as the primary surface anchoring the gas temperature within a variable-volume space when the thermal solution within the solid was not important. It is equivalent to the thin-surface quasi-adiabatic model component in the limit of infinite thermal mass and zero axial conduction. It presumes the positive z surface is subject to time-varying sinusoidal heat flux, with zero mean. The negative z surface along with both x and y surfaces are presumed insulated. It has a discretized temperature distribution $T_s(x)$ with no time variation. T_s serves as both the solid interior temperature and the temperature at the positive z-surface. Sage implicitly solves T_s to achieve zero net (time-mean) surface heat flux.

As it turned out, floating isothermal surfaces were not noticeably faster than other quasi-adiabatic solids and the solid temperature would often drift to physically unrealistic values because of the lack of solid axial conduction. In some cases this led to convergence problems. So the use of floating isothermal surfaces is no longer recommended but they are still documented here for completeness and available in the software for special purposes.

A floating isothermal surface receives its initial temperature distribution T_s from its parent model component, but has no variables of its own to appear in display windows or output listings. Everything you need to know about surface heat transfer and temperature is available from the gas-domain component it is connected to.

29.4.2 Line Temperature Drop

TGxQdrop

This component was originally intended to provide a framework for including an extra temperature drop between an isothermal heat source (external boundary condition) and a distributed conductor representing a duct wall. The need for this component was largely eliminated with the introduction of the independent line heat source component, which provides an independent boundary temperature that can be optimized or recast as a dependent variable. Arguably the line temperature drop can also be used to provide an additional temperature drop between two conductive surfaces or distributed conductors but those components also provide their own temperature drops, with the advantage that they are solved according to physical principals rather than specified as inputs.

This component is a bit like a distributed conductor (section 17.7.2) except instead of solving a temperature distribution as a function of heat flux, physical dimensions and solid properties it simply imposes a fixed temperature drop between negative and positive y-faces, regardless of heat flux. The temperature drop is specified by independent cubic-spline input DeltaT, that may be recast as a dependent variable based on external heat-transfer analysis implemented in terms of user-defined inputs and variables.

A line temperature drop is born with two y-face heat-flow connectors and has no provision for customizing the connectors via child-model attachments.

The solution domain has an axial center-line temperature distribution $T_s(x)$, discretized on a spatial grid, beginning at the negative x end and ending at the positive x end. It also has two additional discretized temperature distributions $T_n(x)$ and $T_p(x)$, parallel to $T(x)$, centered in the negative and positive y faces. There is no axial (x directed) heat flux. Transverse (y directed) heat fluxes q_n and q_p at the negative and positive y faces are determined by adjoining components. The center-line temperature distribution is implicitly solved according to the condition that q_n and q_p are the same

$$q_p - q_n = 0 \tag{29.36}$$

which is the equivalent of a steady solid energy equation ignoring axial conduction. Temperatures T_n and T_p are explicitly solved as

$$T_n = T_s - \frac{\Delta T}{2} \tag{29.37}$$

and

$$T_p = T_s + \frac{\Delta T}{2} \tag{29.38}$$

where ΔT is input DeltaT, interpolated to the location in the grid where ΔT is evaluated.

Distributed conductor variables are:

Delta : (cubic spline, K) y-face temperature drop $T_p - T_n$.

QyNeg : (real, W) Net (x integrated) heat flow through negative y face.

QyPos : (real, W) Net heat flow through positive y face.

AEQy : (real, W) Available energy loss to y directed heat flow, according to internal generation formula (17.2).

AEdiscr : (real, W) Available energy discrepancy of above two losses compared to external generation as calculated by (17.1). (see section 17.3)

The sign of DeltaT is important. It should be positive for negative directed heat flow and vice-versa. In physical terms this means heat flows from hot to cold. If you get it backwards then this component will violate the second law of thermodynamics and produce a negative available energy loss AEQy.

Bibliography

[1] M. M. Abbott, *Cubic Equations of State: An Interpretive Review*, in *Equations of State in Engineering and Research*, Advances in Chemistry Series 182, (1978)

[2] B.J. Abu-Ghannam and R. Shaw, *Natural Transition of Boundary Layers — The Effects of Turbulence, Pressure Gradient, and Flow History*, J. Mech. Engineering Science, V. 22, No. 5, pp. 213–228, (1980)

[3] R. Akhavan, R.D. Kamm and A. H. Shapiro, *An Investigation of Transition to Turbulence in Bounded Oscillatory Stokes Flows, Part 1. Experiments*, J. Fluid Mech. vol. 225, pp. 395-422, (1991)

[4] ASM ready reference. Electrical and magnetic properties of metals, Materials Properties Database Committee, ASM International, (2000)

[5] A. Bejan, *Entropy Generation Through Heat and Fluid Flow*, Wiley-Interscience, (1982)

[6] 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)

[7] H.S. Carslaw and J.C. Jaeger, *Conduction of Heat in Solids*, Oxford at the Clarendon Press, (1959)

[8] J.E. Coppage and A.L. London, *Heat Transfer and Flow Friction Characteristics of Porous Media*, Chemical Engineering Progress, Vol. 52, No. 2, pp. 57-63, Feb. (1956)

[9] J.H. Dymond and E.B. Smith, *The Virial Coefficients of Gases*, Clarendon, (1969)

[10] E.R.G. Eckert and R.M Drake, *Analysis of Heat Mass Transfer*, McGraw-Hill, (1972)

[11] D.K. Edwards and I. Catton, *Prediction of Heat Transfer by Natural Convection in Closed Cylinders Heated from Below*, Int. J. Heat and Mass Transfer, Vol. 12, pp. 23–30, (1969).

[12] J.D. Felske, *Approximate Radiation Shape Factors between Two Spheres*, ASME Journal of Heat Transfer, Vol. 100, pp. 547–548, Aug. (1978)

[13] D. Gedeon, *Mean-Parameter Modeling of Oscillating Flow*, ASME Journal of Heat Transfer, Vol. 108, pp. 513–518, Aug. (1986)

[14] D. Gedeon, *A Cylinder Heat Transfer Model*, personal memorandum, (Tmodel2.pdf), December (1989)

[15] D. Gedeon, *An Approximate Cylinder Nusselt Number*, personal memorandum, (Tmodel3.pdf), December(1989)

[16] D. Gedeon, *Advection-Driven vs Compression-Driven Heat Transfer*, NASA report under PO C-23433-R, Task 2, (AdvectionVsCompressionSansFigures.pdf), May (1992)

[17] D. Gedeon, *Cylinder Heat Transfer Update*, personal memorandum, (TModel4.pdf), October (1995)

[18] D. Gedeon and G. Wood, *Oscillating-Flow Regenerator Test Rig: Hardware and Theory with Correlelations for Screens and Felts*, NASA-Lewis Contractor Report 198422, (1996)

[19] D. Gedeon, *Flow Streaming in Compliance Tubes*, personal memorandum, (FlowStreamingComplTubes.pdf) (1995)

[20] D. Gedeon, *Sage Tortuosity Reformulation*, personal memorandum (SageTortuosityFormulation.pdf), February (2006)

[21] D. Gedeon, *Random Fiber Data Reduction Results: porosities 85%, 90% 96%*, personal memorandum (NASARandomFiberDataReduction.pdf), January (2006)

[22] D. Gedeon, *Random Fiber Correlations with Porosity Dependent Parameters — Updated per New Tests at 90.9%, 93% and 96% Porosity*, personal memorandum (SageRandomFiberMasterCorrelationSept08.pdf), September (2008)

[23] D. Gedeon, *Random Fiber Correlations with Porosity Dependent Parameters — Updated per Revised Fiber Diameter for 90.9% Fecralloy Sample*, personal memorandum (SageRandomFiberMasterCorrelationNov08.pdf), November (2008)

[24] D. Gedeon, *Radiation Loss Modeling in Sage*, personal memorandum (RadiationLossModeling.pdf), January (2009)

[25] D. Gedeon, *Flow Streaming in Compliance Tubes: Stabilized Interior Formulation*, personal memorandum, (FlowStreamingComplianceTubesRevision.pdf) (2015)

[26] D. Gedeon, *Pulse-Tube Free Convection Vibratory Stabilization*, personal memorandum, (PtubeFreeConvectionVibratoryStabilization.pdf) (2015)

[27] D. Goldfarb, A. Idnani, *A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs*, Mathematical Programming, **27**, pp. 1–33, (1983)

[28] Hands B.A., *Cryogenic Engineering*, Academic Press, Chapter 9, (1986)

[29] Heames T.J., Uherka D.J., Zabel J.C., Daley J.G., *Stirling Engine Thermodynamic Analysis: A Users Guide to SEAM1*, Argonne National Laboratory, ANL-82-59, pp. 11–20, Sept. (1982)

[30] K.G.T. Hollands, *Natural Convection in Horizontal Thin-Walled Honeycomb Panels*, Trans. ASME J. Heat Transfer, Vol. 95, pp. 439–444, (1973).

[31] J.P. Holman, *Heat Transfer*, Fourth Edition, McGraw-Hill (1976)

[32] E. Fried, I.E. Idelchik, *Flow Resistance: A Design Guide for Engineers*, Hemisphere, (1989)

[33] D.C. Jiles and D.L. Atherton, *Theory of Ferromagnetic Hysteresis*, Journal of Magnetism and Magnetic Materials 61, North-Holland, Amsterdam, pp. 48–60, (1986)

[34] V.J. Johnson, *Properties of Materials at Low Temperature (Phase 1)*, Pergamon Press, (1961)

[35] W.M. Kays, A.L. London, *Compact Heat Exchangers, 3rd Edition*, McGraw-Hill, (1984)

[36] C.J.C. Kruger, *Constrained Cubic Spline Interpolation for Chemical Engineering Applications*, http://www.korf.co.uk/spline.pdf

[37] B.E. Launder and D.B. Spaulding, *Mathematical Models of Turbulence*, Academic Press, (1972)

[38] K. Lee, *A Simplistic model of Cyclic Heat Transfer Phenomena in Closed Spaces*, 18th IECEC, pp. 720–723, (1983)

[39] J.M. Lee, P. Kittel, K. D. Timmerhaus and R. Radebaugh, *Flow Patterns Intrinsic to the Pulse Tube Refrigerator*, International Cryocooler Conference, (1992)

[40] M.A. Lewis and R. Radebaugh, *Measurement of Heat Conduction through Metal Spheres*, in Cryocoolers 11, edited by R.G. Ross, Jr., Kluwer Academic/Plenum, pp. 419–425, (2001)

[41] Eric Lemmon, Mark McLinden, Marcia Huber, NIST Standard Reference Database 23, *NIST Reference Fluid Thermodynamic and Transport Properties — REFPROP*, U.S. Dept. of Commerce, (2002)

[42] Vitaly Leus and David Elata, *Fringing Field Effect in Electrostatic Actuators*, Technical report ETR-2004-2, Technion – Israel institute of Technology, (2004)

[43] I.F. Macdonald, M.S. El-Sayed, K. Mow and F.A.L. Dullien, *Flow through Porous Media – the Ergun Equation Revisited*, Ind. Eng. Chem. Fundam., Vol. 18, No. 3, pp. 199–208, (1979)

[44] NIST Standard Reference Database 12, *Thermophysical properties of pure fluids*, avaliable from http:\\www.nist.gov\srd, (1998)

[45] E. Oberg, F. D. Jones, *Machinery's Handbook*, Seventeenth Edition, The Industrial Press, (1964)

[46] J.R. Olson G.W. Swift, *Acoustic streaming in pulse tube refrigerators: Tapered pulse tubes*, LA-UR-96-3083, (1996)

[47] S. V. Pantakar, W. E. Ibele, W. J Koehler *Numerical Predictions of Turbulent Oscillating Flow — Initial Studies*, NASA-Lewis Progress Report, (1989)

[48] R.P. Peyret and T. D. Taylor, *Computational Methods for Fluid flow*, Springer-Verlag (1983)

[49] M. J. D. Powell, *A Fast Algorithm for Nonlinearly Constrained Optimization Calculations*, in: *Lecture Notes in Mathematics*, 630, Numerical Analysis (Proc. Biennial Conf. at Dundee, 1977), Springer-Verlag (1978)

[50] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes*, Cambridge, (1986)

[51] O. Redlich, J.N.S. Kwong, *On the Thermodynamics of Solutions. V. An Equation of State. Fugacities of Gaseous Soutions*, Chemical Reviews, V 44, p. 233, (1949)

[52] R. D. Richtmyer and K. W. Morton, *Difference Methods for Initial Value Problems*, Interscience, New York, (1967)

[53] P.A. Rios, *An Approximate Solution to the Shuttle Heat-Transfer Losses in a Reciprocating Machine*, ASME J. Eng. Power, pp. 177–182, (1971)

[54] R.G. Ross, Jr. and D.L. Johnson, *Effect of gravity orientation on the thermal performance of Stirling-type pulse tube cryocoolers*, Space Cryogenics Workshop, Girdwood, Alaska, September 2003.

[55] H. Schlichting, *Boundary-Layer Theory*, Seventh Edition, McGraw-Hill, (1979)

[56] E. Schmidt, *Thermodynamics*, Dover, (1966)

[57] J.R. Seume and T.W. Simon, *Oscillating Flow in Stirling Engine Heat Exchangers*, 21st IECEC, (1986)

[58] J.R. Seume, *An Experimental Investigation of Transition in Oscillating Pipe Flow*, Ph.D. Thesis, University of Minnesota, (1988)

[59] J. Seume, G. Friedman and T.W. Simon *Fluid Mechanics Experiments in Oscillatory Flow*, NASA CR-189127, Lewis Research Center, (1992)

[60] T.W. Simon and J.R. Seume, *A Survey of Oscillating Flow in Stirling Engine Heat Exchangers*, NASA 182108, (1988)

[61] T.W. Simon, M. Ibrahim, M. Kannapareddy, T. Johnson, G. Friedman, *Transition of Oscillatory Flow in Tubes: An Empirical Model for Application to Stirling Engines*, 27th IECEC, (1992)

[62] R. Siegel and J.R. Howell, *Thermal Radiation Heat Transfer*, McGraw-Hill, (1972)

[63] E.M Sparrow and R.D. Cess, *Radiation Heat Transfer*, Brooks/Cole, (1970)

[64] G.W. Swift, *Thermoacoustic Engines*, J. Acoust. Soc. Am., **84** (4), pp. 1145–1180, (1988)

[65] G.W. Swift, *Thermoacoustics: A Unifying Perspective for some Engines and Refrigerators*, Fourth draft, LA-UR-99-895, 1999

[66] G.W. Swift and S.N. Backhause, *The pulse tube and the pendulum*, J. Acoust. Soc. Am, Vol. 126, No. 5 November 2009, pp. 2273–2284.

[67] G.W. Swift and S.N. Backhause, *why High-Frequency Pulse Tubes Can Be Tipped*, Cryocoolers 16, edited by S.D. Miller and R.G. Ross, ICC Press (2010), pp. 183–192.

[68] Y.S. Touloukian and C.Y. Ho, *Thermophysical Properties of Matter*, Purdue Research Foundation, Plenum Publishing Corp., (1970)

[69] N. B. Vargaftik, *Tables on the Thermophysical Properties of Liquids and Gases*, Second Edition, Wiley, (1975)

[70] A.M. Wahl, *Mechanical Springs*, McGraw-Hill (1963)

Index

A

T