Chapter 1

Introduction

Ravel is a data analysis program which is an alternative to spreadsheets and existing business intelligence programs (Tableau, Power BI, etc.). It has two key features that distinguish it from all other programs in this space:

- The Ravel, a visual representation of multidimensional data that is far easier to use than Pivot Tables; and

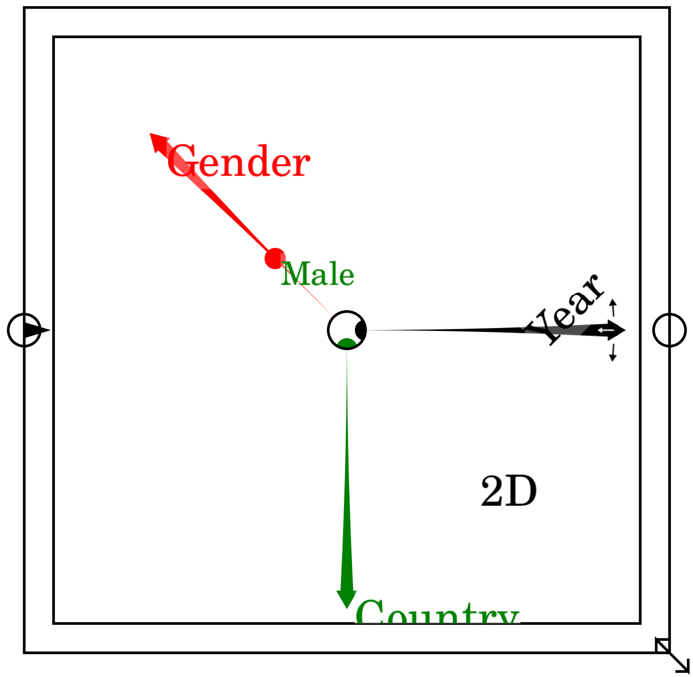
- Equations that are written as flowcharts, rather than the obscure cell references of spreadsheets.

It also includes the system-dynamics engine *Minsky*.

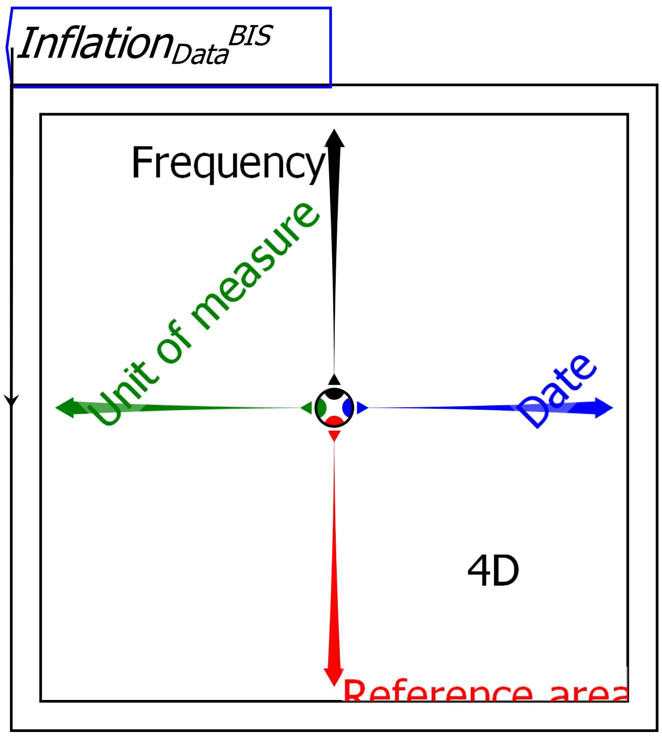
Unlike spreadsheets, which are fundamentally limited to two dimensions (rows and columns), *Ravel* supports an effectively unlimited number of dimensions. *Ravel*'s key feature—also called a Ravel—is a visual tool for manipulating and analysing multidimensional data, which can handle as many dimensions as your data contains. By contrast, to handle more than 2 dimensions in a spreadsheet, the go to tool is a *Pivot table*, which provides some sort of control over the slice through the multidimensional *hypercube*. Needless the say, pivot tables are not for the faint of heart!

Ravel is primarily focussed on numerical data, although it is possible to analyse purely symbolic data in "counter mode".

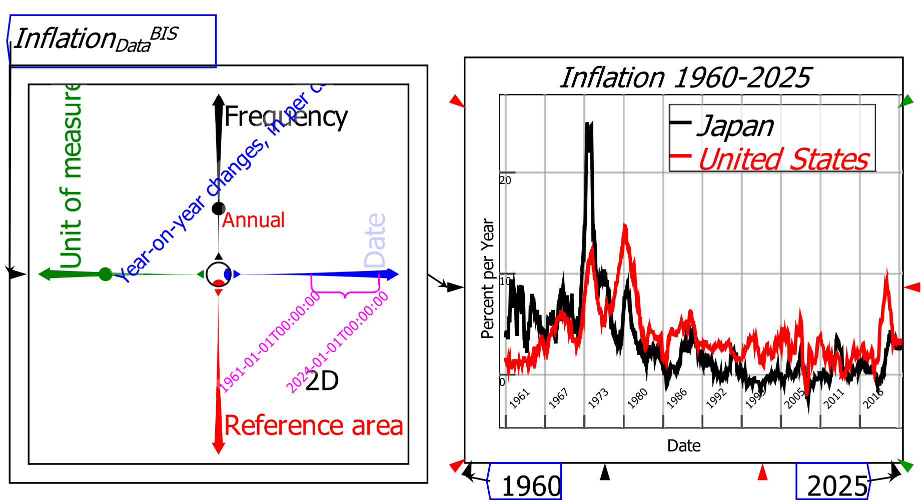
This is a blank Ravel—a Ravel with no data attached to it. The axes are indicative examples, and will be replaced by the dimensions of your data, once you attach a data parameter to the Ravel:



To use a Ravel, you first need to import a data file–at present this must be a CSV file (other data sources will be added in later releases). Once the data is imported, the data object can be attached to a Ravel. This is a Ravel with data attached:

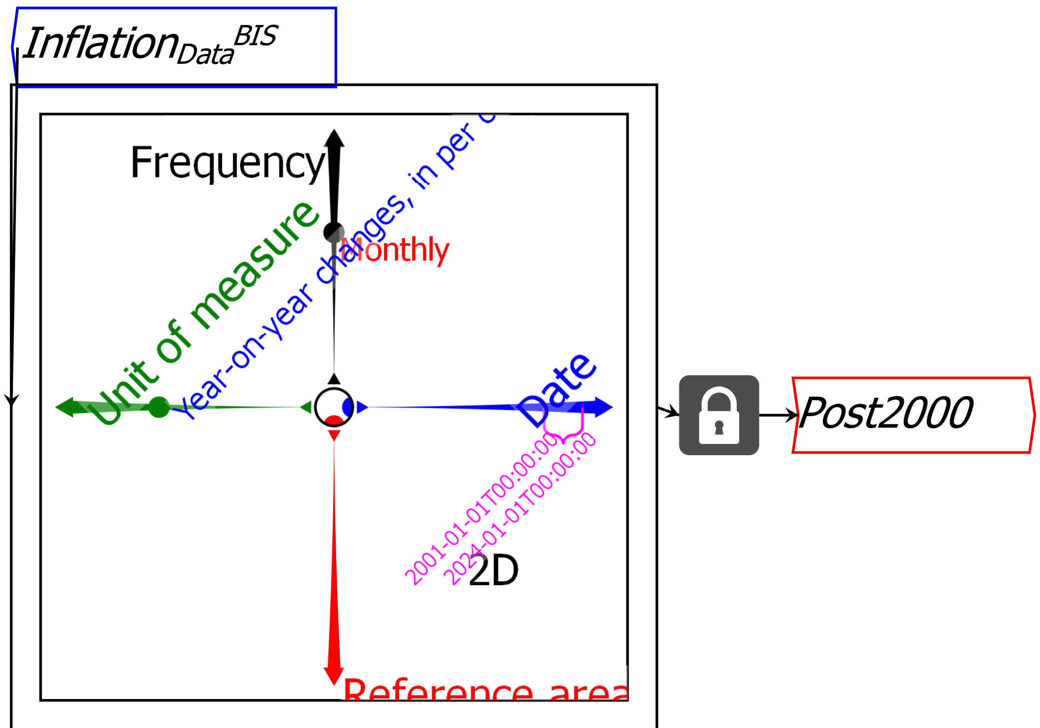


There are many ways to manipulate and display data directly from a Ravel. This is a Ravel with data attached and selected for graphing: the "Year on Year Changes" data is selected from the Unit of Measure axis; two countries (Japan and the United States) are selected from the Reference area axis via the *Pick axis slices* command; Monthly data is chosen from the Frequency axis; and Calipers are applied to the Date axis to select data from 1960 till 2024.

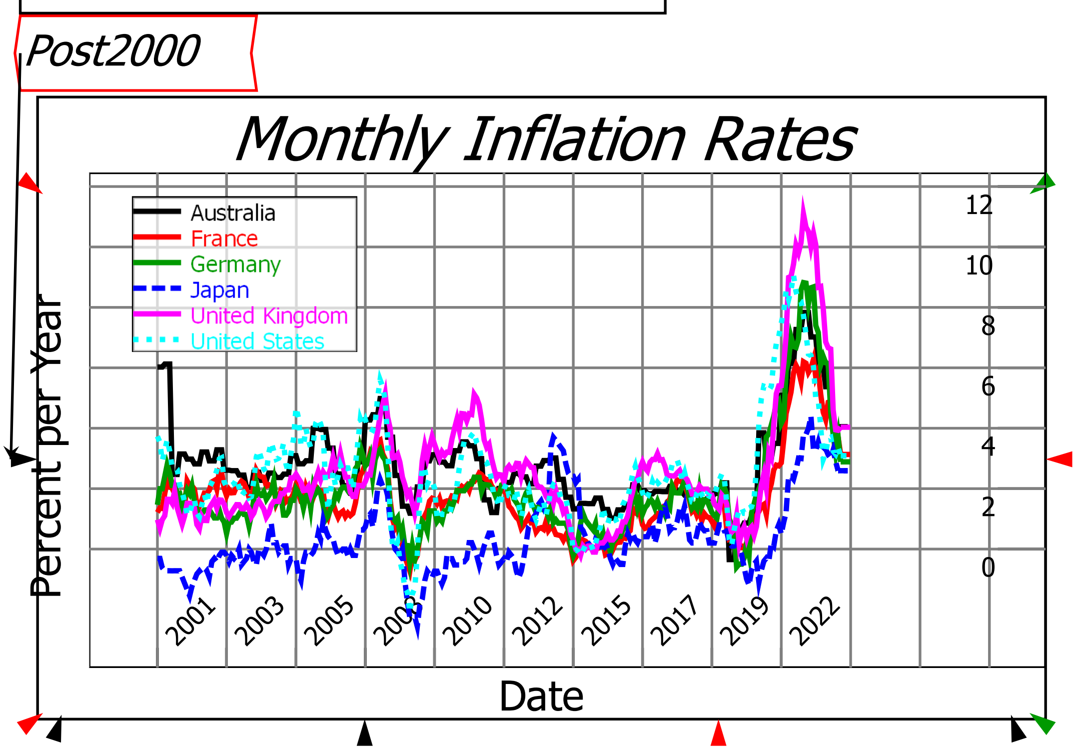


Ravel the object itself makes it far easier to drill down into and visualise data than using either a spreadsheet, or the Pivot Tables that standard Business Intelligence program use.

Ravel the program enables easy analysis of data using self-documenting flowchart formulas. This is a Ravel with data selected–for six countries, on the annualised monthly inflation rate, for dates from January 2001 till January 2024–and assigned to a variable ("Post2000").



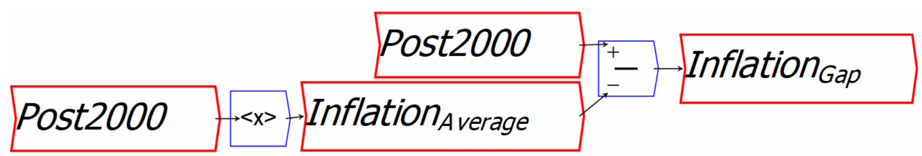
$Inflation_{Data}{}^{BIS}$
Frequency
Monthly
Unit of measure
Year-on-year changes, in per c
Date
2001-01-01T00:00:00
2024-01-01T00:00:00
2D
Reference area
Post2000



Post2000
Monthly Inflation Rates
Australia
France
Germany
Japan
United Kingdom
United States
12
10
8
6
4
2
0
Percent per Year
2001
2003
2005
2008
2010
2012
2015
2017
2019
2022
Date

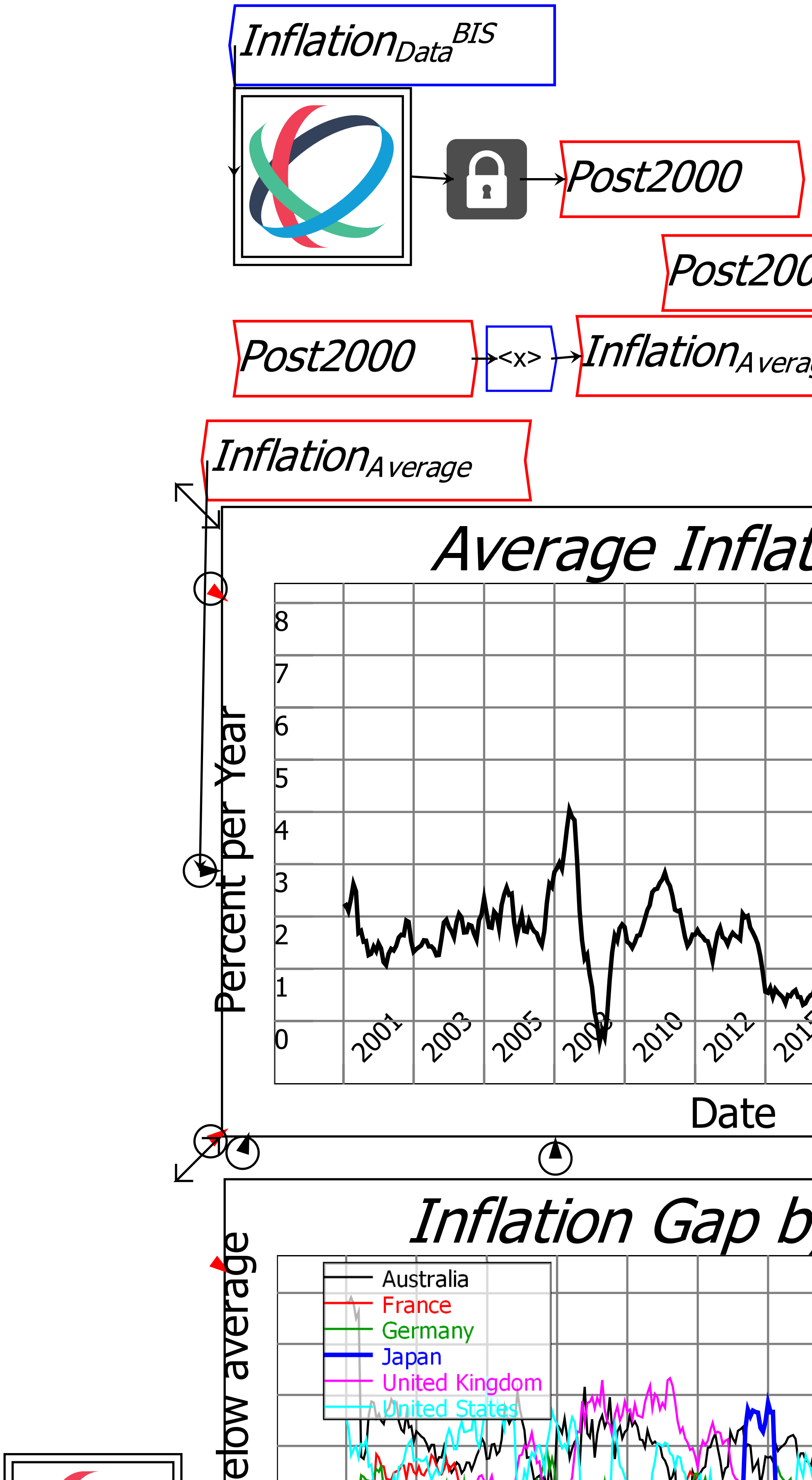
The data is analyzed by (a) working out the average inflation rate for the selected countries, and (b) subtracting the average from the actual inflation rate for each country.

The average inflation rate is calculated using the formula shown below, where the  operator works out the average inflation rate over time for the six countries in the variable *Post2000* to generate the variable $Inflation_{Average}$. This average is then subtracted from the country-specific data in *Post2000*:



This one formula is applied to every country in the Ravel (six countries in this case) and every quarterly data point (80 quarters). Doing the same analysis with a spreadsheet would require writing an obscure cell reference formula and replicating it across 480 cells.

This example shows average inflation outcomes for the six countries, and the deviation of each of them from the average.



InflationData BIS
Post2000
Post200
Post2000
<x>
InflationAvera
InflationAverage
Average Inflat
Percent per Year
8
7
6
5
4
3
2
1
0
2001
2003
2005
2010
2012
Date
Inflation Gap b
elow average
Australia
France
Germany
Japan
United Kingdom
United States

This illustrates the capacity of Ravel to rapidly provide insights from data–in this case, that the best-performing country during the post-Covid inflation was Japan, and the worst performing were the USA and UK. This is noteworthy, because both the USA and UK sharply increased interest rates with the intention of reducing inflation, while Japan kept its interest rate constant. Perhaps then, interest rates aren't effective at controlling inflation rates?

These examples are drawn from economics, mainly because *Ravel*'s inventor is an economist (and a contrarian one at that). But *Ravel* can analyze any data you give it—marketing data, scientific data, production data, whatever. It can also handle enormous data sets, far larger than are manageable with Excel.

Ravel can handle sparse data—it requires 16 bytes per data point, so for example, a 16GB computer will technically be able to handle up to a billion data points, although in practice it is unwise to exceed more than half that. There is no limit to the number of rows and columns it can ingest, subject to the overall limit due to available memory, in contrast to Excel's limits of 1,048,576 rows by 16,384 columns. One other limit to keep in mind is that the size of the datacube (product of sizes of all the dimensions) cannot exceed 1.8×10^{19}. This is something to bear in mind when importing data, as the datacube size increases exponentially with the number of dimensions.

We hope these examples show how easy it is to turn data into information using Ravel.

Chapter 2

Getting Started with *Ravel*

2.1 System requirements

Ravel is a proprietary program for data analysis which currently only runs on Windows (version 8 and later). It consists of a plugin for the *Ravel application*, also known as *Minsky*, which is an open source program available for Windows, Mac OS X,and various Linux distributions. After obtaining Minsky (from the Patreon Store[1]), you will need to select "Upgrade" from the File menu to install the latest version of the Ravel plugin.

Some components of the interface are specific to *Minsky*. This "Getting Started" guide focuses on the components used by *Ravel*. For a getting started guide to see *Minsky*.

2.2 Getting help

Press the F1 key, or select "help" from the context menu. Help is context-sensitive. If you press F1 while the mouse is hovering over a widget–for example, the addition block–then the help window will appear with instructions on how to add elements together.

2.3 The Ravel Interface

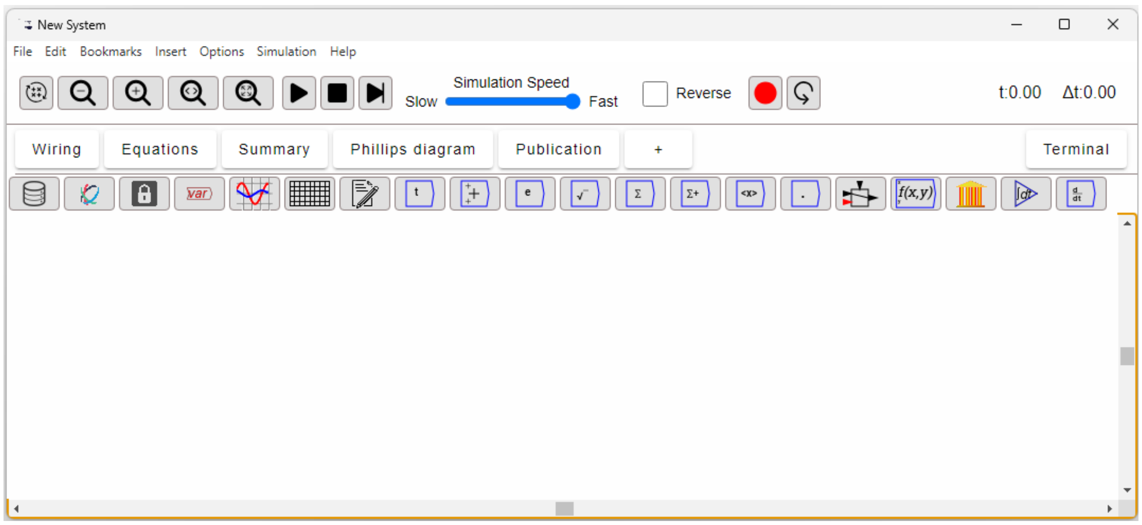
There are 5 main components to the *Ravel* interface:

1. The Menus;
2. The Operations controls;
3. The Tabs: The Design Canvas and its documentation companions Equations, Summary, Phillips Diagram, and Publication(s);

[1] https://www.patreon.com/hpcoder

4. The Design icons (sometimes called Operators or Widgets in this help file); and

5. The Design Canvas.



2.3.1 Menus

The menu controls the basic functions of saving and loading files, default settings for the program, etc. These may alter as the program develops; the current menu items (as of June 2024) are:

File Edit Bookmarks Insert Options Simulation Help

2.3.1.1 File

About Tells you the version of *Ravel* and/or *Minsky* that you are using.

Upgrade Will upgrade you to the latest release of Minsky, and if subscribed for Ravel updates, also install the latest version of Ravel.

New System Clear the design canvas. If you have made changes and haven't saved them, you will be prompted to save before the canvas is cleared.

Open Open an existing *Ravel* or *Minsky* file. *Ravel* files have the suffix of "rvl", *Minsky* files have the suffix of "mky". Technically they are identical, with the only difference being that RVL files contain a Ravel while MKY files do not.

Recent Files Provides a shortcut to some of your previously opened *Ravel* (or *Minsky*) files. *Ravel* files have the suffix RVL and *Minsky* files have the suffix MKY.

Library Opens a web-based repository of models for the *Minsky* simulation system.

Save Save the current file.

Save As Save the current file under a new name.

Insert File as Group Insert a *Ravel/Minsky* file directly into the current model as a Group.

Dimensional Analysis As processing networks get more complex, it is useful to attach units to the data. Dimensional analysis then checks that the mathematical operations are allowable—rejecting, for example, adding feet to metres.

Export Canvas as Export the current canvas in svg, pdf, eps, tex, or m format for use by other programs. The current canvas (which varies depending on which Tab you have open) can be exported in a number of different formats:

- **SVG** This is a "vector graphics" format which can be inserted into word processing (such as Word or OpenOffice) or presentation (Powerpoint etc.) documents.
- **PDF** This saves the currently displayed canvas as an Adobe Acrobat file .
- **EMF** This saves the canvas in an enhanced form of the WMF (Windows Metafile) vector graphics standard, for use in documentation programs like Powerpoint and Word. This is only available in the Windows version of Minsky.
- **Postscript** This saves the canvas in an encapsulated form of PDF.
- **Portable Network Graphics** This saves the canvas in a bitmap file (PNG) for use in paint and photo programs, etc.
- **LaTeX** This exports the equations in a model in a mathematical formatting language called LaTeX. This file can be imported into mathematics programs like MathType to document the mathematical logic in your model. If you are a LaTeX user yourself, you can load this directly into your preferred LaTeX editor.

  If your LaTeX implemention doesn't support breqn style, untick the wrap long equations option, which can be found in the preferences panel under the options menu.
- **Matlab** This exports the model as an ".m file" for importing into the algebraic program Matlab. This enables the analysis and simulation of your model in a MatLab compatible system, such as Matlab itself `https://en.wikipedia.org/wiki/MATLAB` or Octave `http://www.gnu.org/software/octave`.

Log simulation This *Minsky*-specific command outputs the results of simulated variables into a CSV data file for later use in other programs.

Recording This *Minsky*-specific command records the states of a model as it is being built for later replay.

Replay recording This *Minsky*-specific command replays a recording of model states.

Quit Exit the program. *Ravel* will check to see whether you have saved your changes. If you have, the program will close; if not, you will get a reminder to save your changes.

Debugging use Items under the line are intended for developer use, and will not be documented here. All programs have bugs! If you experience one, please report it via the SourceForge ticket system.

Redraw Redraw may be useful if the screen gets messed up because of a display bug. For example, a bug could cause items on the canvas to be scaled differently. Redraw could overcome this problem without requiring you to exit the program.

2.3.1.2 Edit

Undo and Redo allow you to step back and forward in your editing history. If you undo a few edits, and then change the model at that point, the undo history is then reset to commence with your new edit. *Ravel* supports the standard Windows shortcuts of control-Z for undo and control-Y for redo.

Cut/copy/paste Selecting, or lassoing a region of the canvas will select a group of icons, which will be shaded to indicate the selected items. Wires joining two selected items will also be selected. Note that, compatible with X-windows, selecting automatically performs a copy, so the copy operation is strictly redundant, but provided for users familiar with systems where an explicit copy request is required. Cut deletes the selected items. Paste will paste the items in the clipboard as a Group into the current model. *Ravel* supports the Windows-standard shortcut keys of control-C for copy, control-X for cut (which deletes the entity at the current location and creates a copy for pasting elsewhere) and control-V for paste.

Group selection Create a Group using the contents of the selection. Groups allow you to organise more complicated systems components into aggregated modules that make the overall system more comprehensible. In *Ravel*, groups can be used to, for example, collect all the file importing operations into a Group, thus removing the details of these operations from the top level view. This reduces the complexity of a canvas, which can make it easier for both the developer and viewers to focus on the analysis that the document is actually doing.

Dimensions This invokes the Dimensions dialog box, which lists the Dimensions (Ravel axes) in a document, their type (string, time, value), and their units or formatting.

Remove Units removes all units from the system, effectively disably dimensional analysis.

Auto Layout and Random Layout change the layout of your model. These commands are still under development, so are currently not recommended unless you have all items sitting on top of each other. If you do accidently select one of these commands and mess up your model layout, simply press "undo".

2.3.1.3 Bookmarks

Bookmarks store the location and scaling of a model for future reference. This is an alternative to grouping as a means to organize a model. To create a bookmark, either:

- Move the canvas and zoom it to the level at which all the items you wish to bookmark are visible. Then click on the Bookmark menu and choose "Bookmark this position". Give the Bookmark a name and it will be added to this menu. To move to that location in the model, click on this Bookmark name. Bookmarks can be deleted using the "Delete bookmark" sub-menu. Or

- Insert a text box at the upper left location and click the Bookmark? checkbox. The Short Description of the textbox is used as the name for the Bookmark. Both the upper left location shown on the canvas and the zoom level in operation at the time of the Bookmark's creation are saved, and restored when that Bookmark is selected from the Bookmarks menu.

2.3.1.4 Insert

This menu contains all the mathematical operator blocks used in *Ravel*, and enables you to place those operators on the Canvas. You can get the same effect by clicking on the Design Icons. A Ravel can be inserted from this menu, as can Plots, Sheets and *Minsky*-specific tools like Godley Tables.

2.3.1.5 Options

The options menu allows you to customise aspects of *Ravel* and *Minsky.* At the moment most options pertain to *Minsky* rather than *Ravel*, but this will change as *Ravel* increases in complexity. In this section of the manual we ignore options pertaining only to *Minsky*; these are covered in *Introduction to Minsky*

Preferences

- Number of recent files to display — this determines how many previously edited files are displayed on the Recent files menu.
- Wrap long equations in LaTeX export. If ticked, *Ravel* & *Minsky* will use the LaTeX "breqn" package to produce nicer looking automatically line-wrapped formulae. Because not all LaTeX implementations are guaranteed to support breqn, untick this option if you encounter problems.
- select a font for variable names, parameter names, etc.

Background colour — select a colour for the wiring canvas. The default is white.

2.3.1.6 Simulation

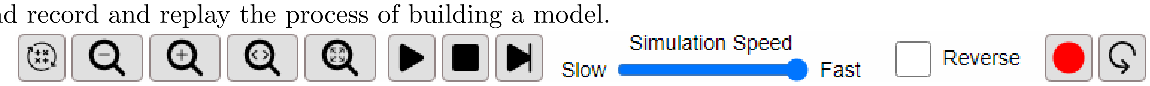
These commands are specific to *Minsky* and control how a model is simulated. They are covered in the *Minsky* component of this manual.

2.3.1.7 Help

Provides an in-program link to this manual. Note that pressing F1 will also launch help windows in a context-sensitive way. That is, it will open the relevant help section for whatever object the mouse is currently hovering over. Similarly, each item on the canvas has a help menu item in the context menu relevant for that item.

2.3.2 The Operation Controls

These controls cause *Ravel* to recalculate a model; zoom in and out on the canvas; run and control the speed and direction of a *Minsky* simulation model; and record and replay the process of building a model.



2.3.2.1 Recalculate button

The recalculate button computes the values of all variables in a model. *Ravel* will periodically recalculate a document, but this is a useful option if you wish to immediately see the results of a calculation.

2.3.2.2 Zoom buttons

The Zoom buttons and zoom in and out on the wiring canvas. The same functionality is accessed via the mouse scroll wheel. The reset zoom button resets the zoom level to 1, and also recentres the canvas. The Zoom to Fit button zooms the model so that it just fits in the current canvas window.

2.3.2.3 Run Buttons

These commands are specific to *Minsky* and are covered in the *Minsky* component of this manual.

2.3.2.4 Speed slider, time direction and model recording

These commands are specific to *Minsky* and are covered in detail in the *Minsky* component of this manual.

The speed slider controls the rate at which a model is simulated. The "Reverse" checkbox causes the simulation to run backwards rather than forwards in time.

The Record and Replay buttons respectively record the process of creating a model, and replay that process.

2.3.2.5 Simulation time

In the right hand top corner is a textual display of the current simulation time t, and the current (adaptive) difference between iterations Δt. This information is specific to a *Minsky* simulation model.

2.3.3 Tabs

2.3.3.1 Wiring Tab

The Wiring Tab is where a *Ravel* document is designed: see *Design Canvas* for its components.

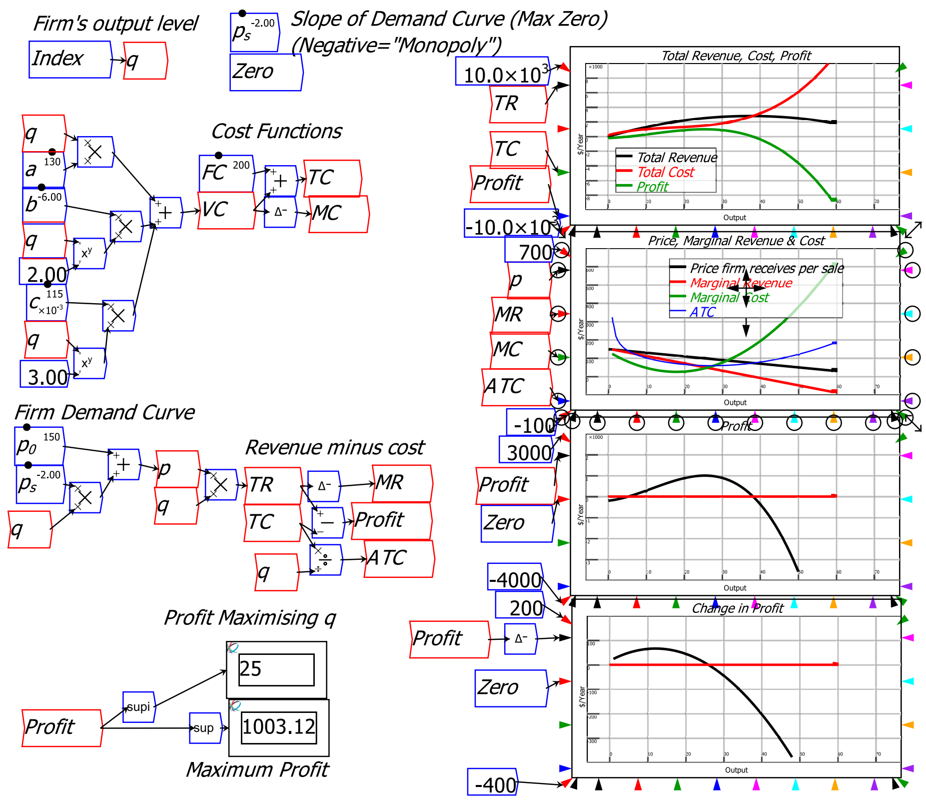
2.3.3.2 Equations tab

This displays the mathematical representation of the document, which enables you to see the complete mathematical logic in a document. You can also export a full LaTeX rendition of the document (from the File:Export Canvas command) to a LaTeX-capable editor.

2.3.3.3 Summary Tab

This tab provides a summary table of all variables in the system, in a heirarchical fashion. Each variable is fully documented including its name, definition, dimensions (for tensor-valued variables), units and current value.

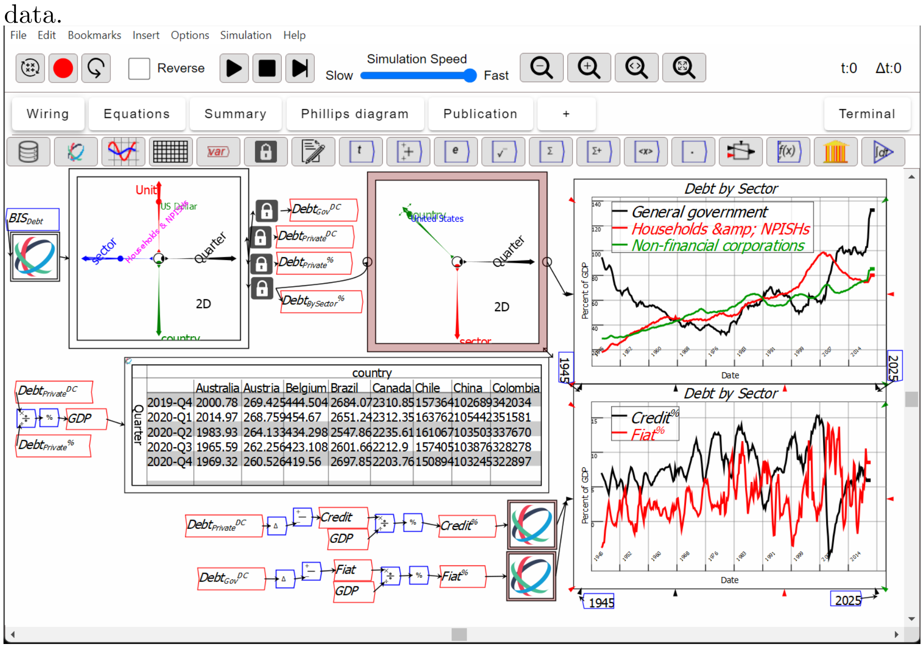
For example, this is a *Ravel* file modelling the economics textbook concept of a profit-maximizing firm.



The next figure shows an extract from the summary tab documentation for this model.

Name	Definition	Dimensions
$: ATC$	$\frac{\mathrm{TC}}{q}$	60
$: MC$	$[\Delta^{-}(\mathrm{VC})_i])$	59
$: MR$	$[\Delta^{-}(\mathrm{TR})_i])$	59
$: Profit$	$\mathrm{TR} - \mathrm{TC}$	60
$: TC$	$\mathrm{FC} + \mathrm{VC}$	60
$: TR$	$p \times q$	60
$: VC$	$q \times a + b \times q^2 + c \times q^3$	60
$: p$	$p_0 + p_s \times q$	60
q	Index	60

This document imports data from the Bank of International Assessments, separates the data into a number of variables, and uses data on debt in domestic currency and debt as a percentage of GDP to derive GDP in domestic currency data.



This is the Summary Tab for that model, showing the variable names, their mathematical definitions, their dimensions, and any initial values assigned to them.

▼ All Variables - 16

▼ flow - 9

Name	Definition	Dimensions	Initial	Units	Value
$: Credit$	$-\left[\Delta\left(\mathrm{Debt}_{\mathrm{Private}^{\mathrm{DC}}}\right)_i\right])$	319,48			
$: Credit^{\%}$	$\left(\frac{:\mathrm{Credit}}{\mathrm{GDP}}\right)\%$	319,45			
$: Debt^{\%}_{BySector}$	*locked*	323,3,43			
$: Debt^{DC}_{Gov}$	*locked*	323,43			
$Debt^{\%}_{Private}$	*locked*	323,4		0	
$Debt^{DC}_{Private}$	*locked*	323,48		0	
$: Fiat$	$-\left[\Delta\left(:\mathrm{Debt}_{\mathrm{Gov}^{\mathrm{DC}}}\right)_i\right])$	319,43			
$: Fiat^{\%}$	$\left(\frac{:\mathrm{Fiat}}{\mathrm{GDP}}\right)\%$	319,45			
$: GDP$	$\left(\frac{\mathrm{Debt}_{\mathrm{Private}\mathrm{DC}}}{\mathrm{Debt}_{\mathrm{Private}\%}}\right)\%$	323,45			

Most of these fields are editable. Changing a variable's name will do a *replace all instances* operation to update all variables of the same name. Changing a variable's definition will replace the wiring graph leading into the variable by a user defined function containing your edited string. At some future point, functionality will be added to convert a user defined function into a wiring graph.

2.3.3.4 Phillips diagram tab

This is a *Minsky*-specific feature which is covered in the *Minsky* sections of this manual.

2.3.3.5 Publication tab

Publication tabs allow the creation of dashboards to emphasise certain aspects of your data analysis in *Ravel* (or simulation in *Minsky*). For example, you may wish to provide different reports on the same data for the Accounting and Marketing sections of your company.

Multiple publication tabs can be created by clicking the '+' tab. See *Publication tabs* for full details.

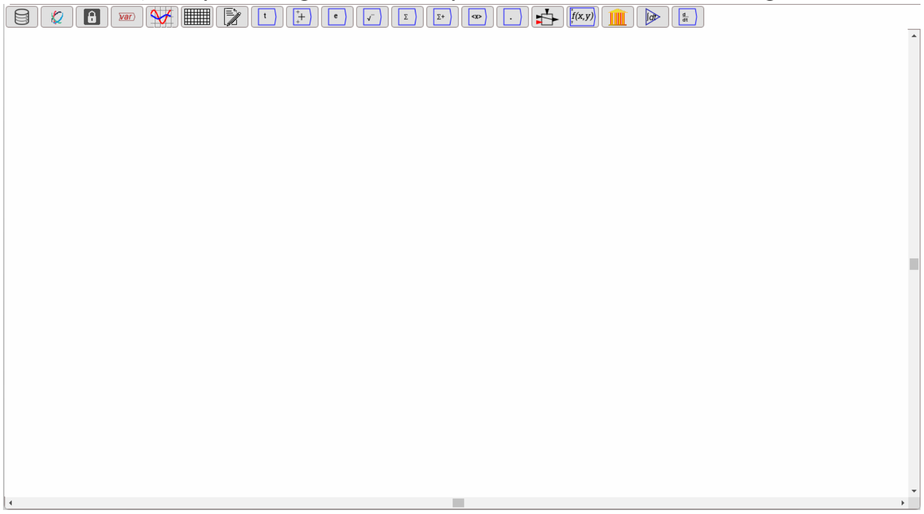
2.3.4 Design Canvas

The Design Canvas is where you develop your model. It has two components:

- The Operations icons, which give access to all of *Ravel*'s objects and mathematical operations; and
- The canvas itself, which is a space on which these objects and operations can be placed and connected by wires to perform data analysis, modelling, and displaying of results in Sheets and Plots.

A model consists of a number of blocks—imported data in parameters, Ravels, user-defined parameters, variables, constants, mathematical operators and the display elements (plots and sheets)—connected by wires.

The canvas is *zoomable*, either via the zoom buttons on the toolbar, or via the mouse scroll wheel. It is also *pannable*, either via the scroll bars on the right and bottom, or by holding the shift key and left mouse button together.



The canvas is effectively unlimited, however the scroll bars treat the canvas as 10000 pixels in size.

2.3.4.1 Wires

The wires in a model connect blocks together to define equations. For details see *Wires*

2.3.4.2 Design Icons

These are the "nuts and bolts" of data analysis using *Ravel*. There are many icons, and more will be added over time as we extend *Ravel*'s capabilities.

The critical icons for *Ravel* are the first four: Import Data ; insert a Ravel ; attach a selected slice of the Ravel to a Lock; and attach that lock to a variable for further analysis.

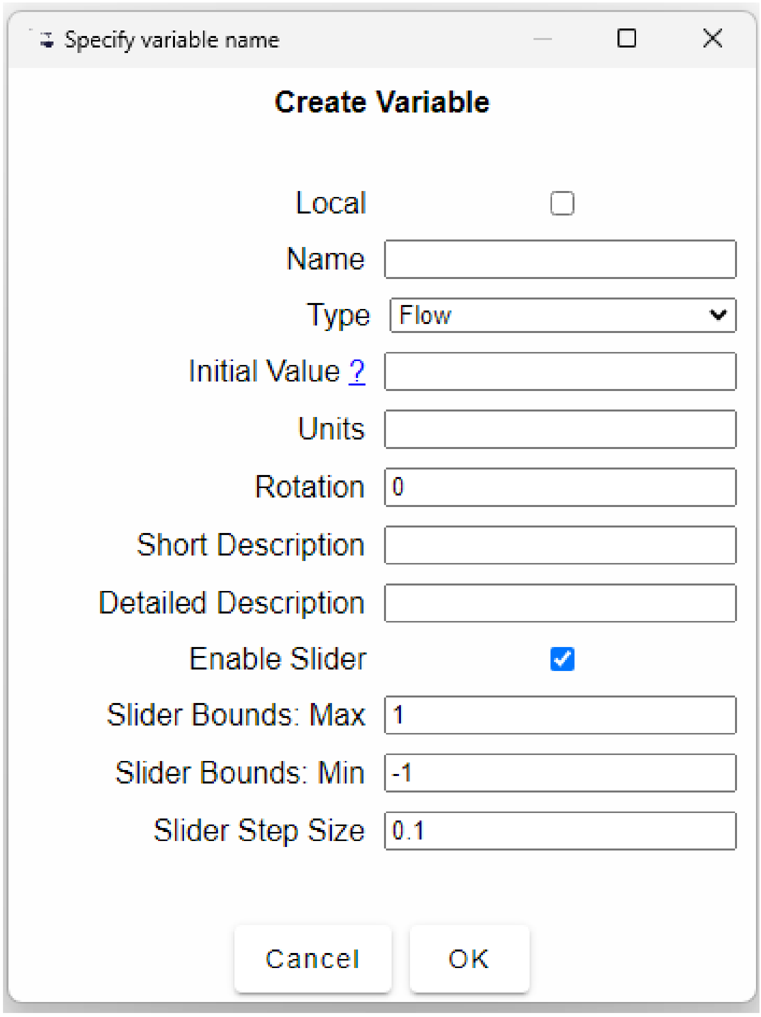
Import data Opens an import CSV file dialog, which allows a CSV file to be loaded into a parameter in Ravel (the default name of the parameter is the name of the file being imported). See *Importing CSV files* for full details. After a data file is imported, the next step is to attach it to a Ravel.

Ravel This places a Ravel on the wiring canvas. The first time this is done in a document, the Ravel is displayed full-size in Edit mode, and a sample set of dimensions are displayed. Subsequent Ravels are displayed in icon mode. For full details on using a Ravel see *Ravel GUI*.

Lock Lock widgets are used with Ravels. A lock keeps a record of the current state of a Ravel: the items selected on its axes, the effect of calipers in selecting data ranges, and so on. You can then manipulate the Ravel without changing the output from the Lock, which can be assigned to a variable for further use. You can also impose the state of a Lock on its associated Ravel—this is useful if you wish to fine tune the output from the Lock. See *Lock* for full details. The output of a Lock will change whenever the attached data file changes.

Variable This is a pop-up menu, which gives access to the form that creates variables, constants and parameters, and access to the Browser, which is a window that lists all the variables and parameters in a model, and enables them to be placed on the wiring canvas.

Variables are entities whose value changes as a function of time and its relationship with other entities in your model. Click on it and a variable definition window will appear:



See Variables for full details on using this form.

Plot widget Add Plots to the canvas. See Plot widget for full details.

Sheet widget Add a Sheet—for the display of numerical data to the canvas. See Sheet for full details.

Notes Add textual annotations to a document. See Notes for full details.

Time embeds a reference to the simulation time on the Canvas. This is a *Minsky*-specific feature.

Fundamental constants. These include e, π, 0, 1 and the percentage operator. See Special constants for full details.

Binary operations . These execute the stated binary mathematical operations: operations that require two (or more) inputs. Where appropriate, each input port to a binary operator can take multiple wires—so that to add five numbers together, for example, you can wire 1 input to one port on the Add block, and the other four to the other port. The same applies to the subtract, multiply, and divide blocks. See Binary Operations for full details.

Unary functions These are a fairly standard complement of mathematical functions which take only one input–though this input can have multiple dimensions. See Unary functions for details.

Reduction operations This menu contains operations such as sum, product, any, all, etc., that reduce a vector to a scalar, or reduce the rank of a tensor. See Tensor operations for details.

Scans This menu contains operations running sum, running product, and the difference operators. See Tensor operations for details.

Miscellaneous tensor operations Any other tensor function not covered elsewhere.

Switch Add a piecewise-defined function block to the canvas. See Switch for details.

User defined function You can define your own function using an algebraic expression, such as `exp(-x^2y)`+. See User defined functionsfor details.

Godley Table . This is the fundamental element of *Minsky* that is not found (yet) in any other system dynamics program. It is covered in the *Minsky* chapter of this manual.

Integration . This inserts a variable whose value depends on the integral of other variables in the system. It is discussed further in the *Minsky* section of the manual.

Derivative Operator This operator symbolically differentiates its input. It is a component of *Minsky* which is explained in the *Minsky* section of this manual.

2.4 Working with *Ravel*

Data analysis is performed in Ravel by connecting components using wires on the Wiring canvas.

2.4.1 Components in *Ravel*

There are several types of components:

1. Parameters, which either load in data from an external CSV file, store values typed in by the user, or create arrays of numbers using simple formulas entered into the *Initial Value* field;

2. Ravels, which take data from a parameter and create a multi-dimensional graphical rendition of the data;

3. Mathematical operators such as plus (+), minus (-), etc. These are all aware of the dimensions of your data, so they act on arrays of data, rather than single cells as with spreadsheet formulas;

4. Constants, which are given a value by the user;

5. Variables whose values are calculated by the program and depend on the values of constants, parameters and other variables, and the formulas applied to them;

6. Text strings and paragraphs, which can be used to document a model; and

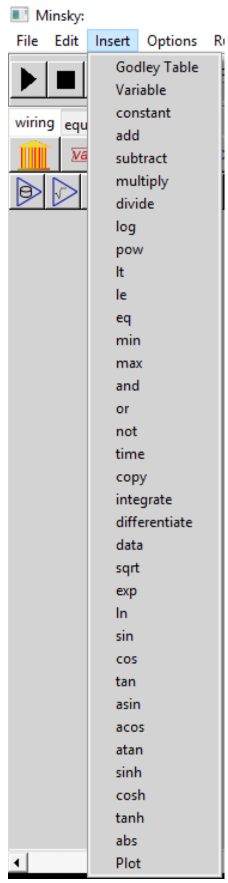
7. Groups, which allow components to be grouped into modules that can be used to construct more complex models.

2.4.2 Inserting a component

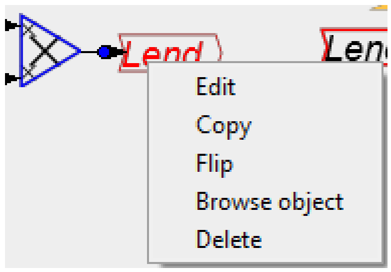
There are five ways to insert a component onto the Canvas:

1. Click on the desired Icon on the Icon Palette, drag the block onto the Canvas and click the mouse where you want to insert it

2. Choose Insert from the menu and select the desired block there


Minsky:
File
Edit
Insert
Options
wiring
Godley Table
Variable
constant
add
subtract
multiply
divide
log
pow
lt
le
eq
min
max
and
or
not
time
copy
integrate
differentiate
data
sqrt
exp
ln
sin
cos
tan
asin
acos
atan
sinh
cosh
tanh
abs
Plot

3. Right-click on an existing block and choose copy. Then place the copy where you want it on the palette.



4. Variables can be inserted by typing the variable name on the canvas, and constants can be entered simply by typing the number on the canvas. Similarly, operations can be inserted by typing the operator name (eg `sin`, or `*`). Notes can be inserted by starting the note with a `#` character.

5. Variables can also be picked from the Variable Browser and placed on the canvas.

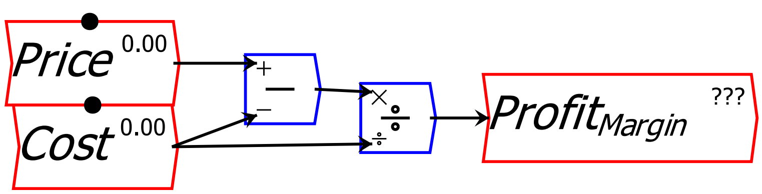
2.4.3 Creating an equation

Equations are entered in *Ravel* graphically. Mathematical operations like addition, multiplication and subtraction are performed by wiring the inputs up to the relevant mathematical block. The output of the block is then the result of the equation.

For example, a simple equation like

$$ProfitMargin = (Price - Cost)/Cost$$

is performed in *Ravel* by :



If you click on the Equation or Summary Tab, you will see this flowchart rendered as an equation:

$$Profit_{Margin} = \frac{Price - Cost}{Cost}$$

If the variables Price and Cost were multi-dimensional—if, for example, they stored price and cost data for different *products* at different *stores* at different *dates*—then this one formula would calculate profit margins *by product by store by date*.

2.4.4 Wiring components together

A model is constructed by wiring one component to another in a way that defines an equation. Wires are drawn from the output port of one block to the input port of another. Ports are circles on the blocks to which wires can be attached, which can be seen when hovering the pointer over the block. Variables have an input and an output port; constants and parameters only have an output port. A mathematical operator has as many input ports as are needed to define the operation. Some operators can be "overloaded"—more than one input can be attached to an input port. Thus you can add more than two variables using the block, simply by wiring more than one variable to each input port. The same applies to the subtraction block , division , minimum and maximum blocks.

Chapter 3

An Overview of *Ravel*

3.1 Slicing, Dicing and Rotating Data

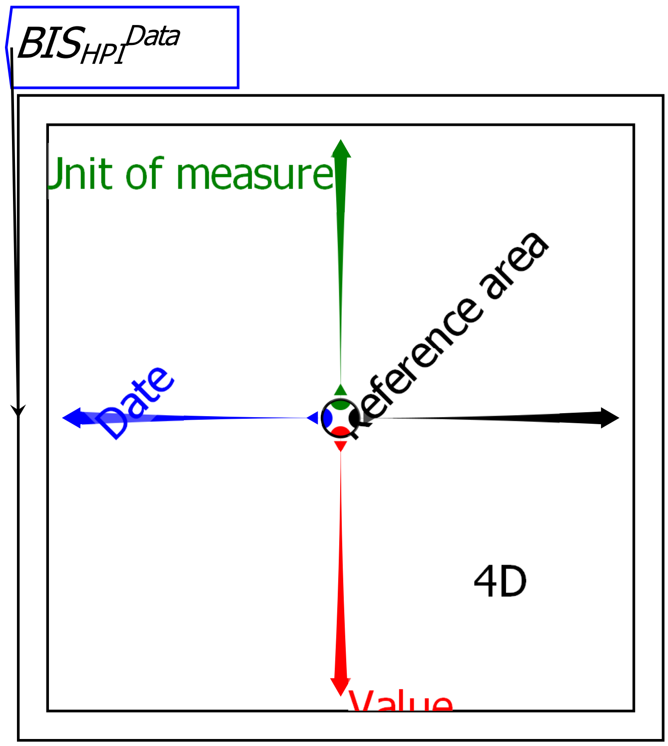
A Ravel is a graphical representation of multi-dimensional data. Unlike a spreadsheet, which has only two dimensions, a Ravel can have as many dimensions as your data. You manipulate the axes of a Ravel to select the components of your data that you wish to see, and those components are then the output of the Ravel, which can be graphed or displayed directly, or attached to variables which can be further analysed using the flowchart equation capabilities of *Ravel*. The Ravel below loads data from the *Bank of International Settlements* on house prices. This data has four dimensions:

Date Quarterly data from 1927 till 2024: 388 entries

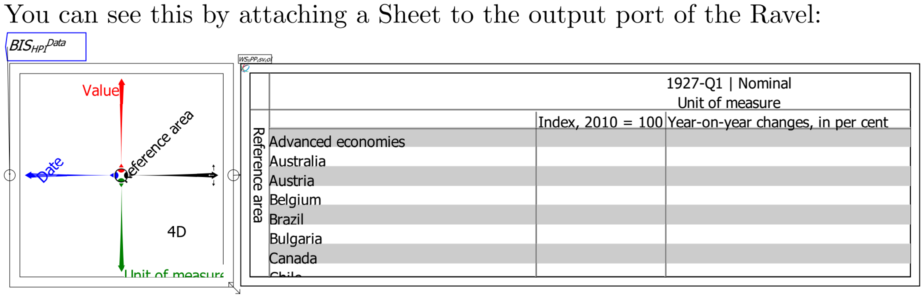
Unit of Measure House Price Index where 2010 = 100; 2 entries

Value Nominal or Real (CPI-deflated) prices: 2 entries

Reference Area Country or Region: 62 entries

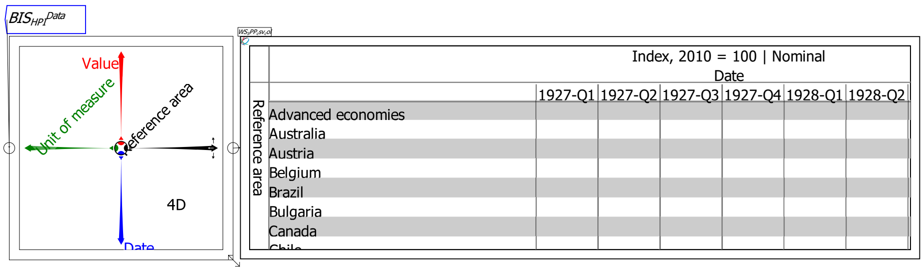


When a Ravel is first attached to data, it outputs the entire data set—which is indicated by the dimension count *4D* in the lower right quadrant of this Ravel. You can see this by attaching a Sheet to the output port of the Ravel:



The right-pointing axis of the Ravel determines what is shown on the rows of the sheet, while the down-pointing Axes of the Ravel determine what is shown by the columns. At present these are Reference Area by Value, and the sheet shows a slice of that data for the first entry in the Date and Value axes–1927-Q3, and Nominal (unadjusted for inflation).

It would be more useful to see the data by Reference Area by Date. To get that view, click the left mouse button on the arrowhead of the Date axis, hold the button down, and rotate the axis into the down direction, which is currently occupied by the Value axis. When you release the mouse button, the Date axis will replace the Unit of measure axis in the down direction, and the data in the Sheet will now have countries by rows and Quarters by columns.



The Sheet is still blank, because there is no data for the current selections–there is no data for Index for Nominal House Prices in the very first Quarters of the data (in the years 1927 and 1928) for the first countries in the file in alphabetical order.

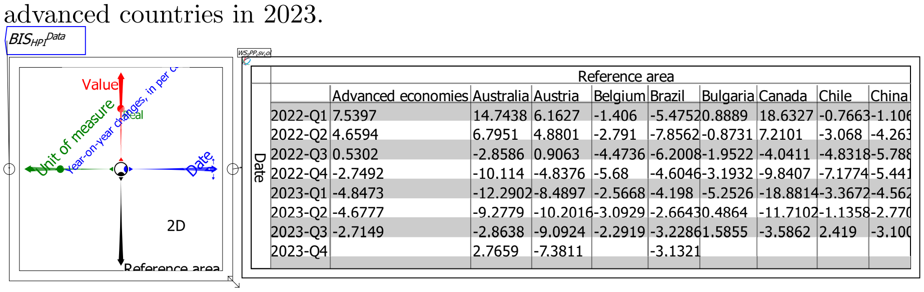
To see data immediately, you can take advantage of a feature of the Sheet: it can display the first few rows and columns of data (the default setting), which we call the Head, the last few (the Tail), or a few of both (Head and Tail). To show the last few rows and columns, right-click on the Sheet and choose Row Slices/Tail and Column Slices/Tail". That will then show you the last countries in the data file in alphabetical order, and the last quarters in the data file.

The data still shows the Nominal Index data, since these are the first entries in the other two axes. You can control the entry shown using the selector dots on those two axes: these are the coloured dots that are currently within the inner circle of the Ravel. Selector dots can be moved:

- By using the mouse. Click on a dot and drag it to the required selection; or

- By using the arrow keys. Use the mouse to move the cursor so that it is hovering over an axis; then use the up or right arrow key to move the dot out towards the arrowhead on an axis, or the down or left arrow key to move back towards the center.

To see the Real (CPI-adjusted) annual rate of change of house prices, use the selector dot on those two axes. That selection is shown below–where Date has also been rotated to the rows so that Countries are shown by the columns. This already gives one interesting insight: house prices were on average falling across advanced countries in 2023.



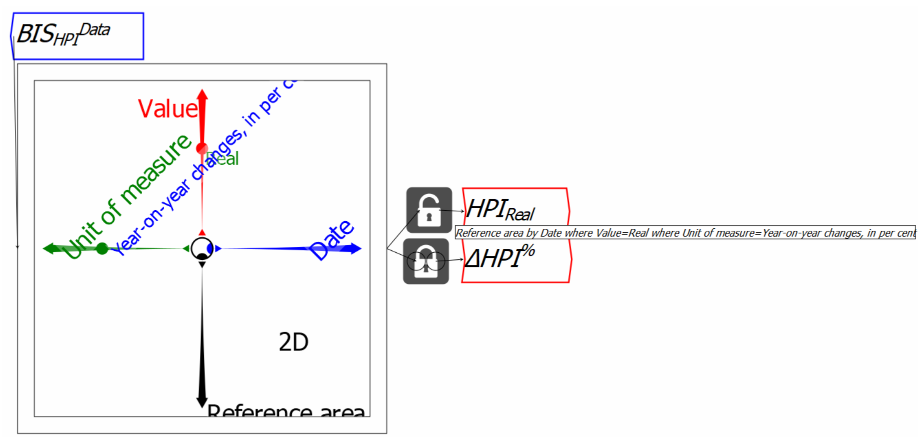
Date	Advanced economies	Australia	Austria	Belgium	Brazil	Bulgaria	Canada	Chile	China
2022-Q1	7.5397	14.7438	6.1627	-1.406	-5.4752	0.8889	18.6327	-0.7663	-1.10
2022-Q2	4.6594	6.7951	4.8801	-2.791	-7.8562	-0.8731	7.2101	-3.068	-4.26
2022-Q3	0.5302	-2.8586	0.9063	-4.4736	-6.2008	-1.9522	-4.0411	-4.8318	-5.78
2022-Q4	-2.7492	-10.114	-4.8376	-5.68	-4.6046	-3.1932	-9.8407	-7.1774	-5.44
2023-Q1	-4.8473	-12.2902	-8.4897	-2.5668	-4.198	-5.2526	-18.8814	-3.3672	-4.56
2023-Q2	-4.6777	-9.2779	-10.2016	-3.0929	-2.6643	0.4864	-11.7102	-1.1358	-2.77
2023-Q3	-2.7149	-2.8638	-9.0924	-2.2919	-3.2286	1.5855	-3.5862	2.419	-3.10
2023-Q4		2.7659	-7.3811		-3.1321				

To really develop insights from your data, you attach the output of the Ravel to variables, and analyse them using *Ravel*'s flowchart mathematics formulas.

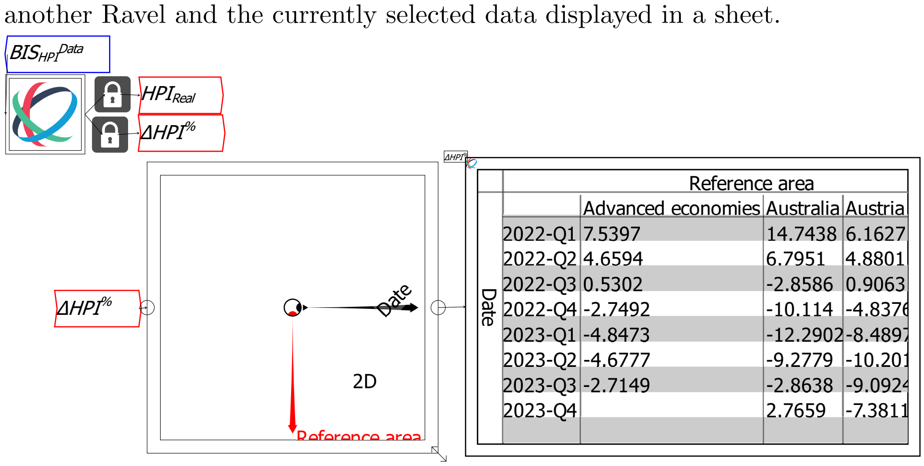
3.2 Attaching Variables to Ravels

The source data file combines information on House Price Indices (where the base year is 2010, so all indices are 100 during 2010), and the annual rate of change of house prices, with data on both Nominal and Real (CPI-deflated) prices. To analyse the data, it is useful to separate it into House Price Index information and House Price Inflation information, and to focus on Real rather than Nominal Prices. That is done by attaching the output of the Ravel to Locks, and the Locks to named Variables that you create.

The next image shows two variables HPI_{Real} and ΔHPI. The lock for ΔHPI has been closed, while the lock for HPI_{Real} is still open. Once the lock is closed, the output from that lock remains the same, even if the selection on the source Ravel is altered.



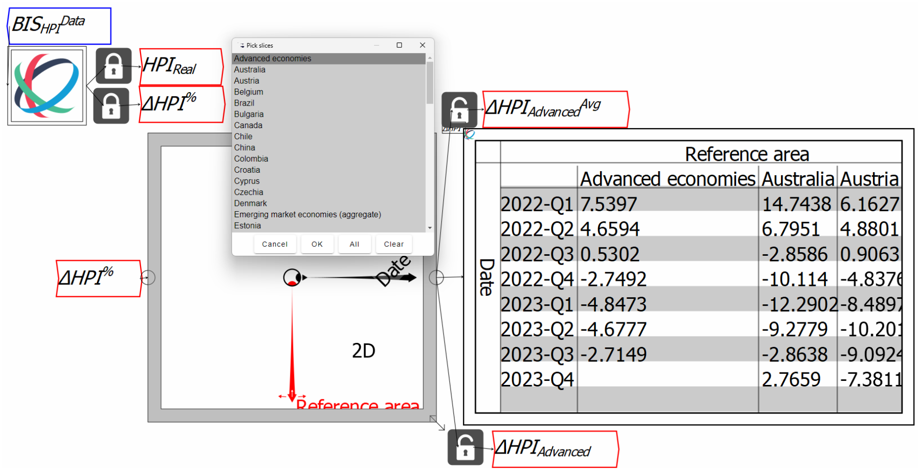
With the data separated into index and inflation data, we can now focus on those subsets of the data rather than the entire source file. The next image shows the source Ravel in icon mode, with the house price inflation data attached to another Ravel and the currently selected data displayed in a sheet.



The information in the sheet suggests a possibly useful piece of analysis:

why not compare the average for all advanced countries (the first entry on the Reference Area axis) to each advanced country?

The image below shows the result of attempting to do this by selecting just the data for "Advanced Economies" using the selector dot, and attaching that to a new variable $\Delta HPI^{Avg}_{Advanced}$, while also selecting a number of advanced economies from the axis (Australia, Austria, Belgium, etc.) and assigning that to another variable $\Delta HPI_{Advanced}$.



3.3 Basic Analysis using *Ravel*

Data analysis can be done using the features of a Ravel alone. As well as enabling the selection of display of different segments of multidimensional data, you can perform basic data analysis by collapsing the axes of a Ravel and applying various aggregations:

1. Summing an axis;

2. Multiplying axis elements by each other;

3. Extracting the maximum or minimum values along an axis;

4. Calculating the average along an axis;

5. Calculating the standard deviation of an axis; and

More advanced analysis can be performed by attaching the output of a Ravel to a variable (preferably via a Lock, so that the contents of the variable are unaffected by any subsequent manipulations of the Ravel), and analysing it using *Ravel*'s mathematical operators. For more details see *Operations*.

3.4 Linking Ravels

Ravels which have the same dimensions can be linked to each other. For example, one Ravel may contain inflation data by country by month, while another contains unemployment data by country by month. If the two or more Ravels are linked, then selections applied to one Ravel are applied to the other, which makes it easy to compare these variables by country over time. For more details see *Linking Ravels*.

3.5 Displaying your results in Sheets and Plots

Ravels enable the selection and some analysis of data, but don't display data themselves (however, we will embed Sheets and Plots into Ravels in future releases of *Ravel*). To see the output of a Ravel, you need to attach the output of the Ravel (or variables derived from it) to a Sheet or a Plot. For more details see *Sheet* and *Plot Widget*.

3.6 Using Publication Tabs

Calculations on the Wiring canvas can become too complicated for non-experts to follow, so Publication tabs exist to enable you to show tailored views of the results for different audiences—say one Tab for the Marketing and another for the Accounting divisions of a company. For details see *Publication tabs*.

3.7 Working with Other Programs

Though *Ravel* can be used to present results very effectively, you will sometimes wish to display your results in publication or presentation software, or to create better-looking plots (Ravel's Plots will improve dramatically in subsequent releases). You can export a variety of formats from Ravel:

1. CSV files, in both columnar and tabular layout;
2. SVG, EPS and EMF vector and PNG bitmapped graphics files, for use in word processor and presentation software;
3. PDF format;
4. LaTeX format, for use in mathematical formatting programs like LaTeX editors and MathType; and
5. .m files, for analysis of the mathematics of a Ravel document in Matlab or Octave.

Exporting is done in two main ways:

1. From the File menu option *Export canvas as*, which exports an entire canvas (this includes the Equations, Summary and Publication Tabs)

2. From the context menu for any specific object, such as a Ravel, Variable, Plot or Sheet.

3.8 Importing Data into Ravel

CSV files typically have some columns that specify the nature of data, while other columns contain the data itself. *Ravel* turns the fields specifying the nature of data into *dimensions* (which become axes of a Ravel), which enables the data to be manipulated much more effectively than can be done with a spreadsheet. For data to be imported into *Ravel*, the dimensions must uniquely specify each data point. This process is controlled by the import tool ,
which then brings up the import form in which you specify which columns to treat as dimensions and which as data. For more details see *Importing data into a parameter from a CSV file*.

Chapter 4

Reference

4.1 Tabs

Ravel has six Tabs:

- *Wiring*, which houses the Operators and the Design canvas on which data analysis (*Ravel*) and dynamic models (*Minsky*) are constructed;

- *Equations*, which shows the equations created by the analysis or model on the Wiring Tab;

- *Summary*, which shows a hierarchical overview of all the definitions of variables and parameters in the model;

- *Phillips Diagram*, a *Minsky*-specific view of the financial flows in a dynamic economic model;

- *Publication*, which enables selected items from a Wiring to be displayed; and

- +, which adds additional Publication Tabs

4.1.1 Wiring

The Wiring Tab consists of:

- The Operations bar, which contains all of *Ravel*'s mathematical operators; and

- The design canvas, a free-form space on which variables, parameters and operations are combined to perform analysis of data in *Ravel*, and to simulate dynamic systems in *Minsky*.

4.1.2 Equations

The Equation Tab shows the equations in your data analysis or model. To export equations in a format that can be inserted into LaTeX editors or programs like MathType, choose “Export canvas as” from the File menu, and choose the format LaTeX. For details see *Working with Other Programs*.

4.1.3 Summary

The Summary Tab shows the equations in your data analysis or model in a structured tabular layout. For details see *Summary tab*.

4.1.4 Phillips Diagram

The Phillips Diagram Tab shows the financial flows in a *Minsky* simulation model in a dynamic flowchart format. This is a *Minsky*-specific feature.

4.1.5 Publication

A Publication Tab enables you to place any item on the Wiring canvas in any location. This is useful for distributing the results of a model without having to show all the workings involved in deriving results.

Any item from the wiring tab can be added to a publication tab, and then moved, resized or rotated independently of the item on the wiring tab. To add an item to a Publication Tab, hover the mouse over the item and choose “Add item to a publication tab” from the context menu. The sub-menu lists available Publication tabs. Click on the one you wish to place the item on, and the item will be placed in the upper-left hand corner of that tab. It can then be moved to anywhere else on that tab.

For items that dynamically update, the publication tab will be updated dynamically when selections on the Wiring Tab are altered.

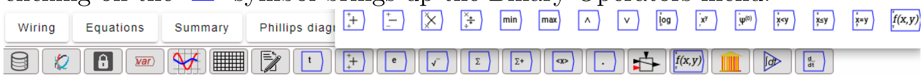
Ravel components can also be manipulated on the Publication tab–for example, a different store can be selected on the “Store” axis of a Ravel documenting retail sales. If a Ravel has a caliper applied to an axis on the Wiring Tab, to, for example, select a range of dates, that caliper can be moved to a different range of dates on the Publication Tab.

Textual annotation can be added to the publication tab, independently of any annotations on the wiring tab.

Wires currently cannot be added to the publication tab. However, you can insert text-based arrows (eg $\rightarrow$) by typing `#\rightarrow` on the canvas. This item can then be rotated and scaled to emulate the wiring diagram connecting parts of the dashboard together.

4.2 Operations

Except for , which invokes a pop-up sideways menu for defining variables, constants and parameters, the first 8 icons on the Operations bar—from data importing to the simulation time–are immediately activated by clicking on them. The remaining mathematical and data operations are stored on pop-up sideway menus, which are activated when you click on the menu's icon. For example, clicking on the symbol brings up the Binary Operators menu:



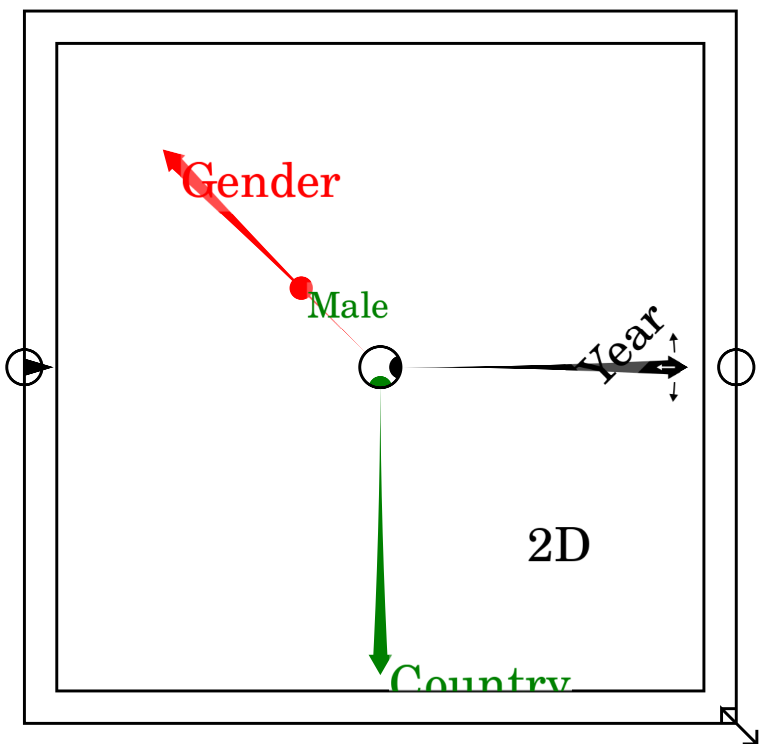
4.2.1 Import Data

The import data operator inserts a parameter on the canvas called , and invokes the data import form. For more details see CSV import.

4.2.2 Ravel

A Ravel is a visual tool for manipulating and analyzing multi-dimensional data. For full details on the concept see Ravel.

The Ravel operator inserts a Ravel onto the Wiring canvas. The first Ravel inserted is in Edit mode, and has sample axis names and a sample data point selected.



Subsequent Ravels are inserted in iconized mode.

To use a Ravel, it must first have a parameter (or variable) attached to it. Normally the parameter will contain data imported from a CSV file—for more details see CSV import.

The key feature of a Ravel is its Axes (or "Arrows"). A Ravel has one axis per dimension of the data attached to it. Axes can be:

Rotated The right-pointing axis determines what is shown on the x-axis of a Plot and columns of a Sheet; the down-pointing axis determines what is displayed on the Y-axis of a plot and the rows of a sheet.

Collapsed An axis is collapsed by dragging the arrow head back to the centre of the Ravel. This applies an aggregation method to the axis, which is determined by the setting for "Set next aggregation" on the context menu. The aggregations are Sum, Product, Average, Standard Deviation, Maximum and Minimum. The aggregation type can be changed *post facto* unser the "Axis properties" context menu.

Selected When the selector dot for an Axis is in the centre of the Ravel, the entire Ravel axis is output (though this can be a selection of values on the axis if "Pick axis slices" has been activated). When the selector dot is on a particular value on an axis, only that value is output.

Sorted The "none" sort order of a Ravel is that of the axis as it is attached (usually the order in which it is read from a CSV file). The axis can also be sorted in either a "forward" or "reverse" direction. The sorting comparison depends on the dimension type — numerical in the case of "value", time order in the case of "time" and *lexicographic* ("alphabetical") in the case of "string".

Sorted by value When the output of a Ravel is 1-dimensional, axes can be sorted by the data values along this 1-dimensional axis. The choices are a combination of Forward and Reverse, and Statically and Dynamically. A Static sort applies the order of the data for a given instance—for example, countries could be sorted by the inflation rate for 2022. This sort order is maintained if the selector dot for date is altered. For example, with a static sort, choosing 2000 on the Date axis would show the inflation rate in 2000, with countries sorted according to their inflation rates in 2022. A Dynamic sort applies the sort order for each selection. For example, a reverse dynamic sort of Country by inflation rates would show countries by descending order of their inflation rate in whichever year was chosen on the Date axis.

Picked The context menu option "Pick axis slices" invokes a pop-down menu showing all the entries on that axis—countries for a Country axis, dates for a Time axis, etc. You can select specific values by clicking on them, and select multiple values by control-clicking on them.

Calipered "Toggle axis calipers" brings up a caliper which initially spans the entire data range. The ends of the caliper can be individually adjusted to select a range of the data, while the whole caliper can be moved to select the same number of data points for different segments of the data.

Ravels can be linked with other Ravels that share the same dimension(s). This means that any operation applies to one Ravel is applied to all Ravels in the linked set. For more details see *Linking Ravels*.

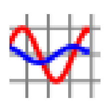
4.2.3 Lock

Locks are attached to the output of Ravels. After a Lock is closed (either by double-clicking on the Lock, or choosing Lock from the context menu), alterations can be made to the source Ravel without altering the output of the Lock. This makes it possible to take multiple slices of the data from one Ravel, and to assign those slices to variables so that they can be further analyzed by *Ravel*. For details see *Lock*

4.2.4 Variable

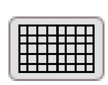
This operator activates a pop-up menu for defining variables, parameters and constants. For details see *Variables*.

4.2.5 Plot



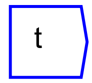
This operator inserts a Plot onto the Wiring canvas. For details see *Plot Widget*.

4.2.6 Sheet



This operator inserts a Sheet onto the Wiring canvas. For details see *Sheet*.

4.2.7 time t

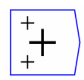


Returns the current value of system time.
The operator can be placed on the canvas in two ways:

- From the time widget on the toolbar  ; or
- By typing the letters "time" on the canvas and then pressing the Enter key.

4.2.8 Binary Operations

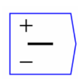
4.2.8.1 add +



Add multiple numbers together.
The input ports allow multiple wires, which are all summed. If an input port is unwired, it is equivalent to setting it to zero.
The operator can be placed on the canvas in two ways:

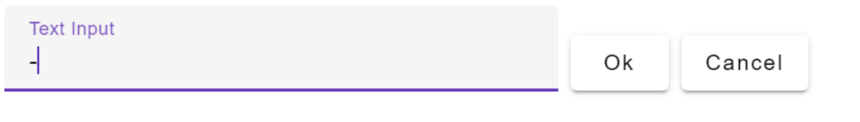
- From the *Binary Operations ("binop")* toolbar; or
- By pressing the plus key anywhere on the wiring canvas.

4.2.8.2 subtract −



Subtract two numbers. The input ports allow multiple wires, which are summed prior to the subtraction being carried out. If an input port is unwired, it is equivalent to setting it to zero. Note the small '+' and '−' signs on the input ports indicating which terms are added or subtracted from the result.
The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By pressing the minus key anywhere on the wiring canvas, *followed by pressing the Enter key, or clicking on OK in the text input window.* The reason for requiring the Enter key to be pressed—rather than immediately placing the minus operator on the keyboard, as with the plus and multiply operators—is that a user may wish to enter a negative number as a constant.



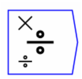
4.2.8.3 multiply ×

Multiply numbers with each other. The input ports allow multiple wires, which are all multiplied together. If an input port is unwired, it is equivalent to setting it to one.

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By pressing the multiply (asterisk: *) key anywhere on the wiring canvas.

4.2.8.4 divide ÷

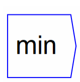


Divide a number by another. The input ports allow multiple wires, which are multiplied together for each port prior to the division being carried out. If an input port is unwired, it is equivalent to setting it to one. Note the small '×' and '÷' signs indicating which port refers to the numerator and which the denominator.

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By pressing the divide key (/) anywhere on the wiring canvas.

4.2.8.5 min



Returns the minimum of x and y. If multiple wires are connected to the ports, the the overall minimum from all inputs is returned.

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By typing the letters "min" on the canvas and then pressing the Enter key.

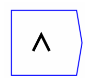
4.2.8.6 max

Returns the maximum of x and y. If multiple wires are connected to the ports, the the overall maximum from all inputs is returned.

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By typing the letters "max" on the canvas and then pressing the Enter key.

4.2.8.7 and $\wedge$

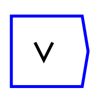


Logical and of x and y, where $x \leq 0.5$ means false, and $x > 0.5$ means true. The output is 1 or 0, depending on the result being true (1) or false (0) respectively.

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By typing the letters "and_" (the word and, *followed by an underscore*) on the canvas and then pressing the Enter key. The underscore is needed because "and" is a reserved word in *C++,* the programming language in which *Ravel* is written.

4.2.8.8 or $\vee$

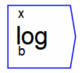


Logical or of x and y, where $x \leq 0.5$ means false, and $x > 0.5$ means true. The output is 1 or 0, depending on the result being true (1) or false (0) respectively.

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By typing the letters "or_" (the word or, *followed by an underscore*) on the canvas and then pressing the Enter key. The underscore is needed because "or" is a reserved word in *C++,* the programming language in which *Ravel* is written.

4.2.8.9 log



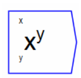
Take the logarithm of the x input port, to base b. The base b needs to be specified — if the natural logarithm is desired ($b = e$), use the natural log instead.

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By typing the word "log" anywhere on the wiring canvas, and then pressing the Enter key.

When you use the direct typing method to enter the Log operator, the text entry window pops up. This allows you to type a variable/parameter name starting with Log (like, for example, "Logical". If you press Enter (or click on OK) with only the word Log in the window, the Log operator will be placed on the canvas.

4.2.8.10 pow x^y

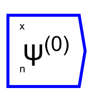


Raise one number to the power of another. The ports are labelled x and y, referring the the formula x^y.

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By typing the letters "Pow" anywhere on the wiring canvas and then pressing the Enter Key (or clicking on OK).

4.2.8.11 Polygamma $\psi^{(n)}(x)$



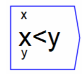
Returns the polygamma function of the first argument x, with the order n being given by the floor of the second argument.

$$\psi^{(n)}(x) = \frac{d^{n+1}}{dx^{n+1}} \ln \Gamma(x)$$

The operator can be placed on the canvas in two ways:

- From the *Binary Operations ("binop")* toolbar; or
- By typing the letters "polygamma" on the canvas and then pressing the Enter key.

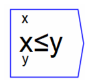
4.2.8.12 lt <



Returns 0 or 1, depending on whether $x < y$ is true (1) or false (0).
The operator can be placed on the canvas in two ways:

- From the *Binary Operations (“binop”)* toolbar; or
- By typing the letters ”lt” on the canvas and then pressing the Enter key.

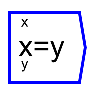
4.2.8.13 le ≤



Returns 0 or 1, depending on whether $x \leq y$ is true (1) or false (0).
The operator can be placed on the canvas in two ways:

- From the *Binary Operations (“binop”)* toolbar; or
- By typing the letters ”le” on the canvas and then pressing the Enter key.

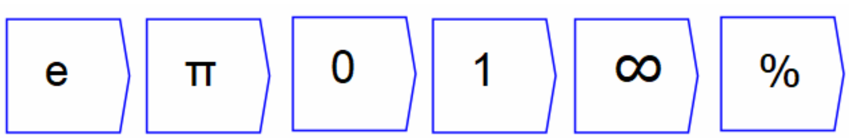
4.2.8.14 eq =



Returns 0 or 1, depending on whether $x = y$ is true (1) or false (0).
The operator can be placed on the canvas in two ways:

- From the *Binary Operations (“binop”)* toolbar; or
- By typing the letters “eq” on the canvas and then pressing the Enter key.

4.2.9 Special constants

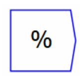


Some special constants ($e = 2.72\ldots$, $\pi = 3.14\ldots$, 0, 1, ∞) can be placed on the canvas, via two methods:

- By clicking on the relevant icon on the *Fundamental Constants (“constop”)* toolbar; or

- By typing the constant name on the canvas, and pressing Enter (or clicking OK) in the variable definition window. These names are: *Euler* for $e = 2.72\ldots$; *pi* for $\pi = 3.14\ldots$; *inf* for ∞; and the percent symbol for the percent operator (which multiplies the input by 100).

4.2.9.1 Percent



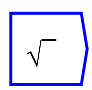
The percent operator takes one input, and multiplies its elements by 100. It is useful for converting fractions into percentages for display purposes.

The operator can be placed on the canvas in two ways:

- From the *Fundamental Constants ("constop")* toolbar; or
- By pressing the percent key anywhere on the wiring canvas.

4.2.10 Functions/Unary Operators

4.2.10.1 sqrt $\surd$



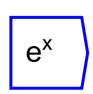
This produces the square root of the input value.

For example, connecting the value of 9 with the "sqrt" block will produce the value of 3. As with all Ravel operators, this function accepts multidimensional arguments.

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "sqrt" on the canvas and then pressing the Enter key.

4.2.10.2 exp

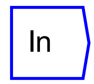


Connecting a variable (for example, "time") to this block will produce the exponential function e^x where x is the input variable.

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "exp" on the canvas and then pressing the Enter key.

4.2.10.3 ln

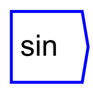


Produces a natural logarithm of the input, to the base of e. This takes the equation $\log_e x$ where x is the input.

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "ln" on the canvas and then pressing the Enter key.

4.2.10.4 sin



Produces a sine function of the input.

For example, connecting a "time" block to this function, and then to a graph, will produce a sine wave.

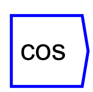
The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "sin" on the canvas and then pressing the Enter key.

For further explanation regarding trigonemtric functions, see Wikipedia's page on trigonometric functions.

4.2.10.5 cos

Produces a cosine function of the input.

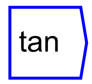


The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "cos" on the canvas and then pressing the Enter key.

For example, connecting a "time" block to this function, and then to a graph, will produce a cosine wave. For further explanation regarding trigonemtric functions, see Wikipedia's page on trigonometric functions.

4.2.10.6 tan

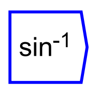


Produces a tangent function of the input.
The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "tan" on the canvas and then pressing the Enter key.

For example, connecting a "time" block to this function, and then to a graph, will produce a tangent graph. For further explanation regarding trigonemtric functions, see Wikipedia's page on trigonometric functions.

4.2.10.7 asin

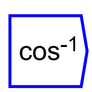


Produces an arc sine function of the input, or the inverse of the sine function.
The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "asin" on the canvas and then pressing the Enter key.

For further explanation regarding trigonemtric functions, see Wikipedia's page on trigonometric functions.

4.2.10.8 acos

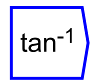


Produces an arc cosine function of the input, or the inverse of the cosine function.
The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "acos" on the canvas and then pressing the Enter key.

For further explanation regarding trigonemtric functions, see Wikipedia's page on trigonometric functions.

4.2.10.9 atan



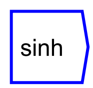
Produces an arc tangent function of the input, or the inverse of the tangent function.

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "atan" on the canvas and then pressing the Enter key.

For further explanation regarding trigonemtric functions, see Wikipedia's page on trigonometric functions.

4.2.10.10 sinh

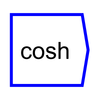


hyperbolic sine function $\frac{e^x - e^{-x}}{2}$

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "sinh" on the canvas and then pressing the Enter key.

4.2.10.11 cosh

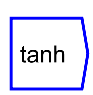


hyperbolic cosine function $\frac{e^x + e^{-x}}{2}$

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "cosh" on the canvas and then pressing the Enter key.

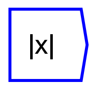
4.2.10.12 tanh



hyperbolic tangent function $\frac{e^x - e^{-x}}{e^x + e^{-x}}$
The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "tanh" on the canvas and then pressing the Enter key.

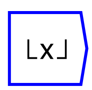
4.2.10.13 abs $|x|$



absolute value function. Returns the magnitude of any entered variable.
The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "abs" on the canvas and then pressing the Enter key.

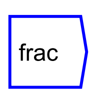
4.2.10.14 floor $\lfloor x \rfloor$



The greatest integer less than or equal to x.
The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "floor" on the canvas and then pressing the Enter key.

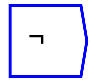
4.2.10.15 frac



Fractional part of x, ie $x - \lfloor x \rfloor$.
The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "frac" on the canvas and then pressing the Enter key.

4.2.10.16 not ¬

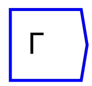


The output is 1 or 0, depending on whether $x \leq 0.5$ is true (1) or false (0) respectively.

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "not_" (the word not, *followed by an underscore*) on the canvas and then pressing the Enter key. The underscore is needed because "not" is a reserved word in *C++,* the programming language in which *Ravel* is written.

4.2.10.17 Gamma Γ



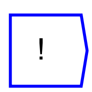
Returns the Gamma function of its argument:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or
- By typing the letters "gamma" on the canvas and then pressing the Enter key.

4.2.10.18 Factorial !



Returns the factorial of its argument:

$$\begin{aligned} 0! &= 1 \\ n! &= \prod_{i=1}^{n} i \end{aligned}$$

The operator can be placed on the canvas in two ways:

- From the *Functions ("function")* toolbar; or

- By typing the letters “fact” on the canvas and then pressing the Enter key.

Note:

$$n! = \Gamma(n+1)$$

which is how it is implemented in Minsky.

Minsky provides this function because of its relationship to the derivative of the Gamma function (and factorials).

4.2.10.19 copy

This just copies its input to its output, which is redundant on wiring diagrams, but is needed for internal purposes.

4.2.11 differentiate d/dt

Symbolically differentiates its input with respect to system time, producing d/dt[input]. For further explanation regarding differentiation, see this wikipedia page.

4.2.12 User defined function

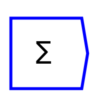
A user defined function is a function defined by an algebraic expression. Support for this feature is courtesy of the wonderful exprtk library developed by Arash Partow.

A user defined function has a *name*, *parameters* and an *expression*. Example expressions are things like `x+y` or `sin(x)`. More details of the sorts of expressions possible can be found in the User Defined Functions section of the manual.

The parameters are specified as part of the name, so a user defined function adding x and y would be called `useradd(a,y)` and the sin example might be called `mysin(x)`. Functions with up to two arguments can be wired on the canvas. User defined functions can call other user defined functions, so specifiying more than 2 parameters can be a useful thing to do.

4.2.13 Tensor operations

4.2.13.1 sum $\sum$



Sum along a given axis.

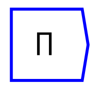
The operator can be placed on the canvas in two ways:

- From the *Reductions Operations (“reduction”)* toolbar  ; or

- By typing the letters “sum” on the canvas and then pressing the Enter key.

The same result can be achieved by collapsing the relevant Ravel axis after choosing Σ from the “Set next aggregation” context menu.

4.2.13.2 product $\prod$

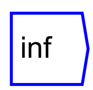


Multiply along a given axis.
The operator can be placed on the canvas in two ways:

- From the *Reductions Operations (“reduction”)* toolbar ; or
- By typing the letters “product” on the canvas and then pressing the Enter key.

The same result can be achieved by collapsing the relevant Ravel axis after choosing Π from the “Set next aggregation” context menu.

4.2.13.3 infimum

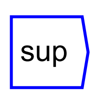


Return the least value along a given axis.
The operator can be placed on the canvas in two ways:

- From the *Reductions Operations (“reduction”)* toolbar ; or
- By typing the letters “infimum” on the canvas and then pressing the Enter key.

4.2.13.4 supremum

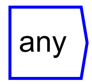
Return the greatest value along a given axis.



The operator can be placed on the canvas in two ways:

- From the *Reductions Operations (“reduction”)* toolbar ; or
- By typing the letters “supremum” on the canvas and then pressing the Enter key.

4.2.13.5 any

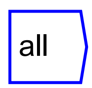


Return 1 if any value along a given axis is nonzero, otherwise return 0 if all are zero.

The operator can be placed on the canvas in two ways:

- From the Reductions Operations toolbar  ; or
- By typing the letters "any" on the canvas and then pressing the Enter key.

4.2.13.6 all



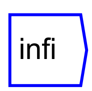
Return 1 if all values along a given axis are nonzero, otherwise return 0 if any are zero.

The operator can be placed on the canvas in two ways:

- From the *Reductions Operations ("reduction")* toolbar  ; or
- By typing the letters "all" on the canvas and then pressing the Enter key.

4.2.13.7 infindex

Return the index of the least value along a given axis.

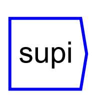


The operator can be placed on the canvas in two ways:

- From the *Reductions Operations ("reduction")* toolbar  ; or
- By typing the letters "infIndex" on the canvas and then pressing the Enter key.

4.2.13.8 supindex

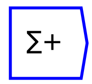
Return the index of the greatest value along a given axis.



The operator can be placed on the canvas in two ways:

- From the *Reductions Operations ("reduction")* toolbar ; or
- By typing the letters "supIndex" on the canvas and then pressing the Enter key.

4.2.13.9 running sum $\sum +$



Computes the running sum of the input tensor along a given axis.

The optional numerical argument (in the "edit" dialog) can be used to specify a window over which the running sum is performed. If negative, the window is over the entire dimension.

The operator can be placed on the canvas in two ways:

- From the *Scan Operations ("scan")* toolbar ; or
- By typing the letters "runningSum" on the canvas and then pressing the Enter key.

For example, take this rank 2 tensor:

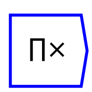
$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 4 & 3 & 2 \\ 8 & 7 & 6 & 5 \end{pmatrix}$$

The running sum of this tensor, along the horizontal dimension, is:

$$\begin{pmatrix} 1 & 3 & 6 & 10 \\ 5 & 9 & 12 & 14 \\ 8 & 15 & 21 & 26 \end{pmatrix}$$

Running sum will normally be applied to a Ravel or a variable defined via a Ravel.

4.2.13.10 running product $\prod +$



Computes the running product of the input tensor along a given axis.

The optional numerical argument (in the "edit" dialog) can be used to specify a window over which the running sum is performed. If negative, the window is over the entire dimension.

The operator can be placed on the canvas in two ways:

- From the *Scan Operations ("scan")* toolbar ; or
- By typing the letters "runningProduct" on the canvas and then pressing the Enter key.

For example, take this rank 2 tensor:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 4 & 3 & 2 \\ 8 & 7 & 6 & 5 \end{pmatrix}$$

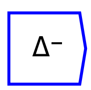
The running product of this tensor, along the horizontal dimension, is:

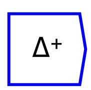
$$\begin{pmatrix} 1 & 2 & 6 & 24 \\ 5 & 20 & 60 & 120 \\ 8 & 56 & 336 & 1680 \end{pmatrix}$$

Running product will normally be applied to a Ravel or a variable defined via a Ravel.

4.2.13.11 difference Δ^-, Δ^+

Computes the nearest neighbour difference along a given direction.





These operators can be placed on the canvas in two ways:

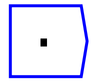
- From the *Scan Operations ("scan")* toolbar ; or
- By typing the letters "difference" or "differencePlus" on the canvas and then pressing the Enter key.

The optional argument (δ) can be used to specify the number of neighbours to skip in computing the differences. For example if your data is monthly and you want to calculate changes per year, you would set δ to 12. The length of the dimension being differenced is reduced by δ in the result.

It comes in two different forms which differ only in how the resultant x-vector is calculated. $\Delta_i^- = x_i - x_{i-\delta}$, and $\Delta_i^+ = x_{i+\delta} - x_i$, where i refers to the x-vector index.

Where there is more than one dimension to data, the user needs to specify the axis along which the difference operation is calculated.

4.2.13.12 inner product ·



Calculates the inner product of two tensors.
The operator can be placed on the canvas in two ways:

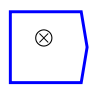
- From the *Tensor Operations ("tensor")* toolbar ; or
- By typing the letters "innerProduct" on the canvas and then pressing the Enter key.

It computes

$$z_{i_1,\ldots,i_{r_x-1},j_1,\ldots,j_{r_y-1}} = \sum_k x_{i_1\ldots,i_{a-1},k,i_{a+1}\ldots,i_{r_x-1}} y_{j_1,\ldots,j_{r_y-1},k},$$

where a is the given axis, and r_x and r_y are the ranks of x and y respectively.

4.2.13.13 outer product ⊗



Calculates the inner product of two tensors.
The operator can be placed on the canvas in two ways:

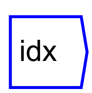
- From the *Tensor Operations ("tensor")* toolbar ; or
- By typing the letters "outerProduct" on the canvas and then pressing the Enter key.

It computes:

$$z_{i_1,i_2,\ldots,i_{r_x},j_1,\ldots,j_{r_y}} = x_{i_1,,i_2,\ldots,i_{r_x}} y_{j_1,\ldots,j_{r_y}}.$$

where r_x and r_y are the ranks of x and y respectively.

4.2.13.14 index



The operator can be placed on the canvas in two ways:

- From the *Tensor Operations ("tensor")* toolbar ; or
- By typing the letters "index" on the canvas and then pressing the Enter key.

Returns the index within the hypecube where the input is true (ie > 0.5). For example, where

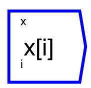
$$\iota(3,3) = \begin{pmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{pmatrix},$$

$$\mathrm{idx}(\iota(3,3) < 5) = \begin{pmatrix} 0 & 3 & nan \\ 1 & 4 & nan \\ 2 & nan & nan \end{pmatrix},$$

Note that the output array has the same shape as the input, with unused values padded with NANs (missing value).

Dimension and argument parameters are unused.

4.2.13.15 gather



Gather collects the values at index locations of the first argument, indexed by the second argument. If the index argument is a scalar, the output tensor's rank is one less than the first argument's rank, ie $[a_0, \ldots a_{j-1}, a_{j+1}, \ldots a_{ar}]$ where j is the axis along which the gather is performed. If the index argument is a tensor, however, it's shape is ignored, and the output tensor is the same rank as the first argument, and has shape $[i, a_0, \ldots a_{j-1}, a_{j+1}, \ldots a_{ar}]$ where i is the number of elements in the index argument's hypercube, and j is the axis along which the gather is performed.

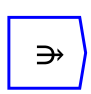
If the index is not an integer, the gather will linearly interpolate between the values on either side. So, for example, $x[2.5] = 0.5(x[2] + x[3])$. The x-vector elements are also linearly interpolated if the x-vector is a value or time type dimension, or the closest string label if a string type.

If the index value is outside the range of the x-vector along the axis being gathered, then NAN is assigned to that tensor element, and the x-vector gets a default assigned element (empty string, NaN or not-a-date).

The operator can be placed on the canvas in two ways:

- From the *Tensor Operations ("tensor")* toolbar  ; or
- By typing the letters "gather" on the canvas and then pressing the Enter key.

4.2.13.16 Meld



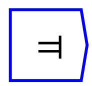
The operator can be placed on the canvas in two ways:

- From the *Tensor Operations ("tensor")* toolbar  ; or

- By typing the letters "meld" on the canvas and then pressing the Enter key.

The meld operation is used to insert additional values in an existing axis of a Ravel. For example, if you had data from two different time periods and you wished to merge them into one time dimension, you would use Meld to combine them.

4.2.13.17 Merge



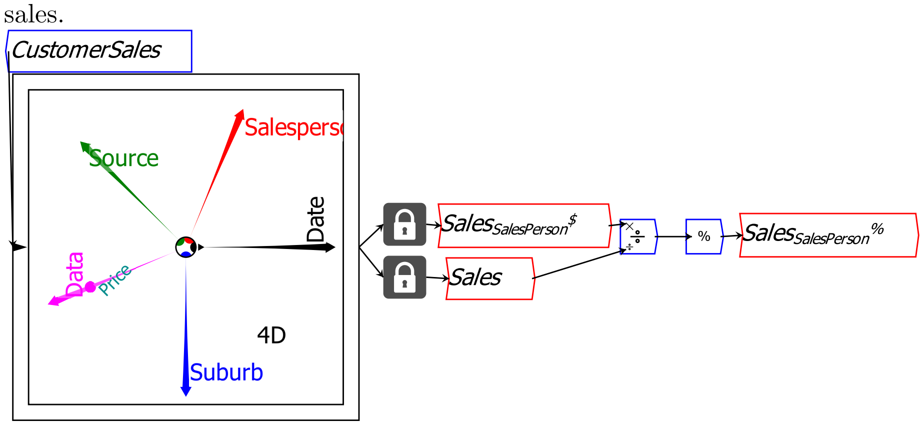
The operator can be placed on the canvas in two ways:

- From the *Tensor Operations ("tensor")* toolbar  ; or

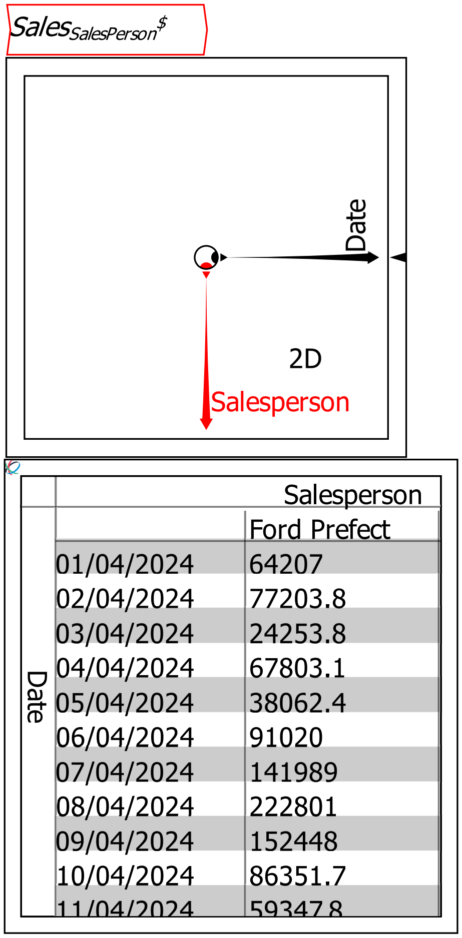
- By typing the letters "merge" on the canvas and then pressing the Enter key.

Merge combines two Ravels sharing *n* dimensions to produce a new Ravel with *n+1* dimensions.

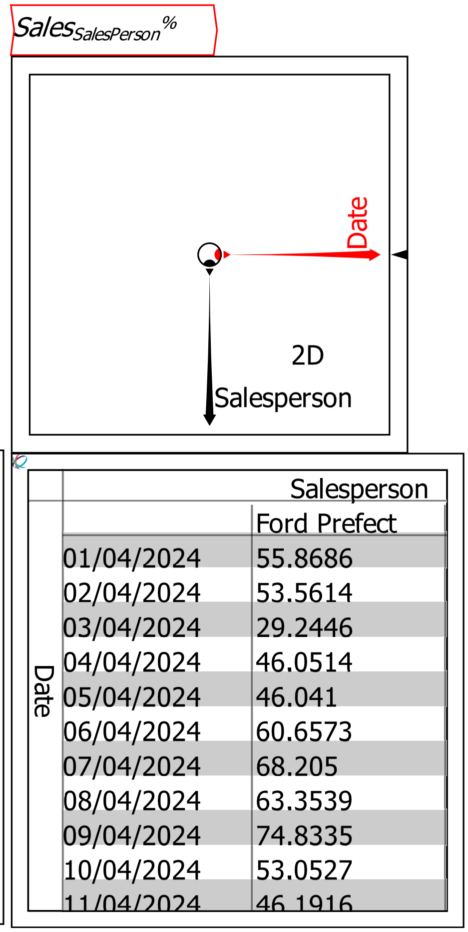
For example, the file below creates two 2-dimensional Ravels, one with sales in dollars and the other with sales by each salesperson as a percentage of total sales.



Sales$_{\mathrm{SalesPerson}^{\$}\mathrm{and\ SalesPerson}^{\%}}$share the same dimensions (SalesPerson and Date), and they can be separately displayed as in the next figure.

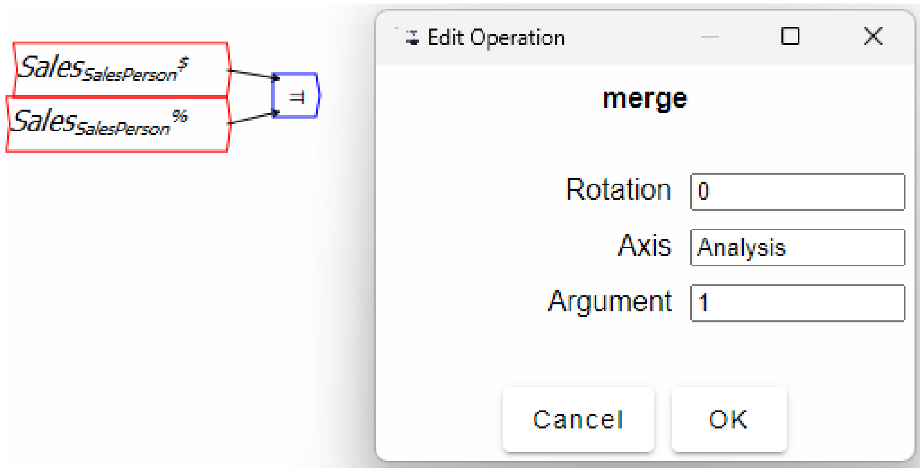


	Salesperson
Date	Ford Prefect
01/04/2024	64207
02/04/2024	77203.8
03/04/2024	24253.8
04/04/2024	67803.1
05/04/2024	38062.4
06/04/2024	91020
07/04/2024	141989
08/04/2024	222801
09/04/2024	152448
10/04/2024	86351.7
11/04/2024	59347.8

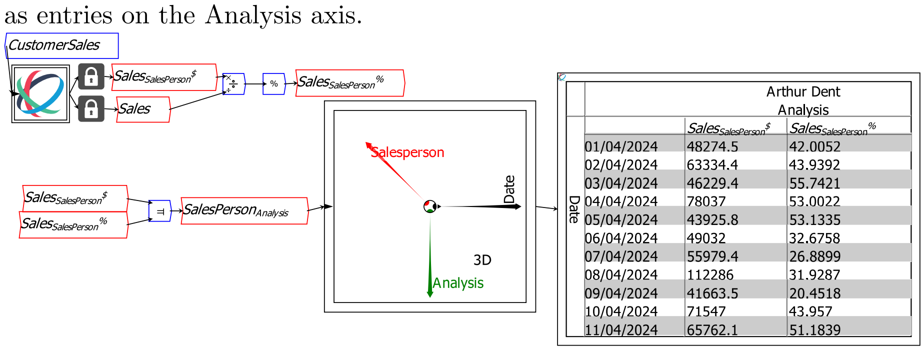


	Salesperson
Date	Ford Prefect
01/04/2024	55.8686
02/04/2024	53.5614
03/04/2024	29.2446
04/04/2024	46.0514
05/04/2024	46.041
06/04/2024	60.6573
07/04/2024	68.205
08/04/2024	63.3539
09/04/2024	74.8335
10/04/2024	53.0527
11/04/2024	46.1916

Using the Merge operator, you can create one new Ravel containing information from both the $ sales and the % of total sales analysis:

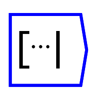


The merged Ravel has 3 dimensions, with the extra dimension Analysis now containing the Sales by dollar amount and Sales as a percentage of total sales as entries on the Analysis axis.



Date	Sales SalesPerson $	Sales SalesPerson %
01/04/2024	48274.5	42.0052
02/04/2024	63334.4	43.9392
03/04/2024	46229.4	55.7421
04/04/2024	78037	53.0022
05/04/2024	43925.8	53.1335
06/04/2024	49032	32.6758
07/04/2024	55979.4	26.8899
08/04/2024	112286	31.9287
09/04/2024	41663.5	20.4518
10/04/2024	71547	43.957
11/04/2024	65762.1	51.1839

4.2.13.18 Slice



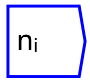
The operator can be placed on the canvas in two ways:

- From the *Tensor Operations (“tensor”)* toolbar  ; or
- By typing the letters “slice” on the canvas and then pressing the Enter key.

Slice will cut off a tensor along a given axis by the argument, as configured in the operation edit dialog. For example slice$(\{x_1, x_2, \ldots x_n\}, 3) = \{x_1, x_2, x_3\}$. If the tensor is rank one (ie a vector), it is not necessary to specify the axis.

If the slice argument is negative, then it refers to the number of elements from the end of that axis. For example slice$(\{x_1, x_2, \ldots x_n\}, -3) = \{x_{n-2}, x_{n-1}, x_n\}$.

4.2.13.19 Size



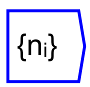
The operator can be placed on the canvas in two ways:

- From the *Tensor Operations ("tensor")* toolbar ; or
- By typing the letters "size" on the canvas and then pressing the Enter key.

Size refers to the number of elements along a named dimension given by the operation axisargument—eg a 3×2 rank 2 tensor with named axes "0" and "1", size("1")==2.

If the axis argument is left blank, the size returns the total number of elements present in the tensor. This may be less than the product of axis sizes if the data is sparse—which will normally be the case.

4.2.13.20 Shape



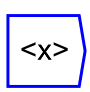
The operator can be placed on the canvas in two ways:

- From the *Tensor Operations ("tensor")* toolbar ; or
- By typing the letters "shape" on the canvas and then pressing the Enter key.

Returns a vector of axis sizes. Coupling this operation with a gather operation and variable would allow you interactively select axis size.

4.2.14 Statistical Operations

4.2.14.1 Mean

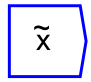


The operator can be placed on the canvas in two ways:

- From the *Statistics ("statistics")* toolbar ; or
- By typing the letters "mean" on the canvas and then pressing the Enter key

Returns the mean (or average) along a named dimension of all elements present. If the dimension is not named, then the mean is over all elements present in the tensor. Note that missing elements are not counted.

4.2.14.2 Median

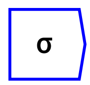


The operator can be placed on the canvas in two ways:

- From the *Statistics ("statistics")* toolbar ; or
- By typing the letters "median" on the canvas and then pressing the Enter key

Returns the median along a named dimension of all elements present. If the dimension is not named, then the median is over all elements present in the tensor.

4.2.14.3 Standard Deviation



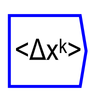
The operator can be placed on the canvas in two ways:

- From the *Statistics ("statistics")* toolbar ; or
- By typing the letters "stdDev" on the canvas and then pressing the Enter key

Returns the standard deviation along a named dimension of all elements present. If the dimension is not named, then the standard deviation is over all elements present in the tensor. Note that missing elements are not counted.

$$\sigma = \frac{1}{N-1}\sum_i^N (x_i - \langle x\rangle)^2$$

4.2.14.4 k-th moment



The operator can be placed on the canvas in two ways:

- From the *Statistics ("statistics")* toolbar ; or
- By typing the letters "moment" on the canvas and then pressing the Enter key

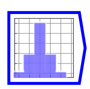
Returns the k-th moment about the mean along a named dimension of all elements present. k is specified by the numerical argument of the operation, which defaults to 1 (hence the result will be 0 in that case). If the dimension is not named, then the k-th moment is over all elements present in the tensor. Note that missing elements are not counted.

$$\langle \Delta x^k \rangle = \frac{1}{N} \sum_i (x_i - \langle x \rangle)^k$$

Also

$$\sigma^2 = \frac{N}{N-1} \langle \Delta x^2 \rangle$$

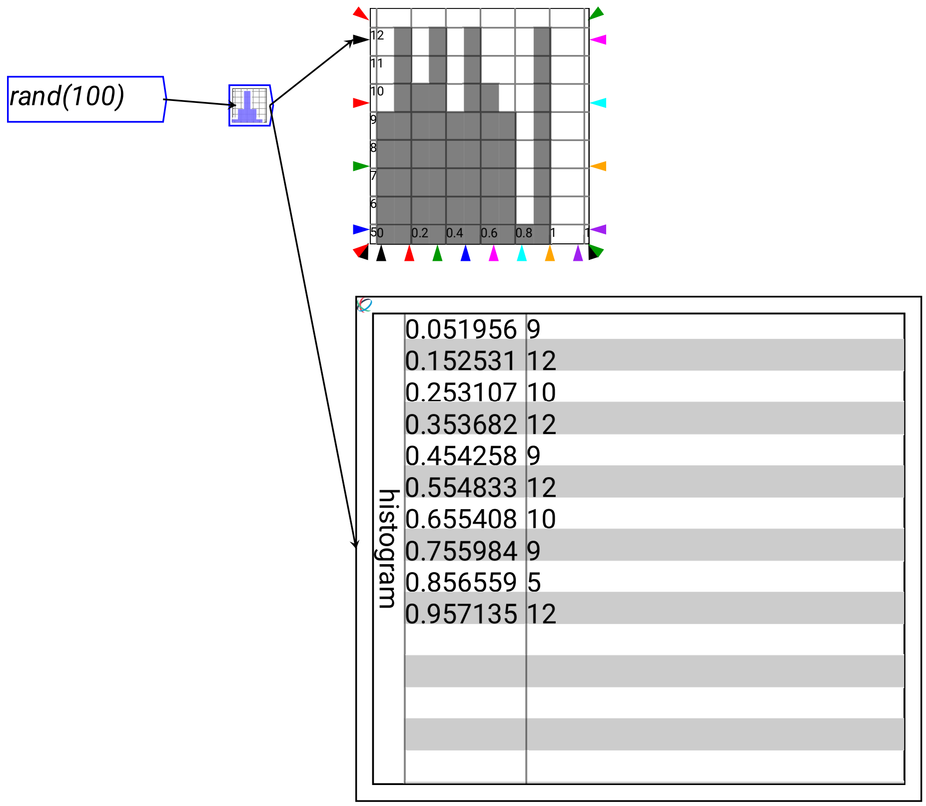
4.2.14.5 Histogram



The operator can be placed on the canvas in two ways:

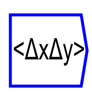
- From the *Statistics ("statistics")* toolbar  ; or
- By typing the letters "histogram" on the canvas and then pressing the Enter key

Computes the histogram along a named dimension of all elements present. If the dimension is not named, then the histogram is over all elements present in the tensor. The number of bins is specified by the numeric argument to the operation.



An example usage of the histogram operation

4.2.14.6 Covariance



The operator can be placed on the canvas in two ways:

- From the *Statistics ("statistics")* toolbar  ; or
- By typing the letters "covariance" on the canvas and then pressing the Enter key

Computes the covariance of two tensors along named dimension. If the inputs are of rank N and M respectively, the output will be a $(N-1)\times(M-1)$ rank tensor, where the (i,j) element is the covariance of the i-th slice of the first argument along the named dimension, and the j-th slice along the named dimension. As such, it is conformant with the definition of `cov` function in Octave, but not with the equivalently named function in Matlab:

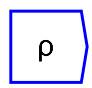
> Compatibility Note:: Octave always treats rows of X and Y as multivariate random variables. For two inputs, however, MATLAB treats X and Y as two univariate distributions regardless of their shapes, and will calculate covariance whenever the number of elements in X and Y are equal. This will result in a 2x2 matrix. Code relying on MATLAB's definition will need to be changed when running in Octave.

If only a single argument x is supplied to the covariance, then the result is equivalent to $\mathrm{cov}(x, x)$, ie each slice is covaried with each other slice.

The formula for covariance between stochastic variables x and y is

$$\mathrm{cov}(x, y) = \frac{1}{N-1}\sum_i (x_i - \langle x\rangle)(y_i - \langle y\rangle)$$

4.2.14.7 Correlation coefficient ρ



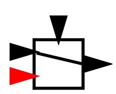
The operator can be placed on the canvas in two ways:

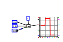
- From the *Statistics ("statistics")* toolbar  ; or
- By typing the letters "correlation" on the canvas and then pressing the Enter key

See covariance for the interpretation of tensor valued arguments. The correlation coefficient is defined as

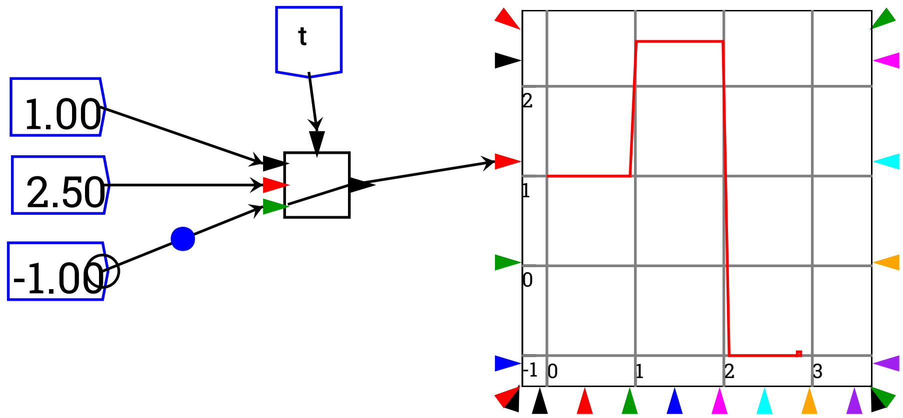
$$\rho(x, y) = \frac{\mathrm{cov(x,y)}}{\sigma(x)\sigma(y)}$$

4.2.15 Switch



The operator can be placed on the canvas by clicking on its icon  on the widget bar.

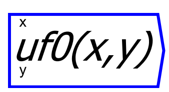
A switch block (also known as a case block, or select in the Fortran world) is a way of selecting from a range of alternatives according to the value of the input, effectively defining a piecewise function.



An example switch block with 3 cases

The default switch has two cases, and can be used to implement an if/then/else construct. However, because the two cases are 0 and 1, or false and true, a two case switch statement will naturally appear "upside down" to how you might think of an if statement. In other words, it looks like:
`if not` *condition* `then`
...`else`
...
You can add or remove cases through the context menu.

4.2.16 User defined function



A user defined function is a functioned defined by an algebraic expression. Support for this feature is courtesy of the wonderful exprtk library developed by Arash Partow.

A user defined function has a *name*, *parameters* and an *expression*. Example expressions are things like `x+y` or `sin(x)`. More details of the sorts of expressions possible can be found in the User Defined Functions section of the manual.

The parameters are specified as part of the name, so a user defined function adding x and y would be called `useradd(a,y)` and the sin example might be called `mysin(x)`. Functions with up to two arguments can be wired on the canvas. User defined functions can call other user defined functions, so specifiying more than 2 parameters can be a useful thing to do.

4.2.17 Godley Tables

A Godley Table employs double-entry bookkeeping to create dynamic models of monetary transactions. It can also be used to build dynamic models of systems based on transitions between exclusive categories—such as epidemiological systems like SEIRD models of disease transmission.

A Godley Table uses the rules of accounting to create consistent equations to describe financial flows. The essential rules are that:

- Each flow must be entered twice on its row; and

- The sum of $Assets - Liabilities - Equity$ must equal zero on each row.

Using these rules, *Minsky* develops stock-flow consistent models of financial transactions. The row entries are flows in differential equations for the stocks. For example, the Godley Table shown in the next figure:

	Asset		Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves* ▼	*Loans* ▼	*Deposits* ▼	*Banks$_{Equity}$* ▼	*0*
Initial Conditions	0	0	0	0	*0*
Bank lending		*Lending*	*Lending*		*0*
Interest on Loans			*-Interest*	*Interest*	*0*
Bank Debt Repayment		*-Repay*	*-Repay*		*0*
Bank Spending			*Spend$_{Banks}$*	*-Spend$_{Banks}$*	*0*

generates the following set of differential equations:

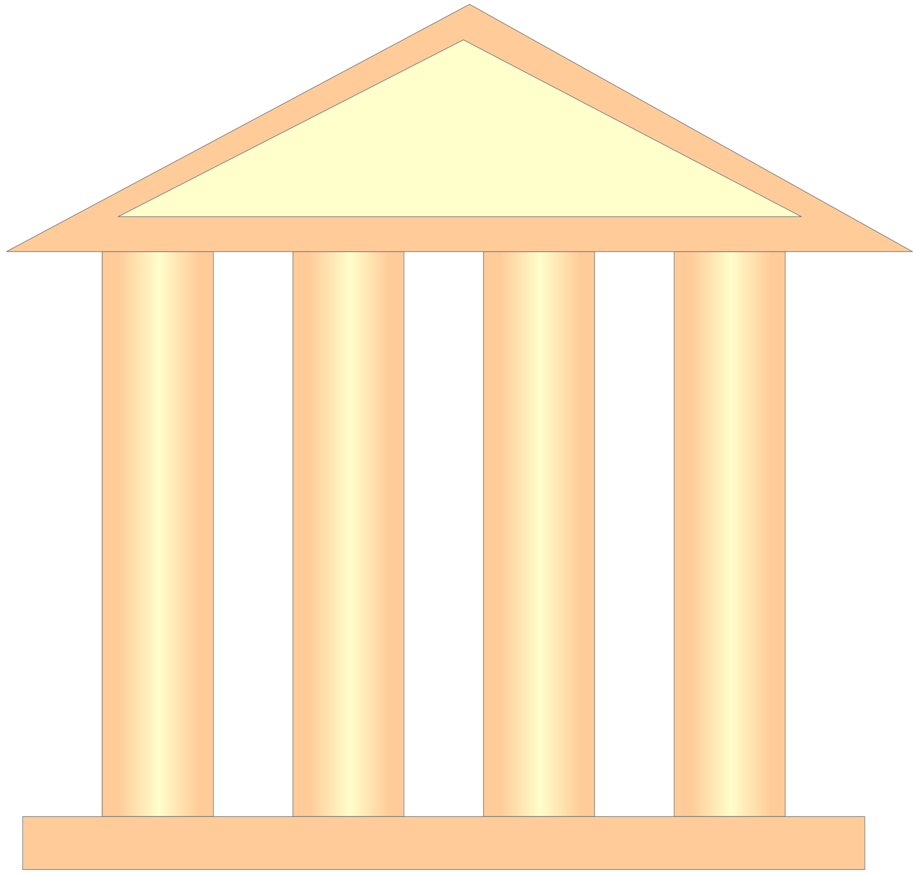
$$\frac{d}{dt}Loans = Lending - Repay$$
$$\frac{d}{dt}Deposits = Lending - Interest - Repay + Spend_{Banks}$$
$$\frac{d}{dt}Banks_{Equity} = Interest - Spend_{Banks}$$
$$\frac{d}{dt}Reserves = 0$$

Stocks are thus completely defined by the Godley Table itself as equations in a set of coupled differential equations. For this reason, when placed on the canvas, each stock has an output but no input. Flows, on the other hand, have both inputs and outputs, and have to be fully defined on the canvas itself.

For complete details see Godley tables.

4.2.17.1 Creating a Godley Table

To insert a Godley Table into a model, click on the Godley icon on the Operations bar. This attaches a Godley Table to your cursor. Click where you wish to place it on the canvas and this object will be inserted:



You can edit this object by either double-clicking on the icon, or by choosing “Edit Godley Table” from its context menu.

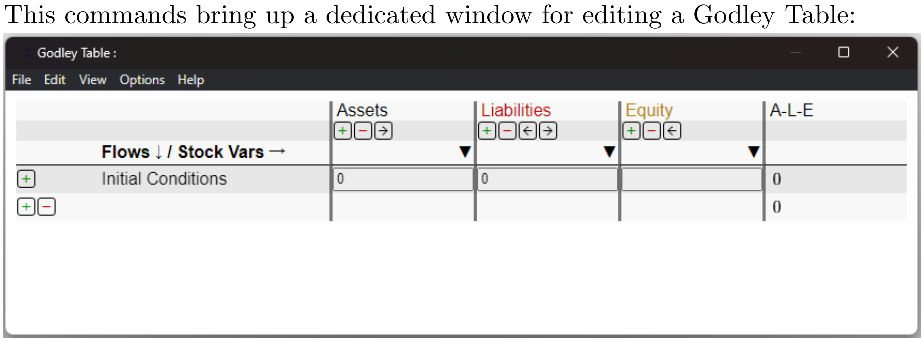
4.2.17.2 Godley Table Context Menu

The Godley-Table-specific commands in the context menu are:

Command	Effect
Open Godley Table	Opens the form for editing Godley Tables
Title	Give the Table a name which is displayed on the icon and edit form
Set currency	If multiple currencies are used in a model, specify the currency for the current table
Editor Mode	Convert the canvas view of the table from an icon to the table itself, which can be edited on the canvas
Row/Col buttons	When Editor Mode is active, show row and column buttons used to add and delete rows and columns
Display Variables	Show the stocks and flows defined in the table as variables attached to the table in icon view
Copy flow variables	Copy all the flow variables in the table to the canvas
Copy stock variables	Copy all the stock variables in the table to the canvas
Export as	Export the table in either CSV or LaTeX format
Rotate GodleyIcon	This command has been deprecated and will be deleted in a future release

4.2.17.3 Open Godley Table

This commands bring up a dedicated window for editing a Godley Table:



Defining Stocks and Flows A Stock is defined by entering its name in the row on which the label *Flows↓/Stock Vars→* is shown. Normally these stocks will be bank accounts, though Godley Tables can be used to model any dynamic system in which the stocks define exclusive categories—such as a epidemiological model of disease transmission, in which the population is divided into mutually exclusive categories such as Infected or Uninfected, in Hospital or Not in Hospital, etc.

Stocks are classified as either *Assets*, *Liabilities* or *Equity*, and the default form enables the entry of only one Asset, Liability and Equity (similarly, there is only one row for recording flows on the default form). To add more columns for accounts, click on the + key below the relevant class (similarly, additional rows for financial transactions are added by clicking on the + key next to a row). This creates an additional column for entering stock names. The - key deletes a stock, and the arrow keys move stocks right and left as desired.

If you press the leftarrow key on the first entry in Liabilities, *Minsky* will warn you that this will change the classification of the stock from a Liability to an Asset.

The final column in the Table, labelled $A - L - E$, checks whether the row sums to zero—which it must do to obey the rules of accounting. The entries in the columns, and the sum itself, are *symbolic*: words are entered rather than numbers. The sum of that row must be zero according to this formula. Normally this will involve two entries of the same word—for example, *Lending*—though fractional entries are possible: $Lending - 0.7Lending - 0.3Lending$ still sums to zero.

Three stocks and flows have been defined in the following figure.

	Asset		Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves* ▼	*Loans* ▼	*Deposits* ▼	*Banks$_{Equity}$* ▼	*0*
Initial Conditions	0	0	0	0	*0*
Bank lending		*Lending*	*Lending*		*0*
Interest on Loans			*-Interest*	*Interest*	*0*
Bank Debt Repayment		*-Repay*	*-Repay*		*0*

Initial Conditions These are the values which the stocks take at the beginning of a simulation. This row must also sum to zero, to ensure stock-flow consistency.

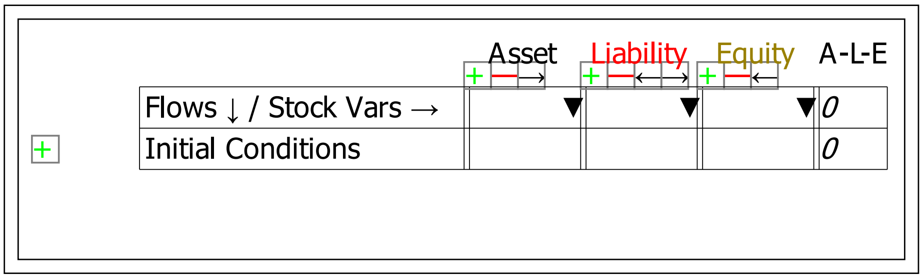
4.2.17.4 Editor Mode

This toggle command converts the canvas display of the Godley table from the icon to a double-entry bookkeeping view:

	Asset	Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	▼	▼	▼	*0*
Initial Conditions				*0*

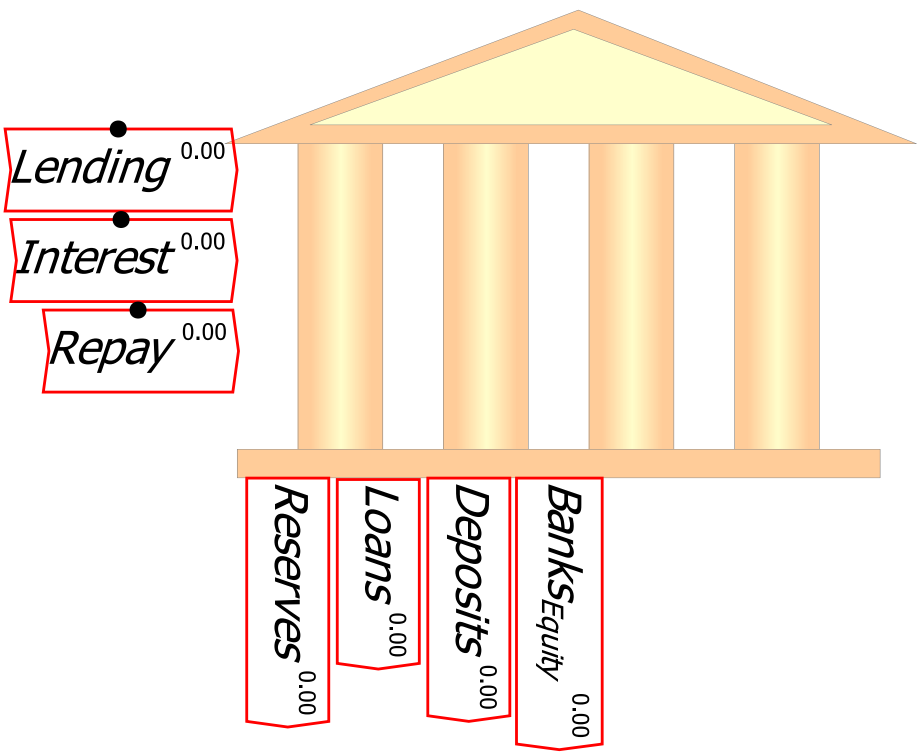
4.2.17.5 Row/Col buttons

This toggle command adds row and column buttons to a double-entry bookkeeping view of the table on the canvas, so that it can be edited on the canvas as an alternative to editing in the dedicated window.



4.2.17.6 Display Variables

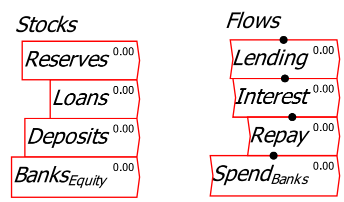
This toggle command places the stocks (bank accounts) and flows (financial transactions) entered in a Godley Table as variables on the Design canvas, attached to the Godley Table icon itself.



4.2.17.7 Copy Stock/Flow Variables

These commands copy all Stocks or all Flows from a Godley Table and attach them to the mouse cursor, for placement on the canvas. The flows can then be defined using *Minsky*'s mathematical operators (Stocks are already completely defined by the Godley Table itself). The next figure shows the result of using both these commands to place all stocks and flows on the canvas.

	Asset		Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves* ▼	*Loans* ▼	*Deposits* ▼	*Banks$_{Equity}$* ▼	*0*
Initial Conditions	0	0	0	0	*0*
Bank lending		*Lending*	*Lending*		*0*
Interest on Loans			*-Interest*	*Interest*	*0*
Bank Debt Repayment		*-Repay*	*-Repay*		*0*
Bank Spending			*Spend$_{Banks}$*	*-Spend$_{Banks}$*	*0*



4.2.17.8 Create multiple related Godley Tables

Minsky uses the fact that one entity's financial asset is another's financial liability to rapidly build interlocked Godley Tables, which describe the financial system from the perspective of every entity or sector modelled in it. The inverted wedge ▼ next to the name of every column is used to check for Assets that have not yet been recorded as a Liability, and vice versa. The next figure shows an expanded Godley Table for the Banking Sector, with additional Tables added for the Non-bank private sector, the Central Bank, and the Treasury.

Banks

	Asset			Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves*	*Loans*	*$Bonds_{Banks}$*	*Deposits*	*$Banks_{Equity}$*	*0*
Initial Conditions	0	0	0	0	0	*0*
Bank lending		*Lending*		*Lending*		*0*
Interest on Loans				*-Interest*	*Interest*	*0*
Bank Debt Repayment		*-Repay*		*-Repay*		*0*
Bank Spending				*$Spend_{Banks}$*	*$-Spend_{Banks}$*	*0*
Net government spending	*Deficit*			*Deficit*		*0*
Bond sales	*$-BondSales_{Treasury}$*		*$BondSales_{Treasury}$*			*0*
Interest on Bonds	*Int_{Bonds}*				*Int_{Bonds}*	*0*
Central Bank bond purchases	*$BondBuying_{CB}$*		*$-BondBuying_{CB}$*			*0*

Private Nonbanks

	Asset	Liability	Equity	A-L-E
Flows ↓ / Stock Vars →				*0*
Initial Conditions				*0*

Central Bank

	Asset	Liability	Equity	A-L-E
Flows ↓ / Stock Vars →				*0*
Initial Conditions				*0*

Treasury

	Asset	Liability	Equity	A-L-E
Flows ↓ / Stock Vars →				*0*
Initial Conditions				*0*

As well as auto-completing much of the additional Godley Tables, the system indicates the need for additional accounts that are not part of the initial Banks Godley Table. Central Bank purchases of bonds from private banks indicate the need for the *Asset* $Bonds_{CB}$, while the deficit indicates the need to add the Treasury's account at the Central Bank, the "Consolidated Revenue Fund", as an additional liability of the Central Bank.

Banks

	Asset			Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves*	*Loans*	*$Bonds_{Banks}$*	*Deposits*	*$Banks_{Equity}$*	*0*
Initial Conditions	0	0	0	0	0	*0*
Bank lending		*Lending*		*Lending*		*0*
Interest on Loans				*-Interest*	*Interest*	*0*
Bank Debt Repayment		*-Repay*		*-Repay*		*0*
Bank Spending				*$Spend_{Banks}$*	*$-Spend_{Banks}$*	*0*
Net government spending	*Deficit*			*Deficit*		*0*
Bond sales	*$-BondSales_{Treasury}$*		*$BondSales_{Treasury}$*			*0*
Interest on Bonds	*Int_{Bonds}*				*Int_{Bonds}*	*0*
Central Bank bond purchases	*$BondBuying_{CB}$*		*$-BondBuying_{CB}$*			*0*

Private Nonbanks

	Asset	Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Deposits*	*Loans*	*$Private_E$*	*0*
Initial Conditions	0	0	0	*0*
Net government spending	*Deficit*		*Deficit*	*0*
Interest on Loans	*-Interest*		*-Interest*	*0*
Bank lending	*Lending*	*Lending*		*0*
Bank Debt Repayment	*-Repay*	*-Repay*		*0*
Bank Spending	*$Spend_{Banks}$*		*$Spend_{Banks}$*	*0*

Central Bank

	Asset	Liability		Equity	A-L-E
Flows ↓ / Stock Vars →	*$Bonds_{CB}$*	*Reserves*	*$Treasury_{CRF}$*	*CB_E*	*0*
Initial Conditions	0	0	0	0	*0*
Central Bank bond purchases	*$BondBuying_{CB}$*	*$BondBuying_{CB}$*			*0*
Bond sales		*$-BondSales_{Treasury}$*	*$BondSales_{Treasury}$*		*0*
Net government spending		*Deficit*	*-Deficit*		*0*
Interest on Bonds		*Int_{Bonds}*	*$-Int_{Bonds}$*		*0*

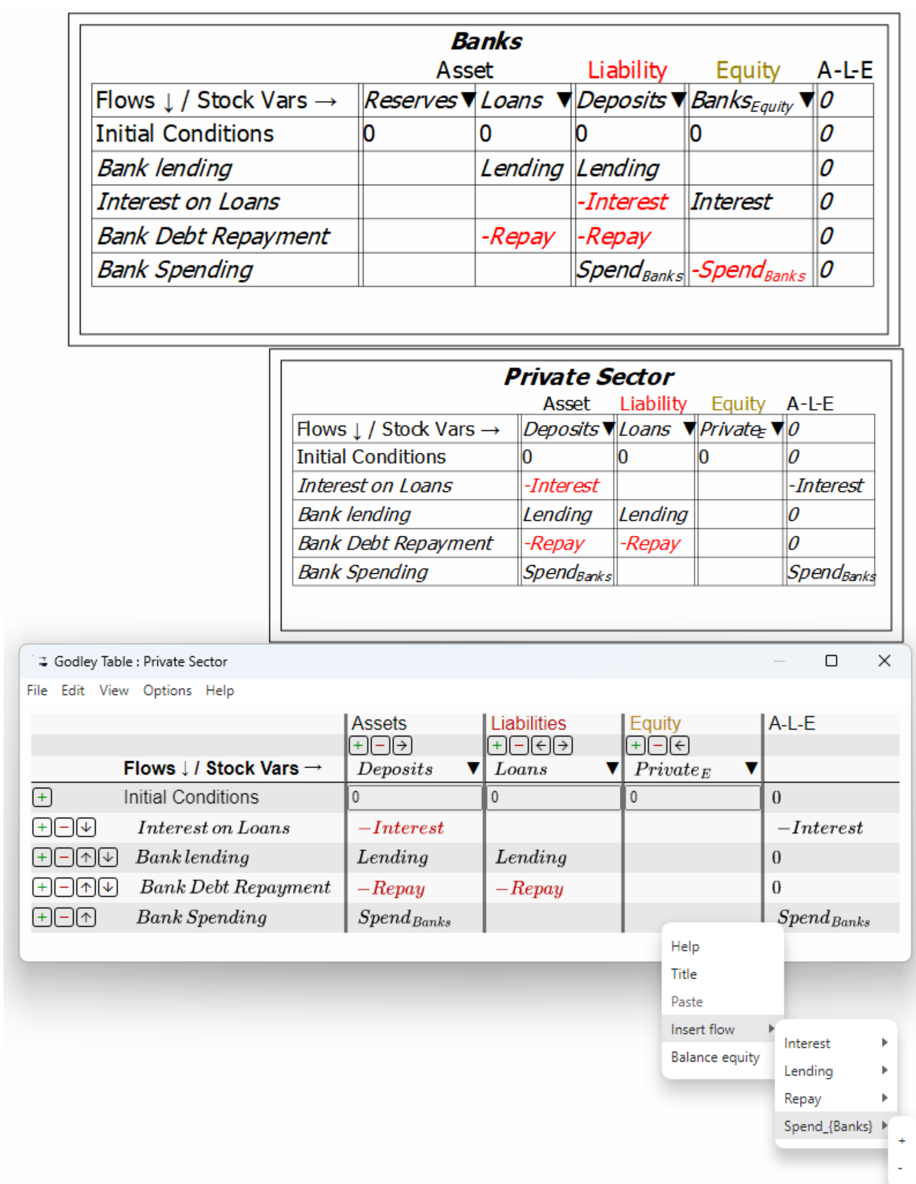
Treasury

	Asset	Liability		Equity	A-L-E
Flows ↓ / Stock Vars →	*$Treasury_{CRF}$*	*$Bonds_{Banks}$*	*$Bonds_{CB}$*	*$Treasury_E$*	*0*
Initial Conditions	0	0	0	0	*0*
Bond sales	*$BondSales_{Treasury}$*	*$BondSales_{Treasury}$*			*0*
Net government spending	*-Deficit*			*-Deficit*	*0*
Interest on Bonds	*$-Int_{Bonds}$*			*$-Int_{Bonds}$*	*0*
Central Bank bond purchases		*$-BondBuying_{CB}$*	*$BondBuying_{CB}$*		*0*

The inverted wedge next to the Equity column supports *nonfinancial* assets—things like houses, and shares valued in excess of their limited liability. If these are included amongst the assets of an entity, then they can also be recorded as an Equity of that entity (this feature is still under development).

4.2.17.9 Godley Table Context Menu

The Godley Table Form has a context menu which assists in the rapid completion of a multi-Godley-Table model. Entries can be copied from one cell and pasted into another, and Flows in the model can be inserted into a Godley Table cell as a positive or a negative entry:



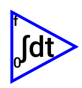
It is also possible to record all as-yet-unclassified flows as additions to or substractions from the Equity of the relevant sector—which is often (but not always) what is needed to complete a set of interlocking Godley Tables. In the figure above, $-Interest$ and $+Spend_{Banks}$ are shown in the $A-L-E$ checksum column. Choosing "Balance equity" from the context menu transfers both these entries to the Equity column.

4.2.17.10 Export Godley Table

This command exports the Godley Table either as a CSV file, or a LaTeX file, which can be imported into other programs for documentation purposes.

4.2.18 integrate $\int dt$

Creates an integration (or stock) variable.



The operator can be placed on the canvas in two ways:

- From the integrate widget  on the toolbar; or
- By typing the letters "integrate" on the canvas and then pressing the Enter key.

Integral blocks require two inputs: the flow being integrated, which is attached to the top input on the block, labelled f; and the initial conditions for the integral, which is attached to the bottom input, labelled 0.

4.2.18.1 Integral Block context menu

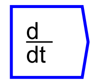
The integral-block-specific commands on the context menu are:

Command	Effect
Edit	Bring up the Edit form for this integrals
Copy item	Copy the variable name, rather than the combined integral block plus name
Toggle var binding	Enable the integral block to be separated from the variable name
Rotate IntOp	Rotate the combined integral block plus name

Editable attributes include the variable's name and its initial value at $t = 0$. The main role of the Edit form is to provide the name for the integrated variable—other details included on the form can be controlled from elsewhere in the program. Type the name you want to give the variable into the Name field and press Enter or click on OK. You can also name the variable via the context menu item "Rename all instances".

The flows to be integrated are connected to the top port of the integral block, labelled 'f'. The bottom port, labelled '0', can optionally be connected to a constant, parameter or variable, which is used to specify the initial value of the integral, following the same rules as the variable initial value field.

4.2.19 differentiate d/dt



Symbolically differentiates its input with respect to system time, producing d/dt[input]. For further explanation regarding differentiation, see this wikipedia page.

4.3 Variables and Parameters

Variables represent values in a calculation, and come in a number of varieties:

Constants represent an explicit numerical value, and do not have a name. Their graphical representation shows the actual value of the constant.

Parameters are named constants. All instances of a given name represent the same value, as with all other named variables, so changing the value of one parameter, either through its edit menu, or through a slider, will affect all the others of that name.

Parameters are used to import data from a CSV file, which is how external data is imported into Ravel for analysis.

Parameters can also be used to generate arrays of numbers for use in a *Ravel* file using built-in formulas which are placed in the "Initial value" field of the parameter definition form. See tensor valued initial conditions.

Flow variables have an input port that defines how the value is to be calculated. Only one flow variable of a given name can have its input port connected, as they all refer to the same quantity. If no input ports are connected, then flow variables act just like parameters.

Integral variables represent the result of integrating its input over time by means of the differential equation solver. The integrand is represented by the input to an integral operator that is attached to the integral variable.

Stock variables are the columns of Godley tables, and represent the integral over time of the sum of the flow variables making up the column.

Variables may be converted between types in the variable edit menu, available from the context menu, subject to certain rules. For example, a variable whose input is wired anywhere on the canvas cannot be changed from "flow". Stock variables need to be defined in a Godley table, and so on.

4.3.1 Variable names

Variable names uniquely identify variables. Multiple icons on the canvas with the same name at the same level all refer to the same variable. Variable names can have local scope, in which case it is specific to the group in which it is contained, othewise its scope is determined by the innermost containing group that has a local variable of that name. If no such variable exists, its scope is global.

A variable which is local to a group can be used as a formula in several locations in a file. For example, if you use a, b anc c as constants in a polynomial $a + bx + cx^2$ and make them local to a group, then the group can be used in several parts of a document, with the argument x and the values of a,b and c being different in every instance.

4.3.2 Initial conditions

Variable initial conditions can be defined through the "init value" field of the variable edit menu, or in the case of Godley table stock variables, through the initial condition row of the Godley table. An initial value can be a simple number, or it can be a multiple of another named parameter. In case of symbolic definitions, it would be possible to set up a circular reference where the initial value of variable A is defined in terms of the initial value of variable B, which in turn depends on the intial value of A. Such a pathological situation is detected when the system is reset.

4.3.3 Tensor valued initial conditions

Ravel provides a simple functional language which allows for the generation of tensor-valued operations. This enables *Ravel* to work with hypothetical as well as actual data—for example, hypothetical sales volumes for a new product from zero to 500,000 sales per year can be generated using the iota formula by putting the expression "iota(500001)" in the initial value field for the variable *Sales*. This generates an axis/dimension named 0 with values from 0 to 500,000.

These functions take the form $func(n_1, n_2, \ldots, n_r)$ where r is the desired rank, and n_1, n_2, etc are the dimensions of the tensor. Available functions include:

name	description
`one`	the tensor is filled with '1'
`zero`	the tensor is filled with '0'
`iota`	the arithmetic sequence $(0, 1, ... \prod_i n_i)$
`eye`	diagonal elements filled with '1', offdiagonal '0'
`rand`	tensor filled with random numbers in the range $[0, 1)$

- `eye` is equivalent to `one` for vectors.
- `rand` generates different random numbers each time the simulation is reset, and uses the clib `rand()` function.

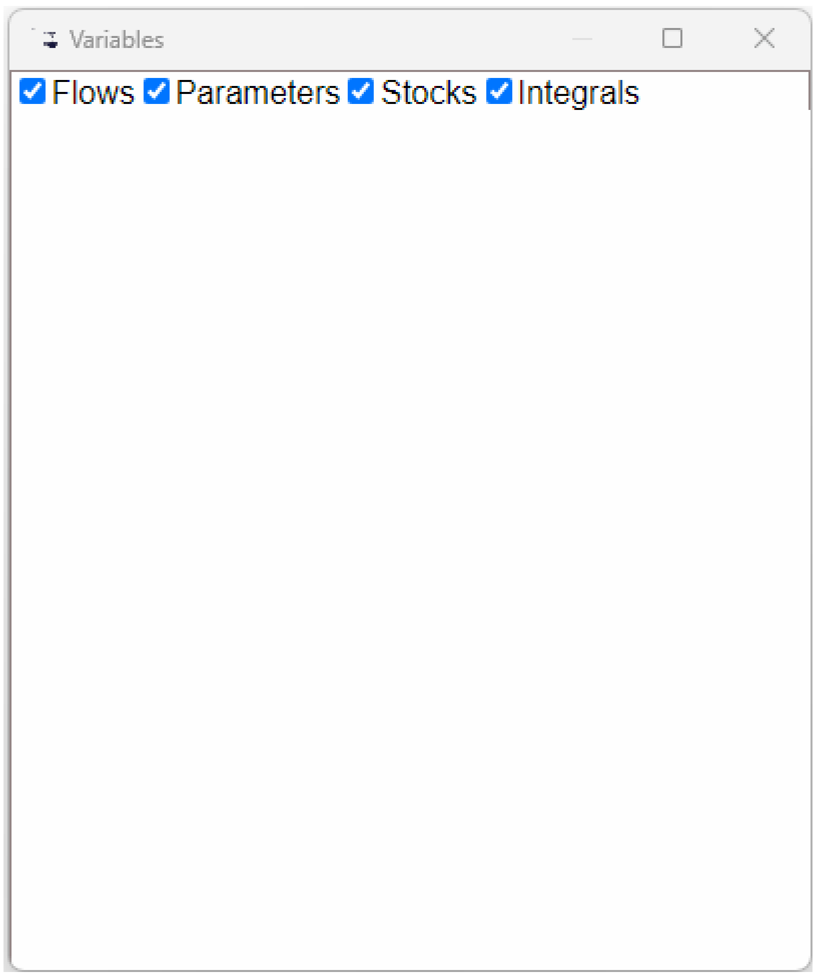
4.3.4 Sliders

Variables have a "slider" feature, which allows the value of the variable (or parameter or constant) to be altered by the user via the mouse or by arrowkeys.

The options Max, Min and Step Size in the Edit form let you control the maximum and minimum values for the variable/parameter, and how much the value changes for each press of an arrowkey. A relative slider means that the step size is expressed as a fraction of max-min.

Adjusting the slider of an integral (or stock) variable while the system is running actually adjusts the present value of the variable. The sliders can also be adjusted using the keyboard arrow keys: uparrow ↑ or rightarrow → increases the value of the parameter by the step size; downarrow ↓or leftarrow ← reduces it.

4.3.5 Variable Browser



The browser can be accessed in two ways:

- From the Insert Menu; or
- From the  widget, which brings up a horizontal menu for defining, constants, parameters and variables. Click on the final option on this menu, "Browser", to bring up the window.

The *variable browser* is a popup window that shows all currently defined variables in the system. It is an "always on top" window, which will be visible over any other window including those for *Ravel/Minsky* (and those for other programs which you have open at the time). This facilitates its use as a means to insert multiple variables into a model.

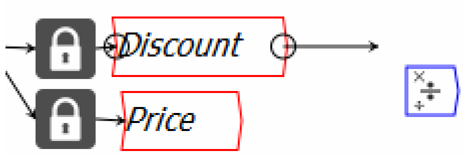
This is a convenience toolbar that allows one to select a variable for insertion into the design canvas, instead of having to type the new variable's name from scratch.

At the top of the variable browser are some filter checkboxes, that allow you filter the variables shown by variable type.

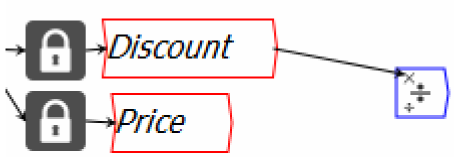
4.4 Wires

4.4.1 Connecting operations with wires

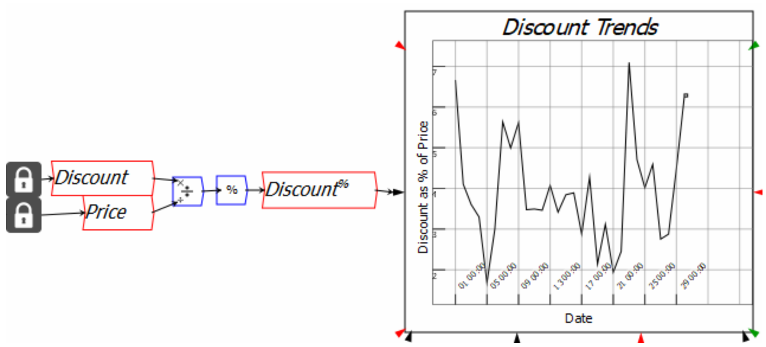
A wire represents the flow of values from one operation to the next. To add a wire to the canvas, click on the output port of an operator or variable (on the right hand side of an icon in its initial unrotated orientation), and then drag it towards an input port (on the left hand side of an unrotated icon); an arrow will be drawn out of the port. The next figure shows an arrow being drawn out of an output port and dragged towards an input port on a divide block.



When the mouse is released, Ravel searches for a nearby input port and attaches the wire there.



When the other variable is wired to the divide operator, the output can be further processed and allocated to another variable for further analysis or display.

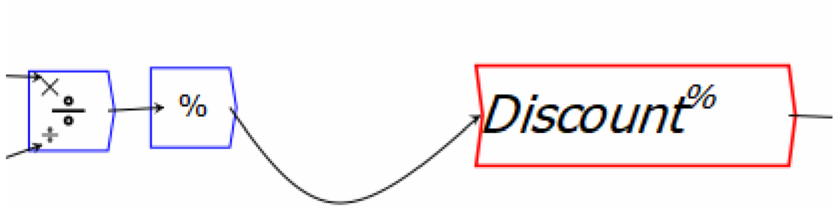


The Equation Tab and LaTeX export shows the equation generated by this wiring: $\mathrm{Discount}^{\%} = \left(\frac{\mathrm{Discount}}{\mathrm{Price}}\right)$.

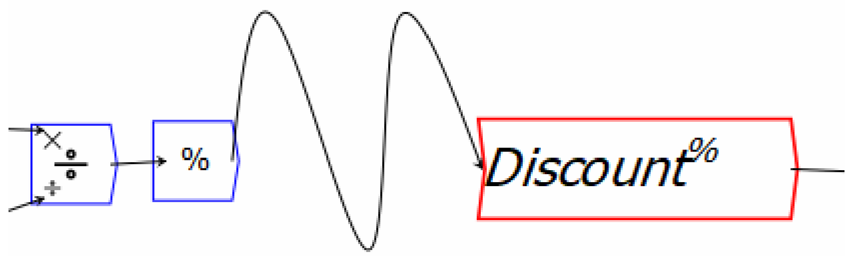
You can't connect an operator to itself (that would be a loop, which is not allowed, unless passing through an integral), nor in general can an input port have more than one wire attached—the exceptions are mathematical operators such as $+/-$ and $\times/\div$ (and similarly max/min), where the multiple wires are summed or multiplied, respectively.

4.4.2 Curving Wires

A wire can be bent by hovering over it, at which point a blue dots ("handle") will appear. Click on the blue dot—or anywhere along the line—and the line will turn into a curve, with 3 "bezier" points showing the degree of curvature.

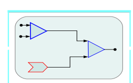


Additional curves can be added to a line, which can allow you to connect variables without drawing a line across some other entity.



However, it can be preferable to copy a variable and paste into a document multiple times, rather than having curved wires cluttering up a diagram. Wires can also become excessively kinked by curving them, and there is a context-menu command to straighten a curved line.

4.5 Groups



Grouping gives the capability to create reusable modules, or subroutines that can dramatically simplify more complicated systems. Groups may be created in the following ways:

- by lassoing a number of items to select them, then selecting "group" from the canvas context menu, or the edit menu.
- by pasting the selection. You may "ungroup" the group from the context menu if you don't desire the result of the paste to be a group.
- by copying another group
- by inserting a Minsky file as a group

Groups are "transparent": at a low level of magnification, the group icon is shown, but as you zoom in on a group, it will become transparent, which allows you see and edit its contents.

Groups may be nested heirarchically, which gives an excellent way of zooming in to see the detail of a model, or zooming out to get an overview of it. The group context menu item "Zoom to display" zooms the canvas in just enough for the group's contents to be visible.

You may also select "Open in canvas" from the context menu. This replaces the current canvas contents with the contents of the group, allowing you to edit the contents of the group directly without the distractions of the rest of the model. Select "Open master group" to return to the top level group occupying the canvas.

Around the edges of a group are input or output variables, which allow one to parameterise the group. One can drag a variable and dock it in the I/O area to create a new input or output for the group.

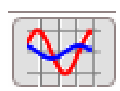
When creating a group, or dragging a variable or operation into or out of a group, if a wire ends up crossing the group boundary, a new temporary variable is added as an I/O variable. You may then edit the I/O variable name to be something more meaningful to your model.

Variable names within groups can be locally scoped to that group. That means that a variable of the same name outside the group refers to a different entity completely. By default, grouped variables refer to entities outside the group scope, but may be marked local by means of context menu option. One can also convert all variables in a group to be local by means of the "Make subroutine" context menu entry.

Nonlocal variables refers to a local variable within an outer scope, going all the way to global scope if no such variable exists. In this way, two groups can share a variable reference to a variable, and you can limit the scope of the shared variable by placing a local variable of the same name in an outer group that both groups are contain within.

A group can also be exported to a file from the context menu. This allows you to build up a library of building blocks. There is a github project "minsky-models" allowing people to publish their building blocks and models for others to use. In the future, we hope to integrate *Minsky* with this github repository, allowing even more seamless sharing of models.

4.6 Plot widget



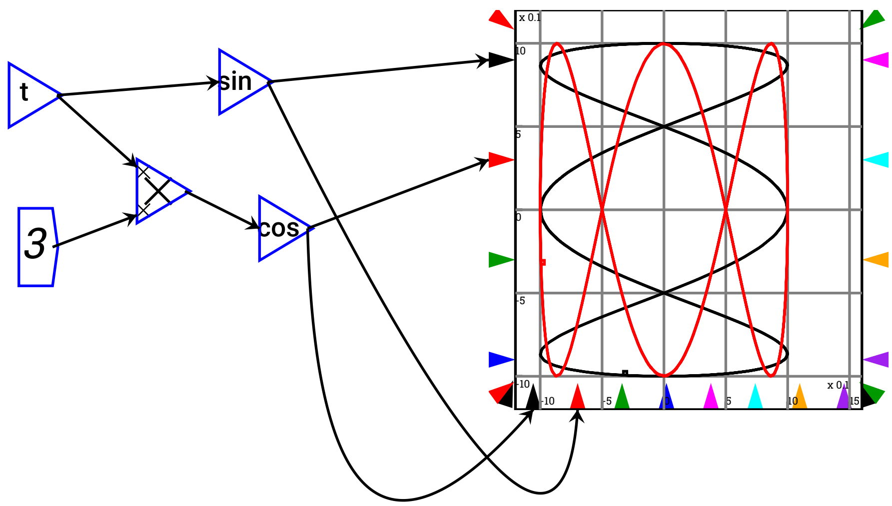
The plot widget embeds a plot into the canvas.

Around the outside of the plot are a number of input ports that can be wired:

- left hand edge Multiple series can be plotted by attaching them as inputs to the black input triangle on the middle of the Y1 axis.

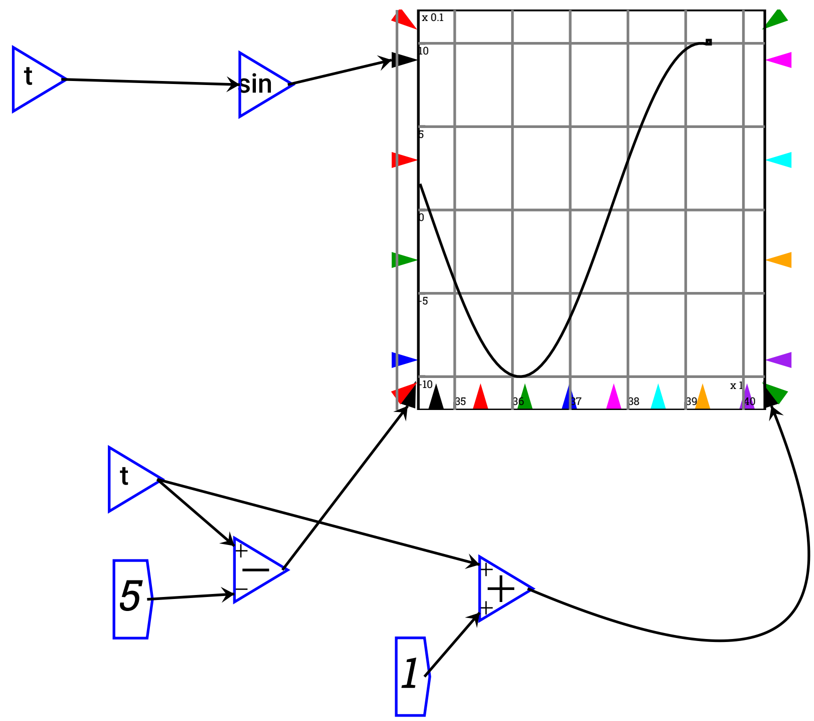
- right hand edge Multiple series can be plotted by attaching them as inputs to the red input triangle on the middle of the Y2 axis. These are shown on a different scale to the left hand Y1 inputs, which allows very different magnitudes to be compared on the one plot.

- bottom edge The default for the X-axis is either time—for a *Minsky* simulation model—or the right-pointing axis of a Ravel. XY plots can be created by attaching an input to the matching input on the X axis: the input to the black triangle will be plotted against the variable(s) inputted on the black Y1 input port; and the input to the red triangle will be plotted against the variable(s) inputted on the red Y2 input port.

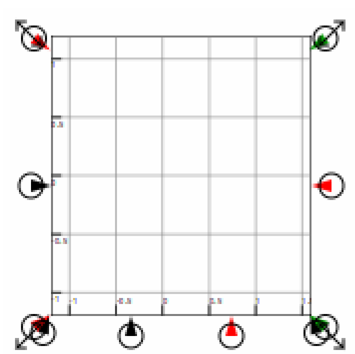


If only one bottom port is connected, then that controls all pens simultaneously, and if no ports are connected, then the simulation time is used to provide the x coordinates

- corners Corner ports control the scale. You can wire up variables controlling minimum and maximum of the x, y and right hand y axes. If left unwired, the scales are determined automatically from the data—or set by fields in the Options form. Corner ports can also be used to, for example, implement a sliding window graph—so that the scale of the graph adapts to the simulation in a *Minsky* model.

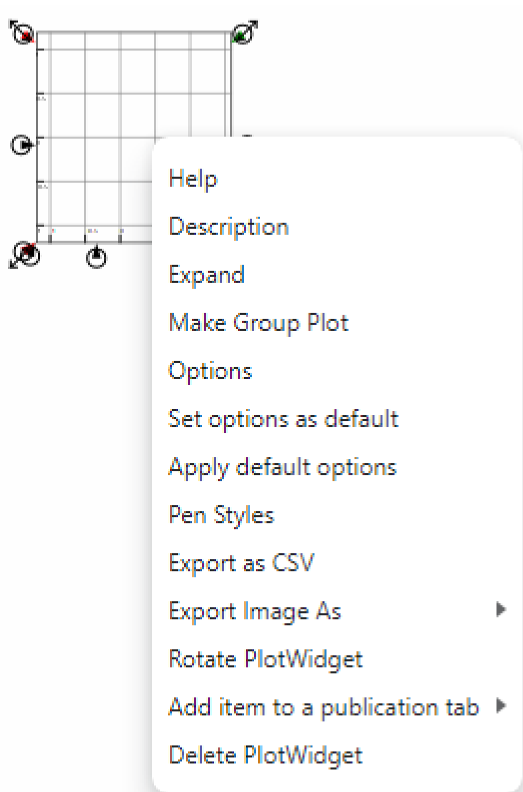


Plots can be resized using the resize arrows that appear when the mouse hovers over a plot. Click on one of these arrows and drag the mouse and the plot will be resized.



4.6.1 Plot context menu

The plot context menu is:



4.6.1.1 Description

Description bring up a text window into which you can enter text to describe a plot. The Short Description field becomes a "Hint" that is displayed when the mouse hovers over a plot. If you check the Bookmark field, the Short Description becomes a bookmark which can be used to navigate the document.

4.6.1.2 Expand

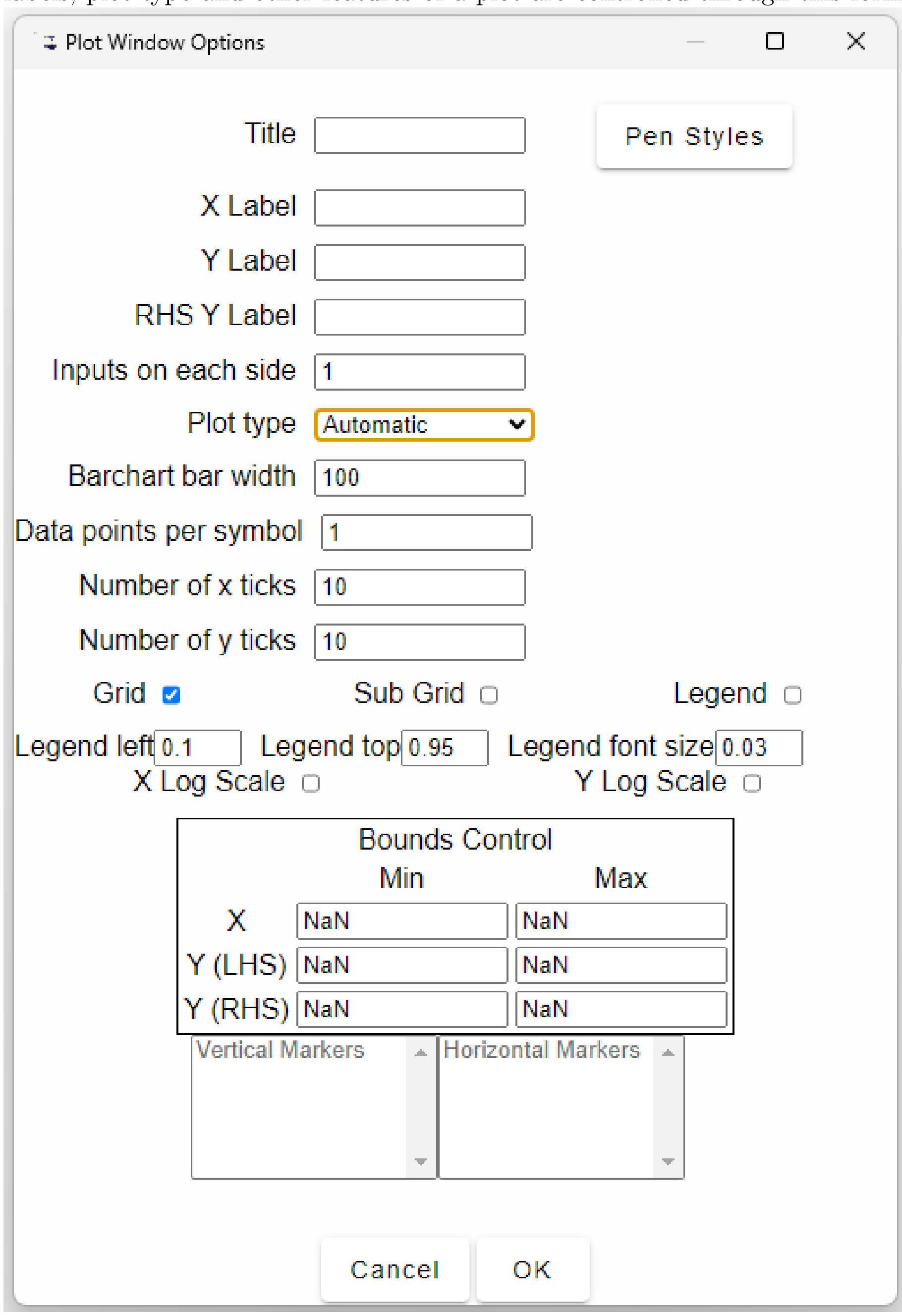
This generates a large display window for the plot alone, independent of the *Ravel/Minsky* canvas itself.

4.6.1.3 Make Group Plot

If this option is checked on a plot within a Group, this plot replaces the default image for a Group. It is a dynamic rendition, so in a *Minsky* model, this can show the plotted variables changing over time.

4.6.1.4 Options

The format of a plot is controlled by the Plot Window Options form. Titles, axis labels, plot type and other features of a plot are controlled through this form.



4.6.1.5 Set options as default

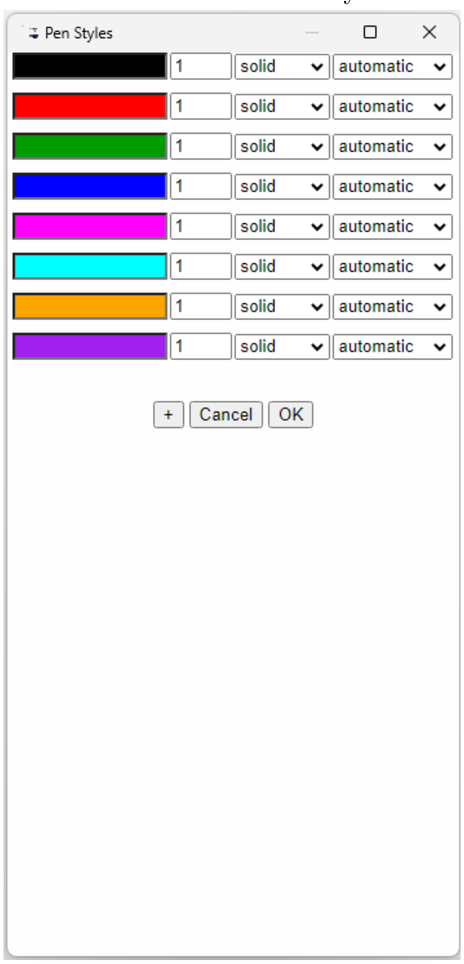
This command reproduces the settings of the selected plot in any subsequent plot. This can expedite plot labelling, as well as setting a default appearance for the style of graph applied to the data in each plot.

4.6.1.6 Apply default options

This applies the default plot options to any selected plot.

4.6.1.7 Pen Styles

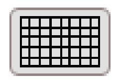
This brings up the pen styles control form. When variables are attached to the input ports, their names will appear above each formatting bar. This menu can also be accessed from the Pen Styles button on the Options form.



4.6.1.8 Export as CSV

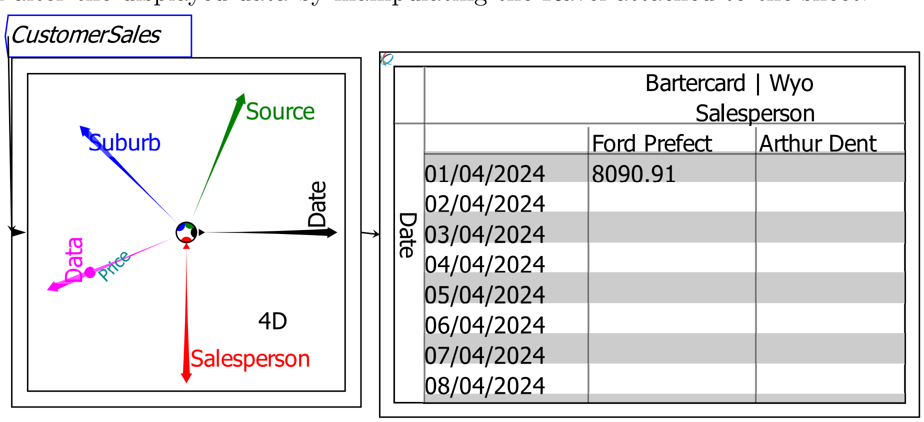
This command exports the data from a Plot as a csv file.

4.7 Sheet Widget



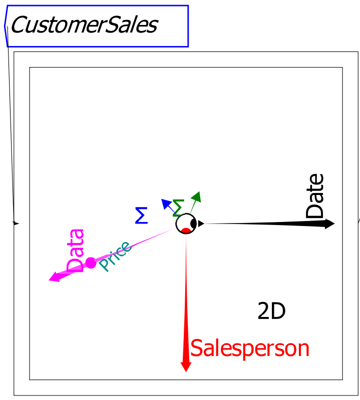
The Sheet widget displays data as in a typical spreadsheet program.

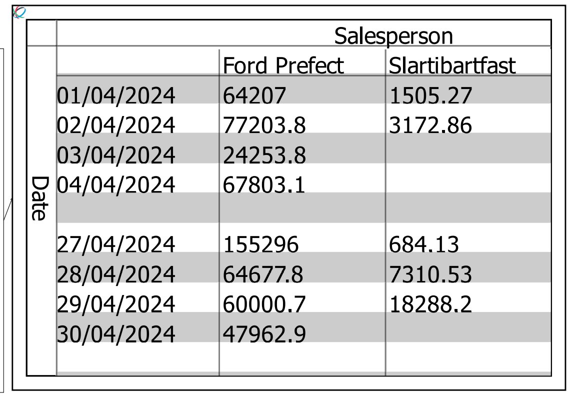
The default sheet is a blank square box, as shown above. Once data is attached to the sheet, formatted and hierarchical data will fill the sheet. The example shown below has 4 dimensions—the data is Price, while the dimensions are Date, Salesperson, Suburb and Source. Data for the first two dimensions—the downward and right-pointing Ravel axes—is shown, while the first entry in the third and higher order dimensions are shown at the head of the sheet. You can alter the displayed data by manipulating the Ravel attached to the sheet.



To use the Sheet widget, simply attach a variable or Ravel to it via a wire. The input is on the left-hand side of the sheet widget box. This sheet displays the input data as a number (in the case of a variable being simulated in a *Minsky* model) or an array of data (in the case of a Ravel). Note that only one wire can be connected to a sheet, as the sheet can only display a single input value– however, the *Ravel* operators *meld* and *merge* enable you to combine data from several Ravels to allow them to be displayed in a single sheet.

By default, a Sheet outputs the beginning of the data series. The context menu allows you to specify that the data displayed comes from the Head of the data (the default), its Tail (the final entries in the data series), or the Head and Tail. The next figure shows the Head and Tail displayed for the rows of this sheet.





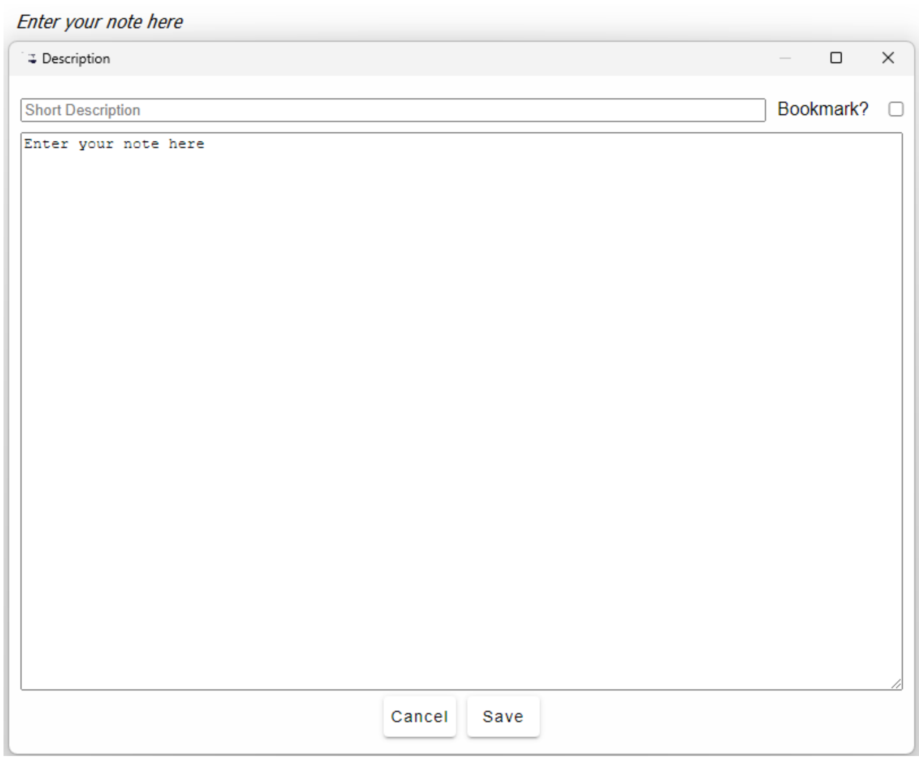
	Salesperson	
Date	Ford Prefect	Slartibartfast
01/04/2024	64207	1505.27
02/04/2024	77203.8	3172.86
03/04/2024	24253.8	
04/04/2024	67803.1	
27/04/2024	155296	684.13
28/04/2024	64677.8	7310.53
29/04/2024	60000.7	18288.2
30/04/2024	47962.9	

4.8 Note Widget

The note widget places the text line “Enter your note here” on the canvas.

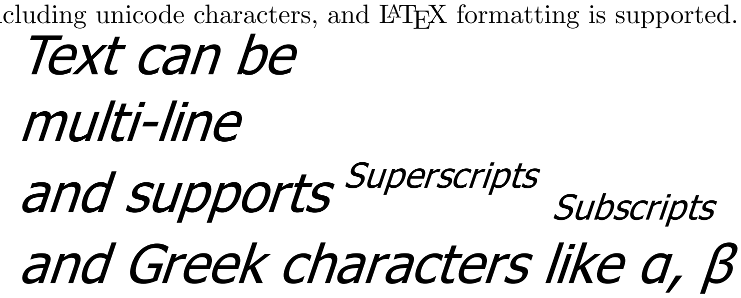
Enter your note here

Double-clicking on the text, or choosing Description from the context menu, brings up the note form:



The Short Description field becomes a tool-tip for this note, and also doubles as a Bookmark should you check the Bookmark? box.

Notes allow arbitrary text to be placed on the canvas for explanatory purposes. Anything that can be entered on the keyboard can be placed here, including unicode characters, and LaTeX formatting is supported.



4.9 Context Menu

All canvas items have a context menu, accessed by clicking the right-mouse button while hovering over an item, which allow a variety of operations to be applied to the canvas item. Common context menu items are explained here:

Help bring up context specific help for the item

Description Attach an annotation to the item. This is only visible by selecting the description item from the context menu, although whatever is set as the "Short Description" will also appear as a tooltip whenever the mouse hovers over the item.

Port values When running a simulation, you can drill down into the actual values at the input and output ports of the variable or operation, which is a useful aid for debugging models.

Edit set or query various attributes of an item. This function can also be accessed by double clicking on the item. (Plot widgets behave slightly differently: double-clicking on a plot brings up an independent window in which the plot is displayed.).

Copy Creates a copy of an item, retaining the same attributes of the original. This is very useful for creating copies of the same variable to reduce the amount of overlapping wiring in a model. This can substantially improve the readability of a *Ravel/Minsky* model when compared to standard system-dynamics programs.

Flip actually rotates an object through 180°. You can specify aribtrary rotations of objects through the edit menu.

Delete delete the object.

Item specific context menu items:

variables, parameters and constants

- **Local** Make the variable's scope local to its group
- **Find definition** Place a red circle on the variable that defines its value.
- **Select all instances** Select all instances of this variable
- **Rename all instance** Do a global search and replace of this variable name with a new name.
- **Export as CSV** Export the current variable's value as a CSV file.
- **Add integral** attach an integration operation, and convert the variable into an integral type.

integrals

- **Copy Var** copy just the integrated variable, but not the integration operation itself.
- **Toggle Var Binding** Normally, integrals are tightly bound to their variables. By toggling the binding, the integral icon can then be moved independently of the variable it is bound to.

Godley tables

Open Godley Table opens a spreadsheet-like window to allow financial flows to be defined in a Godley Table (these can also be defined directly on the canvas when a Godley Table is displayed in Edit Mode).

Resize Godley Table allows the icon to be resized.

Edit/Copy var allows individual stock and flow variables to be copied or edited.

Export to file export table contents as either CSV data, or as a LaTeX table, for importing into other software, such as a spreadsheet or word processor.

Groups

Zoom to Display Zoom the canvas sufficiently to see the contents of the group.

Resize Resize the group icon on the canvas.

Save group as Save the group in it's own Minsky file.

Flip contents Rotate each item within the group by 180°

Ungroup Ungroup the group, leaving its contents as icons on the canvas.

contentBounds Draws a box on the canvas indicating the smallest bounding box containing the group items.

Plot Widgets

Expand By double-clicking, or selecting "Expand" from the context menu, a popup window is created of the plot, which can be used examine the plotting in more detail.

Resize Allows you to resize the plot icon on the canvas

Options Customize the plot by adding a title, axes labels and control the number of axis ticks and grid lines on the detailed plot. You can also add a legend, which is populated from the names of variables attached to the plot.

4.10 Canvas background and keyboard shortcuts

The canvas is not simply an inert place for the canvas items to exist. There is also a background context menu, giving access to the edit menu functionality such as cut/copy/paste, and also keyboard entry.

	F1	context sensitive help
Special keys:	Shift	enter panning mode
	$\leftarrow, \rightarrow, \uparrow, \downarrow$	adjust sliders, adjust Ravel slicers

The following keystrokes insert an operation

`+`	add
`-`	subtract
`*`	multiply
/	divide
`^`	pow
`%`	percent operator
`&`	integral
`=`	Godley table
`@`	plot
`#`	start a text comment, finish with return

Typing any other character, followed by the Return key, will insert an operation (if the name matches), or otherwise a variable with that name.

4.11 Bookmarks

Bookmarks are a useful feature for saving the current position and zoom of the canvas, to be able to come back to that part of the canvas later. This helps to manage more complicated models. To create a new bookmark, click on the "Bookmark" tab in the top left-hand corner (in-between "Edit" and "Insert") and select "Bookmark this position". The program will provide a dialogue box to enter in a name for the new bookmark. After creating the bookmark, all user-created bookmarks can be seen in the Bookmarks menu. To delete a bookmark, simply select "Delete" from the Bookmarks menu and select the desired bookmark. To open an existing bookmark, select one from the menu.

You may also create a bookmark from any item on the canvas by selecting the bookmark checkbox in the "description" dialog. This is especially useful with text boxes, since they provide a visual indicator for what is bookmarked at a given location—Parameter definitions, Plots, etc.

4.12 Dimensional Analysis

Dimensional analysis is the idea of attaching units of measurement (eg metre or second) to the quantities being computed. It provides an additional constraint that the system must satisfy, reducing the chance of wiring errors. Two different units being added together will throw up an error — you cannot add 2 metres to 3 kilograms. But it should be possible add 2 metres to 3 feet, and get the correct answer. You may need to explicitly add a multiply operation to convert from one unit to another, for example, dividing the 3 feet by 3.281 before adding it to the 2 metres, providing a total of 2.914 meters.

Using Dimensional Analysis in *Minsky*

To attach units to quantities in *Minsky*, you use the units field of the variable/parameters/constants edit dialog box. Each word typed in this box describes a separate unit. `^` followed by an integer is used to represent a power. Finally, a single "/" indicates that the following units are on the denominator, dividing the first set of units by the second. So to represent the unit of acceleration, you can equivalently type all of the following:

- `m/s^2`
- `m/s s`
- `m/s^-2`

Or spelling it out in full:

- `metre/second^2`
- `metre second^-2`
- `metre / second second`

Note that metre and m are distinctly different units in *Minsky*.

Note – setting the time dimension is done in the simulation menu.

All function objects require dimensionless inputs. You can use dimensional analysis to avoid, for example, incorrectly feeding a degree measurement into a sine function, by requiring them to be multiplied by a radiansPerDegree parameter.

4.13 Tensor values

Tensors are arrays of data, where the rank of the tensor tells how many dimensions the data has. A single number (a scalar) has a tensor of rank 0. A tensor of rank 1 is a column or row of numbers—a vector. A rank 2 tensor appears is a 2D sequence of numbers (a matrix), for example:

$$\begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}$$

A tensor of rank 3 will appear as a three-dimensional cube, rank 4 as a four-dimensional hypercube, and so on. Tensor variables can be created and used in *Minsky*, but power analysis of multidimensional data requires the installation of *Ravel*.

4.13.1 x-vectors

When two or more tensors are combined with a binary operation (such as addition or multiplication), they must have the same rank. For example, two tensors of rank 2 can be multiplied together, but a tensor of rank 2 and a tensor of rank 3 cannot. They may have differing dimensions, which means the values within each tensor may not necessarily match up 1-to-1 exactly. To understand what happens when a given dimension is mismatched requires understanding the concept of an x-vector.

When Minsky is given tensor values, it sorts the values within each tensor by corresponding dimensions. For example, a rank 2 tensor would have its values sorted into two sets of data. This data can be in the form of numbers, dates (time values), or strings. Minsky will then look at cross-sections of the datasets in order to process the values within. When the dimensions of two tensors match up, for example two rank 2 tensors, the corresponding cross-sections of both tensors should also match up. When they don't, a weighted interpolation of the corresponding values is taken. This involves using an x-vector.

An x-vector is a vector of real values, strings or date/time values. If no x-vector is explicitly provided, then implicitly it consists of the the values $(0, \ldots, n_i - 1)$, where n_i is the dimension size of axis i of the tensor.

For example, if the first tensor consists of three elements (x_0, x_1, x_2) and the second consist of a number of different elements that roughly correspond to the same three elements, these can be added together. The x-vector starts with the first tensor's value of (x_0) and looks for a matching value in the second tensor. If it can't find a direct match, it will search for nearby values which roughly correspond. It can then take those values and interpolate the corresponding value based on where in the tensor it appears. This is weighted, so say there are four values nearby, the program will average those out and find where a value in the middle of those four values would appear, and what that hypothetical value would be. To take another example:

Suppose the first tensor was a vector (x_0, x_1) and had an x-vector (1,3) and the second tensor (y_0, y_1, y_2) had an x-vector (0,2,3), then the resulting tensor will be $(x_0 + 0.5(y_0 + y_1), x_1 + y_2)$. If the x-vector were date/time data, then the tensor values will be interpolated according to the actual time values. If the first tensor's x-vector value lies outside the second tensor's x-vector, then it doesn't result in a value being included in the output. The resultant x-vector's range of values is the intersection of input tensors' x-vector ranges.

If both tensor had string x-vectors, then the resultant tensor will only have values where both input tensors have the same string value in their x-vectors. In the above case, where the x-vectors were ('1','3') and ('0','2','3') the resulting tensor will be the scalar $x_1 + y_2$.

It goes without saying that the type of the x-vector for each axis must also match.

4.14 Ravel

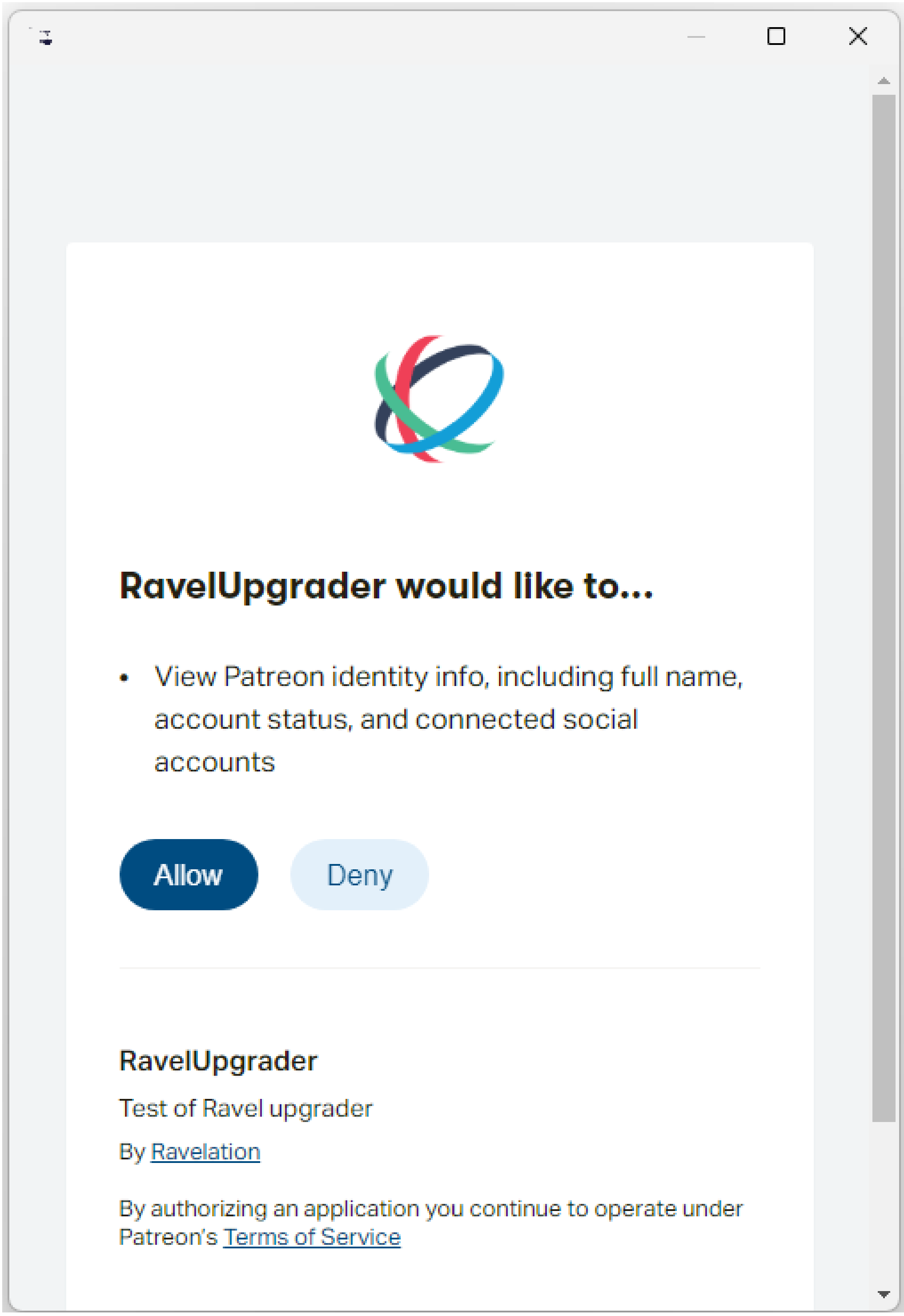
Ravel the program is a commercial data analysis extension to the Open Source system dynamics program *Minsky*. It inherits all of *Minsky*'s features and adds the Ravel™, a graphical object for the manipulation and analysis of multidimensional data. If you are using *Minsky* and would like to check out *Ravel*, go to *Ravel*'s Patreon page[1] and sign up. The monthly fee is US$7, and there is a 7-day trial period during which you can experiment with using *Ravel* before the fee is applied.

[1] https://www.patreon.com/hpcoder

4.14.1 Installing Ravel

Once you are a member of *Ravel*'s Patreon group, you have access to the Ravelation Patreon store, and you can download the latest copy of *Minsky*—either from the store or from attachments to posts announcing updated versions.

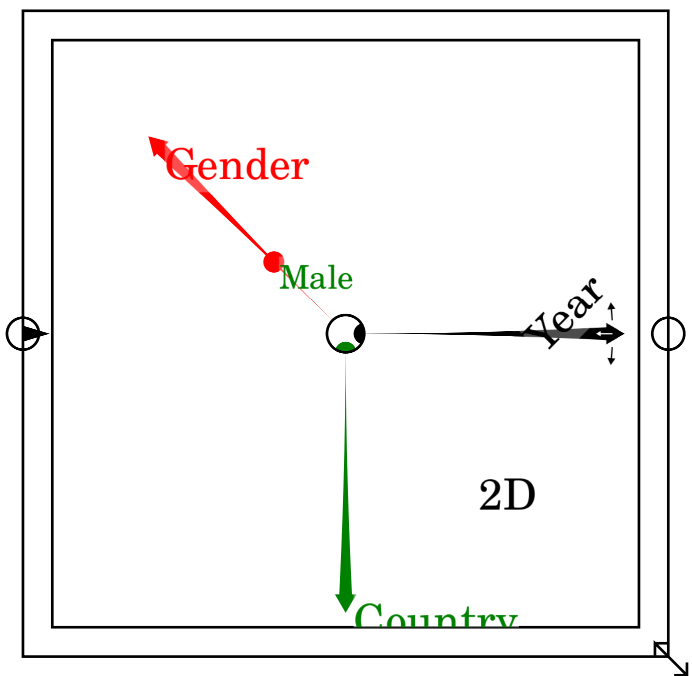
Download and install *Minsky*, which will launch the program once it is installed. Then choose “Upgrade” from the File menu. The RavelUpgrader form will pop up, asking you for confirmation that you are willing to let the process check your name and account details on Patreon.


RavelUpgrader would like to...
View Patreon identity info, including full name, account status, and connected social accounts
Allow
Deny
RavelUpgrader
Test of Ravel upgrader
By Ravelation
By authorizing an application you continue to operate under Patreon's Terms of Service

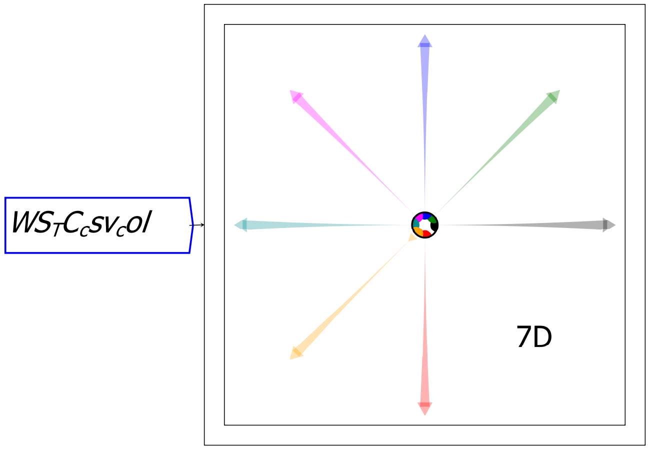
Click on "Allow" and *Minsky* will check that you are a member, and if so then download and install the *Ravel* extension into *Minsky*. Once this is completed, the latest version of *Ravel* will be installed in your *Minsky* subdirectory, and you will have full access to its commands.

4.14.2 Introduction to the Ravel GUI Object

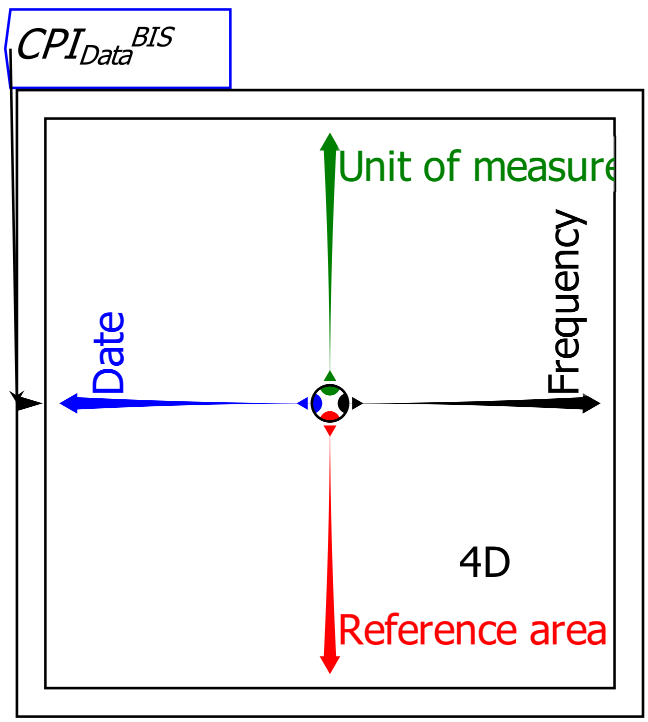
A Ravel is inserted onto the wiring canvas by clicking on the widget from the toolbar. The first Ravel inserted into a document appears in Edit mode as shown below, with indicative Axis names:



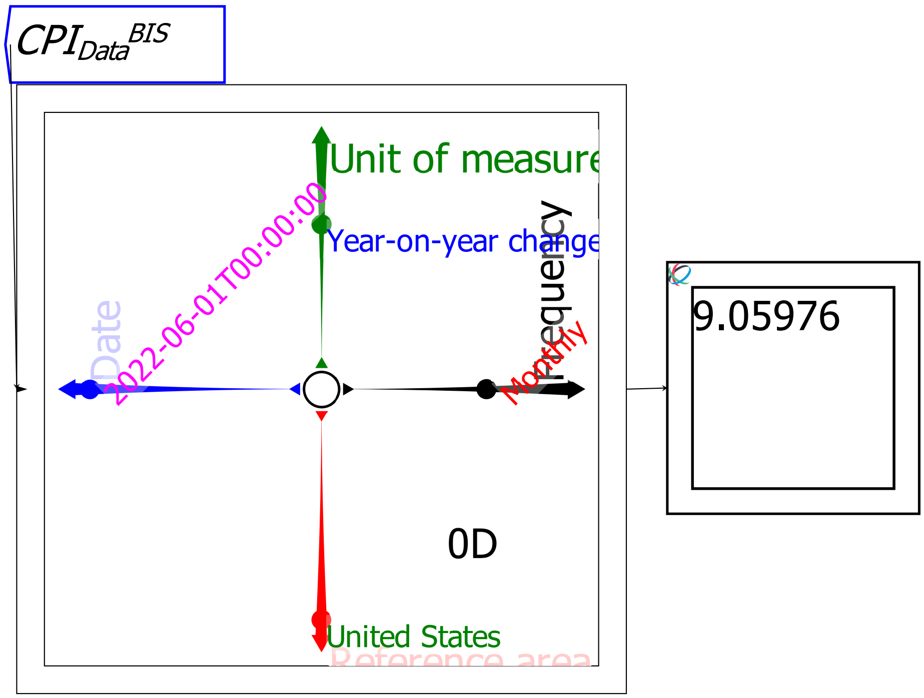
The essential step in using a Ravel is to attach data to it via the triangular input port. See importing CSV files for how to import data. When a data file is attached to the input port of a Ravel, a multi-arrowed object fills the Ravel rectangle. If there are seven dimensions or more to the data, it will appear as shown below, with arrows in a pastel shade and axes labels not shown unless your mouse is hovering over an axis:



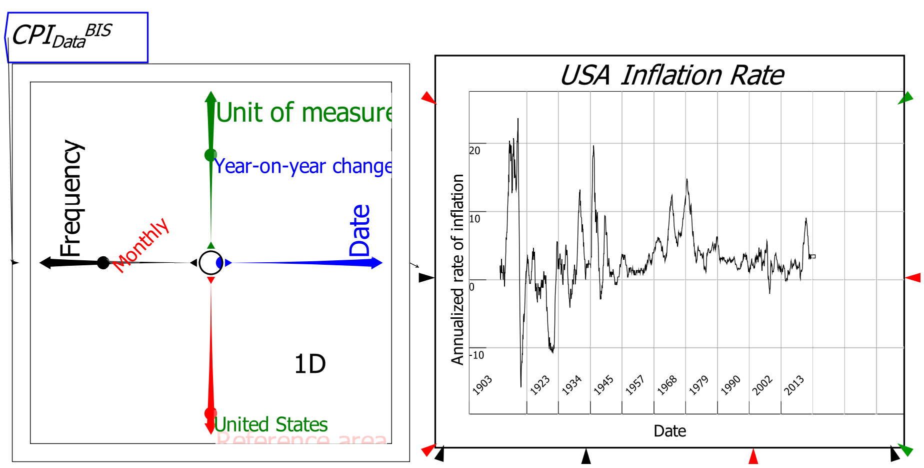
With six or fewer dimensions it will appear as shown below, with full-colour axes and axis labels shown:



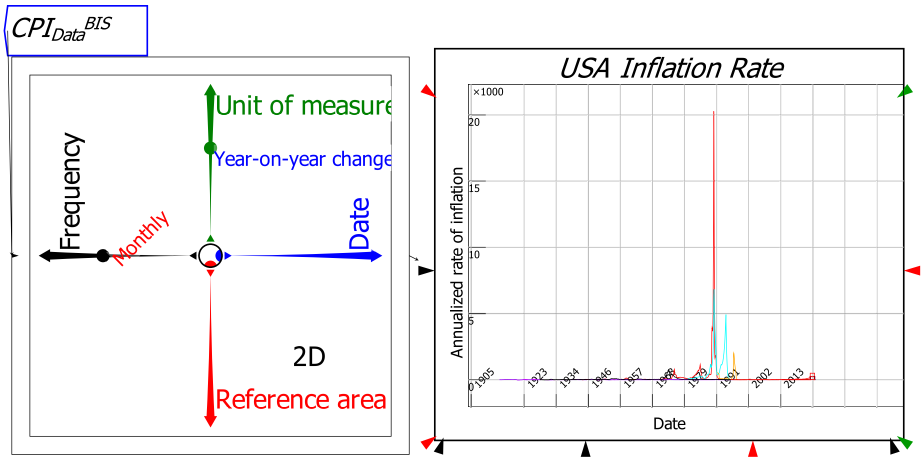
The arrows are the dimensions of the data; the dots are selectors for specific values on each dimension. When a CSV import parameter is first attached to a Ravel, all the selector dots are in the centre of the Ravel, so that all the data in the Ravel is output to any Variable, Sheet or Plot attached to it. The next figure shows all 4 selectors being used so that the Ravel outputs the annualized rate of inflation for the USA in June 2022.



If the selector dot remains in the centre for one dimension, then a column of data for that dimension is output from the Ravel:



With two selector dots remaining in the centre, an array of data is output. This results in multiple countries' inflation rates being plotted:

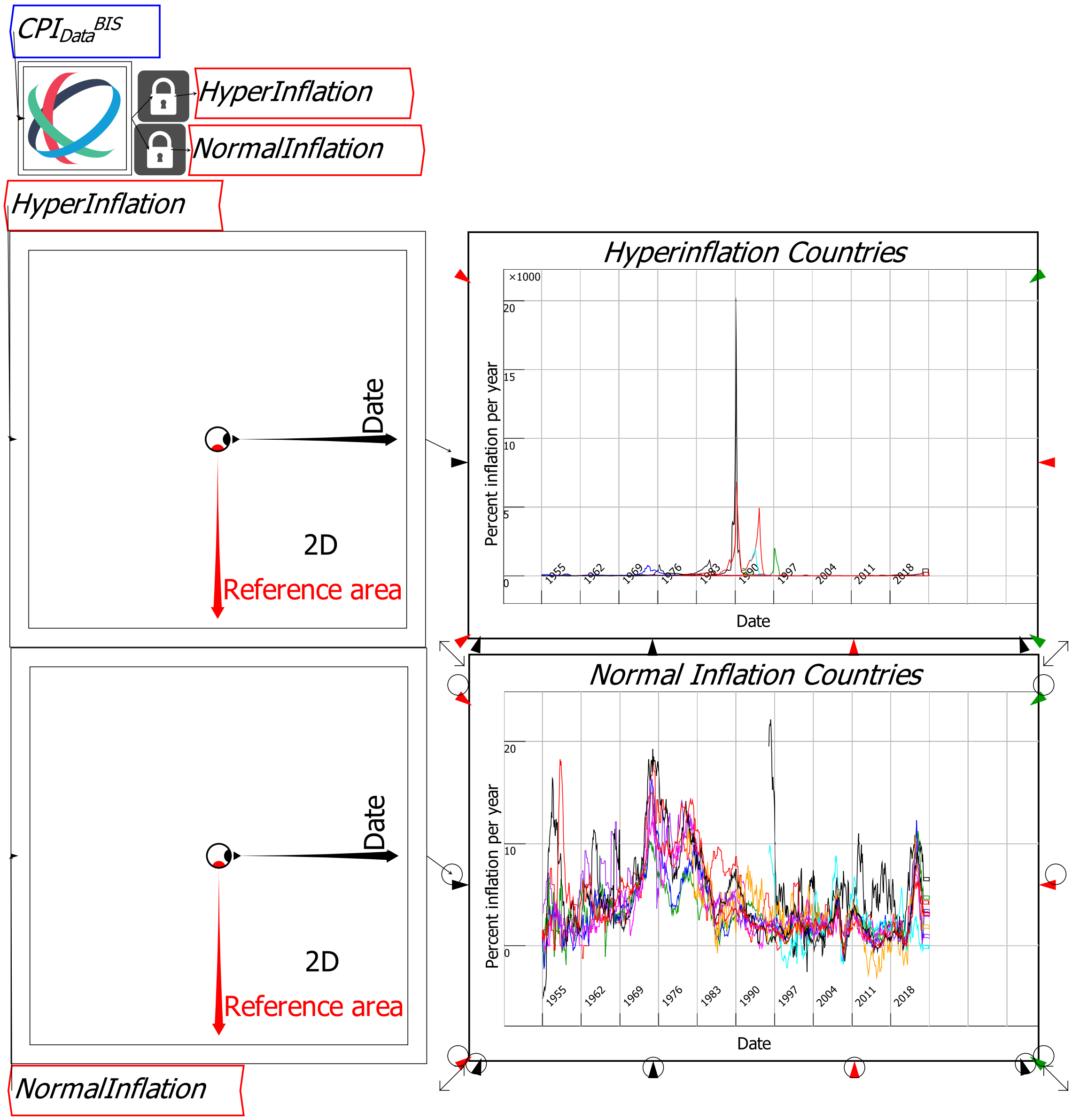


Clearly a number of countries had extreme rates of inflation at various times between the 1970s and the mid-1990s. There are Ravel functions which can be used to isolate these countries (see *Operations*), but it is also easy to identify them visually by moving the selector dot along the Reference Area axis. You can do this by either:

- Clicking on the selector dot with the mouse and dragging it along the axis; or

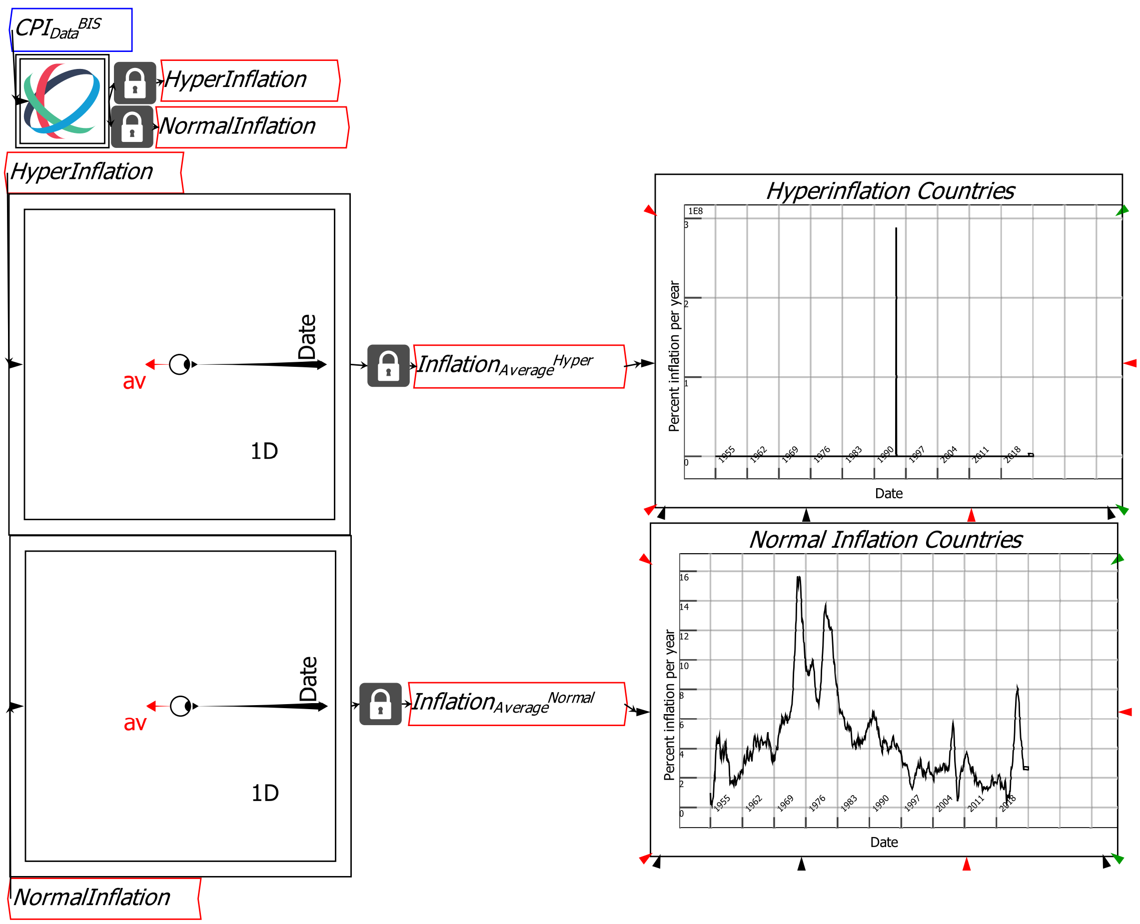
- Hovering the mouse over the axis and using the arrow keys. The up-arrow and right-arrow move you forward along a data axis from the first entry to the last; the down and left arrow keys move you backward.

Eyeballing the data indicates that a 40% annual inflation rate is the dividing line between countries which did and did not experience a hyperinflation in the post-WWII period (defined as from 1955 onwards to exclude WWII-influenced price disturbances—for instance, those associated with the Russian occupation of Austria ending in 1955). That set of countries (Argentina, Brazil, Bulgaria, Chile, Colombia, Croatia, Czechia, Estonia, Iceland, Indonesia, Israel, Latvia, Lithuania, Mexico, North Macedonia, Peru, Philippines, Poland, Romania, Serbia, Slovakia, Slovenia and Turkey) are identified in the next figure as "HyperInflation" countries, while the remainder are "NormalInflation" countries. Except for Iceland, Israel, the Philippines, and Turkey, the hyperinflation countries were either South American, or countries which underwent the transition from socialism to capitalism in the 1990s. This figure also shows the iconized version of a Ravel, which is generated by toggling "Editor mode" on the context menu (also, by default, the first Ravel inserted into a document is in Editor mode, and all subsequent Ravels are inserted in iconized mode).

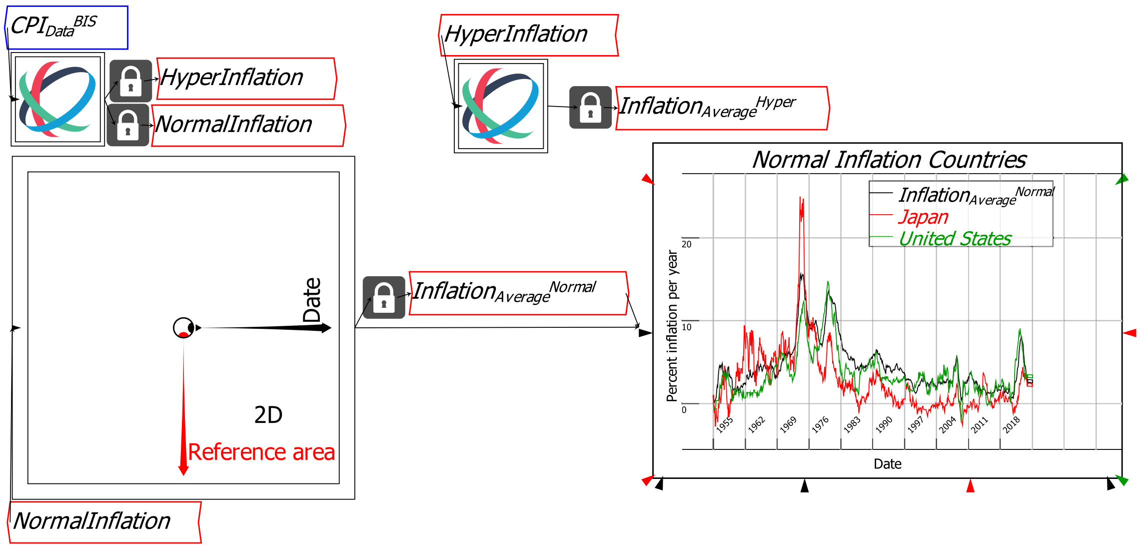


This selection employed one context-menu feature of a Ravel—"Pick Axis Slices" was used on the Reference area axis to choose countries which had annual inflation rate of above 40% between 1955 and 2024—and one Ravel-specific widget, the Lock, which creates a subset of the data which can be allocated to a variable. The selection on the Ravel can then be changed without altering the subset selected by the Lock.

Now that the data is sensibly divided, it is feasible to calculate average inflation rates for normal countries. This is done by choosing "Av" on the context menu command "Set next aggregation", and then dragging the Reference Area axis in to the centre of the Ravel.



It is now possible to compare the inflation of individual countries to the average for all normal inflation countries. In the next figure, the USA and Japan were selected using Pick axis slices, and their data is graphed against the average. This shows that Japan's inflation rate exceeded America's—and the average—until 1975, after which Japan's inflation rate has been consistently lower than that of the USA and the average.



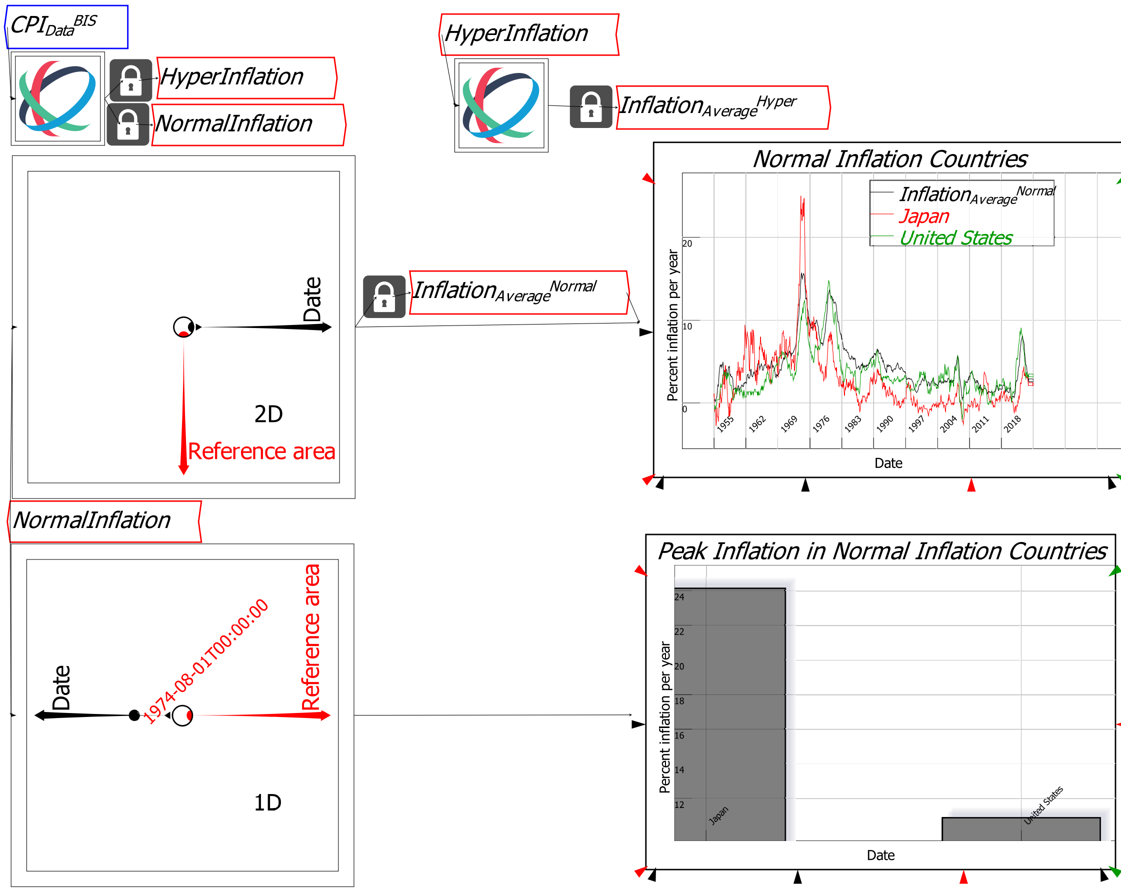
4.14.3 Ravel-specific capabilities

Ravels have a range of features controlled by the Ravel itself, the axes of the Ravel, and context menu commands relevant to the Ravel and its axes.

4.14.3.1 Axis rotation

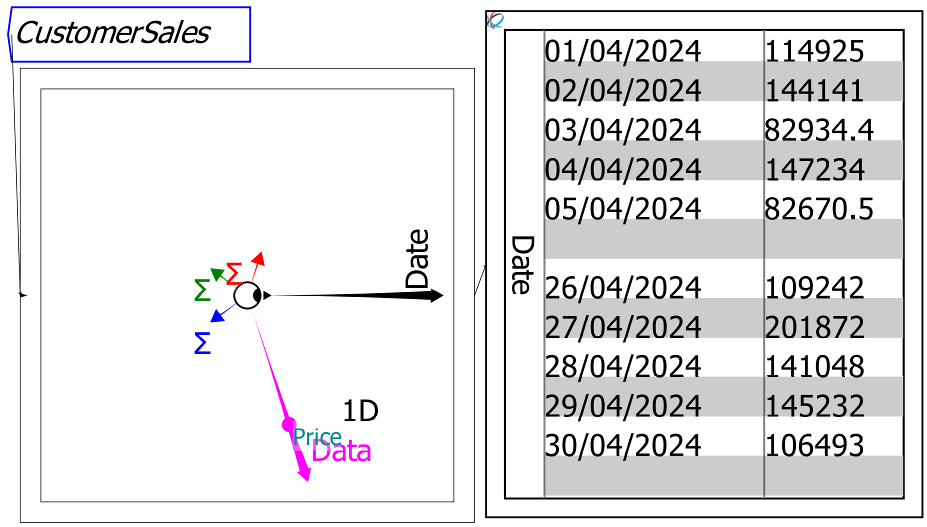
A Ravel outputs the data in the right-pointing axis as the series for graphing in a plot, and as the rows of a sheet. In the previous figures, Date was the right-pointing axis, which generated a set of time series for plotting (the data shown on the Y1 axis). In the next figure, Reference Area is rotated into the right-pointing axis position in the second plot, which means that Date data now generates the x-axis information. The y-axis information is therefore the rate of inflation on a specified date. The second plot in the figure below shows the annual inflation rates for Japan and the USA at the highest point for the inflation data since WWII. Since this date follows the Oil Embargo[2] imposed by OPEC members after the Yom Kippur War[3], this implies that the main cause of the inflationary surge was the four-fold increase in oil prices (from US$2.50 a barrel to $10) caused by the embargo.

[2]https://en.wikipedia.org/wiki/1973_oil_crisis
[3]https://en.wikipedia.org/wiki/Yom_Kippur_War



4.14.3.2 Axis aggregation

Axes can be collapsed as well as rotated, in which case the data on the axis is aggregated according to the choice made on the "Set next aggregation" Ravel context menu. In the next figure, Price is chosen on the Data axis, and all other axes apart from Date are aggregated by the Sum function (the default aggregation method). The Ravel therefore outputs total sales by date.



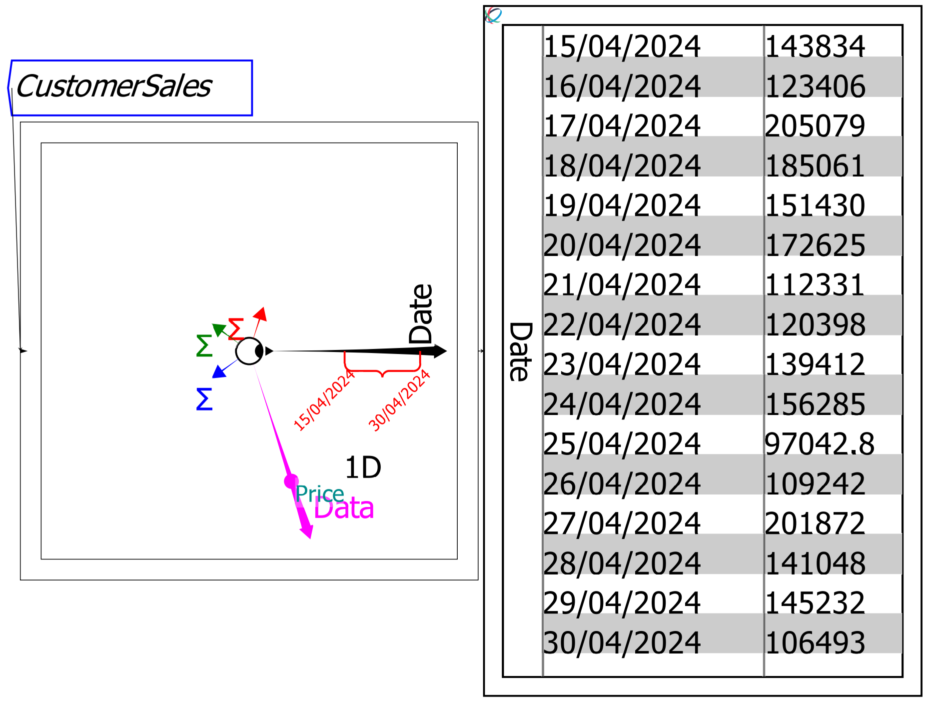
The available aggregations are: Sum, Product, Average, Minimum, Maximum, and Standard Deviation. The default setting is Sum. The other options can be set via the Context (right-mouse click) menu for Ravels.

4.14.4 Context menu for Ravels

Command	Effect
Help	Bring up context help on Ravels
Description	Open a text form for documenting the Ravel
Editor Mode	Toggle between seeing the Ravel in Edit or Icon mode
Export as CSV	Export the data in the Ravel as a CSV file
Set next aggregation	Determine the mathematical operation applied when the next axis is collapsed
Collapse all handles	Aggregate all axes. Ravel will return a single number
Expand all handles	Disaggregate all axes.
Flip	Turn the Ravel around so that the input is on the right and output on the left
Link specific handles	Control over linked Ravels to link only some aspects–orientation, caliper, etc.
Join Link Group	Add this Ravel to an existing set of linked Ravels
Unlink	Remove this Ravel from an existing set of linked Ravels
Toggle Axis Calipers	Turn the caliper on a selected axis on or off
Sort Axis	Sort axis by the value of axis labels.
Sort by value	Sort axis by the value of the data on an axis.
Pick axis slices	Manually choose which axis values are output by the Ravel
Axis properties	Description and Dimension show the dimension name; Set next aggregation
Rotate Ravel	Rotate the Ravel through an arbitrary angle
Add to Publication Tab	Place the Ravel on a specified Publication Tab
Delete Ravel	

4.14.4.1 Calipers

A caliper is a range selector for an axis. Its customary use will be to select a range of dates, but a caliper can be applied to any axis to select a subset of data which is contiguous. In the next figure, calipers have been applied to extract the last 15 days' of data from this Ravel.



When a caliper is initially created, it spans the entire range of a given dimension. It is applied to sections of that data by moving the vertical left or right side of the parenthesis; the entire parenthesis is moved by clicking and dragging on the peak of the parenthesis—the length of the selection is thus maintained while the segment selected is altered.

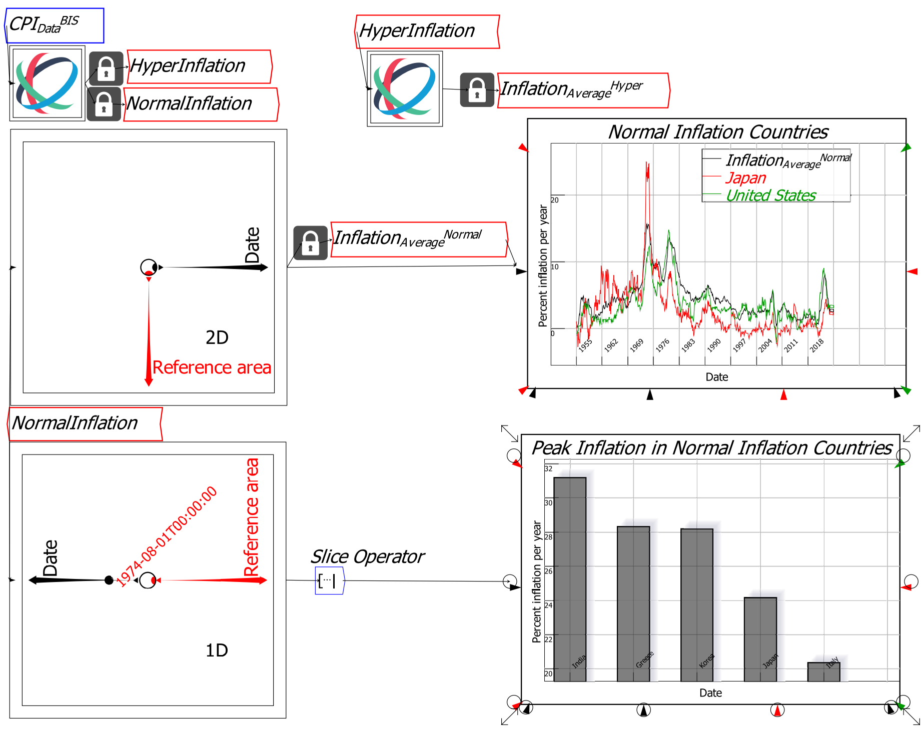
Ravel currently allows one caliper per axis. Future releases will enable multiple calipers, which will allow non-contiguous segments of data to be displayed on the one plot or sheet.

4.14.4.2 Sort axis

The default order of an axis is the same as the data input into it. Once loaded into a Ravel, the data can be sorted by the names of axis entries—for example, countries can be sorted alphabetically. This can be sorted forward (from A to Z) or in reverse.

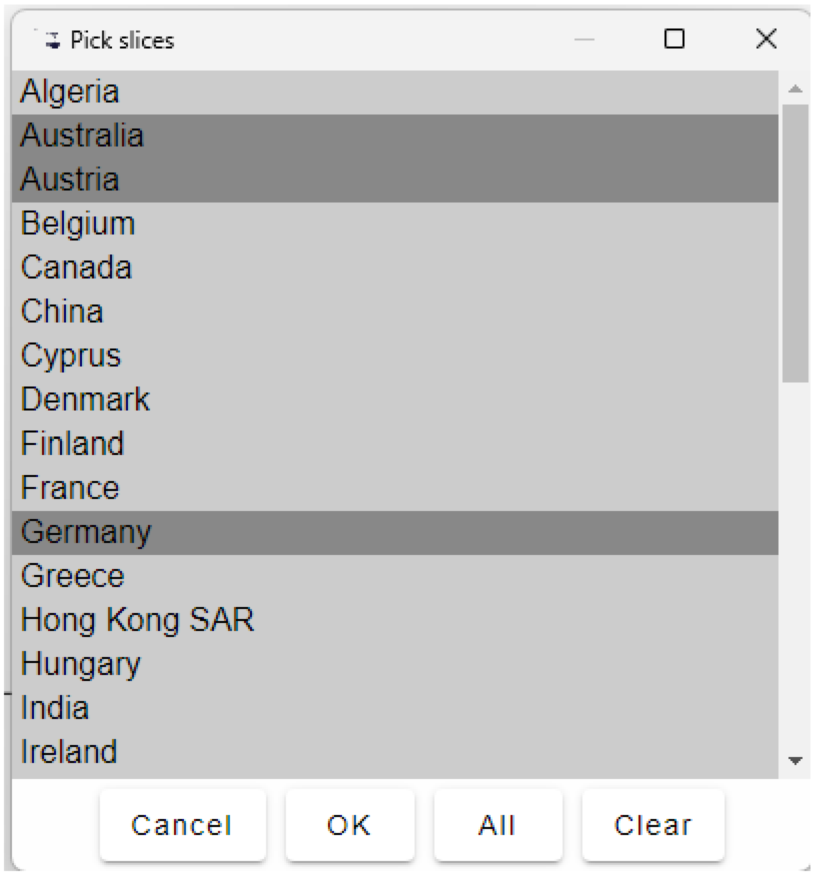
4.14.4.3 Sort by value

The order of an axis can also be determined by the data values for specific axis selections. The command is only available if the axis selections return a single column of data for ranking—for example, in the inflation data, a single date is chosen so that the value of the inflation for all Reference Areas (countries) can be compared. There are four options generated by the combination of dynamic or static sorting, and forward or backward direction. Static sorting sorts the single axis by the value of the data, and does not alter that data as (for example) different dates are chosen using the selector dot. Dynamic sorting alters the order so that the axis is resorted when (for example) a different date is selected. In the next figure, dynamic sorting is applied to the Reference Area axis for the rate of inflation. The output is also put through a Slice operator, so that only the top 5 countries are output.



4.14.4.4 Pick axis slices

Pick axis slices lets you select an arbitrary number of slices—as was done above to choose just the USA and Japan to compare their rates of inflation. Individual slices can be chosen by left-clicking on the slice; to select more than one slice, hold down the control key as you click.



4.14.4.5 Flip Ravel

This is a convenience command, which is convenient when it would be preferable to have the output from a Ravel be on its left side rather than its right–for example, when the Ravel is linked to the Y2 axis of a plot.

4.14.5 Lock

Locks can only be attached to the output of Ravels. The Lock widget is used to fix the output of a Ravel to a particular configuration of data. After the Lock is closed (either by double-clicking on the Lock, or choosing Lock from the context menu), alterations can be made to the source Ravel without altering the output of the Lock. This makes it possible to take multiple slices of the data from one Ravel, and to work with them by assigning those slices to variables, manipulating the subset of data via additional Ravels, etc. It has a number of useful context menu commands.

4.14.5.1 Apply Lock state to Ravel

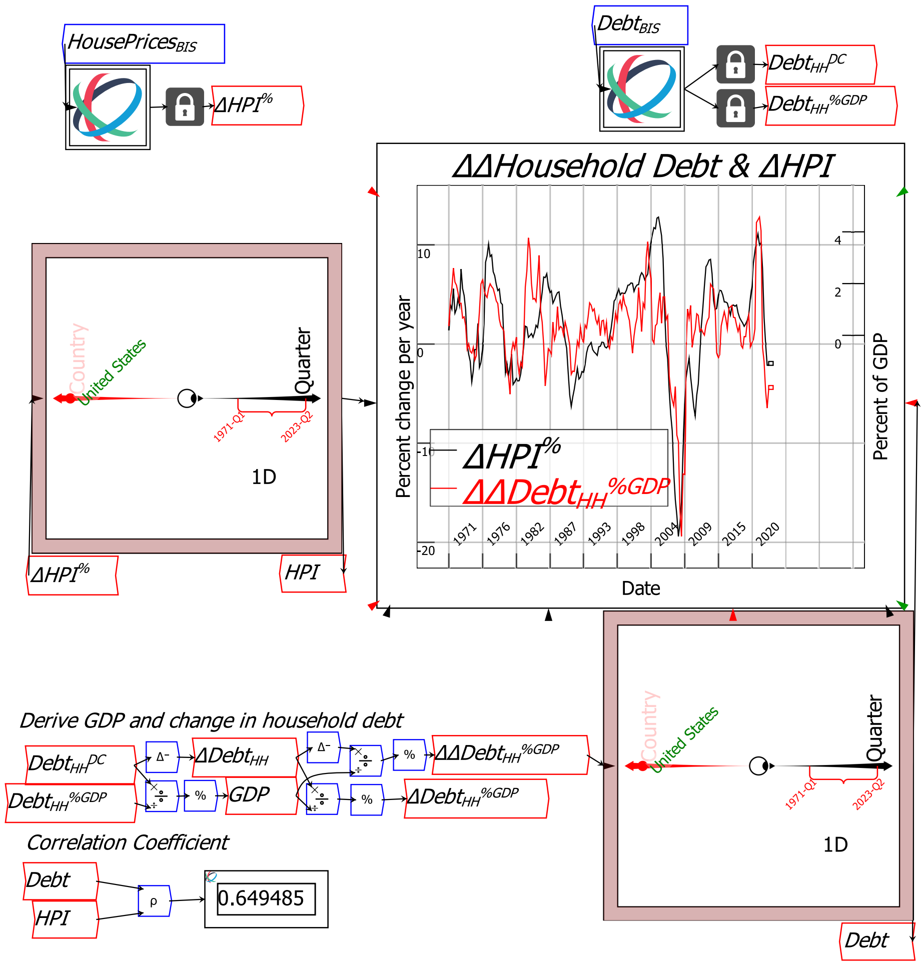
This menu item imposes the lock's state on the source Ravel. Since many locks can be applied to one Ravel, this makes it easy to edit the data as required to fine-tune individual locks. For example, the data for ${Inflation_{Average}}^{Normal}$ initially began in 1950. Since this included a very brief period of hyperinflation for Austria associated with Russia ceasing its posr-WWII occupation, it was advisable to alter the selection to start the caliper in 1955. This was easily done by applying the state of the lock to the Ravel, then editing the caliper, and opening and re-closing the lock.

4.14.6 Linking Ravels

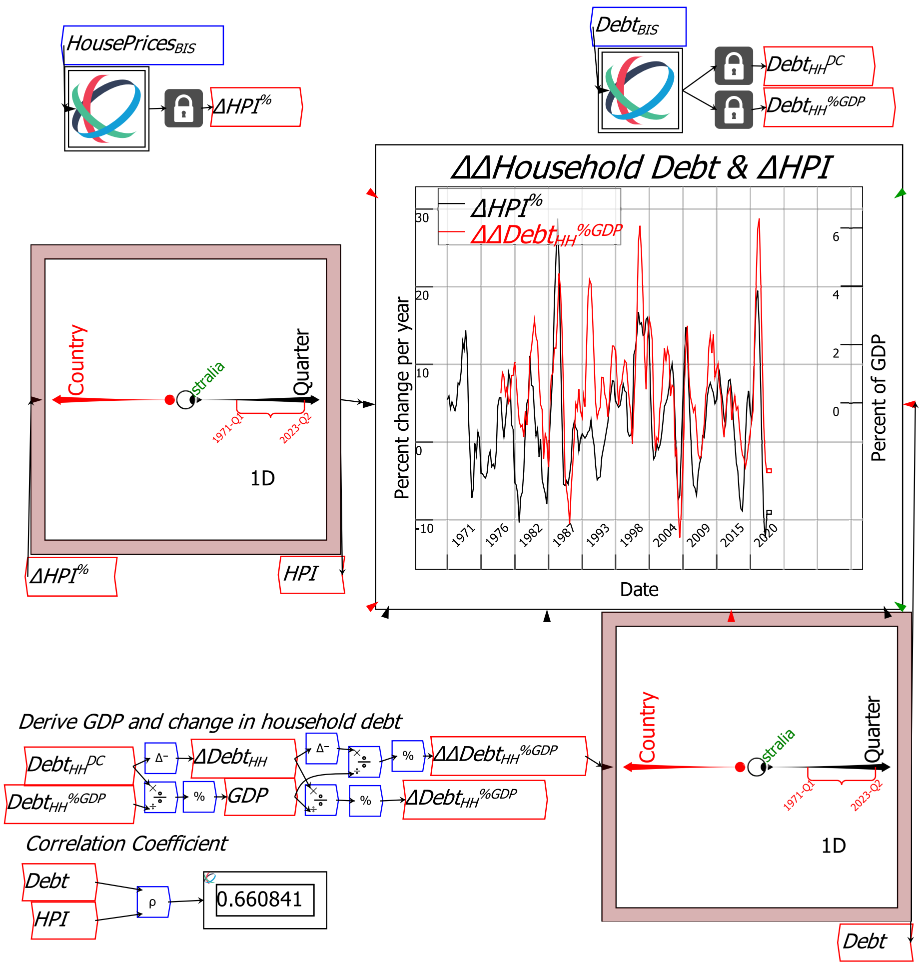
When Ravels share dimensions, it is possible to link them together so that what is done to one Ravel—plotting, selecting data, rotating axes, etc.—is done to the other Ravel. This is very useful when you wish to examine relations between different sets of data.

Ravels are linked by dragging the mouse to select the desired Ravels, then choosing "Link selected Ravels" from the context menu when the mouse is hovering over the canvas.

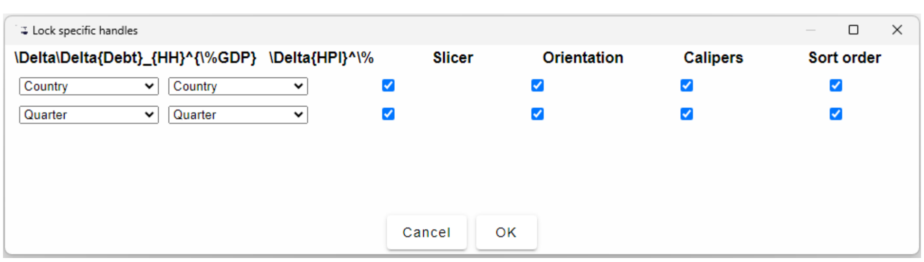
In the next figure, the two Ravels whose borders are highlighted in brown have been linked. When the country "United States" is chosen for the Ravel containing house price data, the same selection is made for the Ravel containing household debt data. The plot and the regression analysis therefore compare United States house price changes to the change in the change in household debt.



When the selector dot for Country is moved to another country, the graph and the correlation result for household debt and house prices for that country replaces the information shown for the USA.



Linking can be refined using the context menu for Ravels—rather than for the canvas. If your mouse is over one of the selected Ravels rather than the canvas, the context menu option is "Lock specific handles". That brings up the detailed lock menu:



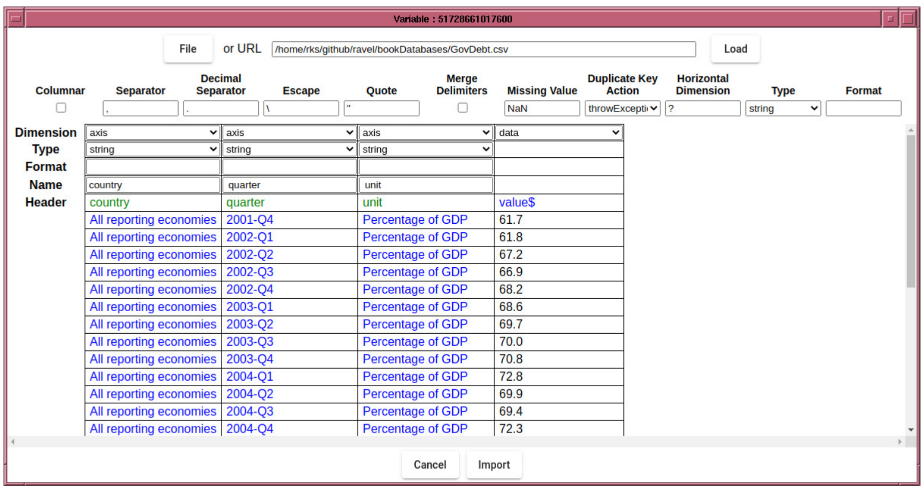
It is thus possible to turn on or off selecting of the same slices, arrow orientation, calipers and sort order if desired, while still linking the Ravels on other characteristics.

4.15 Importing data into a parameter from a CSV file

After creating a parameter from the "Variable" drop-down in the "Insert" menu, right-clicking the parameter and selecting the option to "Import CSV", will open a dialogue box that allows you to select a CSV file. Upon selecting the file, a dialog is opened, allowing you to specify assorted encoding parameters.

An alternative is to click on the ImportData icon , which will create a new parameter for you to import the data into.

The dialog looks somewhat like this (developments in the import routine may change some layout details):



4.15.1 Quick instructions

- Data is typically (but not always) stored in the rightmost columns of a CSV file. Shift click to set the top left cell of the data. Columns to the left will be marked as axes.

- Select "axis" in the Dimension dropdown to include a column as an axis. Column data turns blue.

- Select "ignore" in the Dimension dropdown to exclude a column. Column data turns red.

- Select "data" in the Dimension dropdown to treat a column as a data column. Column data turns black.

- Select Type for included axis columns. Select one of three types:

  string most general type, data treated as is.

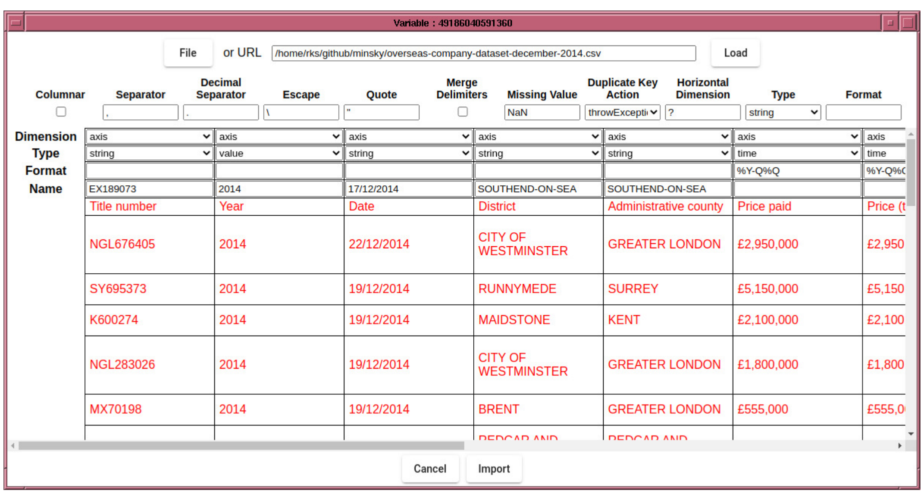
value value data must be numerical, e.g. 1, 2, and not quoted (as in "1", "2", etc.)

time data must refer to date-time data. The format field may be used to control interpretation of this data, though if this field is left blank, Ravel will interpret the data format itself. Blank format assumes data contains year/month/day/hour/minute/second separated by some non-numerical character. If fewer than 6 numerical fields present (from year to second), smallest units are set to minimum value (0 or 1 respectively).

- Click the OK button.
- Data is imported into the parameter.
- You may need to set units for the imported data field, which can be done via the Edit → Dimensions menu item.

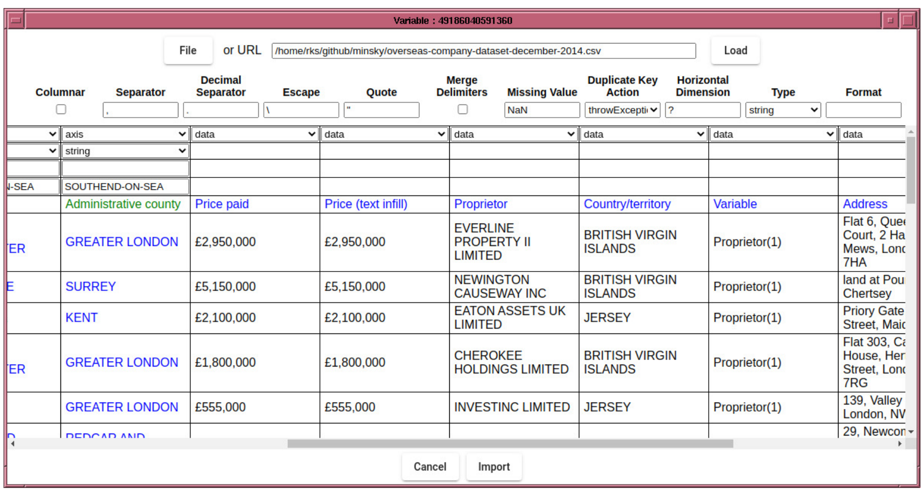
In the case shown above, the system has automatically guessed that the data is 3 dimensional, and that the first 3 columns give the axis labels for each dimension (shown in blue), and the 4th column contains the data. The first row has been automatically determined to be the first row of the file — with the dimension names are shown in green.

In this case, the automatic parsing system has worked things out correctly, but often times it needs help from the computer user. An example is as follows:

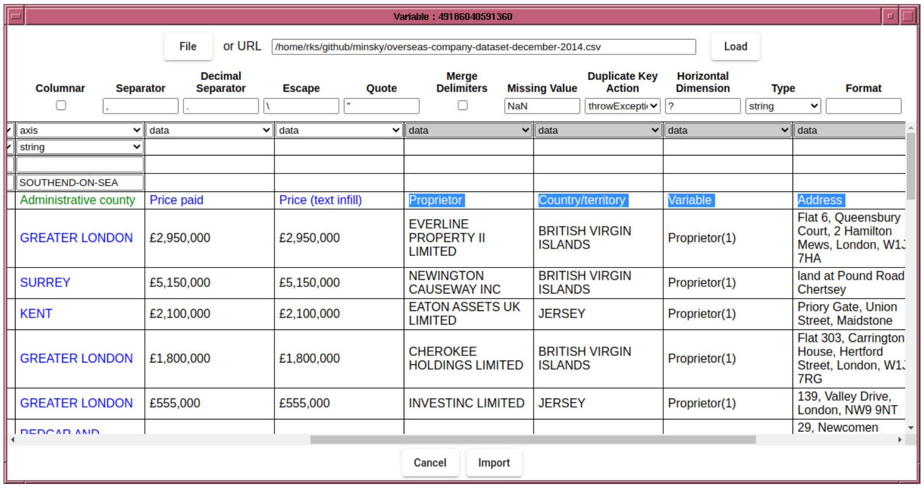


In this example, *Ravel* has failed to determine where the data starts, probably because of the columns to the right of the "Price" columns. So the first thing to do is tell it where the data is located. This is one function of the context menu for data importing. The options are:

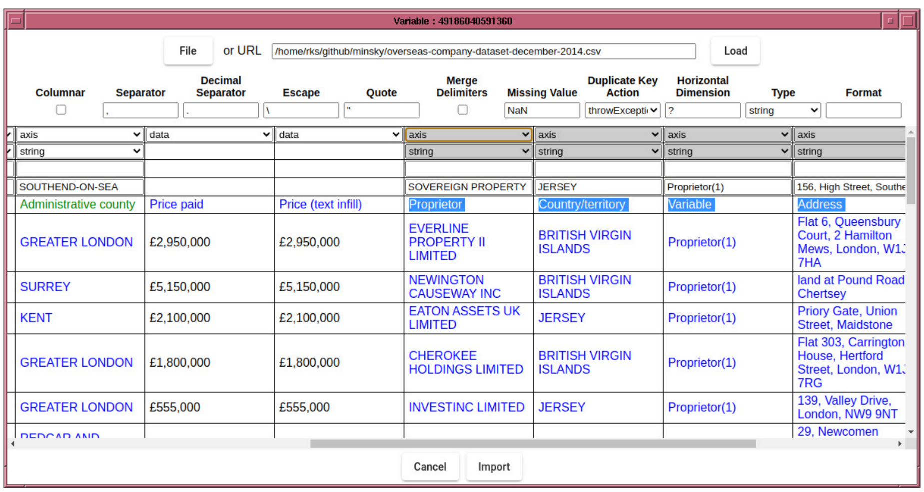
Command	Effect
Set as header row	Tell Ravel that the selected row contains the header information for the import
Auto-classify columns as axis/data	Invoke *Ravel*'s automated column classification routine
Populate column labels	Transfer the information from the header row to the Name row
Populate current column label	Transfer the information from the header to the Name for one column only
Set start of data row, and column	Tell *Ravel* where data begins in the file
Set start of data row	Tell *Ravel* where data begins in the file
Set start of data column	Tell *Ravel* where data begins in the file



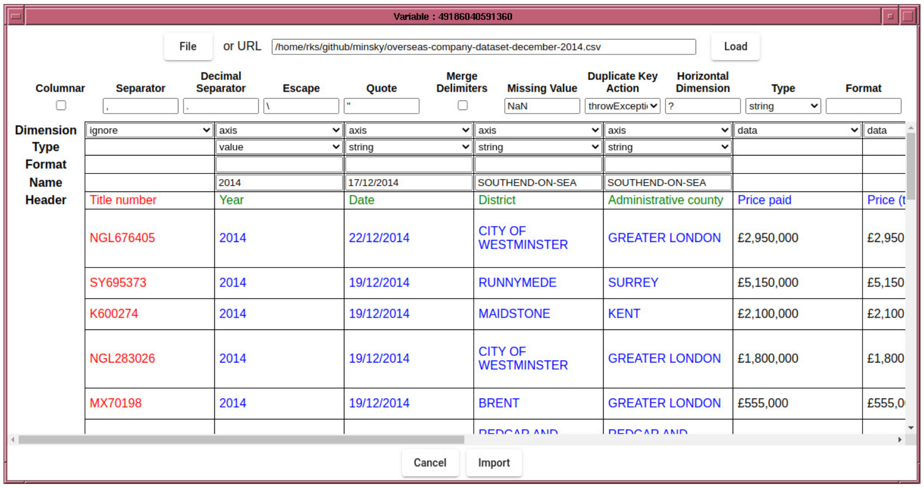
Note that this causes all columns to the right of "Price paid" to be treated as data, which is not right, since the columns to the right of "Price (text infill)" are text based columns, not data. So we need to mark those columns as either "axis" or "ignore". To do that, click and drag your mouse on the header field, which will cause those columns to be selected, like so:



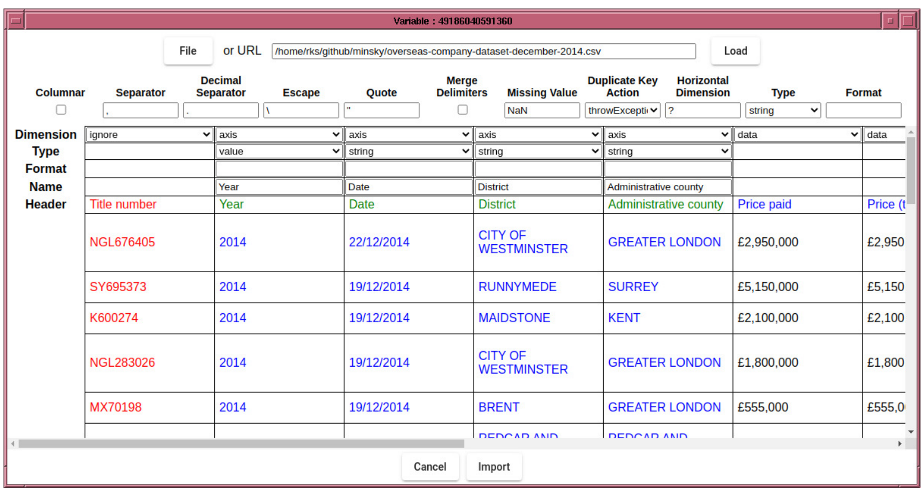
Then in the dimensions row, select "axis", which flips the selected columns:



Now the axes index labels are rendered in blue, the axes names in green and the data is in black. In this example, some axes have unique values, which are not particularly useful to scan over. Other examples might have columns that duplicate others, for example when the data has been entered via a database program that allows the user to type country codes (e.g., LX) rather than the full country name (Luxembourg). Only one of these axes is needed to uniquely specify the country, so you can exclude the unwanted dimension by choosing "ignore" in the "Dimension" row. The deselected columns are rendered in red, indicating that its data will not be imported into *Ravel*:



In this example, the axis names has not been correctly inferred. Whilst, one can manually edit the axis names in the “Name” line, a quick shortcut is to select “Populate column labels” from the context menu.



4.15.2 Temporal data

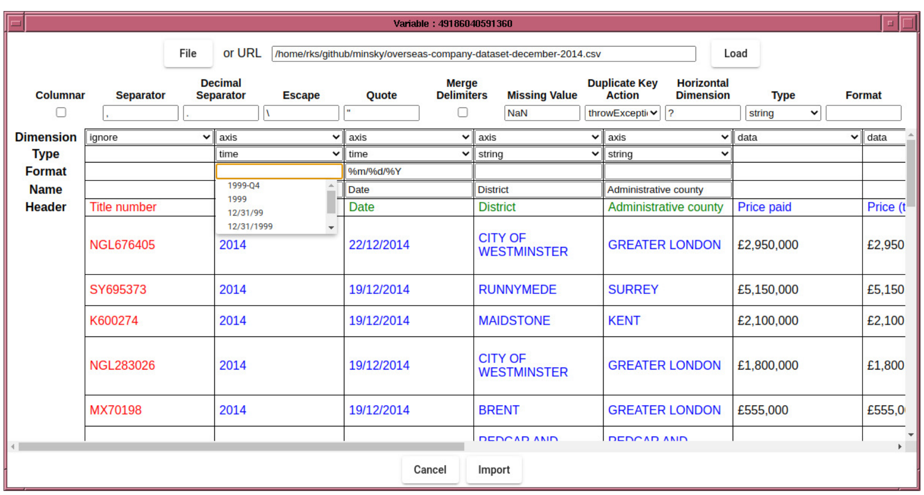
The Date column is current parsed as strings, which not only will be sorted incorrectly, but even if the data were in a YYYYMMDD format which is sorted correctly, will not have a uniform temporal spacing. It is therefore important to parse the Date column as temporal data, which is achieved by changing the column type to “time”, and specifying a format string, which follows strftime conventions with the addition of a quarter specifier (`%Q`).

If your temporal data is in the form Y*M*D*H*M*S, where * signifies any sequence of non-digit characters, and the year, month, day, hour minutes, second

Code	Description
%a or %A	The name of the day of the week according to the current locale, in abbreviated form or the full name.
%b or %B	The month name according to the current locale, in abbreviated form or the full name.
%d	Day of month in range 01 to 31
%e	Day of month in range 1 to 31
%H	Hour in range 0 to 23
%I	Hour in range 1 to 12
%m	Month as a decimal number (01 to 12)
%M	Minute in range 00 to 59
%Q	Quarter (0=1st January, 1=1st March etc)
%p	AM or PM
%s	Number of seconds since epoch (1st January 1970)
%S	Seconds in range 00 to 59
%y	Two digit year YY
%Y	Four digit year YYYY
%z	numerical timezone offset
%Z	Timezone name
%%	Literal % character

Table 4.1: Table of strftime codes

fields are regular integers in that order, then you can leave the format string blank . If some of the fields are missing, eg minutes and seconds, then they will be filled in with sensible defaults.

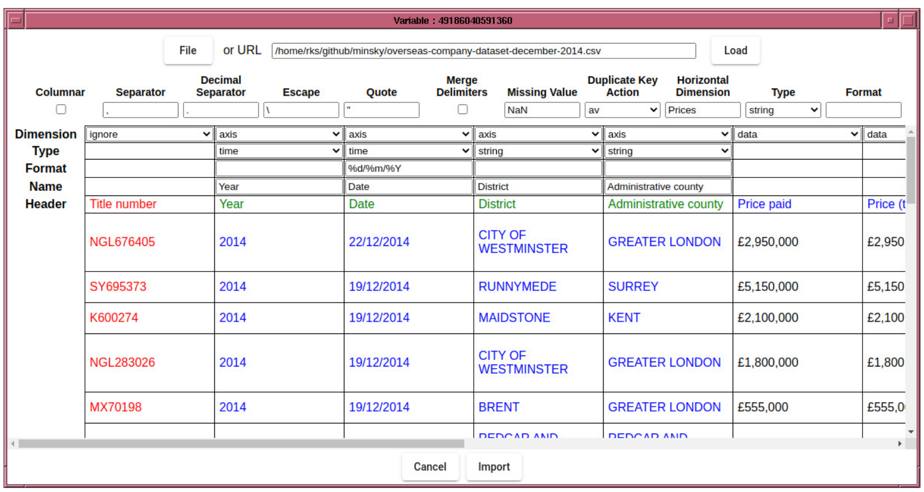


A *Strftime* formatted string consists of escape codes (with leading % characters). All other characters are treated as matching literally the characters of the input. So to match a date string of the format YYYY-MM-DD HH:MM:SS+ZZ

(ISO format), use a format string "`%Y-%m-%d %H:%M:%S+%Z`". Similarly, for quarterly data expressed like 1972-Q1, use "`%Y-Q%Q`". Note that only %Y and %y can be mixed with %Q (nothing else makes sense anyway).

Note that if your month data uses month names instead of numerical month representation, the it is important you select %d or %e depending on whether your day data has leading zeros or not. This is unfortunately a restriction of the underlying date import routine, which may change in the future.

Even in the current settings, you may still get a message "exhausted memory — try reducing the rank", or a similar message about hitting a 20% of physical memory threshold. In some cases, "titles" and "addresses" might be pretty much unique for each record, leading to a large, but very sparse hypercube. If you remove those columns, then you may encounter the "Duplicate key" message. In this case, you can aggregate over these records—which is desirable anyway to analyse trends in aggregate data. You do this by setting "Duplicate Key Action" to sum (or perhaps average for this example). After some additional playing around with dimensions to aggregate over, we can get the data imported.



4.15.3 Duplicate keys

In a hypercube, data is indexed by a list of indices, collectively known as a key. The indices may be strings, integers or date/time values. If more than one value exists in the CSV file for a given key, Minsky throws a "Duplicate key" exception. This exception gives you the option of writing a report, which is basically a sorted version of the original CSV file, with the errors listed at the beginning. You can open this report in a spreadsheet to see if data needs to be corrected or removed.

Duplicates will arise if you load data from a transactional database which stores the exact time of every transaction. Here the correct choice on "Duplicate Key Action" is to sum the entries so that all sales for a particular day (or month, quarter, or year) are aggregated for analysis of trends in the data.

4.15.4 Counter mode

As mentioned previously, Ravel is designed to work with numerical data. However, it is possible to work with purely symbolic data by using *counter mode*. This mode counts the number of times a key is present in the data file — either zero or one, or in the case of duplicate keys, a value greater than one. You can feed this into a Ravel and perform rollups to analyse the symbols statistically, for example as histograms.

To select counter mode, select the "counter" check box in the input dialog form. If no data columns are present at all, Ravel automatically selected counter mode.

4.16 Minsky-specific commands

4.16.1 File Menu Minsky commands

Library Opens a repository of models for the Minsky simulation system.

Recording Record the states of a model as it is being built for later replay. This is useful for demonstrating how to build a model, but bear in mind that recorded logs are not, in general, portable between versions of Minsky.

Replay recording Replay a recording of model states.

4.16.2 Options Menu Commands

The options menu allows you to customise aspects of Minsky.

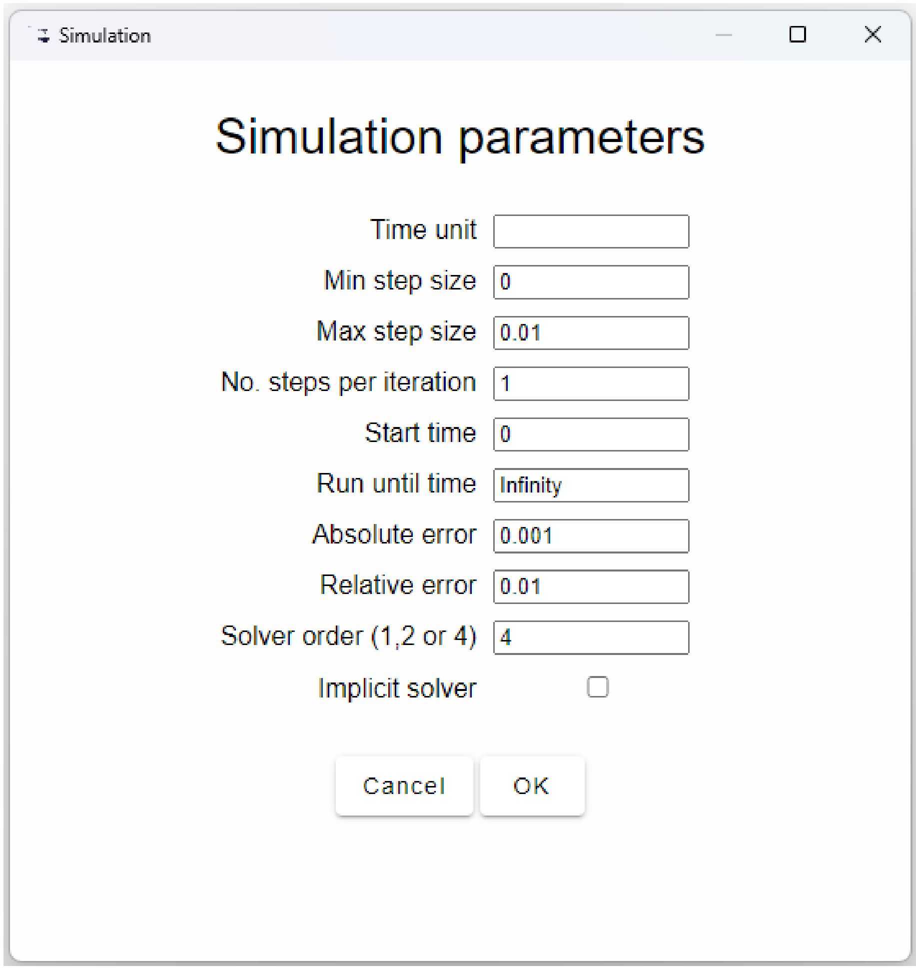
Preferences

- Godley table show values. When ticked, the values of flow variables are displayed in the Godley table whilst a simulation is running. This will tend to slow down the simulation somewhat.
- Godley table output style — whether +/− or DR/CR (debit/credit) indicators are used.
- Enable multiple equity columns - whether Godley table have more than one equity columns.
- Number of recent files to display — affects the recent files menu.
- Wrap long equations in LaTeX export. If ticked, use the breqn package to produce nicer looking automatically line-wrapped formulae. Because not all LaTeX implementations are guaranteed to support breqn, untick this option if you find difficulty.
- select a font for variable names etc.

Background colour — select a colour from which a colour scheme is computed.

4.16.3 Simulation Menu Commands

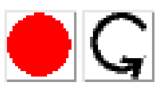
The Simulation Menu invokes the Simulation Parameters form.



- Time unit allows one to specify units for the time dimension for dimensional analysis. eg "year", "s" etc.
- Min and Max step size control aspect of the adaptive Runge-Kutta equation solver, which trade off performance and accuracy of the model. The algorithm is adaptive, so the step size will vary according to how stiff the system of equations is.
  - Specifying a minimum step size prevents the system from stalling, at the expense of accuracy when the step size reaches that minimum.
  - Specifying a maximum step size is useful to ensure one has sufficient data points for smooth plots.

- An iteration is the time between updates to plots. Increasing the number of solver steps per iteration decreases the overhead involved in updating the display, at the expense of smoothness of the plots.
- Start time is the value of the system t variable when the system is reset.
- Run until time can be used to pause the simulation once t reaches a certain value. Setting this to “Infinity” causes the simulation to run indefinitely, or until some arithmetic error occurs.
- The algorithm is adaptive, so the step size will vary according to how stiff the system of equations is.
- Solver order determines how high order a polynomial is used for approximating the actual system. A first order explicit solver is the classic Jacobi method, which is the fastest, but least accurate solver. 4th order is the adaptive Runge-Kutta equation solver.

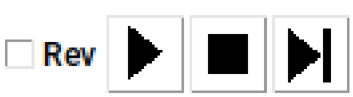
4.16.4 Record/Replay Buttons



These buttons control the recording / replay mode of *Minsky*. You can record your interactions with *Minsky*, and replay those interactions for demonstration/presentation purposes.

1. Record a session of building/modifying a model. Note that replaying a recorded session always starts from a blank canvas, so if you’re recording the modification of a model, ensure that the first thing recorded is to load the model being modified. This button is a toggle button, so clicking it again finishes the session, and closes the file.
2. Simulate/Replay button. Pressing this button puts *Minsky* into replay mode, and asks for a recording file. You may use the run buttons (run/pause,stop and step), as well as the speed slider, to control the replay. This button is a toggle button, so clicking it again returns *Minsky* back to the default simulation mode.

4.16.5 Run Buttons



The Run buttons respectively:

1. Start a simulation–when started the button changes to a pause icon, allowing you to pause the simulation  .

2. Stop a simulation and reset the simulation time to zero
3. Step through the simulation one iteration at a time.
4. Reverse checkbox changes the simulation time direction. Bear in mind that a simulation will eventually diverge from its original trajectory due to chaotic effects.

4.16.6 Speed slider

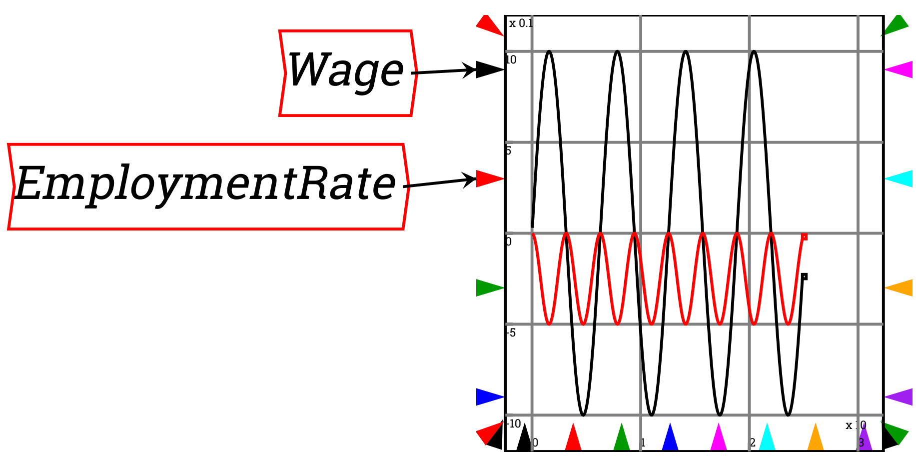
The speed slider controls the rate at which a model is simulated. The default speed is the maximum speed your system can support, but you can slow this down to more closely observe dynamics at crucial points in a simulation.

4.16.7 Simulation time

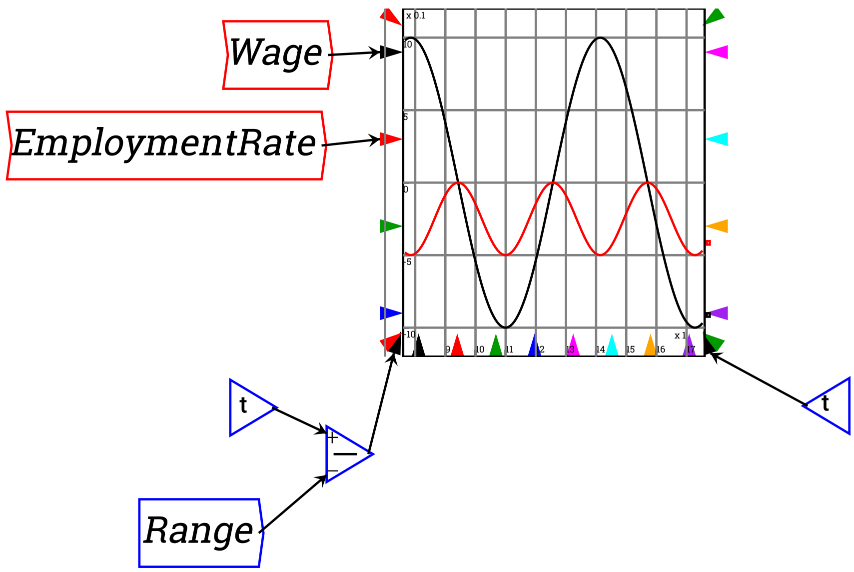
In the right hand top corner is a textual display of the current simulation time t, and the current (adaptive) difference between iterations Δt.

Time embeds a reference to the simulation time on the Canvas. This is not necessary in most simulations, but can be useful if you want to make a time-dependent process explicit, or control the appearance of a graph.

For example, by default a graph displays the simulation time on the horizontal axis, so that cycles get compressed as a simulation runs for a substantial period:

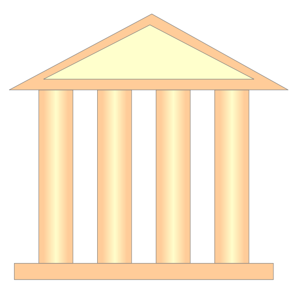


If a Time block is added to the marker for the x-axis range, you can control the number of years that are displayed. This graph is set up to show a ten year range of the model only:



Godley Table  . This is the fundamental element of *Minsky* that is not found (yet) in any other system dynamics program.

Clicking on it and placing the resulting Bank Icon on the Canvas enters a Godley table into your model:

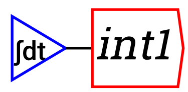


Double-click on the Bank Icon (or right-click and choose "Open Godley Table" from the context menu) and you get a double-entry bookkeeping table we call a Godley Table, which looks like the following onscreen:

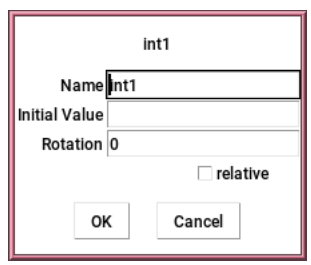
	Asset	Liability	Equity	
Flows ↓ / Stock Vars →	▼	▼	▼	A-L-E
Initial Conditions				*0*

Use this table to enter the bank accounts and financial flows in your model. We discuss this later.

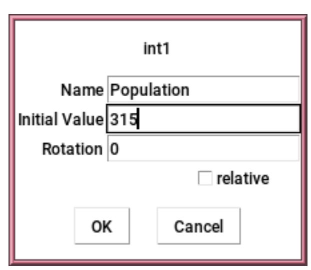
Integration  . This inserts a variable whose value depends on the integral of other variables in the system. This is the essential element for defining a dynamic model. Click on it and the following entity will appear at the top left hand side of the canvas (and move with your mouse until you click to place it somewhere:



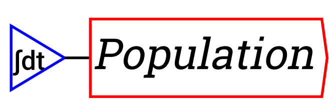
"int1" is just a placeholder for the integration variable, and the first thing you should do after creating one is give it a name. Double-click on the "int1", or right click and choose Edit. This will bring up the following menu:



Change the name to something appropriate, and give it an initial value. For example, if you were building a model that included America's population, you would enter the following:



The integrated variable block would now look like this:



To model population, you need to include a growth rate. According to Wikipedia, the current US population growth rate is 0.97 percent per annum. Expressed as an equation, this says that the annual change in population, divided by its current level, equals 0.0097:

$$\frac{1}{\text{Population}(t)} \cdot \left(\frac{d}{dt}\text{Population}(t)\right) = 0.0097$$

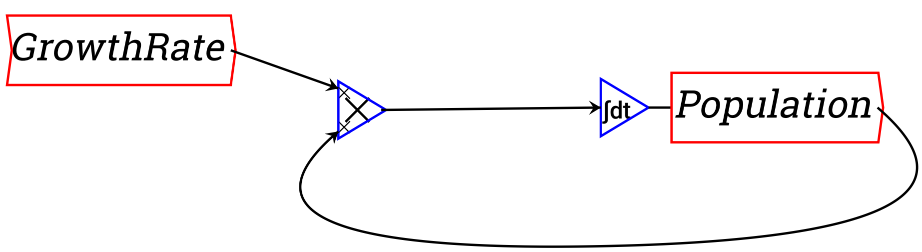
To express this as an integral equation, firstly we multiply both sides of this equation by Population to get:

$$\frac{d}{dt}\text{Population}(t) = 0.0097 \cdot \text{Population}(t)$$

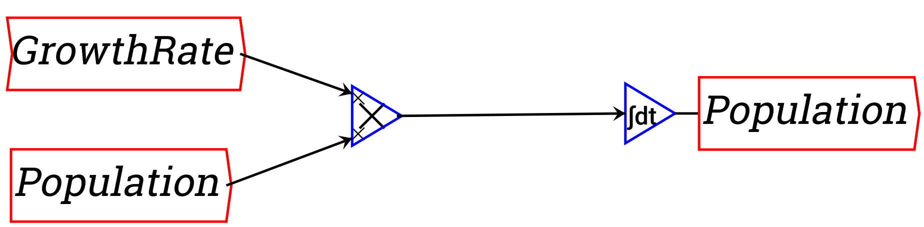
Then we integrate both sides to get an equation that estimates what the population will be T years into the future as:

$$\text{Population}(T) = 315 + \int_0^T 0.0097 \cdot \text{Population}(t)dt$$

Here, 315 (million) equals the current population of the USA, the year zero is today, and T is some number of years from today. The same equation done as a block diagram looks like this:



Or you can make it look more like the mathematical equation by right-clicking on "Population" and choosing "Copy item". This creates a copy of the Population variable, enabling you to wire up the equation this way:



Derivative Operator This operator symbolically differentiates its input, provided the input is differentiable. An error is generated if the input is not differentiable.

Chapter 5

User defined functions

Much of this chapter is exerpted from exprtk's read.txt file

5.1 Introduction

The C++ Mathematical Expression Toolkit Library (ExprTk) is a simple to use, easy to integrate and extremely efficient run-time mathematical expression parsing and evaluation engine. The parsing engine supports numerous forms of functional and logic processing semantics and is easily extensible.

With Minsky's user defined functions, expressions can refer to Minsky variables accessible from the current scope (ie local Minsky variables will hide global variables), and also parameters declared as part of the function name. One can also call other user defined functions, which is the only way a user defined function with more than 2 parameters can be used. For 0-2 parameters, user defined functions can be wired into a Minsky computation.

ExprTk identifiers (such as variable names and function names) consist of alphanumeric characters plus '_' and '.'. They must start with a letter. Minsky is reserving the underscore and full stop to act as an escape sequence, in order to refer to the full range of possible Minsky variable identifiers, including all unicode characters. This section will be updated once that feature is in place — for now, please avoid using those characters in identifiers.

5.2 Capabilities

The ExprTk expression evaluator supports the following fundamental arithmetic operations, functions and processes:

Types: Scalar, Vector, String

Basic operators: `+, -, *, /, %, ^`

Assignment: `:=, +=, -=, *=, /=, %=`

Equalities & Inequalities: `=, ==, <>, !=, <, <=, >, >=`

Logic operators: and, mand, mor, nand, nor, not, or, shl, shr, xnor, xor, true, false

Functions: abs, avg, ceil, clamp, equal, erf, erfc, exp, expm1, floor, frac, log, log10, log1p, log2, logn, max, min, mul, ncdf, nequal, root, round, roundn, sgn, sqrt, sum, swap, trunc

Trigonometry: acos, acosh, asin, asinh, atan, atanh, atan2, cos, cosh, cot, csc, sec, sin, sinc, sinh, tan, tanh, hypot, rad2deg, deg2grad, deg2rad, grad2deg

Control structures: if-then-else, ternary conditional, switch-case, return-statement

Loop statements: while, for, repeat-until, break, continue

String processing: in, like, ilike, concatenation

Optimisations: constant-folding, simple strength reduction and dead code elimination

Calculus: numerical integration and differentiation

5.3 Example expressions

The following is a short listing of infix format based mathematical expressions that can be parsed and evaluated using the ExprTk library.

- `sqrt(1 - (3 / x^2))`
- `clamp(-1, sin(2 * pi * x) + cos(y / 2 * pi), +1)`
- `sin(2.34e-3 * x)`
- `if(((x[2] + 2) == 3) and ((y + 5) <= 9),1 + w, 2 / z)`
- `inrange(-2,m,+2) == if(({-2 <= m} and [m <= +2]),1,0)`
- `({1/1}*[1/2]+(1/3))-{1/4}^[1/5]+(1/6)-({1/7}+[1/8]*(1/9))`
- `a * exp(2.2 / 3.3 * t) + c`
- `z := x + sin(2.567 * pi / y)`
- `u := 2.123 * {pi * z} / (w := x + cos(y / pi))`
- `2x + 3y + 4z + 5w == 2 * x + 3 * y + 4 * z + 5 * w`
- `3(x + y) / 2.9 + 1.234e+12 == 3 * (x + y) / 2.9 + 1.234e+12`
- `(x + y)3.3 + 1 / 4.5 == [x + y] * 3.3 + 1 / 4.5`

- `(x + y[i])z + 1.1 / 2.7 == (x + y[i]) * z + 1.1 / 2.7`

- `(sin(x / pi) cos(2y) + 1) == (sin(x / pi) * cos(2 * y) + 1)`

- `75x^17 + 25.1x^5 - 35x^4 - 15.2x^3 + 40x^2 - 15.3x + 1`

- `(avg(x,y) <= x + y ? x - y : x * y) + 2.345 * pi / x`

- `while (x <= 100) { x -= 1; }`

- `x <= 'abc123' and (y in 'AString') or ('1x2y3z' != z)`

- `((x + 'abc') like '*123*') or ('a123b' ilike y)`

- `sgn(+1.2^3.4z / -5.6y) <= {-7.8^9 / -10.11x }`

5.4 Copyright notice

Free use of the C++ Mathematical Expression Toolkit Library is permitted under the guidelines and in accordance with the most current version of the MIT License

5.5 Built-in operations & functions

5.5.1 Arithmetic & Assignment Operators

OPERATOR	DEFINITION
`+`	Addition between x and y. (eg: `x + y`)
`-`	Subtraction between x and y. (eg: `x - y`)
`*`	Multiplication between x and y. (eg: `x * y`)
`/`	Division between x and y. (eg: `x / y`)
`%`	Modulus of x with respect to y. (eg: `x % y`)
`^`	x^y. (eg: `x ^ y`)
`:=`	Assign the value of x to y. Where y is either a variable or vector type. (eg: `y := x`)
`+=`	Increment x by the value of the expression on the right hand side. Where x is either a variable or vector type. (eg: `x += abs(y - z)`)
`-=`	Decrement x by the value of the expression on the right hand side. Where x is either a variable or vector type. (eg: `x[i] -= abs(y + z)`)
`*=`	Assign the multiplication of x by the value of the expression on the righthand side to x. Where x is either a variable or vector type. (eg: `x *= abs(y / z)`)
`/=`	Assign the division of x by the value of the expression on the right-hand side to x. Where x is either a variable or vector type. (eg: `x[i + j] /= abs(y * z)`)
`%=`	Assign x modulo the value of the expression on the right hand side to x. Where x is either a variable or vector type. (eg: `x[2] %= y ^ 2`)

5.5.2 Equalities & Inequalities

OPERATOR	DEFINITION
`==` or `=`	True only if x is strictly equal to y. (eg: `x == y`)
`<>` or `!=`	True only if x does not equal y. (eg: `x <> y` or `x != y`)
`<`	True only if x is less than y. (eg: `x < y`)
`<=`	True only if x is less than or equal to y. (eg: `x <= y`)
`>`	True only if x is greater than y. (eg: `x > y`
`>=`	True only if x greater than or equal to y. (eg: `x >= y`)

5.5.3 Boolean Operations

<table>
<tr><th>OPERATOR</th><th>DEFINITION</th></tr>
<tr><td>true</td><td>True state or any value other than zero (typically 1).</td></tr>
<tr><td>false</td><td>False state, value of exactly zero.</td></tr>
<tr><td>and</td><td>Logical AND, True only if x and y are both true. (eg: x and y)</td></tr>
<tr><td>mand</td><td>Multi-input logical AND, True only if all inputs are true. Left to right short-circuiting of expressions. (eg: mand(x > y, z < w, u or v, w and x))</td></tr>
<tr><td>mor</td><td>Multi-input logical OR, True if at least one of the inputs are true. Left to right short-circuiting of expressions. (eg: mor(x > y, z < w, u or v, w and x))</td></tr>
<tr><td>nand</td><td>Logical NAND, True only if either x or y is false. (eg: x nand y)</td></tr>
<tr><td>nor</td><td>Logical NOR, True only if the result of x or y is false (eg: x nor y)</td></tr>
<tr><td>not</td><td>Logical NOT, Negate the logical sense of the input. (eg: not(x and y) == x nand y)</td></tr>
<tr><td>or</td><td>Logical OR, True if either x or y is true. (eg: x or y)</td></tr>
<tr><td>xor</td><td>Logical XOR, True only if the logical states of x and y differ. (eg: x xor y)</td></tr>
<tr><td>xnor</td><td>Logical XNOR, True iff the biconditional of x and y is satisfied. (eg: x xnor y)</td></tr>
<tr><td>&</td><td>Similar to AND but with left to right expression short circuiting optimisation. (eg: (x & y) == (y and x))</td></tr>
<tr><td>|</td><td>Similar to OR but with left to right expression short circuiting optimisation. (eg: (x | y) == (y or x))</td></tr>
</table>

5.5.4 General Purpose Functions

FUNCTION	DEFINITION
`abs`	Absolute value of x. (eg: `abs(x)`)
`avg`	Average of all the inputs. (eg: `avg(x,y,z,w,u,v) == (x + y + z + w + u + v) / 6`)
`ceil`	Smallest integer that is greater than or equal to x.
`clamp`	Clamp x in range between r0 and r1, where r0 ¡ r1. (eg: `clamp(r0,x,r1)`)
`equal`	Equality test between x and y using normalised epsilon
`erf`	Error function of x. (eg: `erf(x)`)
`erfc`	Complimentary error function of x. (eg: `erfc(x)`)
`exp`	e^x (eg: `exp(x)`)
`expm1`	e^{x-1} where x is very small. (eg: `expm1(x)`)
`floor`	Largest integer that is less than or equal to x. (eg: `floor(x)`)
`frac`	Fractional portion of x. (eg: `frac(x)`)
`hypot`	$\sqrt{x^2+y^2}$ (eg: `hypot(x,y) = sqrt(x*x + y*y)`)
`iclamp`	Inverse-clamp x outside of the range r0 and r1. Where r0 ¡ r1. If x is within the range it will snap to the closest bound. (eg: $\texttt{iclamp(r0,x,r1)} = \begin{cases} r0 & \text{if} \quad x \le r0 \\ x & \text{if} \quad r0 \le x \le r1 \\ r1 & \text{if} \quad x \ge r1 \end{cases}$)
`inrange`	In-range returns 'true' when x is within the range $[r_0, r_1]$. Where $r_0 < r_1$. (eg: `inrange(r0,x,r1)`)
`log`	Natural logarithm $\ln x$. (eg: `log(x)`)
`log10`	$\log_{10} x$. (eg: `log10(x)`)
`log1p`	$\ln(1+x)$, where x is very small. (eg: `log1p(x)`)
`log2`	$\log_2 x$. (eg: `log2(x)`)
`logn`	$\log_n x$, where n is a positive integer. (eg: `logn(x,8)`)
`max`	Largest value of all the inputs. (eg: `max(x,y,z,w,u,v)`)
`min`	Smallest value of all the inputs. (eg: `min(x,y,z,w,u)`)
`mul`	Product of all the inputs. (eg: `mul(x,y,z,w,u,v,t) == (x * y * z * w * u * v * t)`)
`ncdf`	Normal cumulative distribution function. (eg: `ncdf(x)`)
`nequal`	Not-equal test between x and y using normalised epsilon
`pow`	x^y. (eg: `pow(x,y) == x ^ y`)
`root`	$\sqrt[n]{x}$, where n is a positive integer. (eg: `root(x,3) == x^(1/3)`)
`round`	Round x to the nearest integer. (eg: `round(x)`)
`roundn`	Round x to n decimal places (eg: `roundn(x,3)`) where $n > 0$ is an integer. (eg: `roundn(1.2345678,4) == 1.2346`)
`sgn`	Sign of x, -1 where $x < 0$, $+1$ where $x > 0$, else zero. (eg: `sgn(x)`)
`sqrt`	$\sqrt{x}$, where $x >= 0$. (eg: `sqrt(x)`)
`sum`	Sum of all the inputs. (eg: `sum(x,y,z,w,u,v,t) == (x + y + z + w + u + v + t)`)
`swap` `<=>`	Swap the values of the variables x and y and return the current value of y. (eg: `swap(x,y)` or `x <=> y`)
`trunc`	Integer portion of x. (eg: `trunc(x)`)

5.5.5 Trigonometry Functions

FUNCTION	DEFINITION
`acos`	Arc cosine of x expressed in radians. Interval $[-1, +1]$ (eg: `acos(x)`)
`acosh`	Inverse hyperbolic cosine of x expressed in radians. (eg: `acosh(x)`)
`asin`	Arc sine of x expressed in radians. Interval $[-1, +1]$ (eg: `asin(x)`)
`asinh`	Inverse hyperbolic sine of x expressed in radians. (eg: `asinh(x)`)
`atan`	Arc tangent of x expressed in radians. Interval $[-1, +1]$ (eg: `atan(x)`)
`atan2`	Arc tangent of (x/y) expressed in radians. $[-\pi, +\pi]$ (eg: `atan2(x,y)`)
`atanh`	Inverse hyperbolic tangent of x expressed in radians. (eg: `atanh(x)`)
`cos`	Cosine of x. (eg: `cos(x)`)
`cosh`	Hyperbolic cosine of x. (eg: `cosh(x)`)
`cot`	Cotangent of x. (eg: `cot(x)`)
`csc`	Cosecant of x. (eg: `csc(x)`)
`sec`	Secant of x. (eg: `sec(x)`)
`sin`	Sine of x. (eg: `sin(x)`)
`sinc`	Sine cardinal of x. (eg: `sinc(x)`)
`sinh`	Hyperbolic sine of x. (eg: `sinh(x)`)
`tan`	Tangent of x. (eg: `tan(x)`)
`tanh`	Hyperbolic tangent of x. (eg: `tanh(x)`)
`deg2rad`	Convert x from degrees to radians. (eg: `deg2rad(x)`)
`deg2grad`	Convert x from degrees to gradians. (eg: `deg2grad(x)`)
`rad2deg`	Convert x from radians to degrees. (eg: `rad2deg(x)`)
`grad2deg`	Convert x from gradians to degrees. (eg: `grad2deg(x)`)

5.5.6 String Processing

FUNCTION	DEFINITION
`=` , `==`, `!=`, `<>`, `<=`, `>=`, `<` , `>`	All common equality/inequality operators are applicable to strings and are applied in a case sensitive manner. In the following example x, y and z are of type string. (eg: `not((x <= 'AbC') and ('1x2y3z' <> y)) or (z == x)`)
`in`	True only if x is a substring of y. (eg: `x in y` or `'abc' in 'abcdefgh'`)
`like`	True only if the string x matches the pattern y. Available wildcard characters are '*' and '?' denoting zero or more and zero or one matches respectively. (eg: `x like y` or `'abcdefgh' like 'a?d*h'`)
`ilike`	True only if the string x matches the pattern y in a case insensitive manner. Available wildcard characters are '*' and '?' denoting zero or more and zero or one matches respectively. (eg: `x ilike y` or `'a1B2c3D4e5F6g7H' ilike 'a?d*h'`)
`[r0:r1]`	The closed interval $[r0, r1]$ of the specified string. eg: Given a string x with a value of 'abcdefgh' then: 1. `x[1:4] == 'bcde'` 2. `x[ :5] == x[:10 / 2] == 'abcdef'` 3. `x[2 + 1: ] == x[3:] =='defgh'` 4. `x[ : ] == x[:] == 'abcdefgh'` 5. `x[4/2:3+2] == x[2:5] == 'cdef'` Note: Both r0 and r1 are assumed to be integers, where r0 ¡= r1. They may also be the result of an expression, in the event they have fractional components truncation will be performed. (eg: $1.67 \rightarrow 1$)

FUNCTION	DEFINITION
`:=`	Assign the value of x to y. Where y is a mutable string or string range and x is either a string or a string range. eg: 1. `y := x` 2. `y := 'abc'` 3. `y := x[:i + j]` 4. `y := '0123456789'[2:7]` 5. `y := '0123456789'[2i + 1:7]` 6. `y := (x := '0123456789'[2:7])` 7. `y[i:j] := x` 8. `y[i:j] := (x + 'abcdefg'[8 / 4:5])[m:n]` Note: For options 7 and 8 the shorter of the two ranges will denote the number characters that are to be copied.
`+`	Concatenation of x and y. Where x and y are strings or string ranges. eg 1. `x + y` 2. `x + 'abc'` 3. `x + y[:i + j]` 4. `x[i:j] + y[2:3] + '0123456789'[2:7]` 5. `'abc' + x + y` 6. `'abc' + '1234567'` 7. `(x + 'a1B2c3D4' + y)[i:2j]`
`+=`	Append to x the value of y. Where x is a mutable string and y is either a string or a string range. eg: 1. `x += y` 2. `x += 'abc'` 3. `x += y[:i + j] + 'abc'` 4. `x += '0123456789'[2:7]`
`<=>`	Swap the values of x and y. Where x and y are mutable strings. (eg: `x <=> y`)

FUNCTION	DEFINITION
`[]`	The string size operator returns the size of the string being actioned. eg: 1. `'abc'[] == 3` 2. `var max_str_length := max(s0[],s1[],s2[],s3[]` 3. `('abc' + 'xyz')[] == 6` 4. `(('abc' + 'xyz')[1:4])[] == 4`

5.5.7 Control Structures

STRUCTURE	DEFINITION
`if`	If x is true then return y else return z.eg: 1. `if (x, y, z)` 2. `if ((x + 1) > 2y, z + 1, w / v)` 3. `if (x > y) z;` 4. `if (x <= 2*y) { z + w };`

`if-else`	The if-else/else-if statement. Subject to the condition branch the statement will return either the value of the consequent or the alternative branch. eg:
	1. `if (x > y) z; else w;`
	2. `if (x > y) z; else if (w != u) v;`
	3. `if (x < y) { z; w + 1; } else u;`
	4. `if ((x != y) and (z > w))`
	`{`
	`  y := sin(x) / u;`
	`  z := w + 1;`
	`}`
	`else if (x > (z + 1))`
	`{`
	`  w := abs (x - y) + z;`
	`  u := (x + 1) > 2y ? 2u : 3u;`
	`}`

`switch`	The first true case condition that is encountered will determine the result of the switch. If none of the case conditions hold true, the default action is assumed as the final return value. This is sometimes also known as a multi-way branch mechanism. eg:
	`switch`
	`{`
	`  case x > (y + z) : 2 * x / abs(y - z);`
	`  case x < 3       : sin(x + y);`
	`  default          : 1 + x;`
	`}`

`while`	The structure will repeatedly evaluate the internal statement(s) 'while' the condition is true. The final statement in the final iteration will be used as the return value of the loop. eg:
	`while ((x -= 1) > 0)`
	`{`
	`  y := x + z;`
	`  w := u + y;`
	`}`

<table>
<tr><th>FUNCTION</th><th>DEFINITION</th></tr>
<tr><td><code>repeat/until</code></td><td>The structure will repeatedly evaluate the internal statement(s) 'until' the condition is true. The final statement in the final iteration will be used as the return value of the loop. eg:
<pre>
repeat
  y := x + z;
  w := u + y;
until ((x += 1) > 100)
</pre></td></tr>
<tr><td><code>for</code></td><td>The structure will repeatedly evaluate the internal statement(s) while the condition is true. On each loop iteration, an 'incrementing' expression is evaluated. The conditional is mandatory whereas the initialiser and incrementing expressions are optional. eg:
<pre>
for (var x := 0; (x < n) and (x != y); x += 1)
{
  y := y + x / 2 - z;
  w := u + y;
}
</pre></td></tr>
<tr><td><code>break/break[]</code></td><td>Break terminates the execution of the nearest enclosed loop, allowing for the execution to continue on external to the loop. The default break statement will set the return value of the loop to NaN, where as the return based form will set the value to that of the break expression. eg:
<pre>
while ((i += 1) < 10)
{
  if (i < 5)
    j -= i + 2;
  else if (i % 2 == 0)
    break;
  else
    break[2i + 3];
}
</pre></td></tr>
<tr><td><code>continue</code></td><td>Continue results in the remaining portion of the nearest enclosing loop body to be skipped. eg:
<pre>
for (var i := 0; i < 10; i += 1)
{
  if (i < 5)
    continue;
  j -= i + 2;
}
</pre></td></tr>
</table>

FUNCTION	DEFINITION

`return`	Return immediately from within the current expression. With the option of passing back a variable number of values (scalar, vector or string). eg:
	1. `return [1];`
	2. `return [x, 'abx'];`
	3. `return [x, x + y,'abx'];`
	4. `return [];`
	5. `if (x < y)`
	`    return [x, x - y, 'result-set1', 123.456];`
	`   else`
	`    return [y, x + y, 'result-set2'];`

`?:`	Ternary conditional statement, similar to that of the above denoted if-statement. eg: 1. `x ? y : z` 2. `x + 1 > 2y ? z + 1 : (w / v)` 3. `min(x,y) > z ? (x < y + 1) ? x : y : (w * v)`

`~`	Evaluate each sub-expression, then return as the result the value of the last sub-expression. This is sometimes known as multiple sequence point evaluation. eg:
	`~(i := x + 1, j := y / z, k := sin(w/u)) == (sin(w/u)))`
	`~{i := x + 1; j := y / z; k := sin(w/u)} == (sin(w/u)))`

`[*]`	Evaluate any consequent for which its case statement is true. The return value will be either zero or the result of the last consequent to have been evaluated. eg:
	`[*]`
	`{`
	`  case (x + 1) > (y - 2)    : x := z / 2 + sin(y / pi);`
	`  case (x + 2) < abs(y + 3) : w / 4 + min(5y,9);`
	`  case (x + 3) == (y * 4)   : y := abs(z / 6) + 7y;`
	`}`

`[]`	The vector size operator returns the size of the vector being actioned. eg: 1. `v[]` 2. `max_size := max(v0[],v1[],v2[],v3[])`

Note: In the tables above, the symbols x, y, z, w, u and v where appropriate may represent any of one the following:

1. Literal numeric/string value
2. A variable
3. A vector element
4. A vector
5. A string
6. An expression comprised of [1], [2] or [3] (eg: `2 + x /vec[3]`)

5.6 Fundamental types

ExprTk supports three fundamental types which can be used freely in expressions. The types are as follows:

Scalar Type The scalar type is a singular numeric value. The underlying type is that used to specialise the ExprTk components (float, double, long double, MPFR et al).

Vector Type The vector type is a fixed size sequence of contiguous scalar values. A vector can be indexed resulting in a scalar value. Operations between a vector and scalar will result in a vector with a size equal to that of the original vector, whereas operations between vectors will result in a vector of size equal to that of the smaller of the two. In both mentioned cases, the operations will occur element-wise.

String Type The string type is a variable length sequence of 8-bit chars. Strings can be assigned and concatenated to one another, they can also be manipulated via sub-ranges using the range definition syntax. Strings however can not interact with scalar or vector types.

Chapter 6

Introduction to Minsky

Minsky is a system-dynamics program. If you are familiar with these programs, please go to *If you are experienced in system dynamics*. If you are not, please go to *If you are new to system dynamics*

6.1 If you are new to system dynamics

Minsky is one of many "system dynamics" programs[1]. These programs allow a dynamic model of a system to be constructed, not by writing mathematical equations or computer code, but by laying out a model in a block diagram, which can then be simulated. These programs are now one of the main tools used by engineers to design complex products, and by management consultants to advise on corporate management, product marketing, local government projects, etc.

Minsky has many features in common with these programs, and adds another unique means to create dynamic equations—the "Godley Table"—that is superior to block diagrams for modelling monetary flows.

The main advantages of the block diagram representation of dynamic equations over a list of equations are:

- They make the causal relationships in a complex model obvious. It takes a specialized mind to be able to see the causal relations in a large set of mathematical equations; the same equations laid out as diagrams can be read by anyone who can read a stock and flow diagram—and that's most of us;
- The block diagram paradigm makes it possible to structure a complex model via groups. For example, the fuel delivery system in a car can be treated as one group, the engine as another, the exhaust as yet another. This reduces visual complexity, and also makes it possible for different components of a complex model to be designed by different teams, and then "wired together" at a later stage.

[1] https://en.wikipedia.org/wiki/System_dynamics

Though these programs differ in appearance, they all work the same way: variables in a set of equations are linked by wires to mathematical operators. What would otherwise be a long list of equations is converted into a block diagram, and the block diagram makes the causal chain in the equations explicit and visually obvious. They are also explicitly tailored to producing numerical simulations of models.

6.1.1 Block diagram example

One of the very first models of a dynamic system was developed independently by the mathematicians Lokta[2] and Volterra[3] and is now known as the Lotka-Volterra model[4]. This model simulates interacting populations of predators and prey, which had been seen to display fluctuations that either could not be explained by exogenous shocks, or which displayed counterintuitive responses to exogenous shocks—such as the observed increase in the proportion of predators (sharks, rays, etc.) in the fishing catch in the Adriatic Sea during WWI, when biologists had expected the proportion of predators to fall[5].

The basic dynamics of the predator-prey model are that the number of prey is assumed to grow exponentially in the absence of predators, while the number of predators is assumed to fall exponentially in the absence of prey. Using Rabbits as our example of prey and Foxes as our example of predators, the initial dynamic equations are :

$$\begin{aligned} \tfrac{d}{dt}Rabbits &= r \times Rabbits \\ \tfrac{d}{dt}Foxes &= -f \times Foxes \end{aligned}$$

When interaction between predators and prey is considered, the simplest assumption is that predators reduce the growth rate of prey (r) by some constant (ρ) times how many predators there are, while prey reduce the death rate of predators (f) by another constant (ϕ) times how many prey there are:

$$\begin{aligned} \tfrac{d}{dt}Rabbits &= (r - \rho \times Foxes) \times Rabbits \\ \tfrac{d}{dt}Foxes &= (-f + \phi \times Rabbits) \times Foxes \end{aligned}$$

The non-zero equilibrium of this system is easily calculated by setting the differential equations to zero:

$$\begin{aligned} Rabbit_{Eq} &= \tfrac{f}{\phi} \\ Fox_{Eq} &= \tfrac{r}{\rho} \end{aligned}$$

Mathematicians initially expected that this model would converge to this equilibrium over time, but this was not the case:

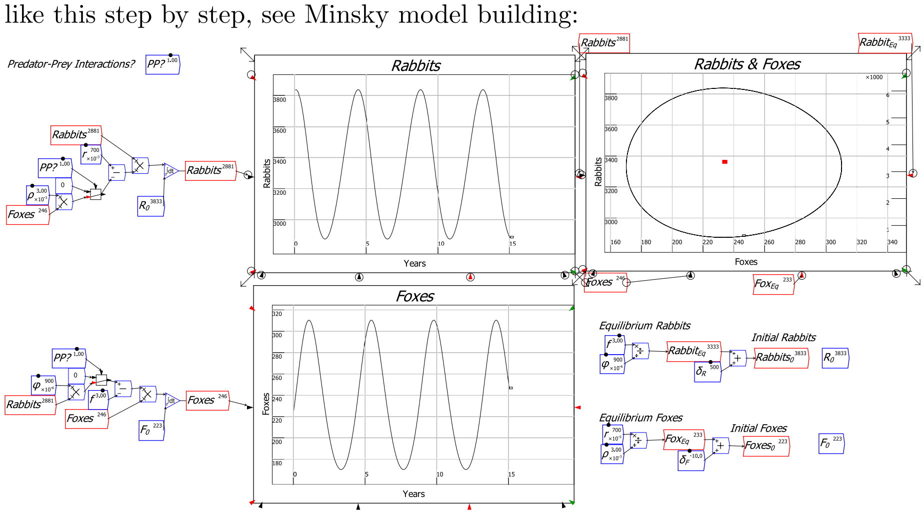
[2] https://www.jstor.org/stable/84156
[3] https://www-nature-com.libproxy.ucl.ac.uk/articles/118558a0
[4] https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations
[5] https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations#History

> Periodic phenomena play an important role in nature, both organic and inorganic . . . it appeared, from the nature of the solution obtained, improbable that undamped, permanent oscillations would arise . . . It was, therefore, with considerable surprise that the writer, on applying his method to certain special cases, found these to lead to undamped, and hence indefinitely continued, oscillations[6].
>
> The mathematical reason for this phenomenon is fairly easily derived by mathematical stability analysis. For a system as simple as this, the main advantage of a system dynamics program is that it enables the numerical simulation of this model. To do this, this system *of ordinary differential equations* needs to be converted into *integral equations,* since numerical integration is a more accurate process than numerical differentiation. In mathematical form, these integral equations are:

$$Rabbits = Rabbits_0 + \int ((r - \rho \times Foxes) \times Rabbits)$$
$$Foxes = Foxes_0 + \int ((-f + \phi \times Rabbits) \times Foxes)$$

Here $Rabbits_0$and $Foxes_0$ represent the initial number of Rabbits and Foxes.

The next figure shows this model in *Minsky*, with the initial conditions set up so that they differ slightly from the equilibrium. To see how to build a model like this step by step, see Minsky model building:



Programs like Vensim and Simulink have been in existence for decades, and they are now mature products that provide everything their user-base of management consultants and engineers want for modeling and analyzing complex dynamic systems. So why has *Minsky* been developed? The key reason is its unique feature of Godley Tables, which allow dynamic equations to be developed for monetary transactions via double-entry bookkeeping tables.

[6]https://www.jstor.org/stable/84156

6.2 If you are experienced in system dynamics

As an experienced system dynamics user (or if you've just read the Introduction to Minsky), the most important thing you need to know is what *Minsky* provides that other system dynamics programs do not. That boils down to one feature: The Godley Table.

6.2.1 Godley Tables

Godley tables are a unique feature of *Minsky*. They are based on what could be called the world's first GUI ("Graphical User Interface"), the accountant's double-entry bookkeeping table.

Double-entry was invented in 15th century Italy[7] to enable accurate recording of financial transactions. The essence of this method is (a) to classify all of an entity's accounts as either Assets or Liabilities, with the difference between them representing the Equity of the entity; and (b) to record every transaction twice, once as a Credit (CR)and once as a Debit (DR), where the definitions of Credit and Debit entries for Assets, Liabilities and Equity are designed to ensure that the transaction was accurately recorded. Minsky uses this GUI to generate stock-flow consistent models of financial flows, but by default uses plus ($+$) and minus ($-$) operators (though the accountant's convention of Credit (CR)and Debit (DR) can be chosen via the Options menu). This system guarantees that, no matter how complex the model is, the equations are internally consistent.

Minsky uses this GUI to generate systems of ordinary differential equations to model financial flows. The columns specify stocks, while the entries in rows are flows. *The symbolic sum of a column is thus the rate of change of the associated stock*. *Minsky* takes a table like the next figure:

Banking Sector

	Asset			Liability		Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves*	*Loans*	*$Bonds_B$*	*Firms*	*Consumers*	*$Banks_E$*	*0*
Initial Conditions	0	0	0	0	0	0	*0*
Net Bank Lending		*Credit*		*Credit*			*0*
Interest on Loans				$-Int_L$		Int_L	*0*
Pay Wages				*-Wages*	*Wages*		*0*
Taxation	*-Tax*			*-0.3Tax*	*-0.7Tax*		*0*
Government spending	$Spend_G$			$0.4Spend_G$	$0.6Spend_G$		*0*
Treasury Bond Sales	$-BondSale_T^B$		$BondSale_T^B$				*0*
Central Bank Bond Purchases	$BondBuy_{CB}$		$-BondBuy_{CB}$				*0*
Bank Bond Sales			$-BondSale_B^{NB}$	$-0.8BondSale_B^{NB}$	$-0.2BondSale_B^{NB}$		*0*
Bond Interest to Banks	Int_B^B					Int_B^B	*0*
Bond Interest to Nonbanks	Int_B^{NB}			$0.8Int_B^{NB}$	$0.2Int_B^{NB}$		*0*
Consumption				*Consume*	*-Consume*		*0*
Bank spending				$Spend_B$		$-Spend_B$	*0*

And converts it into a set of dynamic equations, which by construction are "stock-flow consistent":

[7]https://www.janegleesonwhite.com/double

$$
\begin{aligned}
\text{Consume} &= 0 \\
\text{Wages} &= 0 \\
\text{Tax} &= 0 \\
\text{Spend}_G &= 0 \\
\text{Spend}_B &= 0 \\
\text{Int}_L &= 0 \\
\text{Int}_{B^{\text{NB}}} &= 0 \\
\text{Int}_{B^B} &= 0 \\
\text{Credit} &= 0 \\
\text{BondSale}_{T^B} &= 0 \\
\text{BondSale}_{B^{\text{NB}}} &= 0 \\
\text{Banks}_E(0) &= 0 \\
\frac{d\text{Banks}_E}{dt} &= \text{Int}_L + \text{Int}_{B^B} - \text{Spend}_B \\
\text{Bonds}_B(0) &= 0 \\
\frac{d\text{Bonds}_B}{dt} &= \text{BondSale}_{T^B} - \text{BondSale}_{B^{\text{NB}}} \\
\text{Consumers}(0) &= 0 \\
\frac{d\text{Consumers}}{dt} &= \text{Wages} + 0.6 \times \text{Spend}_G + 0.2 \times \text{Int}_{B^{\text{NB}}} - \\
&\quad (0.7 \times \text{Tax} + 0.2 \times \text{BondSale}_{B^{\text{NB}}} + \text{Consume}) \\
\text{Firms}(0) &= 0 \\
\frac{d\text{Firms}}{dt} &= \text{Credit} + 0.4 \times \text{Spend}_G + 0.8 \times \text{Int}_{B^{\text{NB}}} + \text{Consume} + \text{Spend}_B - \\
&\quad (\text{Int}_L + \text{Wages} + 0.3 \times \text{Tax} + 0.8 \times \text{BondSale}_{B^{\text{NB}}}) \\
\text{Loans}(0) &= 0 \\
\frac{d\text{Loans}}{dt} &= \text{Credit} \\
\text{Reserves}(0) &= 0 \\
\frac{d\text{Reserves}}{dt} &= \text{Spend}_G + \text{Int}_{B^B} + \text{Int}_{B^{\text{NB}}} - (\text{Tax} + \text{BondSale}_{T^B})
\end{aligned}
$$

The model is completed by defining the flows on the canvas.

In addition, because one entity's financial asset is another's financial liability, *Minsky* enables the construction of a multi-sectoral view of financial transactions. The wedge next to every account name is used on other tables to search for Assets that have not yet been recorded as a Liability, and vice versa. For example, the previous Table recorded the financial system from the point of view of the Banking Sector, for which Reserves—the deposit accounts of private banks at the Central Bank–are an Asset. Reserves are also a liability of the Central Bank, and that can be shown in *Minsky* by adding an additional Godley Table, naming it *Central Bank*, and recording Reserves as a Liability:

Central Bank

	Asset	Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	▼	*Reserves* ▼	*CentralBank$_E$* ▼	*0*
Initial Conditions		0	0	*0*
Treasury Bond Sales		*-BondSale$_T^B$*		*BondSale$_T^B$*
Bond Interest to Banks		*Int$_B^B$*		*-Int$_B^B$*
Bond Interest to Nonbanks		*Int$_B^{NB}$*		*-Int$_B^{NB}$*
Government spending		*Spend$_G$*		*-Spend$_G$*
Taxation		*-Tax*		*Tax*
Central Bank Bond Purchases		*BondBuy$_{CB}$*		*-BondBuy$_{CB}$*

At this stage, the financial transactions have been entered only once—against the Central Bank's Liability of Reserves. Each transaction has to be entered a second time to complete its record, but at present there are no available accounts in which to record the transactions a second time.

Central Bank

	Asset	Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	▼	*Reserves* ▼	*CentralBank$_E$* ▼	*0*
Initial Conditions		0	0	*0*
Treasury Bond Sales		*-BondSale$_T^B$*		*BondSale$_T^B$*
Bond Interest to Banks		*Int$_B^B$*		*-Int$_B^B$*
Bond Interest to Nonbanks		*Int$_B^{NB}$*		*-Int$_B^{NB}$*
Government spending		*Spend$_G$*		*-Spend$_G$*
Taxation		*-Tax*		*Tax*
Central Bank Bond Purchases		*BondBuy$_{CB}$*		*-BondBuy$_{CB}$*

Bond purchases by the Central Bank indicate the need for to show bonds owned by the Central Bank as an asset, while the other transactions indicate the need to show the Treasury's account at the Central Bank, the "Consolidated Revenue Fund", as an additional liability:

Central Bank	Asset	Liability		Equity	A-L-E
Flows ↓ / Stock Vars →	$Bonds_{CB}$	$Reserves$	$Treasury_{CRF}$	$CentralBank_E$	0
Initial Conditions	0	0	0	0	0
Treasury Bond Sales		$-BondSale_T^B$	$BondSale_T^B$		0
Bond Interest to Banks		Int_B^B	$-Int_B^B$		0
Bond Interest to Nonbanks		Int_B^{NB}	$-Int_B^{NB}$		0
Government spending		$Spend_G$	$-Spend_G$		0
Taxation		$-Tax$	Tax		0
Central Bank Bond Purchases	$BondBuy_{CB}$	$BondBuy_{CB}$			0

This is an inherently better way to generate a dynamic model of financial flows than the flowchart method used by all other system dynamics programs, for at least four reasons:

- All financial transactions are flows between entities. The tabular layout captures this in a very natural way: each row shows where a flow originates, and where it ends up;
- The program imposes the rules of double-entry bookkeeping, in which entries on each row balance to zero according to the *accounting equation* ($Assets = Liabilities + Equity$). If you don't ensure that each flow starts somewhere and ends somewhere, then the program will identify your mistake;
- The double-entry perspective assists in the completion of a model, since the requirement of a matching entry for each transaction indicates the accounts that are needed to complete the accounting; and
- Double-entry bookkeeping acts as a prohibition against recording invalid transactions.

Minsky thus adds an element to the system dynamics toolkit which is essential for modeling the monetary flows that are an intrinsic aspect of a market economy.

6.2.2 *Minsky*'s unusual system dynamics features

Minsky differs from conventional system dynamics models in various ways:

- Most programs store their mathematical formulas within a text block, which must be opened to see its equation. The mathematics of a *Minsky* model is shown explicitly on the design canvas;
- Most programs require every entity used to build an equation to be wired to the block in which the equation is defined, while each entity is shown only once on the design canvas. This generates the familiar system dynamics "spaghetti diagram", as even practitioners—rather than merely

critics—often describe their models. *Minsky* enables entities to be shown multiple times on the design canvas, which substantially reduces the number of wires needed to build a model. *Minsky*'s equations therefore resemble small "spider webs", rather than large bowls of spaghetti. This makes it easier to understand what a model does; and

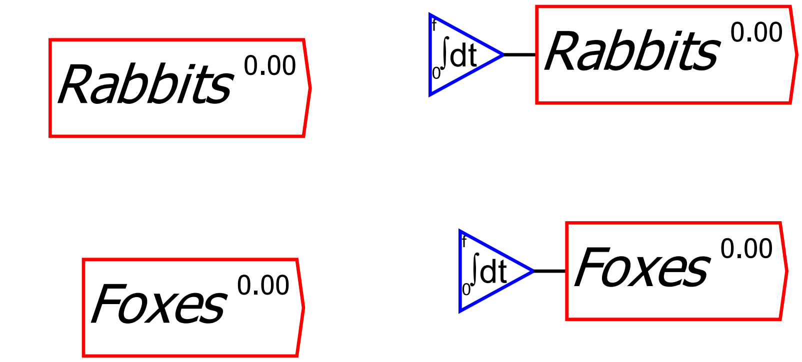
- Most programs show the end results of a simulation. *Minsky* shows simulations dynamically, and the numerical values of parameters and variables are shown on the parameters and variables themselves. Parameters can be altered during a simulation, which enables them to be used as a "control panel" for simulation of policy proposals, etc.

These unusual features are best illustrated by building a simple model in *Minsky*. The next section explains how to build a simpler version of the predator-prey model.

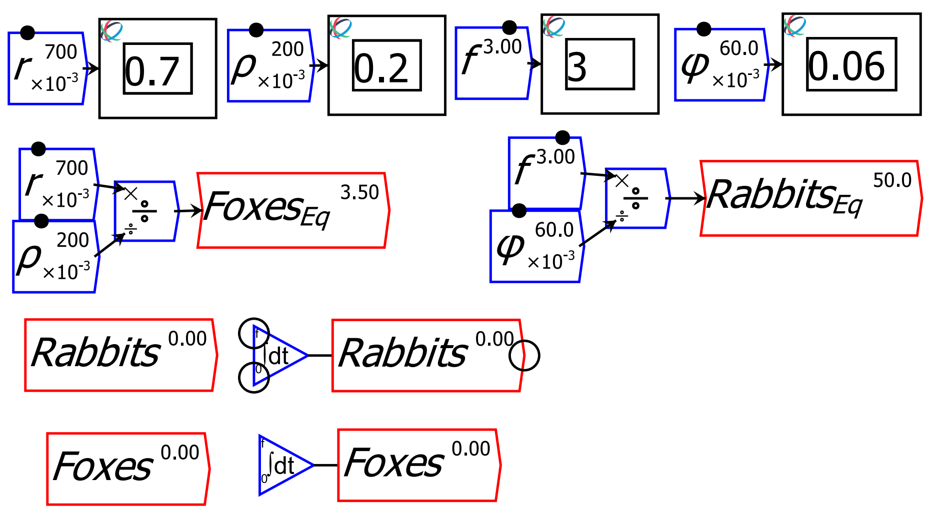
6.2.3 Building a predator-prey model in Minsky

The first step in building the Rabbits and Foxes model in *Minsky* is to insert two integral blocks on the canvas—one for Rabbits and the other for Foxes. An integral block is inserted by clicking on the operator. To alter its name from the generic "int1", either double-click on the widget, which brings up the context menu for integral blocks, or hover the mouse over the block and click the right-mouse button, and choose "Edit" to bring up the edit form (another method is to choose "Rename all instances" from the context menu). Name the two variables Rabbits and Foxes respectively, and you will have the following objects on your canvas:

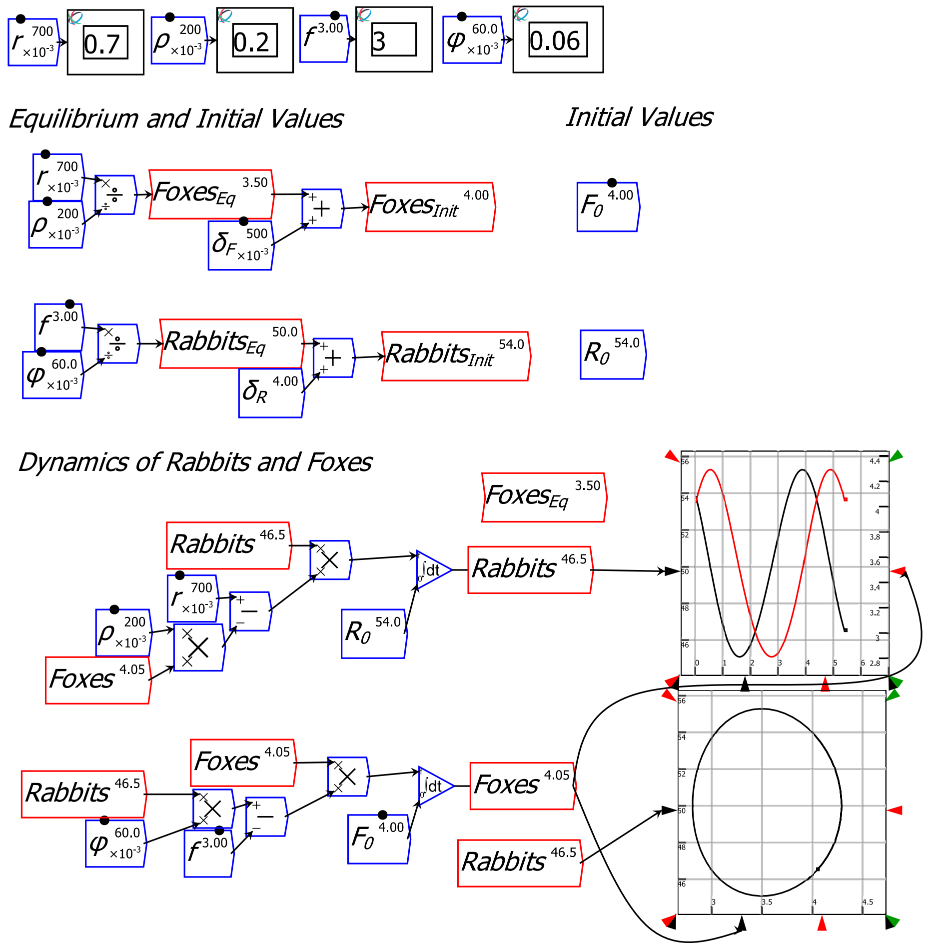
As is typical of dynamic models, the rates of change of the variables Rabbits and Foxes depends on their current values. You therefore need to copy the variable names, which you do using the context menu command "Copy item". Do this and place both on the canvas:



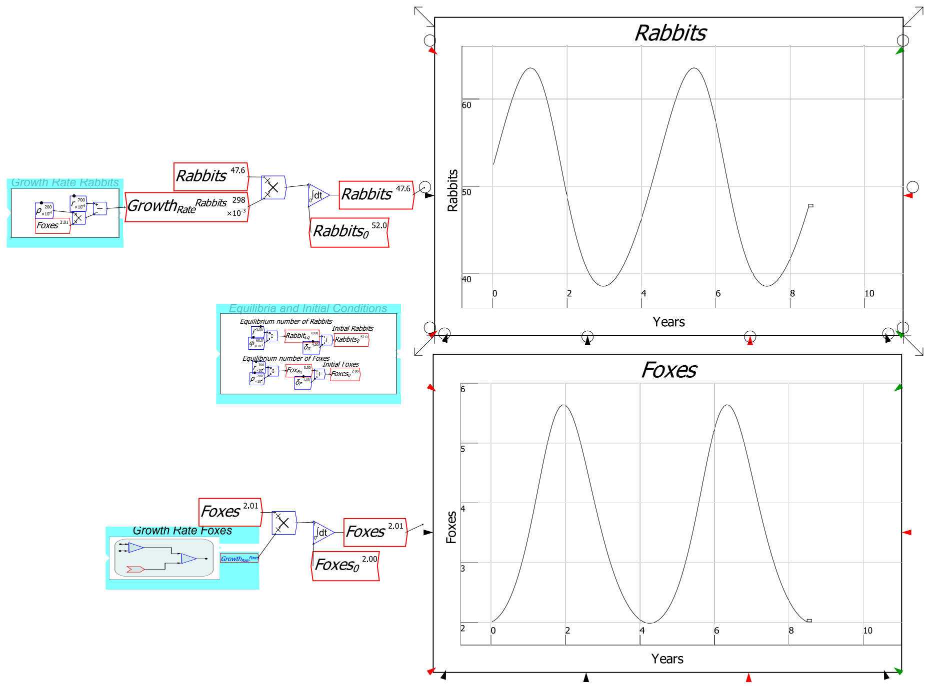
Next add the parameters that determine the rates of growth and death of the two species. The non-zero equilibrium is set by the zeros of $(r - \rho \times Foxes)$ and $(-f + \phi \times Rabbits)$.



Finally, the equations for the rates of growth of the two species are wired up, and the model can be simulated.



If you don't want to display the complete equations on the canvas, you can use grouping to hide the complexity. The next figure illustrates several ways to use groups, and the feature that groups become transparent (and can be edited *in situ*) as the zoom level increases.

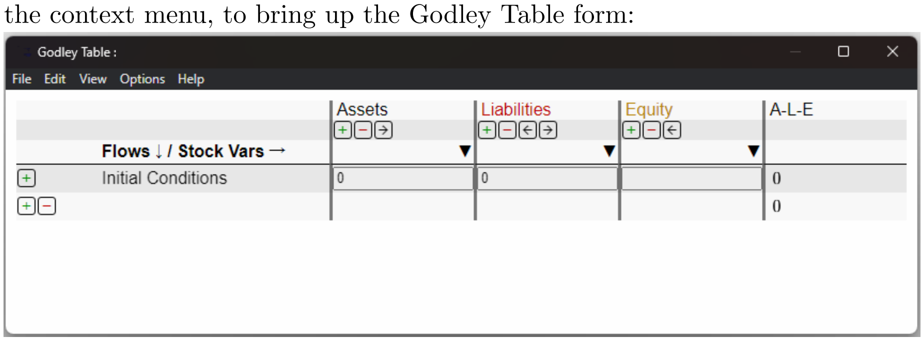


6.2.4 Building monetary models in *Minsky*

Monetary models are actually simpler to build in *Minsky* than block diagram models, because all you have to do is to name the accounts in the Godley Table, and then define the flows on the canvas. *Minsky* automatically generates the differential equations for you.

Building the model starts with placing a Godley Table on the canvas by clicking on the Godley Table operator  . The figure below shows a single Godley Table on the canvas, which has been labelled "Banking Sector" using the context menu "Title" command.

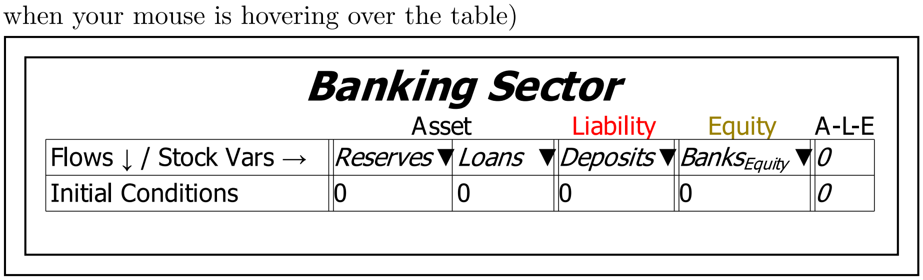
Next either double-click on the icon, or choose "Open Godley Table" from the context menu, to bring up the Godley Table form:



We'll build a model of private money creation in this first example.

6.2.4.1 Modelling private money creation

Add four Stocks to this table: *Reserves* and *Loans* as Assets, *Deposits* as a Liability, and $Banks_{Equity}$ as the Equity of the Banks (type Banks_{Equity} to subscript the word Equity). Go back to the Design canvas and choose "Editor mode" from the context menu, and you will see the table displayed as follows (you will need to resize the table using the Resizing arrows, which are visible when your mouse is hovering over the table)



Banking Sector

	Asset		Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves*	*Loans*	*Deposits*	$Banks_{Equity}$	*0*
Initial Conditions	0	0	0	0	*0*

Now enter numbers into the "Initial Conditions" row. Put 90 in *Reserves*, 10 in *Loans*, 80 in *Deposits*, and 20 in *BanksEquity*.

Banking Sector

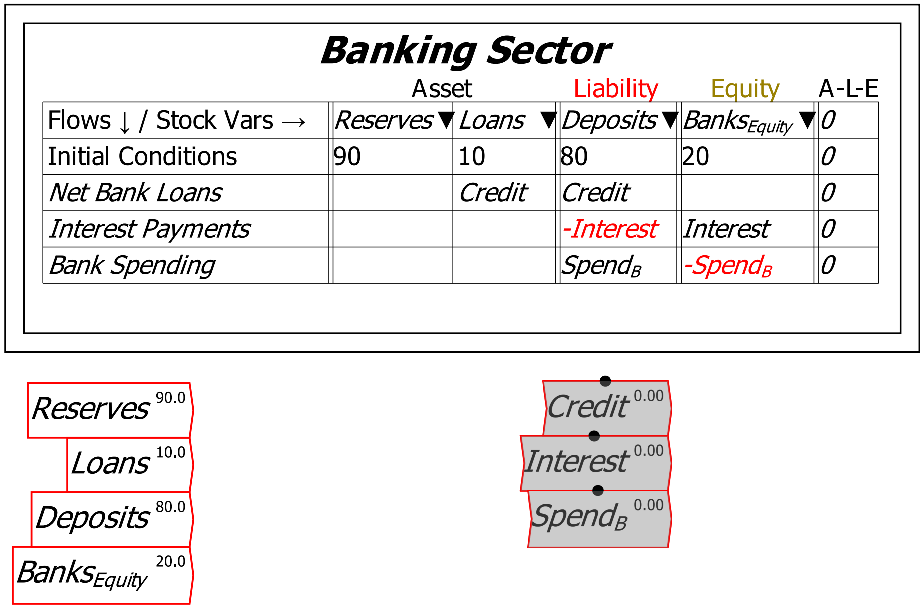
	Asset		Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves*	*Loans*	*Deposits*	*$Banks_{Equity}$*	*0*
Initial Conditions	90	10	80	20	*0*

Then add three flows to the model: Net Bank Loans (which can be negative if people in the aggregate are paying their debts down rather than taking on new debt), Interest Payments, and Bank Spending. Call Net Bank Loans "Credit", Interest Payments "Interest_L" (the subscript is to distinguish interest on loans from interest on bonds, which we'll introduce later), and Bank Spending "Spend_B" (the subscript is to distinguish Bank Spending from Government Spending,which we'll introduce later).

Banking Sector

	Asset		Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves*	*Loans*	*Deposits*	*$Banks_{Equity}$*	*0*
Initial Conditions	90	10	80	20	*0*
Net Bank Loans		*Credit*	*Credit*		*0*
Interest Payments			*-Interest*	*Interest*	*0*
Bank Spending			*$Spend_B$*	*$-Spend_B$*	*0*

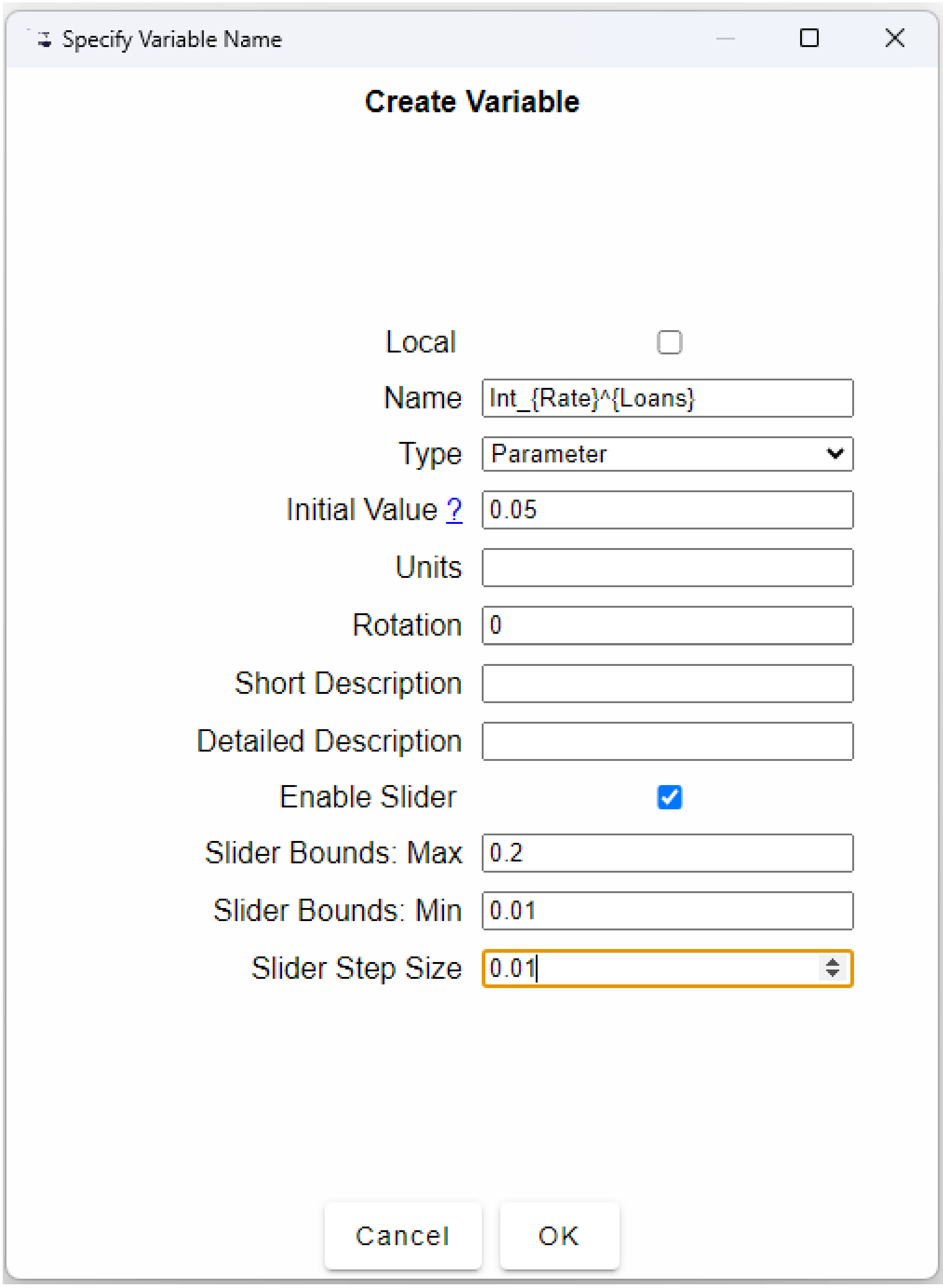
Next, return to the canvas and use the Godley Table context menu commands *Copy flow variables* and *Copy stock variables* to copy all flows and stocks and place them on the canvas, where they can be defined.



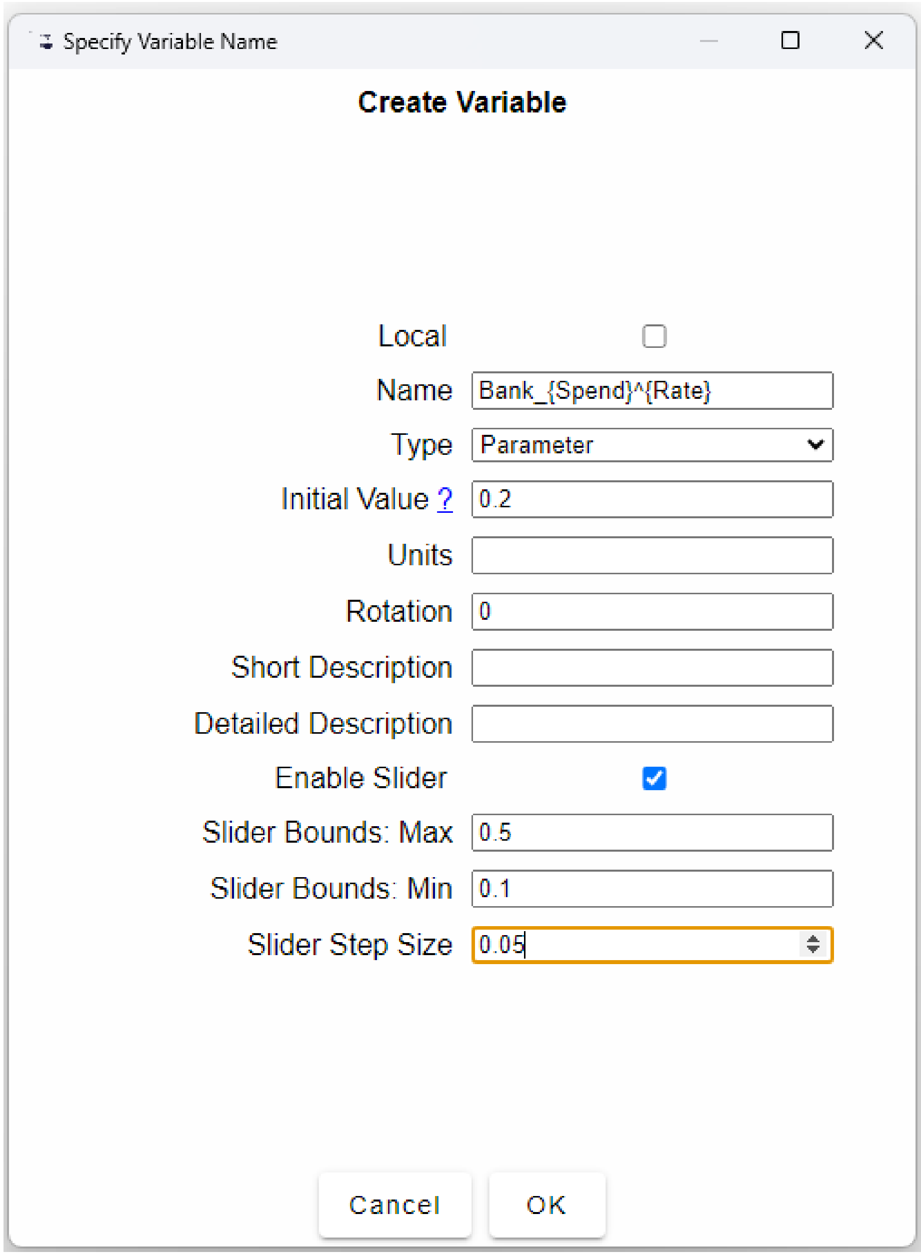
Banking Sector

	Asset		Liability	Equity	A-L-E
Flows ↓ / Stock Vars →	*Reserves*	*Loans*	*Deposits*	*$Banks_{Equity}$*	*0*
Initial Conditions	90	10	80	20	*0*
Net Bank Loans		*Credit*	*Credit*		*0*
Interest Payments			*-Interest*	*Interest*	*0*
Bank Spending			*$Spend_B$*	*$-Spend_B$*	*0*

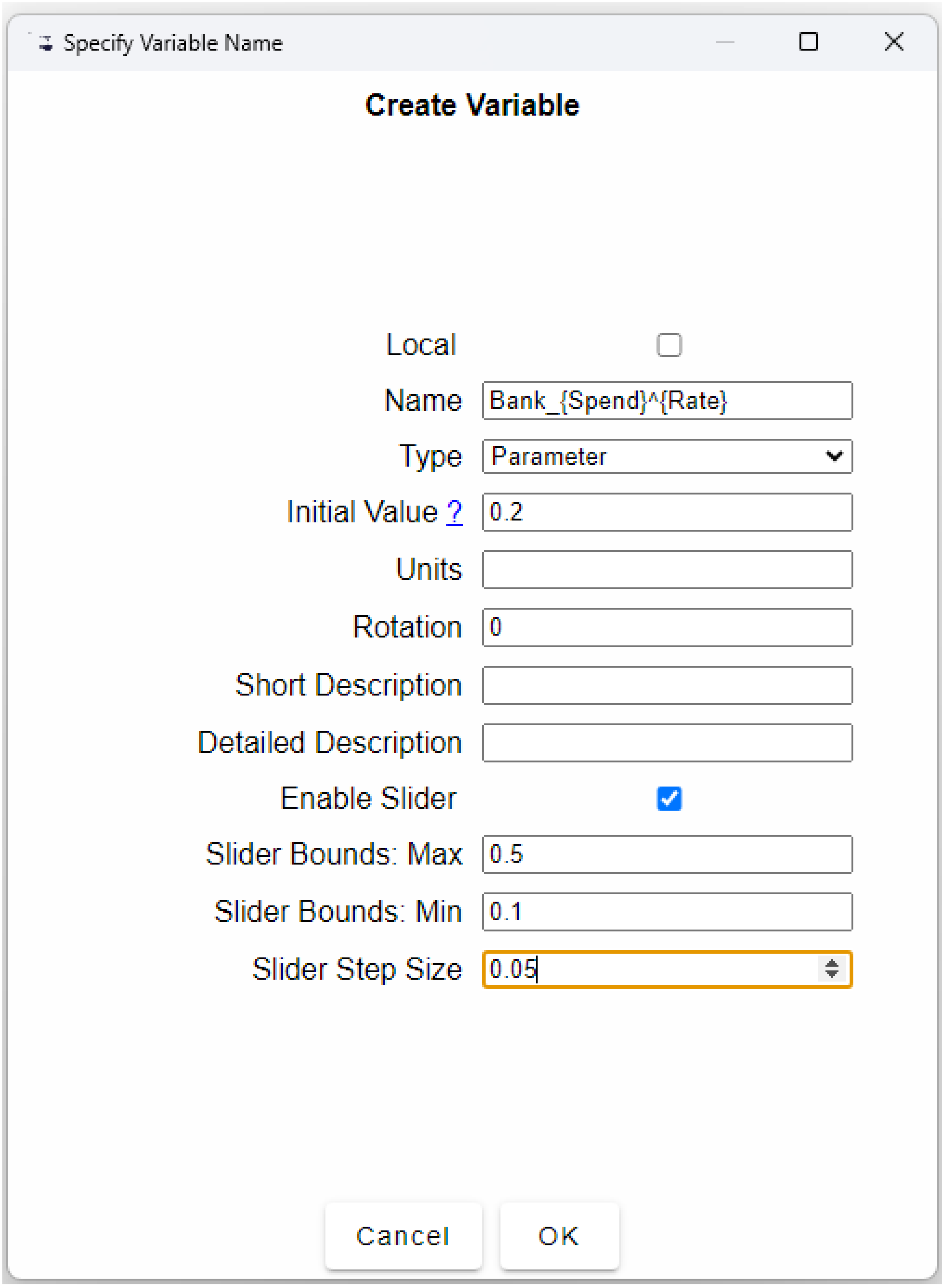
The easiest flow to define is *Interest*, since this is the rate of interest times *Loans*. To define this, first define the parameter $Int_{Rate}{}^{Loans}$: just type Int_{Rate}^{Loans} on the canvas and, once you've finished entering the text, the Variable/Parameter definition form will appear. Put 0.05 as the Initial Value, 0.2 as the *Slide Bounds: Max*, 0.02 as the *Min*, and 0.01 as the *Slider Step Size*.



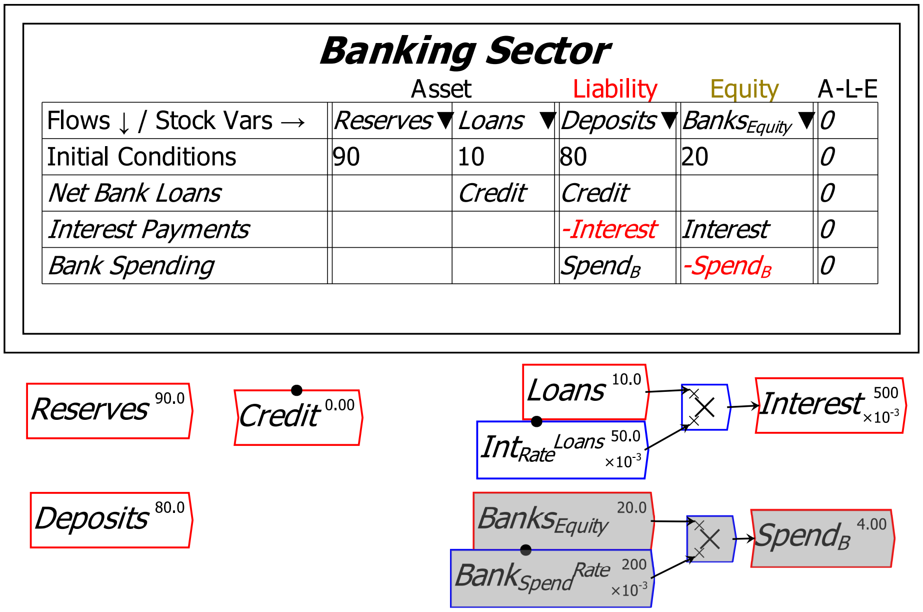
Wire this parameter and *Loans* up to a multiply block and attach it to the flow Interest, and you have defined the first of three flows in this extremely simple model.

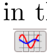


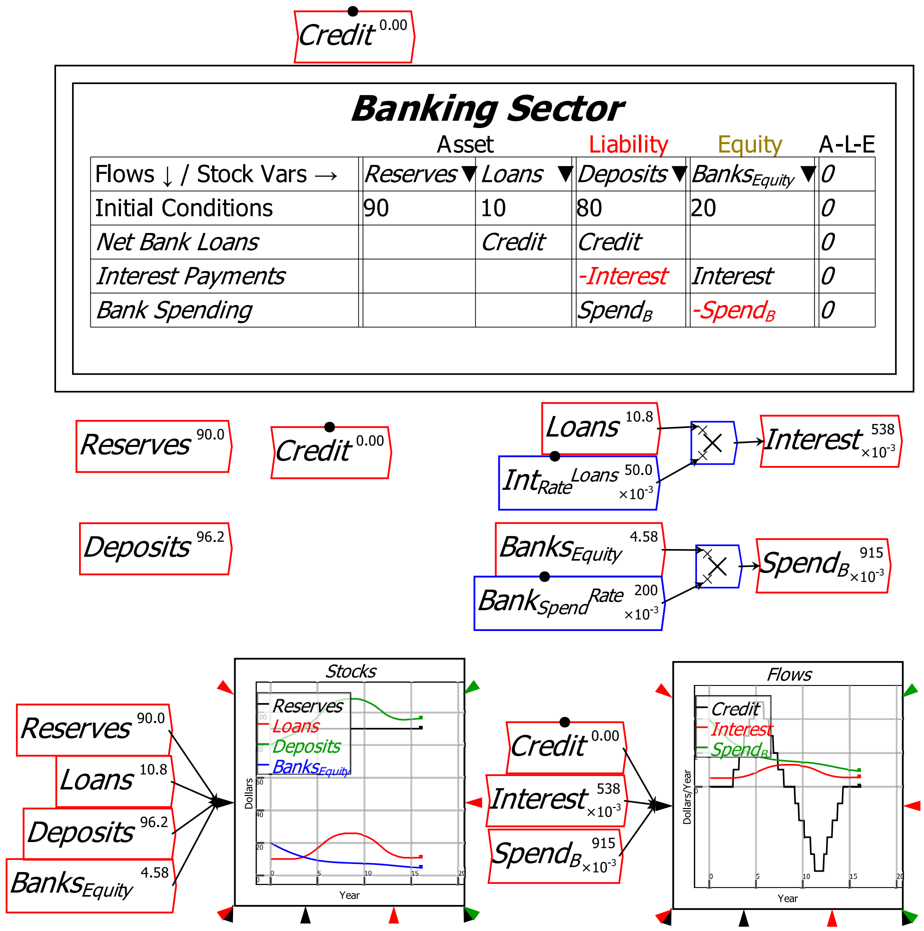
Now define $Spend_B$ as a multiple $Bank_{Spend}{}^{Rate}$ of the $Banks_{Equity}$. Give the parameter a small Initial Value—say 0.2—which means that Banks spend 20% of their equity into the economy every year—this represents paying dividends, wages and bonuses to households, buying goods and services off firms, etc. Set Max to 0.5, Min to 0.1, and the Step Size to 0.05.



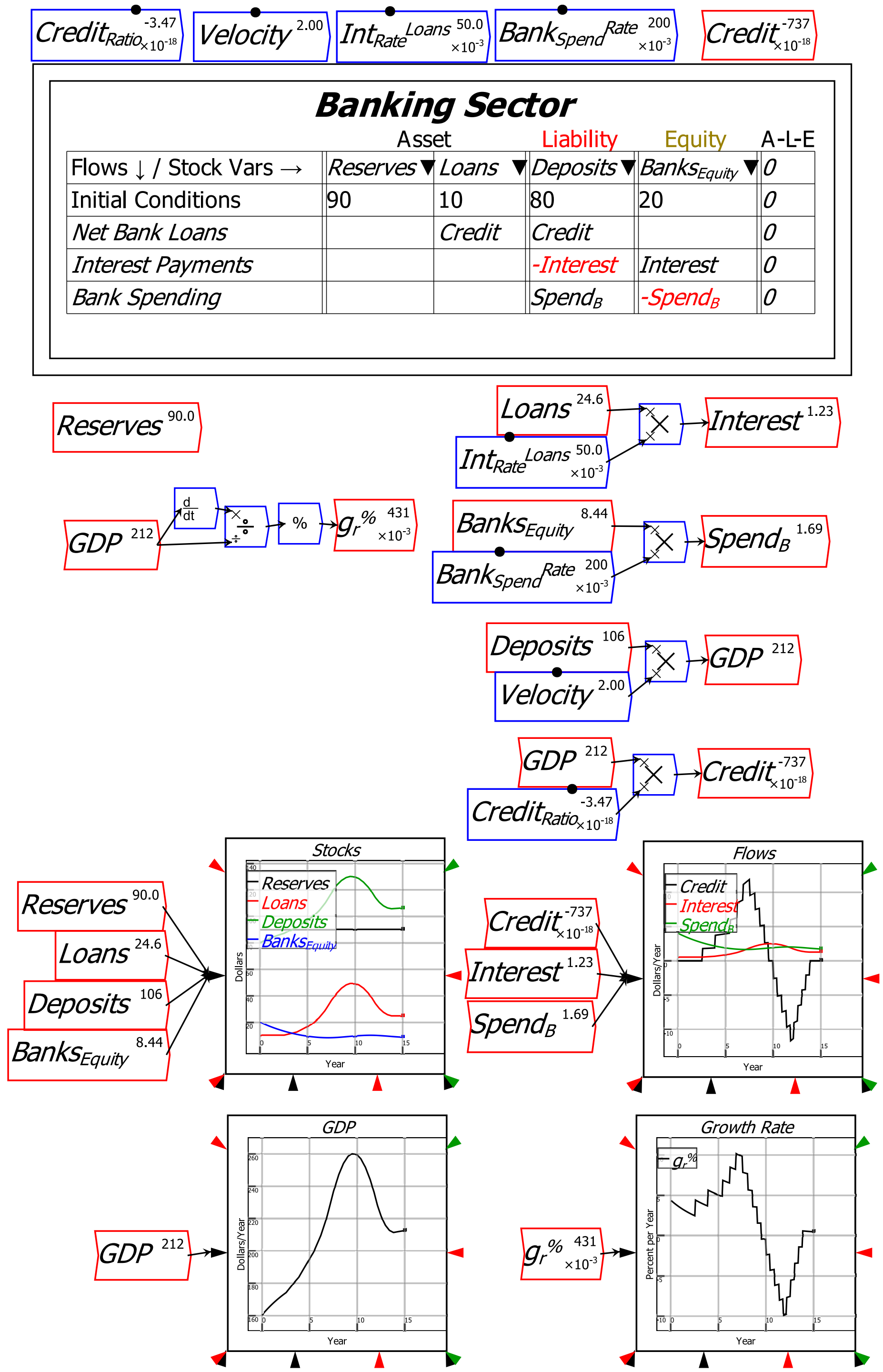
Your canvas should now look like this:



This is sufficient to build a model of just financial stocks and flows, if you edit *Credit* to give it a Max, Min and Step Size in keeping with the other magnitudes in the model–say a Max of 10, Min of -10, and Step Size of 1. Add two Plots  , one for Stocks and one for Flows, attach the Stocks and Flows to their inputs, put a copy of *Credit* at the top of the model for ease of access, click on the Run button ▶, and vary the value of *Credit* to see what happens:



To link this very simple model to economic concepts, make the "Friedmanite" assumption that GDP equals Money times Velocity. Define Money as the sum of Deposits plus $Banks_{Equity}$, and define *Velocity* as a parameter with an Initial Value of 2, Max of 4, Min of 0.5, and Step Size of 0.1. Define a parameter $Credit_{Ratio}$ and give it an Initial Value of 0, Max of 0.1, Min of minus 0.1, and Step Size of 0.01. Also define the percentage growth rate as the differential of GDP divided by itself and put through a percentage operator. Run the model and vary $Credit_{Ratio}$ using the arrow keys as you run a simulation.


Banking Sector
Asset
Liability
Equity
A-L-E
Flows ↓ / Stock Vars →
Reserves
Loans
Deposits
Banks_Equity
0
Initial Conditions
90
10
80
20
0
Net Bank Loans
Credit
Credit
0
Interest Payments
-Interest
Interest
0
Bank Spending
Spend_B
-Spend_B
0
Stocks
Flows
GDP
Growth Rate
Year
Dollars
Dollars/Year
Percent per Year

Much more sophisticated models can be built using *Minsky*, including models that mix the flowchart method with Godley Tables to generate models of both the physical and monetary systems.