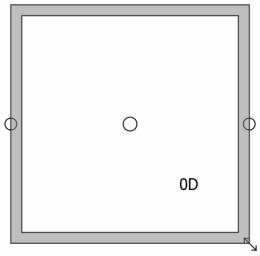
Chapter 1

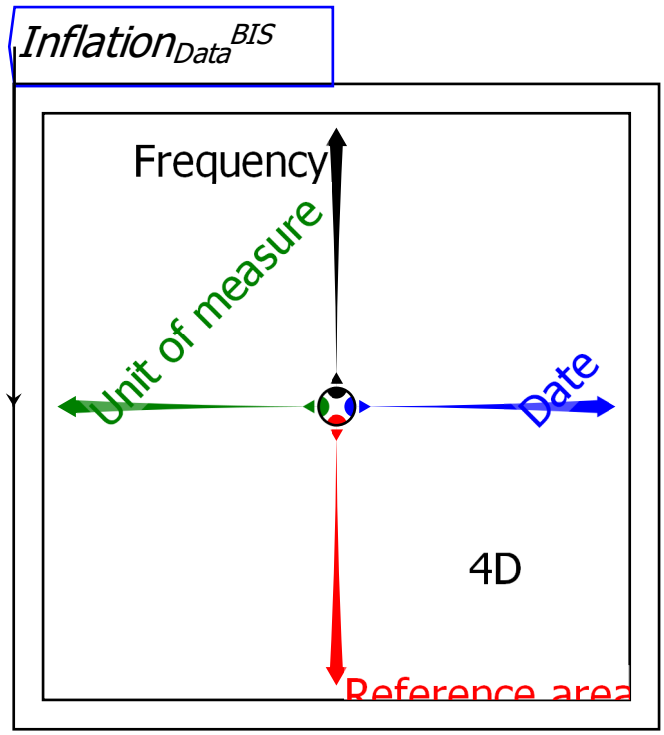
Introduction

Ravel is an intuitive and powerful way to analyse data. Its key feature is the Ravel: a visual tool for manipulating and analysing data.

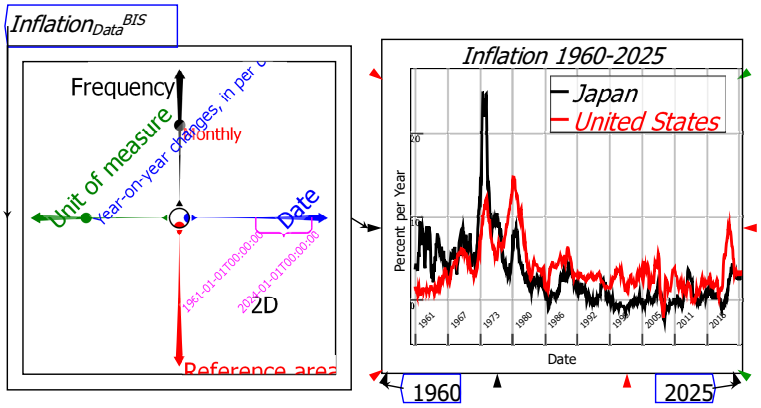
This is a blank Ravel—a Ravel with no data attached to it:



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:

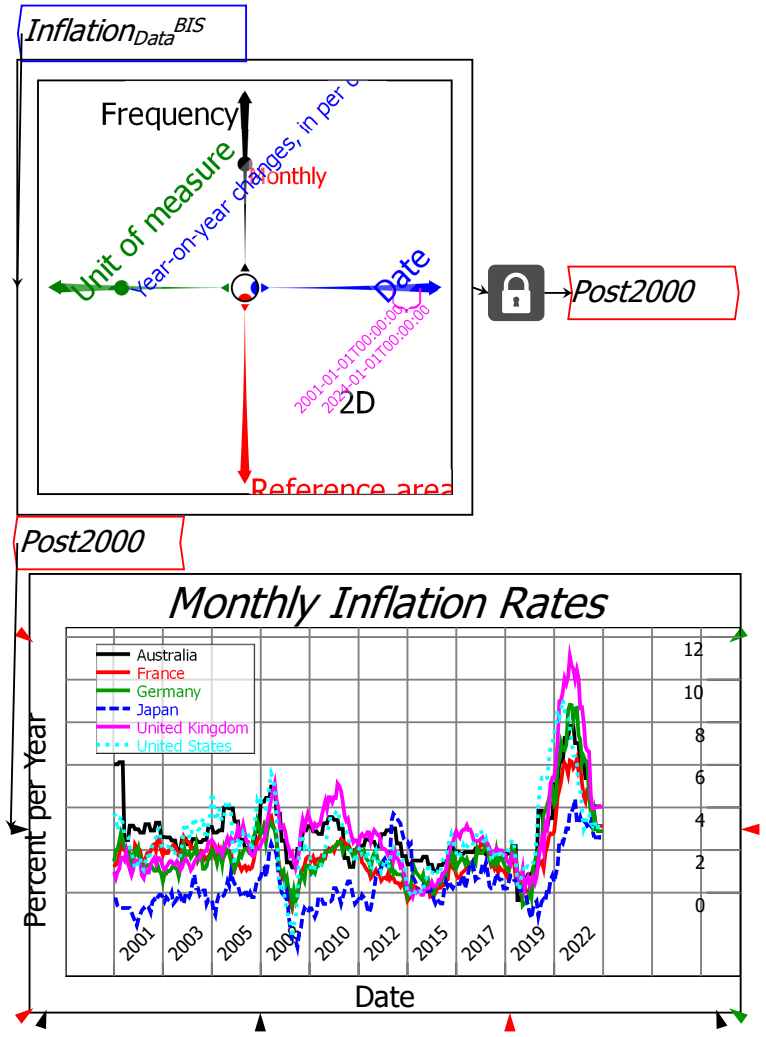


There are many ways to manipulate and display data directly from a Ravel. This is a Ravel with data attached and selected for graphing: the "Year on Year Changes" data is selected from the Unit of Measure axis; two countries (Japan and the United States) are selected from the Reference area axis; Monthly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



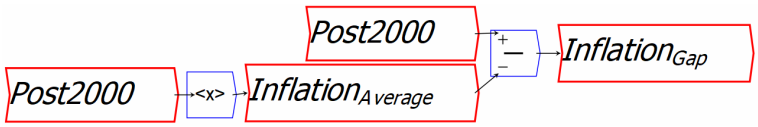
Ravel the object itself makes it far easier to drill down into and visualise data than using either a spreadsheet, or the Pivot Tables that standard Business Intelligence program use.

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").



Finally, 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.

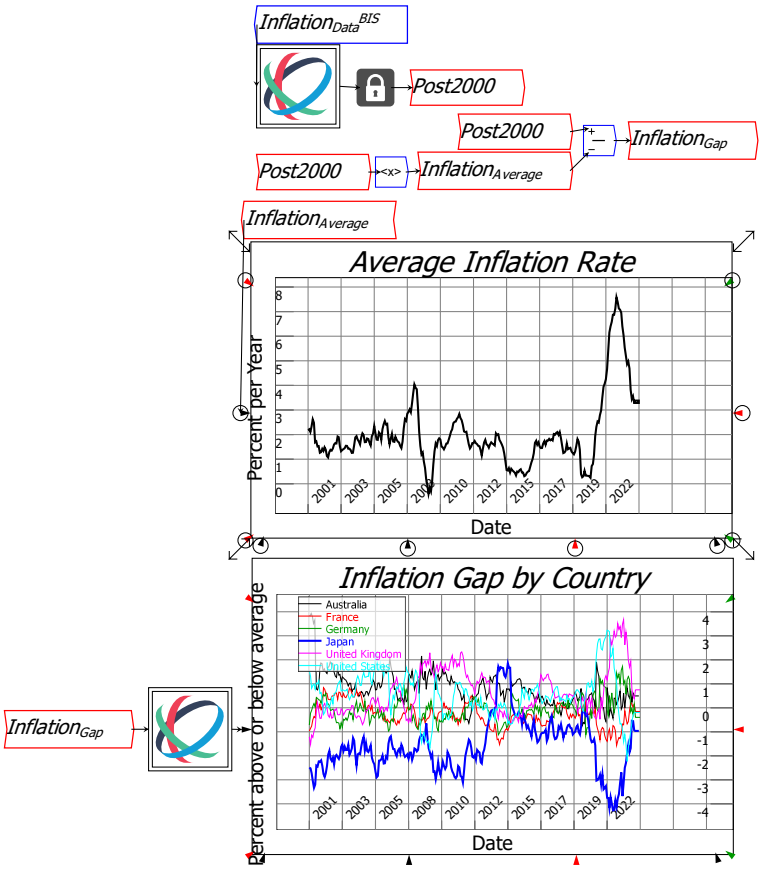
The average inflation rate was calculated using the formula:



¿

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.

The final example is the comparison of average inflation outcomes for the six countries, and the deviation of them from the average. 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 interest 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.

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

Chapter 2

Getting Started with Ravel

Ravel has three main components:

- The Ravel, a visual representation of your data;
- Equations expressed as flowcharts; and
- The system-dynamics engine Minsky

2.1 System requirements

Ravel is a proprietary program for data analysis which currently only runs on Windows. It is built on top of Minsky, which is an open source program available for Windows, Mac OS X,and various Linux distributions. 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 Minsky, click here.

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 Components of the Program

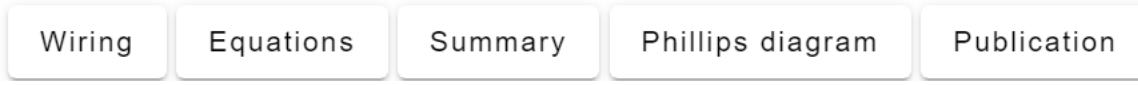
There are 5 main components to the Ravel interface:

1. The menus.

   File Edit Bookmarks Insert Options Simulation Help

2. The Operations controls

3. The Tabs.

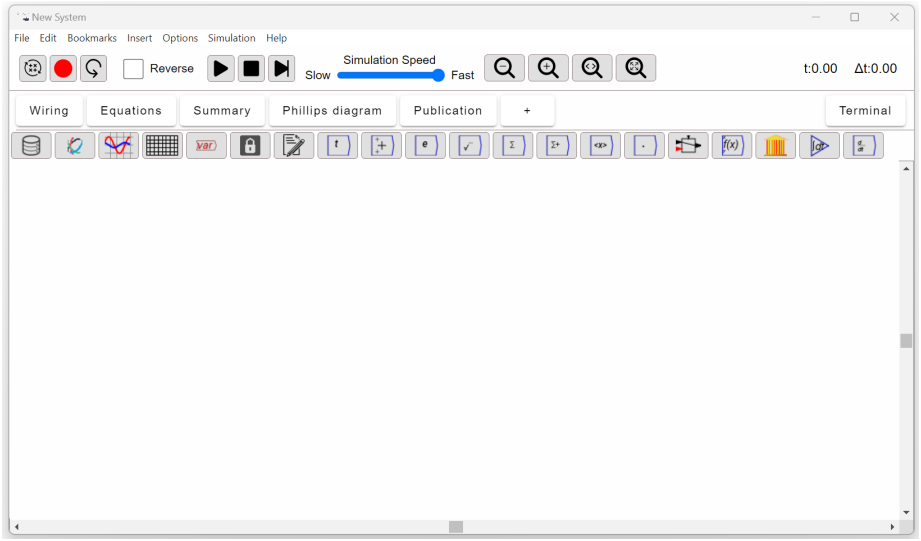


These are

- Wiring, where a model is defined;
- Equations, which shows you a bitmapped image of all the equations in your model (to see well-formatted equations, use the "File/Export Canvas As" menu, choose LaTeX as the output format, and load the resulting .TeX file into a LaTeX editor);
- Summary, which provides a detailed and mathematically formatted table of all the components of your model;
- "Phillips Diagram", which is a Minsky-specific feature not used by Ravel;
- "Publication", which is a default blank canvas onto which any component of your model can be placed for documentation purposes; and
- The "+" tab, which enables you to create and name a new Publication tab. Any number of Publication tabs can be created, which lets you save multiple views of a file for different audiences–the one for auditing, say, would be different to the one for marketing, and so on.

4. The design icons. These are "widgets" which are placed on the canvas to perform various operations. These include data import parameters, Ravel objects, plots and sheets for displaying data, mathematical operators for analysing your data, and so on. These are explained in detail in the reference sections of the online help.

5. And finally, the Design Canvas–the large drawing area beneath the buttons and icons.



Some of the elements of the interface are specific to the simulation engine Minsky–for example, the Record and Replay buttons, the Reverse checkbox, "Run-Stop-Step" buttons and the simulation speed slider. These aspects of the interface are explained in the Minsky section of this manual.

Others are used by both Ravel and Minsky–for example, the Recalculate button:

And the Zoom controls, which enable you to Zoom out and In, go to standard Zoom, and fit the document to the visible screen:

2.3.1 Menu

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

File Edit Bookmarks Insert Options Simulation Help

File

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

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").

Recent Files Provides a shortcut to some of your previously opened Ravel (or Minsky) files.

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 This is a Minsky-specific command to check whether definitions in a simulation model are consistent

Export Canvas as Export the current canvas in svg, pdf, eps, tex, or m format. 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.
- 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, 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[1] or Octave[2].

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.

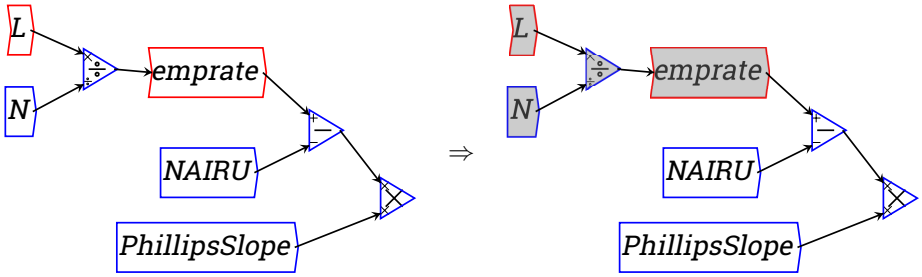
Redraw Redraw may be useful if the screen gets messed up because of a display bug (all programs have them!). For example, a bug could cause items on the canvas to be scaled differently. Redraw could overcome this problem without requiring you to exit the program.

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.

[1] https://en.wikipedia.org/wiki/MATLAB
[2] http://www.gnu.org/software/octave/



- 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 these detail of these operations from the top level view. This reduces the complexity of a canvas, which can make it easier for a viewer to focus on the analysis that the program is actually doing.

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, 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.

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 Minsky-specific tools like Godley table items and Plots.

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 Getting-Started-Minsky

Preferences

- Number of recent files to display — this determines how many previously edited files are displayed on the recentfiles 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 from which a colour scheme is computed.

Simulation

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

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

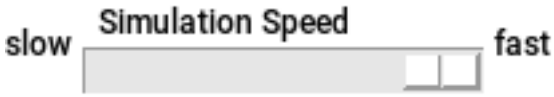
2.3.3 Recalculate button

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

2.3.4 Run Buttons

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

2.3.5 Speed slider



The speed slider controls the rate at which a model is simulated. These commands are specific to Minsky and are covered in the Minsky component of this manual.

2.3.6 Zoom buttons

The Zoom buttons 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. It can also be used to recentre the equation view.

The Zoom to Fit button zooms the model so that it just fits in the current canvas window.

2.3.7 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. This information is specific to a Minsky simulation model.

2.3.8 Wiring and Equations Tabs

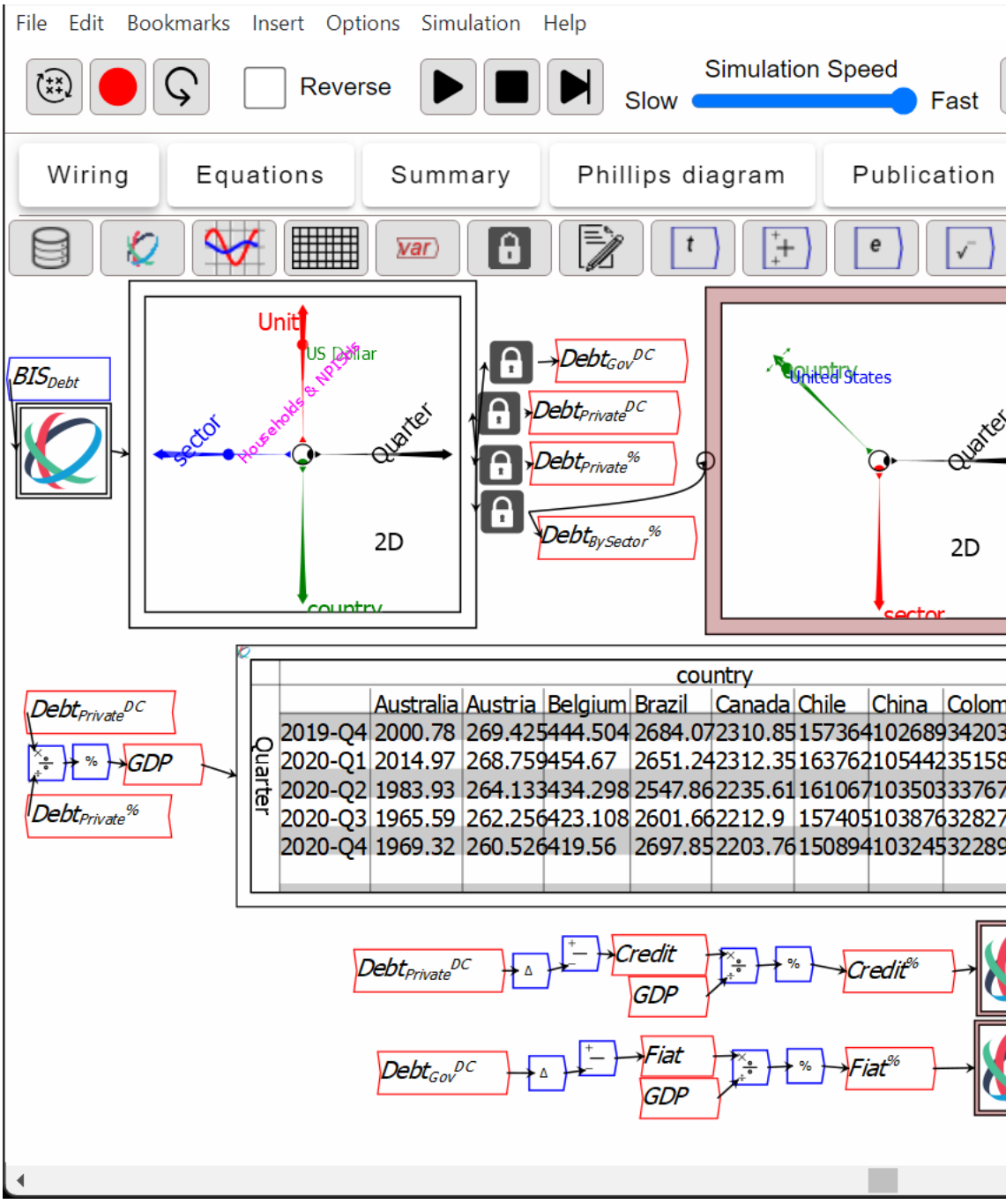
This allows you to switch between the visual block diagram wiring view and other views of your document.

Wiring This is Ravel's design canvas, where you import data, analyse it using Ravel's mathematical operators and functions, and produce visualisations using Plots and Sheets.

Equations As you analyse a model using the flowchart operators in Ravel, the program converts the flowchart logic into standard mathematical formulas. This Tab gives you an instant view of the mathematical logic in your model.

Summary This tab provides a structured view of all the equations in a model, with details about the number of dimensions and values in a formula.

For example, this model 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
$:Credit$	$-\left[\Delta\left(\text{Debt}_{\text{Private}^{\text{DC}}}\right)_i\right])$	319,48
$:Credit^{\%}$	$\left(\frac{:\text{Credit}}{\text{GDP}}\right)\%$	319,45
$:Debt^{\%}_{BySector}$	*locked*	323,3,43
$:Debt^{DC}_{Gov}$	*locked*	323,43
$Debt^{\%}_{Private}$	*locked*	323,4
$Debt^{DC}_{Private}$	*locked*	323,48
$:Fiat$	$-\left[\Delta\left(:\text{Debt}_{\text{Gov}^{\text{DC}}}\right)_i\right])$	319,43
$:Fiat^{\%}$	$\left(\frac{:\text{Fiat}}{\text{GDP}}\right)\%$	319,45
$:GDP$	$\left(\frac{\text{Debt}_{\text{Private}^{\text{DC}}}}{\text{Debt}_{\text{Private}^{\%}}}\right)\%$	323,45

Phillips Diagram This is a Minsky-specific feature which is covered in the Minsky section of this manual.

Publication Publication Tabs allow you to place specific items from the Wiring diagram onto a documentation canvas. The default Tab is called "Publication", and additional user-named Tabs can be added using the "+" Tab.

To place an item on a Publication Tab, right-click on it on the Wiring Tab, and then click on "Add item to a publication tab". Click on the desired named tab from the resulting menu. The item will then be placed in the top-left-hand corner of the selected Tab. You can then place it wherever you want on the Publication Tab.

\+ This creates a new tab. Provide a name in the form and press Enter or click OK and the new Tab will be created.

2.3.9 Design Icons

These are the "nuts and bolts" of data analysis using Ravel, and model-building using Minsky. There is a substantial number of icons, and this number will grow over time as more data analysis features are added.

The essential icons for Ravel are the first two: Import Data and insert a Ravel .

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. Subsequent Ravels are displayed in icon mode. For full details on using a Ravel see Ravel.

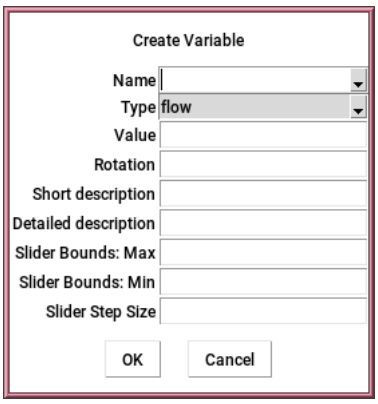
Plot widget Add plots to the canvas.

Sheet widget Add a sheet to the canvas.

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:



The only essential step here is providing a name for the Variable.

Ravel supports the LaTeX language for naming variables (and parameters), which enables you to:

- Use subscripts and superscripts. The character following an underscore character _ is subscripted, while the character following a caret

symbol ^ is superscripted. If the _ or ^ is followed by text in parentheses (curly brackets) {}, then all the text within the parentheses is superscripted or subscripted; and

- Use Greek characters. The English name for a Greek letter (alpha, beta, gamma), preceded by a backslash character \ generates the equivalent Greek letter as part of a variable's name (α, β, γ);

This enables more readable variables to be defined, such as, for example $\Delta Sales^{State}_{Product}$.

You can also enter a value for it (and a rotation in degrees), but these can be omitted. In a dynamic model, the value will be generated by the model itself, provided its input is wired.

When you click on OK (or press Enter), the newly named variable will appear in the top left hand corner of the Canvas. Move the mouse cursor to where you want to place the variable on the Canvas, click, and it will be placed in that location.

Constants are entities whose value is unaffected by the simulation or other entities in the model. Click on it and a constant definition window will appear:

The only essential element here is its value. You can also specify its rotation on the Canvas in degrees. You can vary the value of a constant or parameter using the arrow keys or the slider button on top of a constant or parameter.

A constant is just a type of variable, which also include parameters (named constants), flow variables, stock variables and integration variables. In fact there is no real conceptual difference between creating a constant or creating a variable, as you can switch the type using the type field.

Like the variable and constant button, the parameter button creates a variable defaulting to the parameter type. Parameters differ from flow

variables in not having an input port, and differ from constants in having a name and being controllable by a slider during simulation.

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.

Notes Add textual annotations

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

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: multiple inputs to the input ports on the subtract operator are added together, and then the sum of inputs attached to the "-" port is subtracted from the sum of the inputs attached to the "+" port. For the divide block, the product of the inputs to the "*" port is divided by the product of the inputs to the "/" port.

Min & Max Functions. These take the minimum and maximum values, respectively. These also allow multiple wires per input. The sum of the inputs to one port is compared to the sum of the inputs on the other.

Pow and log. These are binary operations (taking just two arguments). In the case of the power operation, the exponent is the top port, and the argument to be raised to that exponent is the bottom port. This is indicated by the x and y labels on the ports. In the case of logarithm, the bottom port (labelled b) is the base of the logarithm.

Logical Operators $<$ $\leq$, $=$, $\wedge$ $\vee$ $\neg$ (and, or, not)] These return 0 for false and 1 for true.

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

Reduction operations This menu contains operations that reduce a vector to a scalar, or reduce the rank of a tensor. Typically sum, product, any, all etc.

Scans These are running sums and the difference operator

Miscellaneous tensor operations Any other tensor function not covered elsewhere.

Switch Add a piecewise-defined function block to the canvas. Also known as a hybrid function.

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

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.

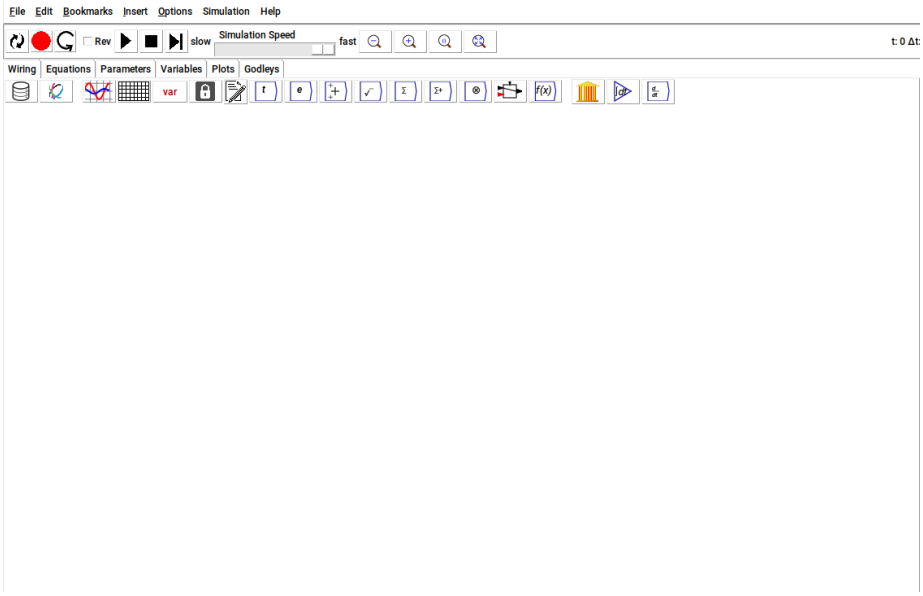
Integration . This inserts a variable whose value depends on the integral of other variables in the system. It is discussed further in the Minsky section of the manual.

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

2.3.10 Design Canvas

The Design Canvas is where you develop your model. 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 a 10000 pixels in size.



2.3.11 Equations tab

This displays the mathematical representation of the model

2.3.12 Summary Tab

This tab provides a summary table of all variables in the system, in a heirarchical fashion that can be navigated by expanding or hiding sections by toggling the caret. Each variable shows its name, it definition, dimensions (for tensor-valued variables), initial expression, units and current value.

Most of these fields are editable, and usually do the obvious thing. 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.

2.3.13 Phillips diagram tab

This tab implement's Minsky's take on a Phillips diagram showing the stocks and flows in a monetary economy. The Phillips tab shows all the information contained in the Godley tables of the model - if there aren't any, this tab will be blank.

Initially, all the stocks will be arranged around a circle, with the flows shown as connecting arrows showing the current direction of the flow. You can move and rotate the stocks, and bend the flows to make a pleasing layout. The stocks

will be coloured as though filled with a fluid like Bill Phillip's original analogue computer, and dynamically updated as the simulation proceeds.

2.3.14 Publication tab

Publication tabs allow the creation of dashboards to emphasise certain aspects of a simulation. For example, you may wish to focus on a particular plot or Godley table when running the simulation.

Multiple publication tabs can be created by clicking the '+' tab.

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. For items that dynamically update, the publication tab will be updated dynamically during the simulation.

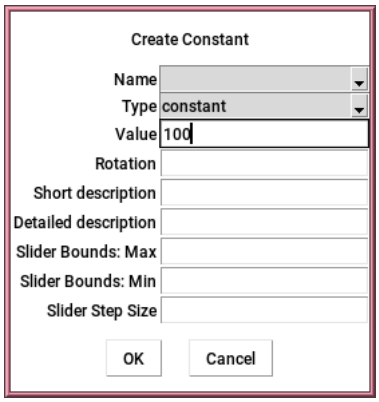
Textual annotation can be added to the publication tab, independently of any annotations on the wiring tab.

Wires cannot be added to the publication tab, however you can insert arrows (eg →) by typing the LaTeX text `\rightarrow`, and then rotate and scale the arrow to connect parts of the dashboard together.

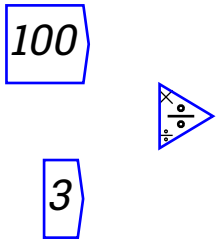
2.3.15 Wires

The wires in a model connect blocks together to define equations. For example, to write an equation for 100/33, you would place a 

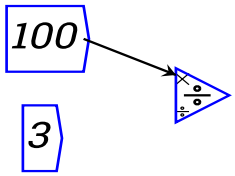
on the canvas, and give it the value of 100:



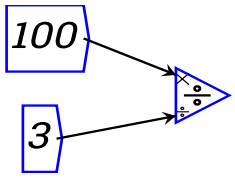
Then do the same for 33, and place a divide block on the canvas:



Then click on the right hand edge of 100 and drag to extend the wire to the numerator (×) port of the divide operation.



Finally, add the other wire.



2.4 Working with Ravel

2.4.1 Components in Ravel

There are several types of components in Ravel

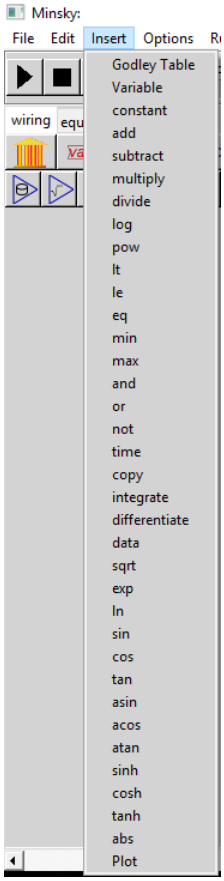
1. Data storage parameters, which load in data from an external CSV file;
2. Ravels, which take data from a data storage parameter and create a multi-dimensional graphical rendition of the data, with one axis per dimension;
3. Mathematical operators such as plus (+), minus (-), etc. These are all aware of the dimensions of your data, so they act on arrays of data, rather than single cells as with spreadsheet formulas;
4. Constants (or parameters, which are named constants) which are given a value by the user;
5. Variables whose values are calculated by the program during a simulation and depend on the values of constants and other variables; and
6. Groups, which allow components to be grouped into modules that can be used to construct more complex models.

2.4.2 Inserting a model component

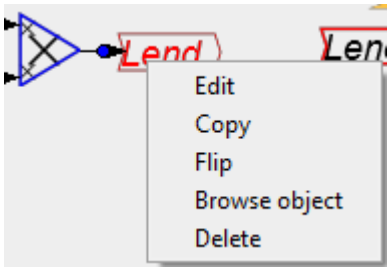
There are five ways to insert a component onto the Canvas:

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



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.

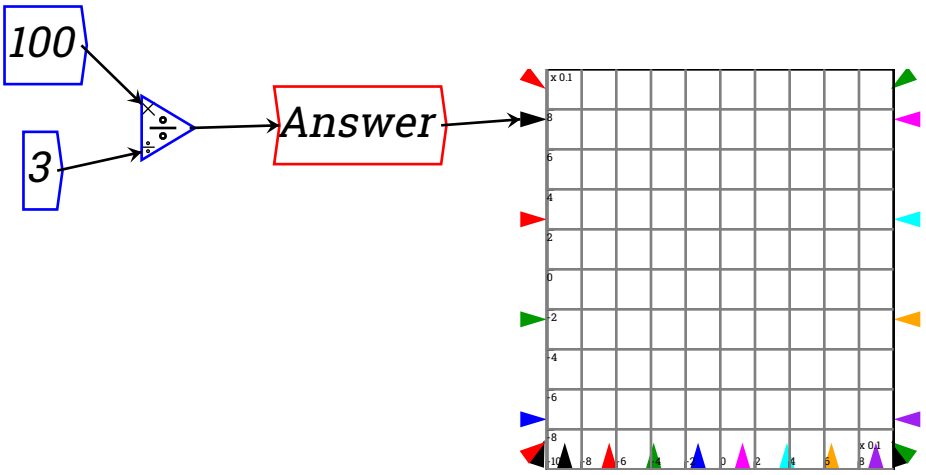
2.4.3 Creating an equation

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

$$100/3 = 33.3$$

is performed in Minsky by defining a constant block with a value of 100, defining another with a value of 3, and wiring them up to a divide-by block. Then attach the output of the divide block to a variable, and run the model by clicking on  :



If you click on the equation tab, you will see that it is:

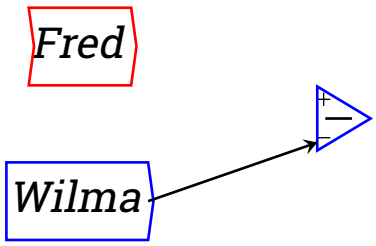
$$\text{Answer} = \frac{100}{3}$$

2.4.4 Wiring components together

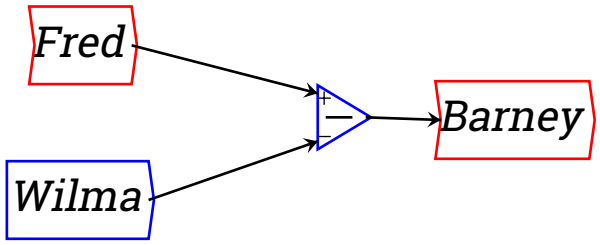
A model is constructed by wiring one component to another in a way that defines an equation. Wires are drawn from the output port of one block to the input port of another. Ports are circles on the blocks to which wires can be attached, which can be seen when hovering the pointer over the block. Variables 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.

To construct an equation, such as Fred - Wilma = Barney:

Click the mouse near the output port of one block and drag the cursor to the input port of another while holding the mouse button down. An arrow extends out from the output port. Release the mouse button near the required input port of the operator. A connection will be made.



The equation is completed by wiring up the other components in the same way.


Fred
Wilma
+
−
−
Barney

Chapter 3

A Ravel Tutorial

3.1 Slicing, Dicing and Rotating Data

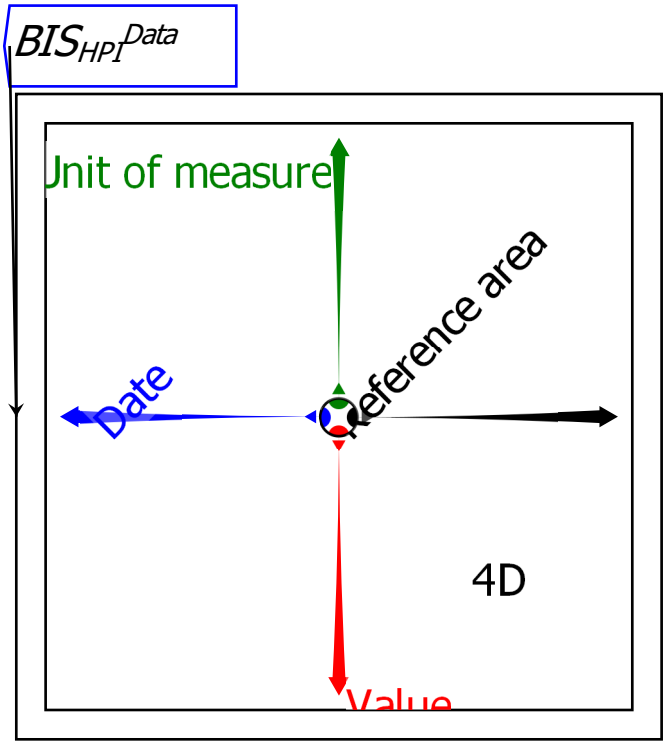
A Ravel is a graphical representation of multi-dimensional data. Unlike a spreadsheet, which has only two dimensions, a Ravel can have as many dimensions as your data. You manipulate the axes of a Ravel to select the components of your data that you wish to see, and those components are then the output 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

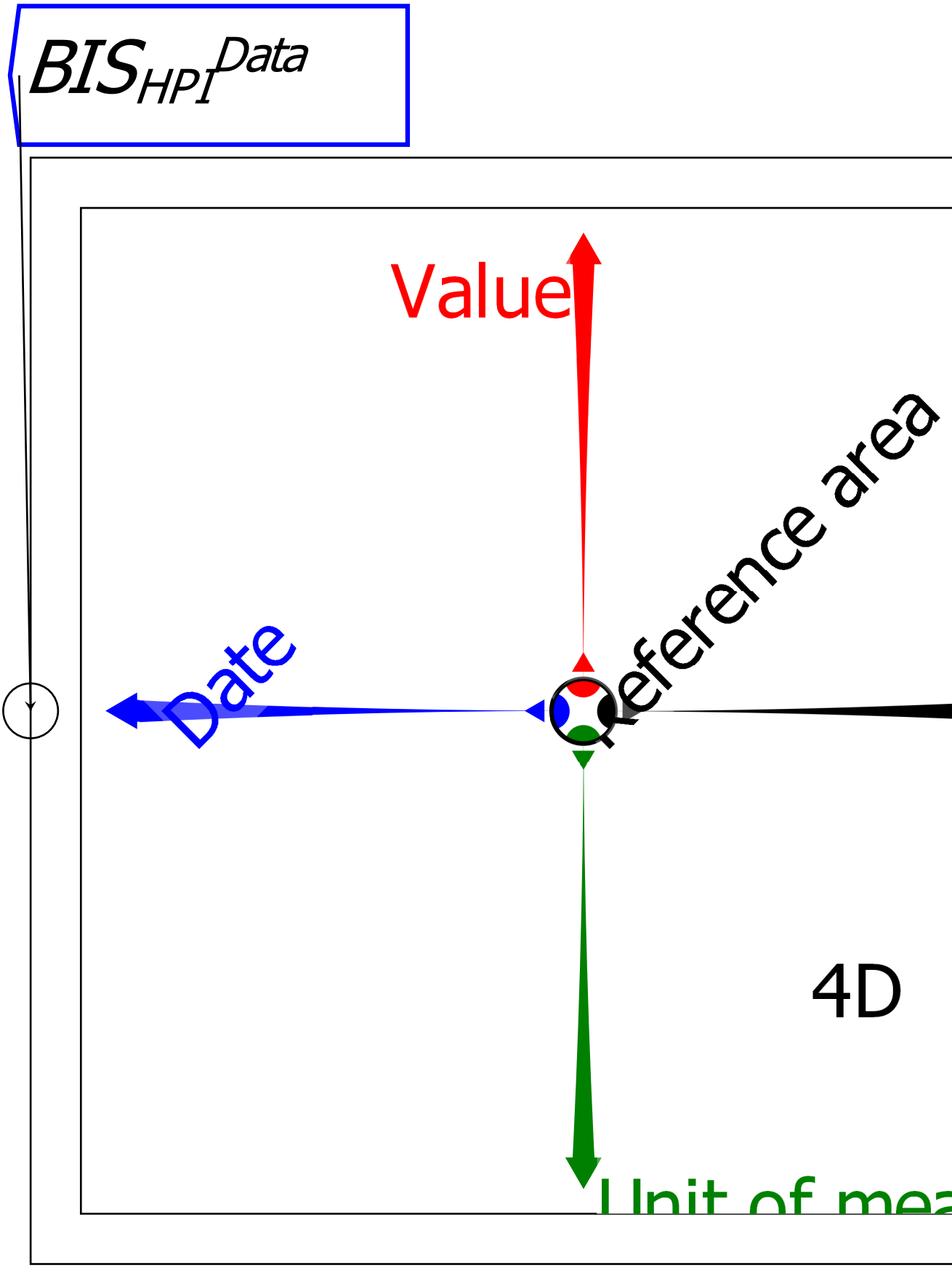
Unit of Measure House Price Index where 2010 = 100; 2 entries

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

Reference Area Country or Region: 62 entries

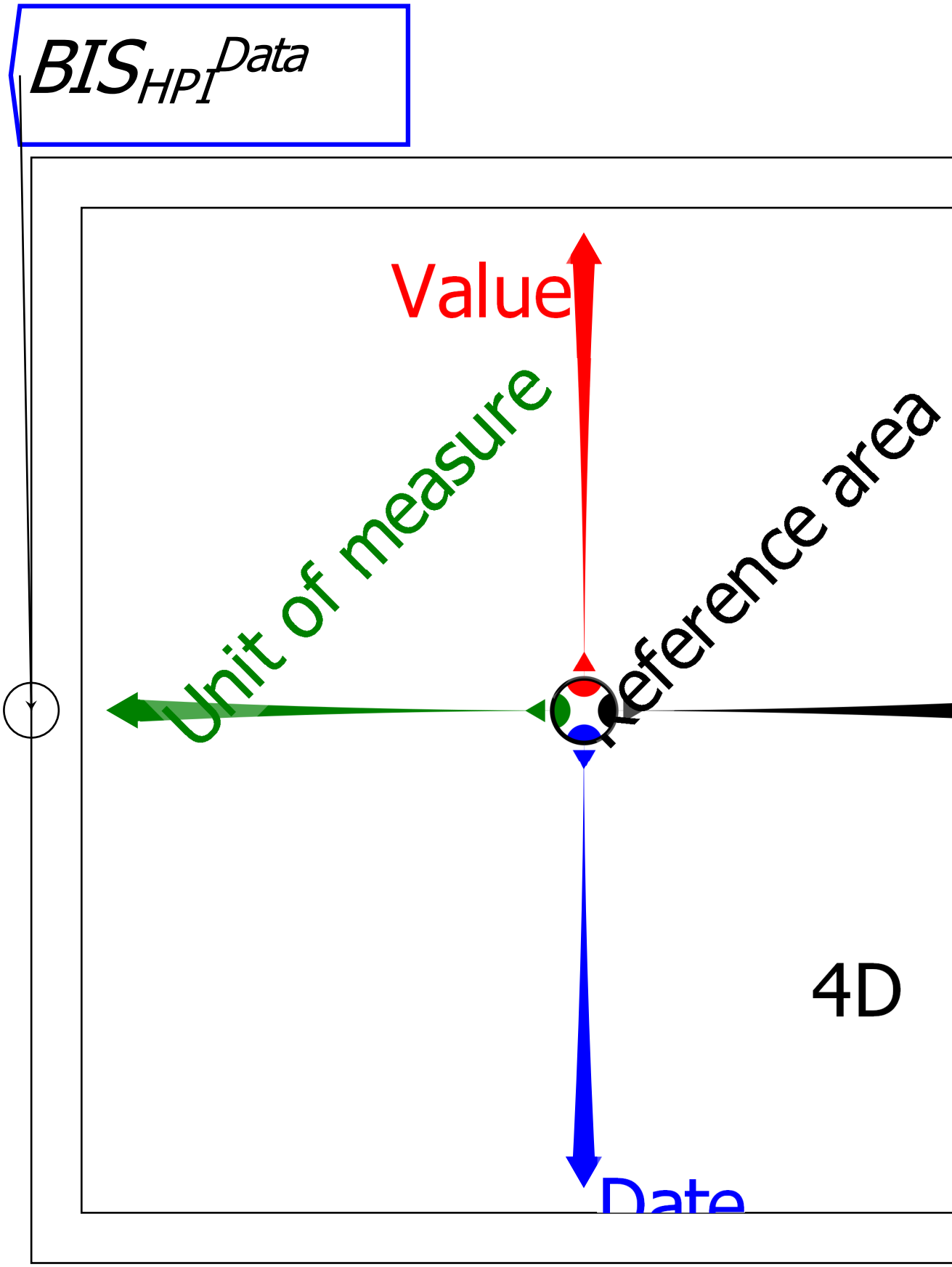


When a Ravel is first attached to data, it outputs the entire data set—which is indicated by the dimension count *4D* in the lower right quadrant of this Ravel. You can see this by attaching a Sheet to the output port of the Ravel:


BIS_HPI Data
Value
Reference area
Date
4D
Unit of mea

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.

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.


BIS_HPI^Data
Value
Unit of measure
Reference area
4D
Date

The Sheet is still blank because there is no data for the current selections–the 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.

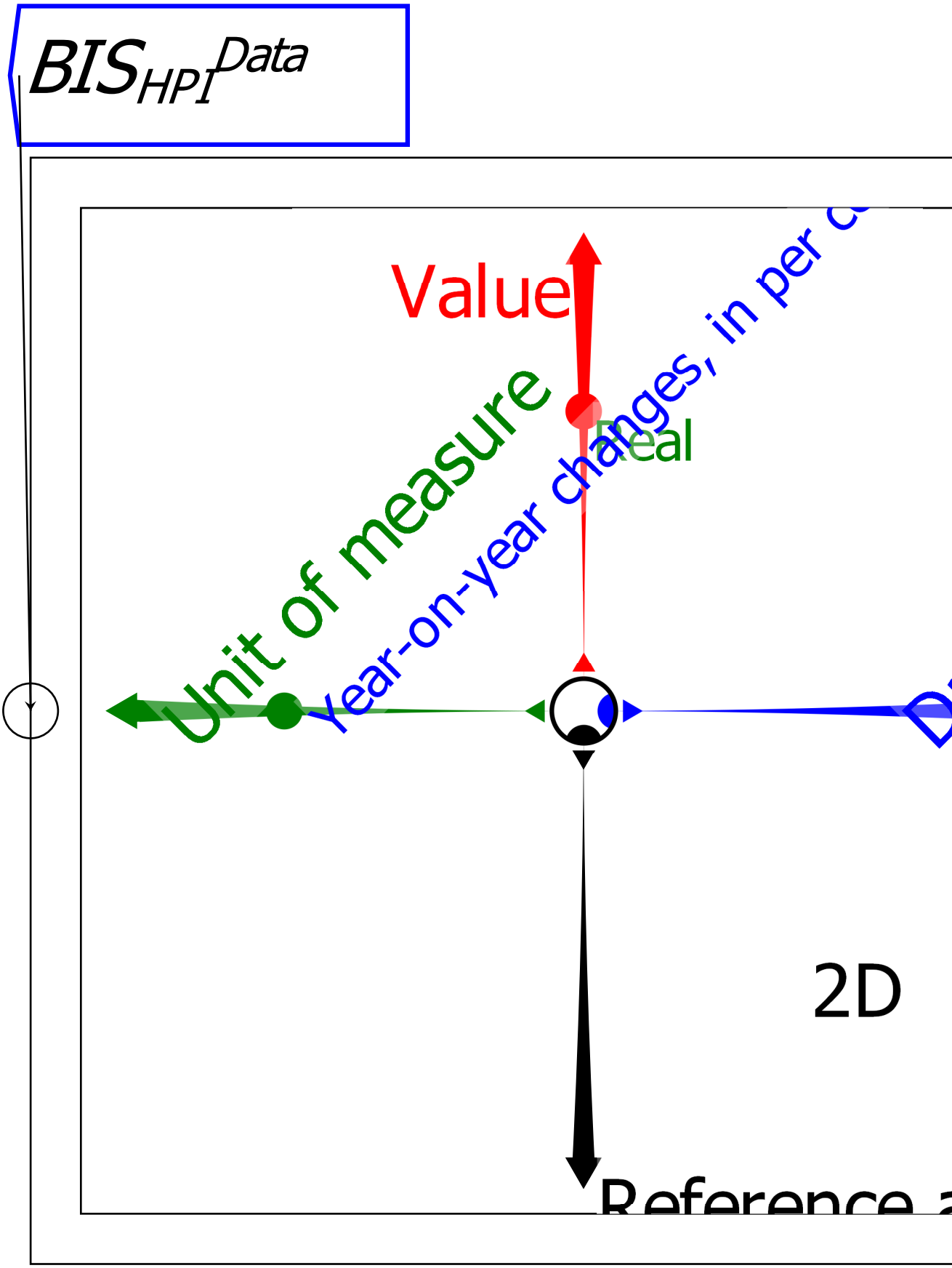
To see data immediately, you can take advantage of a feature of the Sheet: it can display the first few rows and columns of data (the default setting), which we call the Head, the last few (the Tail), or a few of both (Head and Tail). To show the last few rows and columns, right-click on the Sheet and 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.

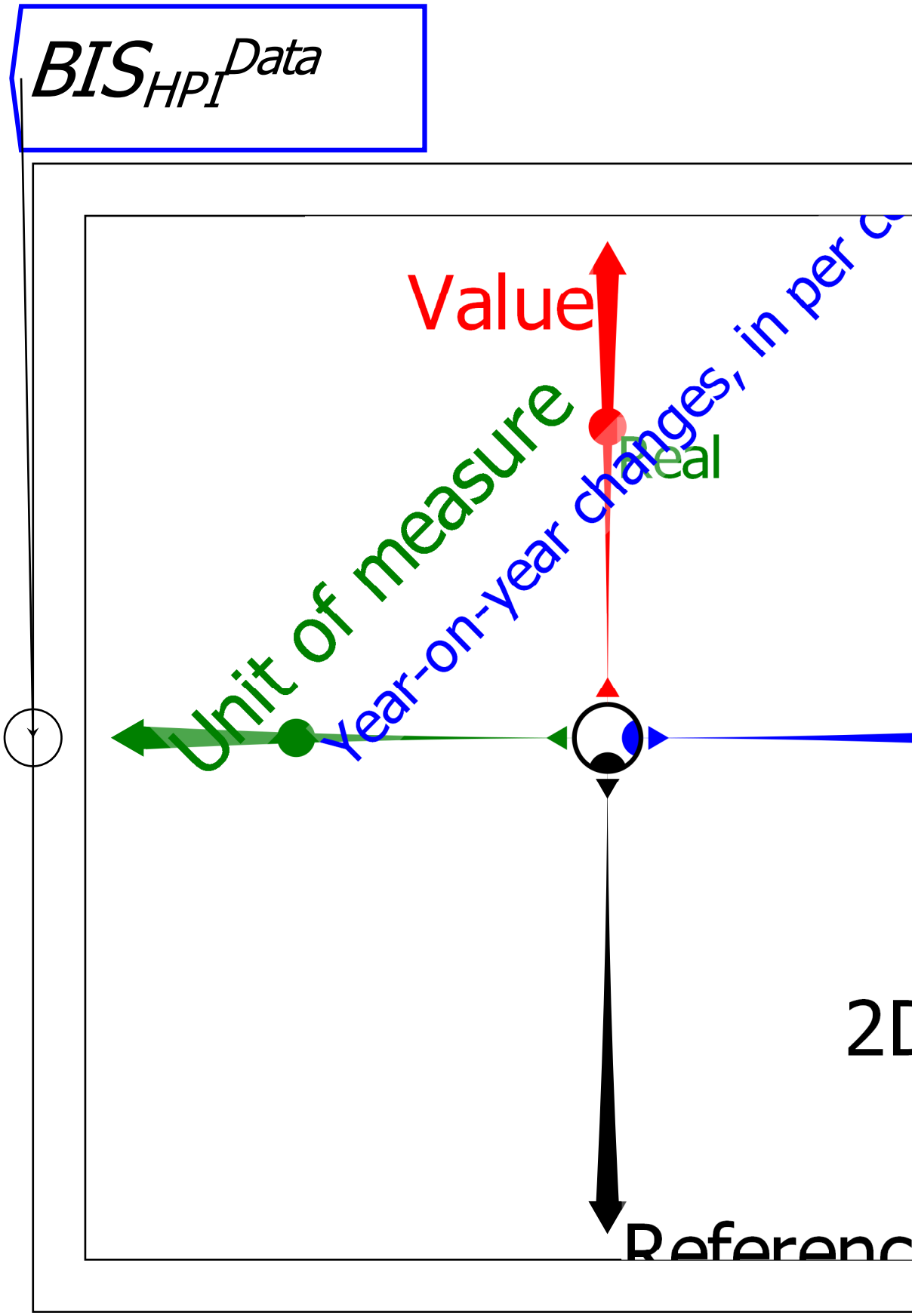
The data still shows the Nominal Index data, since these are the first entries in the other two axes. You can control the entry shown using the selector dots on those two axes: these are the coloured dots that are currently within the inner circle of the Ravel. Selector dots can be moved:

- 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 falling across the world in 2023.


BIS_HPI^Data
Value
Unit of measure
Real
Year-on-year changes, in per c
2D
Reference a

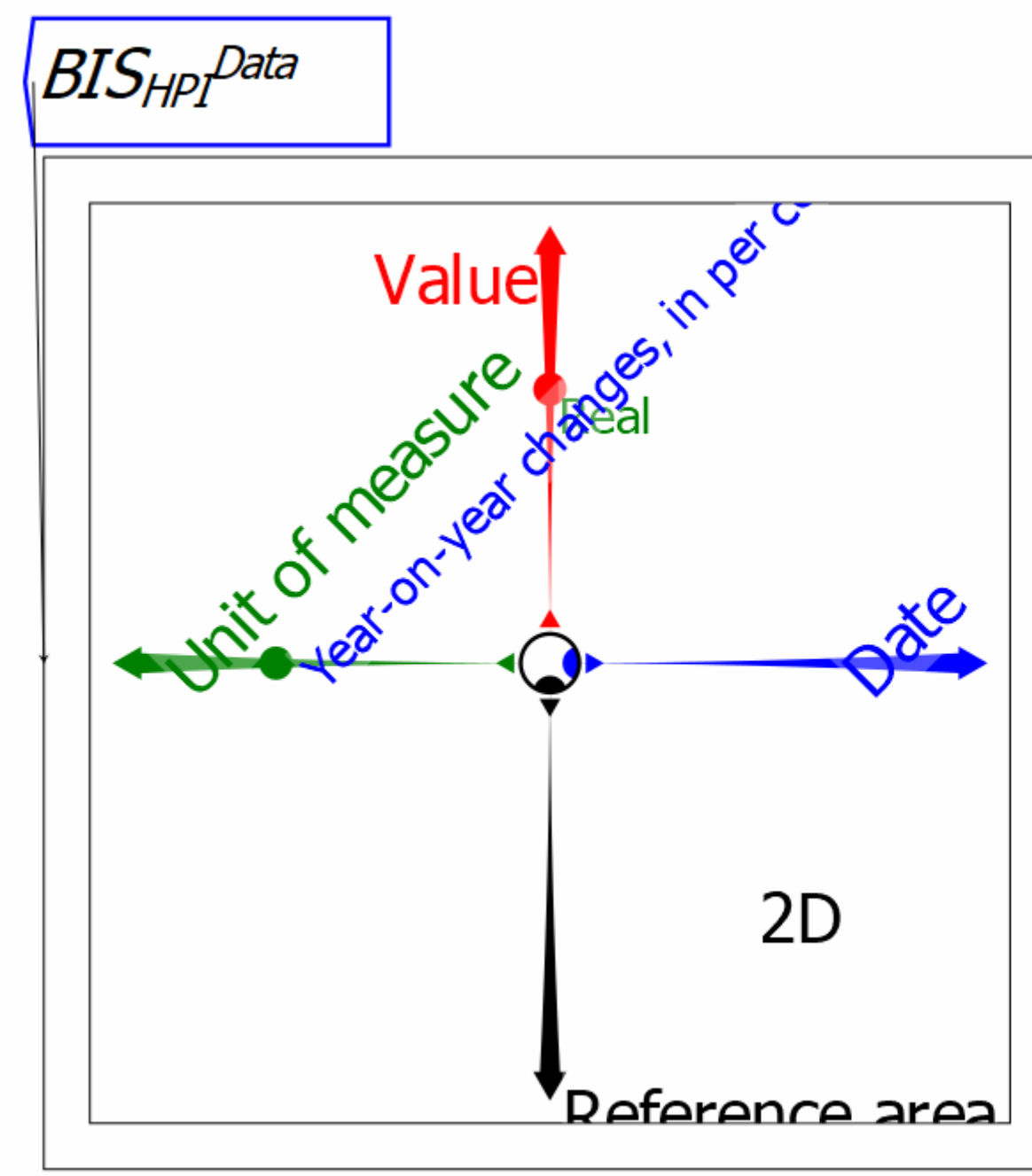

BIS_HPI^Data
Value
Unit of measure
Real
Year-on-year changes, in per
2D
Referenc

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.

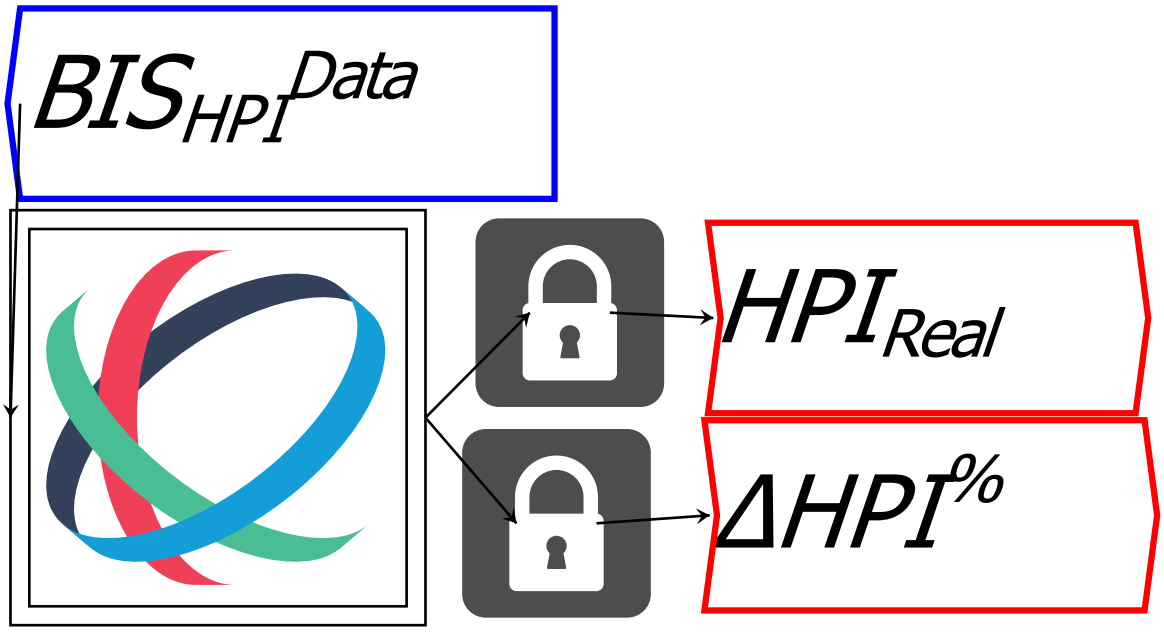
3.2 Attaching Variables to Ravels

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.



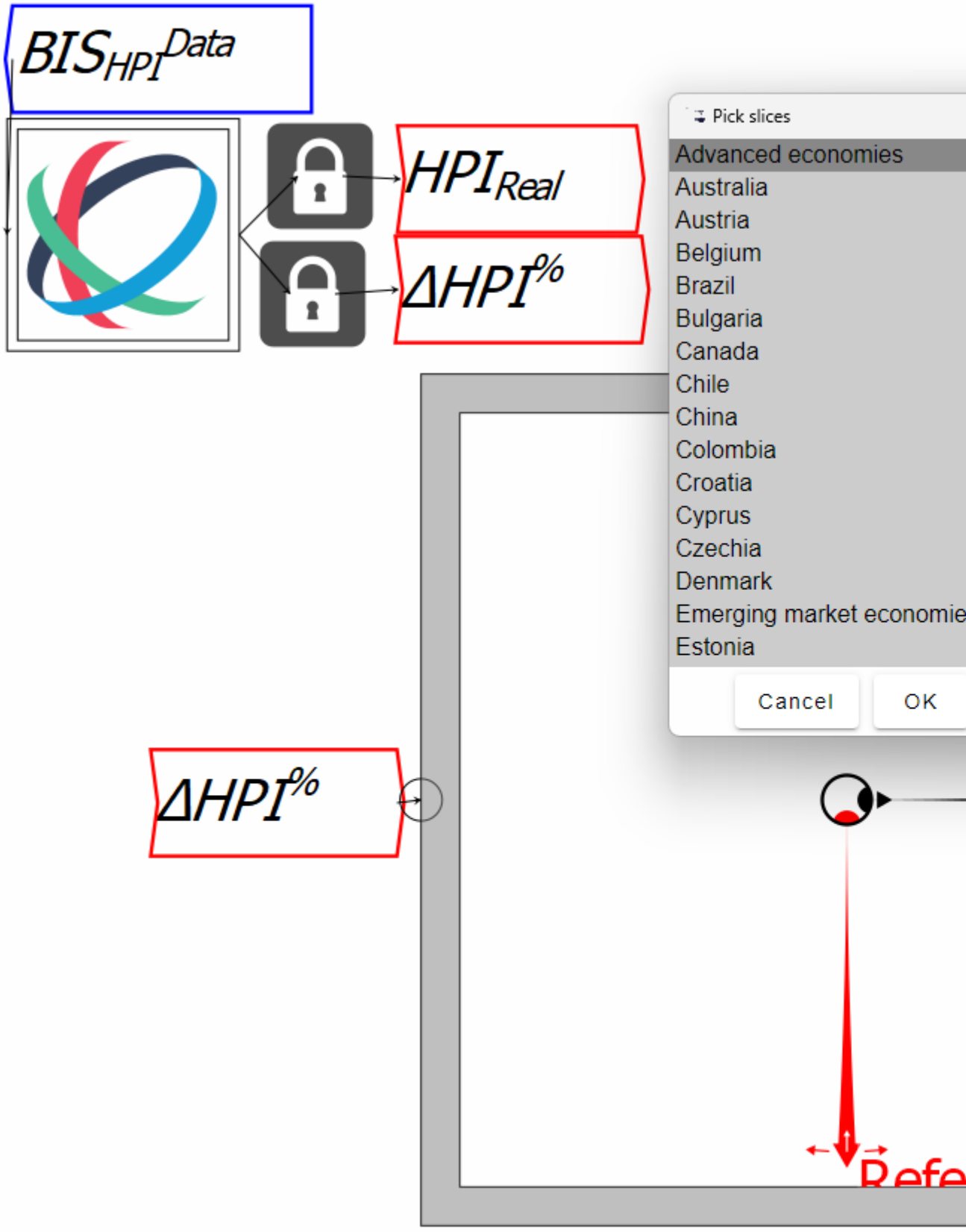
With the data separated into index and inflation data, we can now focus on those subsets of the data rather than the entire source file. The next image shows the source Ravel in icon mode, with the house price inflation data attached to another Ravel and the currently selected data displayed in a sheet.


BIS HPI Data
HPI Real
ΔHPI%


ΔHPI%

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" (this is done using the "Pick Axis Slices" option on the right-click menu for the Reference Area axis, and then choosing several entries by using control-click) and attaching that to a new variable $\Delta HPI^{Avg}_{Advanced}$, while selecting a number of advanced economies from the axis (Australia, Austria, Belgium, etc.) and assigning that to another variable $\Delta HPI_{Advanced}$.


BIS HPI Data
HPI Real
ΔHPI%
Pick slices
Advanced economies
Australia
Austria
Belgium
Brazil
Bulgaria
Canada
Chile
China
Colombia
Croatia
Cyprus
Czechia
Denmark
Emerging market economie
Estonia
Cancel
OK
ΔHPI%

3.3 Basic Analysis using Ravel

3.4 Linking Ravels

3.5 Displaying your results in Sheets and Plots

3.6 Using Publication Tabs

3.7 Working with Other Programs

3.8 Importing Data into Ravel

Chapter 4

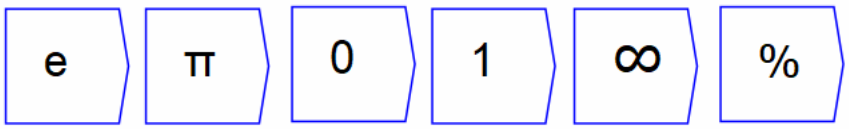
Reference

4.1 Operations

4.1.1 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 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 $\pi = 3.14\ldots$; and *inf* for ∞.



4.1.2 Percent

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

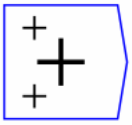
The operator can be placed on the canvas in two ways:

- From the Fundamental Constants toolbar; or
- By pressing the percent key anywhere on the wiring canvas.

4.1.3 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.

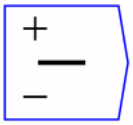


The operator can be placed on the canvas in two ways:

- From the Binary Operations toolbar; or
- By pressing the plus key anywhere on the wiring canvas.

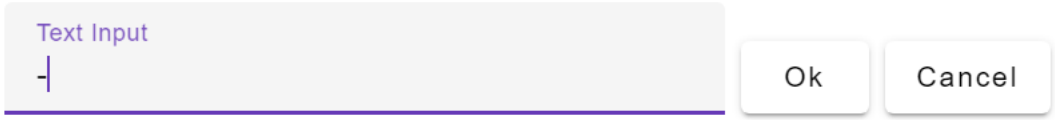
4.1.4 subtract −

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.



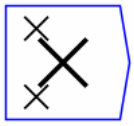
The operator can be placed on the canvas in two ways:

- From the Binary Operations 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.



4.1.5 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.

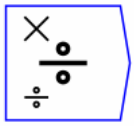


The operator can be placed on the canvas in two ways:

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

4.1.6 divide ÷

Divide a number by another. The input ports allow multiple wires, which are multiplied together 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.

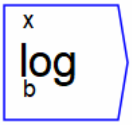


The operator can be placed on the canvas in two ways:

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

4.1.7 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 ln operator instead.



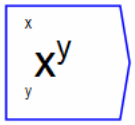
The operator can be placed on the canvas in two ways:

- From the Binary Operations 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 operations, 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 operation will be placed on the canvas.

4.1.8 pow x^y

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

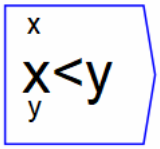


The operator can be placed on the canvas in two ways:

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

4.1.9 lt $<$

Returns 0 or 1, depending on whether $x < y$ is true (1) or false (0).

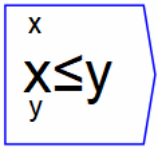


The operator can be placed on the canvas in two ways:

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

4.1.10 le $\leq$

Returns 0 or 1, depending on whether $x \leq y$ is true (1) or false (0).

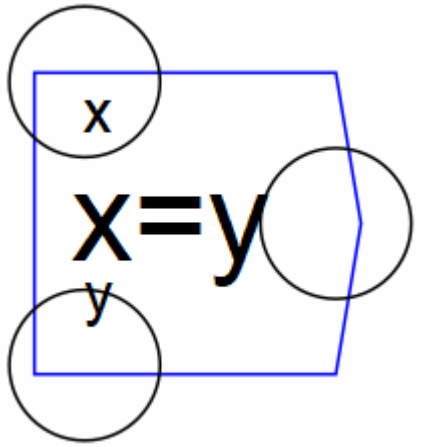


The operator can be placed on the canvas in two ways:

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

4.1.11 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 toolbar; or
- By typing the letters "eq" on the canvas and then pressing the Enter key.

4.1.12 min

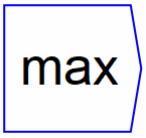
Returns the minimum of x and y.

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

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

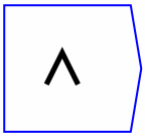
4.1.13 max

Returns the maximum of x and y.



4.1.14 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 toolbar; or
- By typing the letters "and_" on the canvas and then pressing the Enter key. Note the trailing underscore is required because `and` is a reserved word in C++!

4.1.15 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.

4.1.16 not $\neg$

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

4.1.17 time t

Returns the current value of system time.

4.1.18 Gamma Γ

Returns the Gamma function of its argument:

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

4.1.19 Factorial !

Returns the factorial of its argument:

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

Note:

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

which is how it is implemented in Minsky.

4.1.20 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.

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

It relationship to the derivative of the Gamma function (and factorials) is why Minsky provides this function.

4.1.21 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.

4.1.22 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.

4.1.23 copy

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

4.1.24 integrate $\int dt$

Creates an integration (or stock) variable. Editable attributes include the variable's name and its initial value at $t = 0$. The function to be integrated needs to be connected to the top port, 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.

Note that the units of the integral operator are given by the units of the input, multiplied by the time unit, but the units of the *integral's variable* are user specified. It is an error for these to be mismatched, and running *dimensional analysis* from the File menu will check for the consistency of this. Note that

the user specified units in the integral variable can be used to dimension up the integral if the integral variable is connected to the integral's input.

4.1.25 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.

4.1.26 exp

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

4.1.27 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.

4.1.28 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. For further explanation regarding trigonemtric functions, see this wikipedia page[1].

4.1.29 cos

Produces a cosine function of the input. 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 this wikipedia page[2].

4.1.30 tan

Produces a tangent function of the input. 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 this wikipedia page[3].

4.1.31 asin

Produces an arc sine function of the input, or the inverse of the sine function. For further explanation regarding trigonemtric functions, see this wikipedia page[4].

[1] https://en.wikipedia.org/wiki/Trigonometric_functions
[2] https://en.wikipedia.org/wiki/Trigonometric_functions
[3] https://en.wikipedia.org/wiki/Trigonometric_functions
[4] https://en.wikipedia.org/wiki/Trigonometric_functions

4.1.32 acos

Produces an arc cosine function of the input, or the inverse of the cosine function. For further explanation regarding trigonemtric functions, see this wikipedia page[5].

4.1.33 atan

Produces an arc tangent function of the input, or the inverse of the tangent function. For further explanation regarding trigonemtric functions, see this wikipedia page[6].

4.1.34 sinh

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

4.1.35 cosh

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

4.1.36 tanh

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

4.1.37 abs $|x|$

absolute value function

4.1.38 floor $\lfloor x \rfloor$

The greatest integer less than or equal to x.

4.1.39 frac

Fractional part of x, ie $x - \lfloor x \rfloor$.

4.2 Tensor operations

In the following operations, an axis argument can be supplied in the operation edit dialog. The axis name is symbolic and available in a drop down box. If the axis name is not specified, then the operation will be applied as though the input was flattened (unrolled to a vector), and then the result reshaped to the original tensor.

[5] https://en.wikipedia.org/wiki/Trigonometric_functions
[6] https://en.wikipedia.org/wiki/Trigonometric_functions

4.2.1 sum $\sum$

Sum along a given axis.

4.2.2 product $\prod$

Multiply along a given axis.

4.2.3 infimum

Return the least value along a given axis.

4.2.4 supremum

Return the greatest value along a given axis.

4.2.5 any

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

4.2.6 all

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

4.2.7 infindex

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

4.2.8 supindex

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

4.2.9 running sum $\sum+$

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

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

4.2.10 running product $\prod +$

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

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

4.2.11 difference Δ^-, Δ^+

Computes the nearest neighbour difference along a given direction. The optional argument (δ) can be used to specify the number of neighbours to skip in computing the differences. 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.

4.2.12 index

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

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

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 gather

Gather collects the values at index locations indexed by the second argument. The output tensor has shape $[i_0, \ldots i_{ir}, a_0, \ldots a_{j-1}, a_{j+1}, \ldots a_{ar}]$ where $[a_0, \ldots, a_{ar}]$ is the shape of the first argument, and $[i_0, \ldots, i_{ir}]$ is the shape of the second second (index) argument, 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, $x[2.5] = 0.5(x[2] + x[3])$.

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.

4.2.14 inner product ·

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.

4.2.15 outer product ⊗

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.

4.2.16 Meld

The meld operation merges the hypercubes of its input tensors. The value at a given hypercube value is given by the value of the first tensor that has a value defined at that hypercube point. So ordering of input tensors does matter where the data is inconsistent between input tensors.

For example, consider the following inputs and x-vectors:

1. $\{1.0, 2.0, 3.0\}$ with x-vector $\{1, 2, 3\}$, and
2. $\{0.0, 1.5, 4.0\}$ with x-vector $\{0, 2, 4\}$

then the resultant output has x-vector $\{0, 1, 2, 3, 4\}$ and the values are $\{0.0, 1.0, 2.0, 3.0, 4.0\}$ if 1 is connected to port 1 and 2 connected to port 2. If they were connected the other way around, the the values would be $\{0.0, 1.0, 1.5, 3.0, 4.0\}$.

4.2.17 Merge

The merge operation takes n tensors, finds the union hypercube (ie the hypercube that contains all the input tensor hypercubes) and spreads the tensors as necessary to make them conformant. Finally, the resultant tensor has an additional string dimensioned axis, each element of which is one of the input tensors. This dimension should be named using the operation edit dialog. It is an error for a hypercube to contain more than one axis with the same name.

4.2.18 Slice

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($\{x_1, x_2, \ldots x_n\}, -3$) = $\{x_{n-3}, x_{n-2}, x_{n-1}\}$.

4.2.19 Size

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.

4.2.20 Shape

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

4.2.21 Mean

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.

4.2.22 Median

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.

4.2.23 Standard Deviation

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.

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

4.2.24 k-th moment

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.

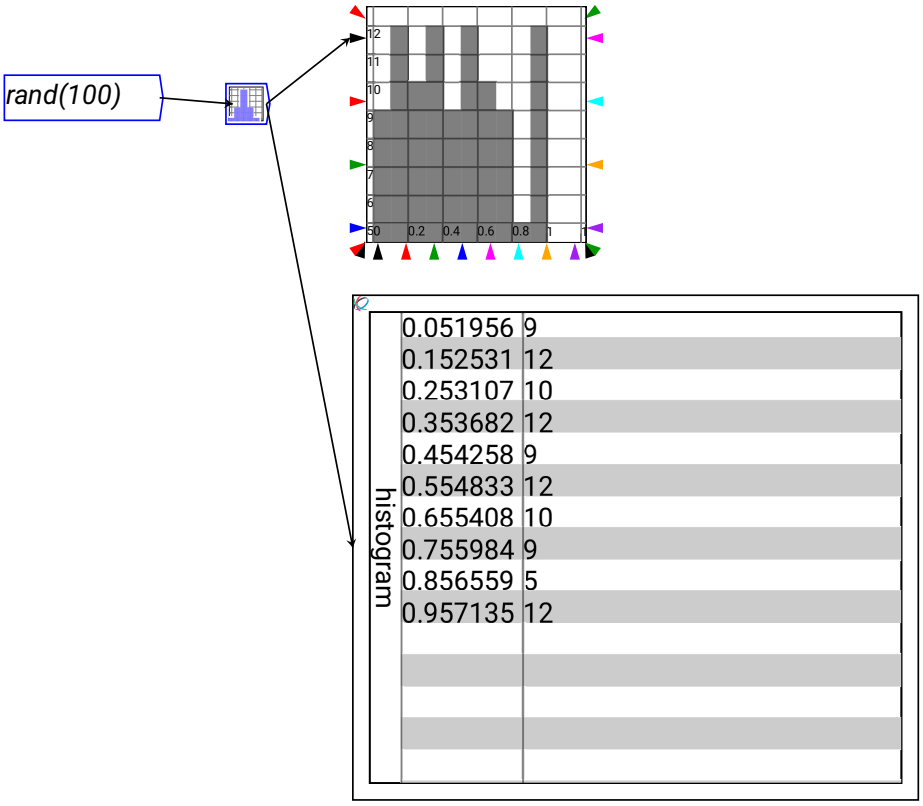
$$\langle \Delta x^k \rangle = \frac{1}{N} \sum_i (x_i - \langle x \rangle)^k$$

Also

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

4.2.25 Histogram

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

4.2.26 Covariance

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 'cov ([X(:), Y(:)])' 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

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

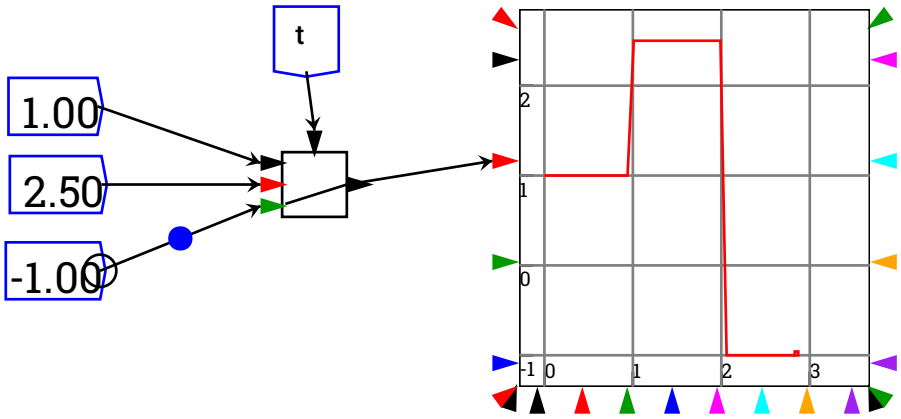
4.2.27 Correlation coefficient ρ

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

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

4.3 Switch

 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.

4.4 Variables

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 may be imported from a CSV file, which is one way of inserting a tensor into the simulation.

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.

4.4.1 Variable names

Variable names uniquely identify variables. Multiple icons on the canvas may have the same name — they all refer to the same variable. Variable names have scope, which is either local (no initial ':'), belonging to an outer group (indicated by a leading ':' on the inner group variable, and the outer group variable having no such leading ':'), or completely global otherwise. You may select a variable name from a drop down list in the "name" combo box, which makes for an easier way of selecting exactly which variable you want.

4.4.2 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 variable (or 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.

4.4.3 Tensor valued initial conditions

There is also a simple functional language, which allows for the generation of tensor-valued operations. 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.

4.4.4 Sliders

From the context menu, one can select a slider to be attached to a variable, which is a GUI "knob" allowing one to control a variable's initial value, or the

value of a parameter or constant. 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.

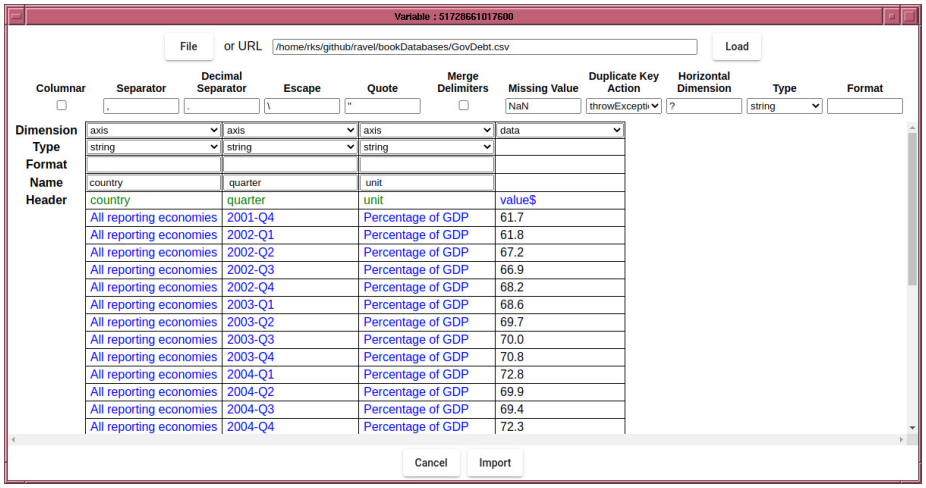
Slider parameters are specified in the edit menu: max, min and step size. A relative slider means that the step size is expressed as a fraction of max-min.

4.4.5 Importing 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:



Quick instructions:

- Data is typically rightmost columns. 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.

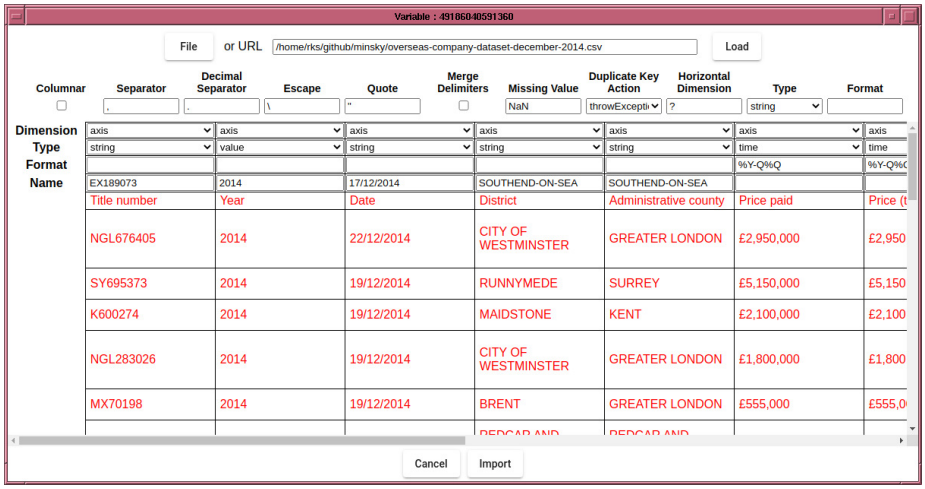
value value data must be numerical and not quoted, e.g. 1, 2

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

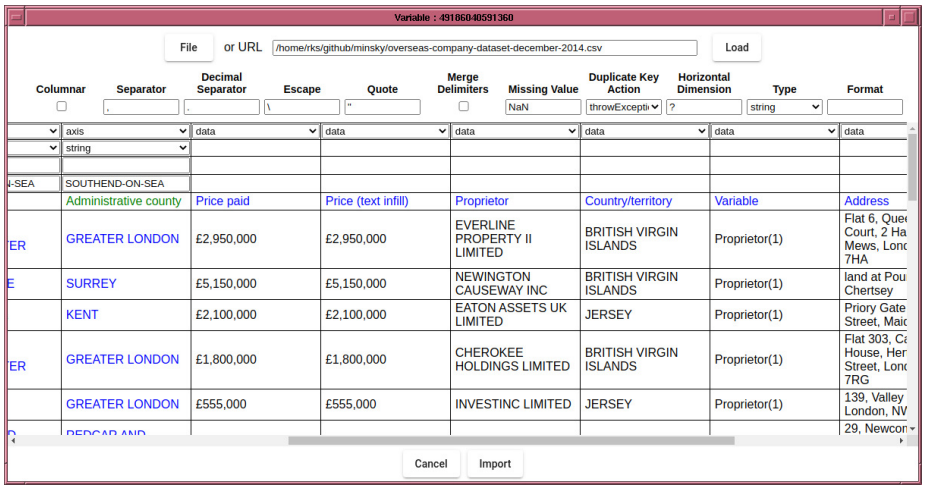
- Click OK button.
- Data is imported into the parameter.
- You may now need to set units for the imported data field, which is located at Edit → Dimensions.

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.

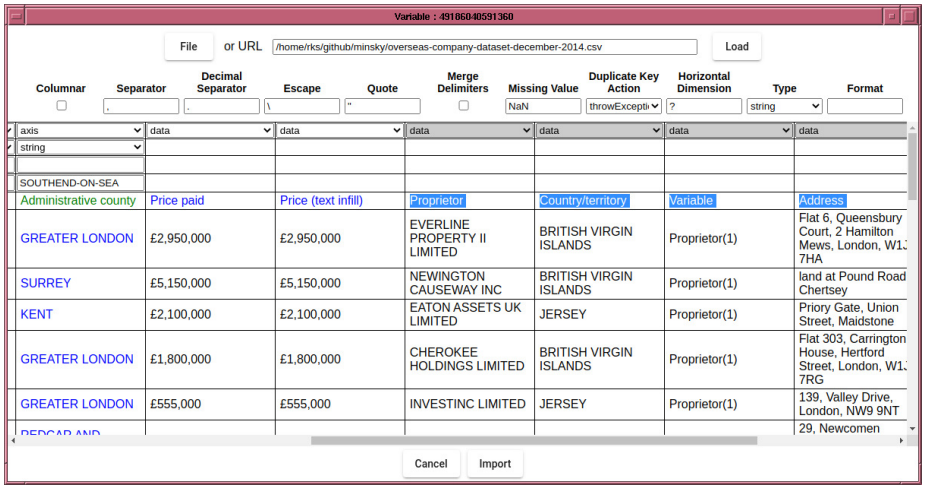
In this case, the automatic parsing system has worked things out correctly, but often times it needs help from the computer user. An example is as follows:



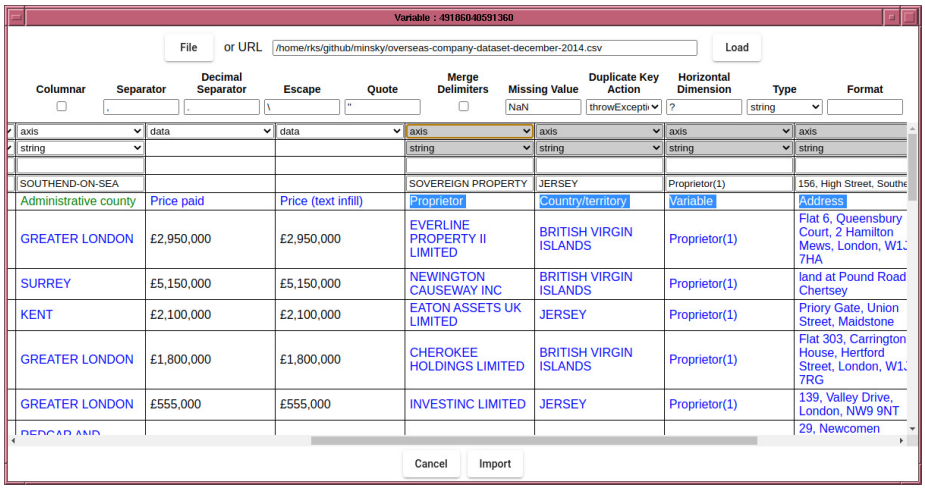
In this example, Minsky has failed to determine where the data starts, probably because of the columns to the right of the "Price" columns. So the first thing to do is tell it where the data is located by clicking on the first cell of the data region.



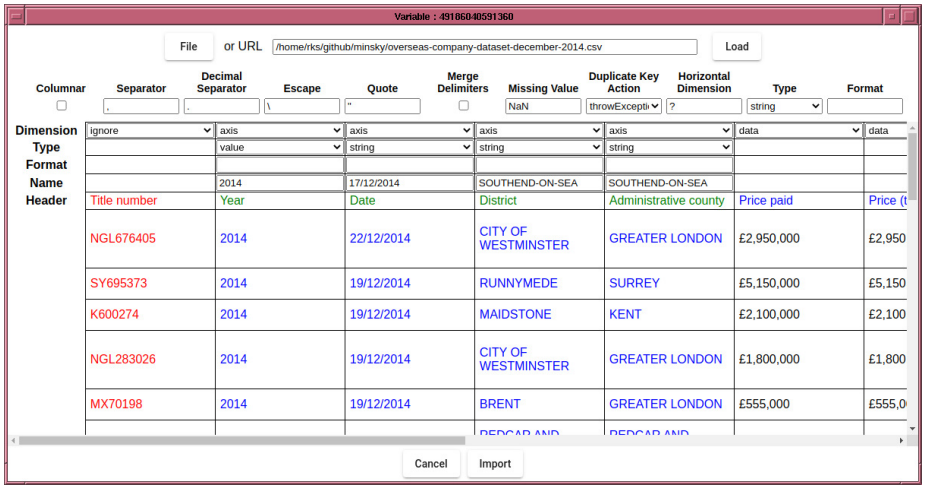
Note that this causes all columns to the right of "Price paid" to be treated as data, which is not right since the columns to the right of "Propieter" are text based columns, not data. So we need to mark those columns as either "axis" or "ignore". To do that, select drag on the header field, which will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



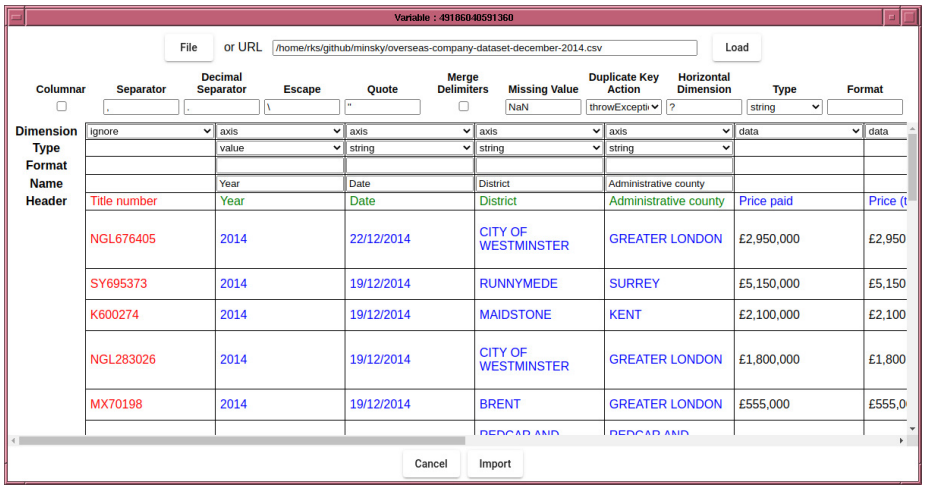
Then in the dimensions row, select "axis", which flips the selected columns:



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, in effect the data is a planar slice through the hypercube. We can remove these axes from the data by marking the column "ignore" in the "Dimension" row. The deselected columns are rendered in red, indicating data that is commented out:

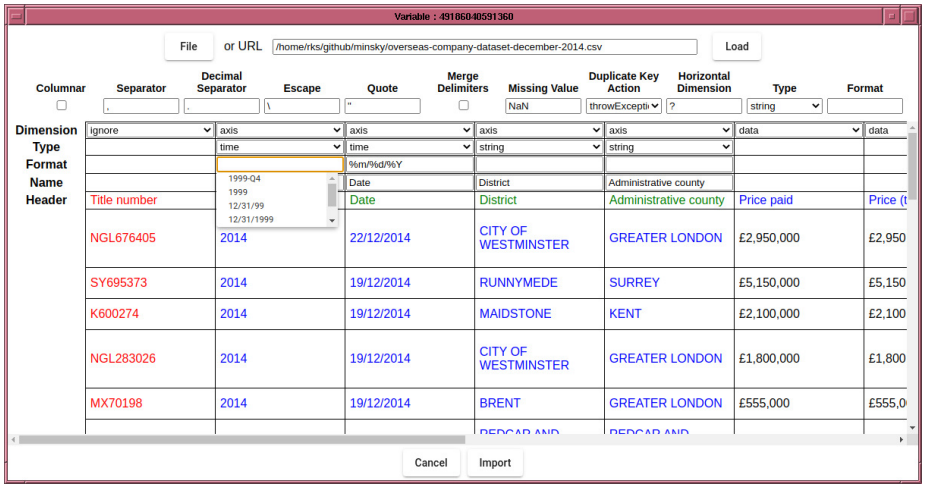


In this example, the axis names has not been correctly inferred. Whilst, one can manually edit the axis names in the "Name" line, a quick shortcut is to drag "Header" and drop it on "Name". (Note the intention is for this to be the case - currently each column name has to be set individually).



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



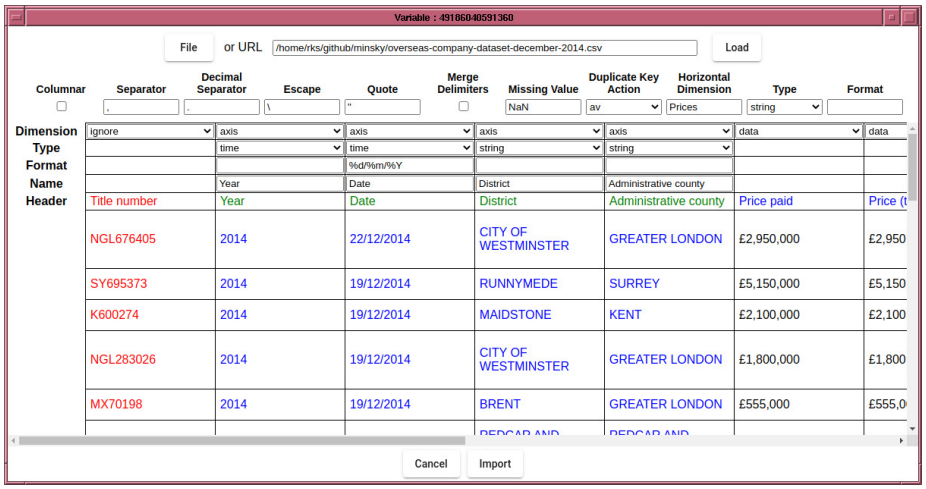
Strftime formatted string consists of escape codes (with leading % characters). All other characters are treated as matching literally the characters of the

Code	Description
%a or %A	The name of the day of the week according to the current locale, in abbreviated form or the full nam
%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
%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

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).

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, we want to aggregate over these records, which we can do by setting "Duplicate Key Action" to sum or maybe average for this example. After some additional playing around with dimensions to aggregate over, we can get the data imported.



4.4.6 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.

In the case where the data is correct, but there are still duplicate keys, such as the example in the previous section, the duplicate keys may be aggregated over by setting the "Duplicate Key action" option.

4.4.7 Variable Browser

The *variable browser* is a popup window that shows all currently defined variables in the system. 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.

4.5 Wires

Wire represent the flow of values from one operation to the next. To add a wire to the canvas, click on the output port of an operation or variable (right hand side of the icon in its initial unrotated orientation), and then drag it towards an input port (on the left hand side of an unrotated icon). You can't connect an operator to itself (that would be a loop, which is not allowed, unless passing

through an integral), nor can an input port have more than one wire attached, with the exception of $+/-$ and $\times/\div$, where the multiple wires are summed or multiplied, respectively, and similarly max/min.

Wires can be bent by dragging the blue dots ("handles"). Every time a handle is dragged out of a straight line with its neighbours, new handles appear on either side. Handles can be removed by double-clicking on them.

4.6 Tensor values

Variables may have tensor values, or sets of data. Different tensors are sorted by rank. For example, a tensor of rank 0 may appear as a single number, let's refer to it as x. A tensor of rank 1 may appear as a sequence of numbers, let's say $(xxxx)$. Rank 2 means a tensor appears as a 2D sequence of numbers, 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. Two ways of getting tensor values into Minsky are via tensor-valued initial conditions (§4.4.3), or by importing a CSV file into a parameter (§4.4.5). Scalar operations are extended to operating elementwise over tensors, and a number of operations exist for operating on tensors (§4.2).

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.

4.7 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

Zooming in on a group 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.

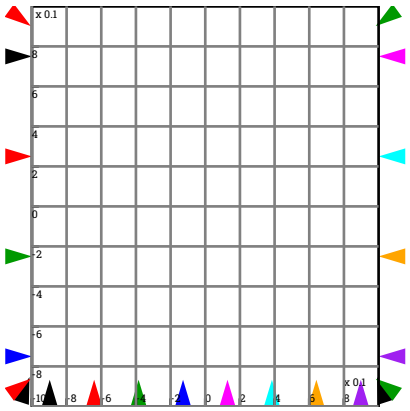
Variable names within groups can be locally scoped to that group. That means that a variable of the same name outside the group refers to a different entity completely. By default, grouped variables refer to entities outside the group scope, but may be marked local by means of context menu option. One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

4.8 Plot widget

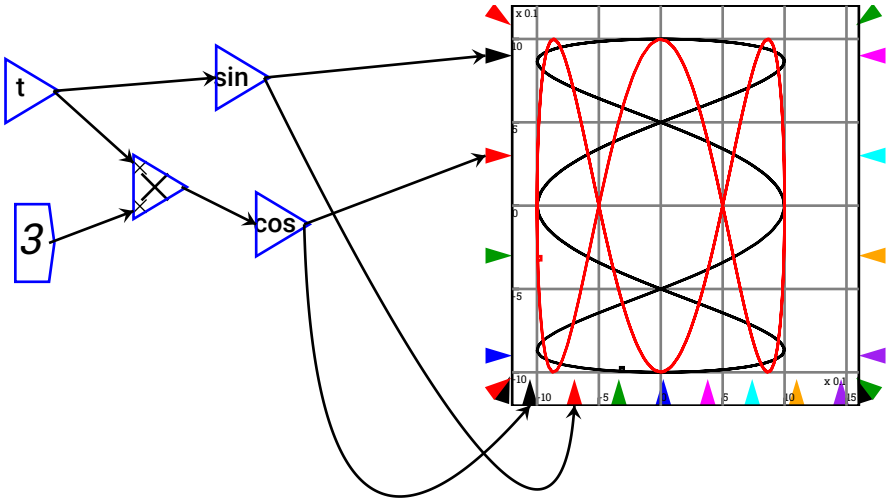
A plot widget embeds a dynamic plot into the canvas. Around the outside of the plot are a number of input ports that can be wired.



left hand edge Up to 4 quantities can be plotted on the graph simultaneously, with line colour given by the colour of the input port

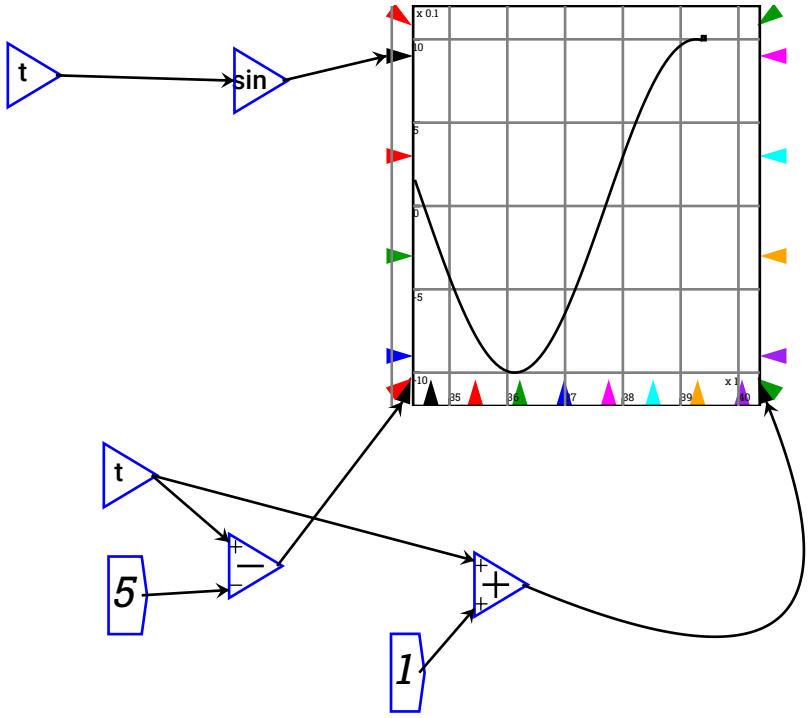
right hand edge Another 4 quantities can be added to the plot. These are shown on a different scale to the left hand inputs, allowing very different magnitudes to be compared on the one plot.

bottom edge Quantities controlling the x-coordinates of the curves. The colours match up with the colour of the pen being controlled.



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. This can be used, for example, to implement a sliding window graph



4.9 Sheet Widget

The Sheet widget displays input data as a number, rather than as a 2D graph, as in the case of the plot widget. To use the Sheet widget, simply wire a variable or other item on your canvas to the left-hand side of the sheet widget box. This will diplay the input data as a number. Note that only one wire can be connected to a sheet, as the sheet can only display a single input value.

The sheet widget can also display rank 0, 1 and rank 2 tensors. These ranks are single values, a string of values, or a 2D matrix of values, respectively. For example, if you create a parameter, and set the initial condition to rand(3,5) (for reference, see section (§4.4.3)), you can wire that into a sheet. The sheet will then diplay the data in a grid display within the widget box.

If you have Ravel™ installed, you will see a small Ravel icon in the top left corner. Clicking this causes a ravel window to pop up, allowing you to manipulate the input data to the sheet, so as to change slices or rotate a multidimensional datacube.

4.10 Note Widget

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. A note widget, like all canvas items, allow short additional tooltips to be specified. It is also possible to annotate an ordinary block with some text that is accessed through the edit menu, or as a tooltip.

You may also use a note as bookmark anchor by ticking the appropriate checkbox.

4.11 Godley Tables

Godley tables describes sets of financial flows from the point of view of a particular economic agent, such as a bank. The columns of the table represent accounts (possibly aggregated), which are treated as integration variables by the system. Accounts may be assets, liabilities or equities. Assets may appear as liabilities in another agent's Godley table, and vice versa, with the sense of the financial flows treated oppositely (a credit flow increasing the asset of one entity will appear as a debit flow, increasing the value of a liability). Transfers between accounts should satisfy the *accounting equation* (Assets-Liabilities-Equities = 0). So if the transfer is between an asset and a liability, then it should appear with the same sign (both positive or both negative), otherwise between two accounts of the same type, or between a liability and an equity, the terms should have opposite signs.

Instead of signed flows, one can optionally use CR and DR prefixes, as specified in the options panel. Each row of the table should have have one CR entry, and one DR entry. The row sum column should be zero if it is done correctly.

The first row specifies the stock variables, after which follow the flow rows. Usually, the row marked "Initial Conditions" comes next, but may be placed in any position. These specify the initial conditions of the stock variables, and may refer to a multiple of another variable, just like the initial condition field, or just be a numerical value.

Finally come the flows. The first column is a simple textual label (the phrase "Initial Conditions", regardless of capitalisation, is a reserved phrase for setting stock variable initial conditions) identifying the flow. The flows themselves are written as a numerical multiplier times a flow variable. For example, if you wanted to transfer an amount between the asset and liability column, you might write "Amount" in both columns, which would satisfy the equation A-L-E=0. It would also be possible to write "2Amount" in the asset column, along with "Amount" in both the Liability and Equity columns. This would still satisfy A-L-E=0.

The Godley table also shows the value of the entered variable, displayed within the table. For example, if you set "Amount" to equal the value of system time, on opening the Godley table, wherever you entered "Amount" in the table the cell would show "Amount = 0.00" if the system time was set to 0.00. This provides a helpful tool for displaying the value of the variable at that point in the simulation. This feature can be enabled or disabled in the preferences panel.

4.12 Context Menu

All canvas items have a context menu, 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).

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 (aka "rats nest") in a model.

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. Obviously only really useful when the variable contains a tensor

Add integral attach an integration operation, and convert the variable into an integral type

integrals

Copy Var copy just the integration variable, not the integration operation

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 to allow financial flows defining the Godley table to be entered or modified.

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 import into other software.

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 it's 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.

4.13 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, then return will insert an operation (if the name matches), or otherwise a variable with that name.

4.14 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

Consider the network introduced in the New to System Dynamics section of the Minsky manual. For GDP, one could enter $/year for the units. Labor Productivity should be expressed in terms of $ per person year. If the system does not accept $/person year, you can enter this as `$ person^-1 year^-1`. Finally, Population has units of person. Press reset, and the Workers variable automatically has units of person, and EmpRate is dimensionless.

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

4.15 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 managing 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.

4.16 Ravel

Ravel is a skunkworks project enabling the interactive manipulation of multi-dimensional datacubes. This is a commercial technology, and you will need a license to use the software, as well as a copy of the Ravel plugin. This is still under active development, but you can read some more about it at Ravelation.

The Lock widget is used to fix the output of a Ravel at a particular point in time, making it easy to compare two different ravel settings.

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

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

OPERATOR	DEFINITION
`true`	True state or any value other than zero (typically 1).
`false`	False state, value of exactly zero.
`and`	Logical AND, True only if x and y are both true. (eg: `x and y`)
`mand`	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)`)
`mor`	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)`)
`nand`	Logical NAND, True only if either x or y is false. (eg: `x nand y`)
`nor`	Logical NOR, True only if the result of x or y is false (eg: `x nor y`)
`not`	Logical NOT, Negate the logical sense of the input. (eg: `not(x and y) == x nand y`)
`or`	Logical OR, True if either x or y is true. (eg: `x or y`)
`xor`	Logical XOR, True only if the logical states of x and y differ. (eg: `x xor y`)
`xnor`	Logical XNOR, True iff the biconditional of x and y is satisfied. (eg: `x xnor y`)
`&`	Similar to AND but with left to right expression short circuiting optimisation. (eg: `(x & y) == (y and x)`)
`|`	Similar to OR but with left to right expression short circuiting optimisation. (eg: `(x | y) == (y or x)`)

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: `iclamp(r0,x,r1)` $= \begin{cases} r0 & \text{if} \quad x \leq r0 \\ x & \text{if} \quad r0 \leq x \leq r1 \\ r1 & \text{if} \quad x \geq r1 \end{cases}$)
`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;`
	`}`

FUNCTION	DEFINITION

`repeat/until`	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:
	
	`repeat`
	`  y := x + z;`
	`  w := u + y;`
	`until ((x += 1) > 100)`

`for`	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:
	
	`for (var x := 0; (x < n) and (x != y); x += 1)`
	`{`
	`  y := y + x / 2 - z;`
	`  w := u + y;`
	`}`

`break/break[]`	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:
	
	`while ((i += 1) < 10)`
	`{`
	`  if (i < 5)`
	`    j -= i + 2;`
	`  else if (i % 2 == 0)`
	`    break;`
	`  else`
	`    break[2i + 3];`
	`}`

`continue`	Continue results in the remaining portion of the nearest enclosing loop body to be skipped. eg:
	
	`for (var i := 0; i < 10; i += 1)`
	`{`
	`  if (i < 5)`
	`    continue;`
	`  j -= i + 2;`
	`}`

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

6.1 New to system dynamics?

Minsky is one of a family of "system dynamics" computer programs. These programs allow a dynamic model to be constructed, not by writing mathematical equations or numerous lines of computer code, but by laying out a model of a system in a block diagram, which can then simulate the system. These programs are now the main tool used by engineers to design complex products, ranging from small electrical components right up to passenger jets.

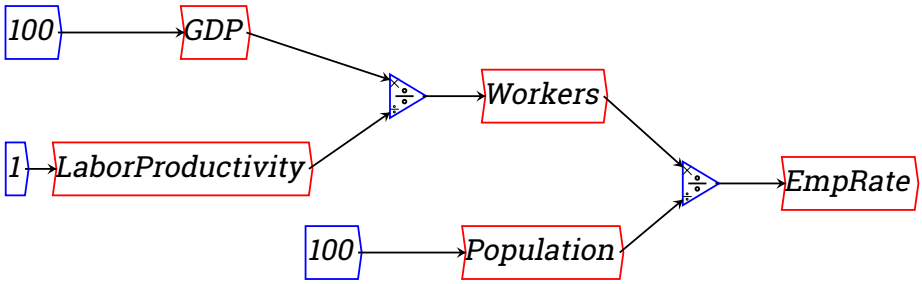
Minsky adds another means to create the dynamic equations that are needed to define monetary flows—the "Godley Table"—which is discussed in the next section for users who are experienced in system dynamics. In this section, we'll give you a quick overview of the generic system dynamics approach to building a model.

Though they 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.

For example, say you wanted to define the rate of employment as depending on output (GDP), labor productivity and population. Then you could define a set of equations in a suitable program (like Mathcad):

$$\begin{aligned} \text{GDP} &:= 100 \\ \text{LaborProductivity} &:= 1 \\ \text{Population} &:= 100 \\ \text{Workers} &:= \text{GDP} \div \text{LaborProductivity} \\ \text{EmpRate} &:= \text{Workers} \div \text{Population} \\ \text{EmpRate} &= 1 \end{aligned}$$

Or you could define it using a block diagram, such as Minsky:



For a simple algebraic equation like this, modern computer algebra programs like Mathcad are just as good as a block diagram programs like Vissim or Minsky. But the visual metaphor excels when you want to describe a complex causal chain.

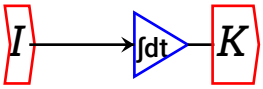
These causal chains always involve a relationship between stocks and flows. Economists normally model stocks and flows by adding an increment to a stock. For example, the level of capital K is defined as a difference equation, where capital in year t is shown as being capital in year $t-1$ plus the investment that took place that year:

$$K_t = K_{t-1} + I_{t-1}$$

The problem with this approach is that in reality, capital is accumulating on a daily, or even hourly, basis. It is better to model stock as continuous quantities and for this reason, all stocks and flows in Minsky are handled instead as integral equations. The amount of capital at time t is shown as the integral of net investment between time 0 and today:

$$K(t) = \int_0^t I(s)ds$$

However, rather than being shown as an equation, the relationship is shown as a diagram:

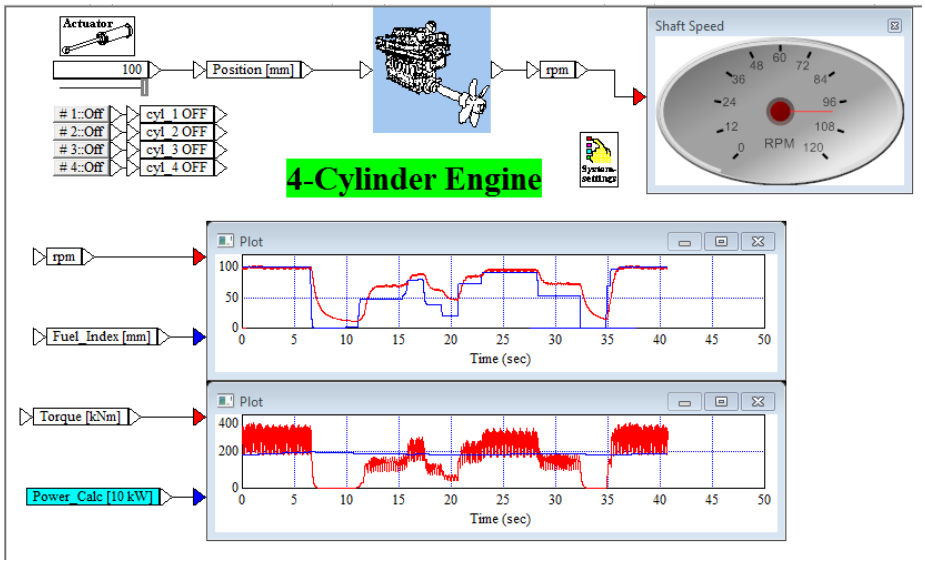


The advantages of the block diagram representation of dynamic equations over a list of equations are:

- 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 store components of a complex block diagram in a group. 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 groups and then "wired together" at a later stage.

For example, here's a model of a 4 cylinder engine car—one of the simple examples distributed with the program Vissim:



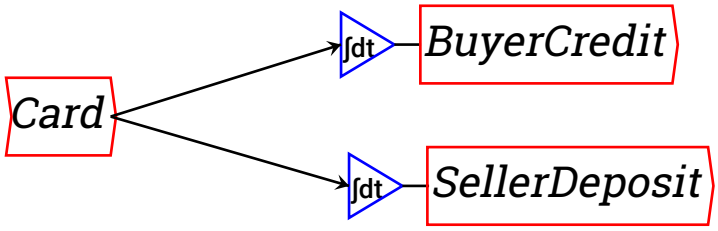
Programs like Vissim and Simulink have been in existence for almost 2 decades, and they are now mature products that provide everything their user-base of engineers want for modeling and analyzing complex dynamic systems. So why has Minsky been developed?

6.2 Experienced in system dynamics?

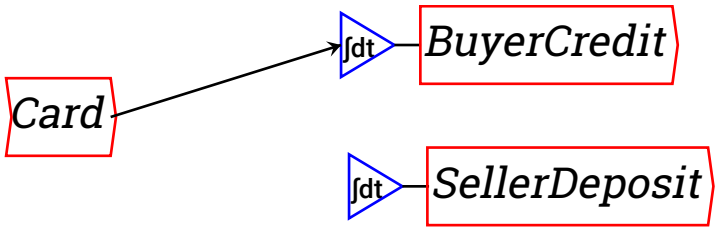
As an experienced system dynamics user (or if you've just read "New to system dynamics?"), what you need to know is what Minsky provides that other system dynamics programs don't. That boils down to one feature: The Godley Table. It enables a dynamic model of financial flows to be derived from a table that is very similar to the accountant's double-entry bookkeeping table.

The dynamics in financial flows could be modeled using the block diagram paradigm. But it would also be very, very easy to make a mistake modeling financial flows in such a system, for one simple reason: every financial flow needs to be entered at least twice in a system—once as a source, and once as a sink.

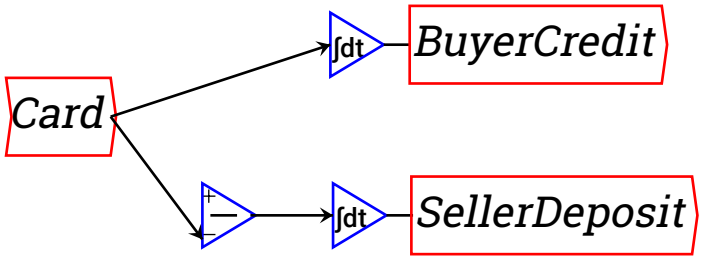
For example, if you go shopping and buy a new computer with your credit card, you increase your debt to a bank and simultaneously increase the deposit account of the retailer from whom you buy the computer. The two system states in this model—your credit card ("BuyerCredit") and the retailer's deposit account ("SellerDeposit")—therefore have to have the same entry (let's call this "Card") made into them. Such a transaction would look like this:



That would work, but there's nothing in the program that warns you if you make a mistake like, for example, wiring up the BuyerCredit entry, but forgetting the SellerDeposit one:



Or, perhaps, wiring up both blocks, but giving one the wrong sign:



In a very complex model, you might make a mistake like one of the above, run the simulation and get nonsense results, and yet be unable to locate your mistake.

Minsky avoids this problem by using the paradigm that accountants developed half a millennium ago to keep financial accounts accurately: double-entry bookkeeping. Here is the same model in Minsky:

Flows ↓ / Stock Variables →	*BuyerCredit*	*SellerDeposit*	Row Sum
	asset	liability	
Initial Conditions	0	0	0
Buyer Accesses Credit	*Card*	*Card*	0

This is an inherently better way to generate a dynamic model of financial flows, for at least two 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 adopts the accounting practice of double-entry bookkeeping, in which entries on each row balance to zero according to the *accounting equation* (Assets=Liabilities+Equities). The source is shown as a positive value increasing the value of assets, the sink is a positive value increasing a corresponding liability. If you don't ensure that each flow starts somewhere and ends somewhere—say you make the same mistake as in the block diagram examples above, then the program will identify your mistake.

If you forget to enter the recipient in this transaction, then the Row Sum identifies your mistake by showing that the row sums to "Card" rather than zero:

Flows ↓ / Stock Variables →	$BuyerCredit$	$SellerDeposit$	Row Sum
	asset	liability	
Initial Conditions	0	0	0
Buyer Accesses Credit	$Card$		$Card$

And it also identifies if you give the wrong sign to one entry:

Flows ↓ / Stock Variables →	$BuyerCredit$	$SellerDeposit$	Row Sum
	asset	liability	
Initial Conditions	0	0	0
Buyer Accesses Credit	$Card$	$-Card$	$2Card$

Minsky thus adds an element to the system dynamics toolkit which is fundamental for modeling the monetary flows that are an intrinsic aspect of a market economy. Future releases will dramatically extend this capability.

Chapter 7

Getting Started with Minsky

7.1 System requirements

Minsky is an open source program available for Windows, Mac OS X, and various Linux distributions, as well as compilable on any suitable Posix compliant system. Go to our SourceForge page to download the version you need. Linux packages are available from the OpenSUSE build service.

7.2 Getting help

Press the F1 key, or select "help" from the context menu. Help is context-sensitive.

7.3 Components of the Program

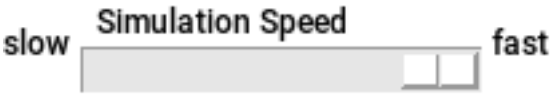
There are 6 components to the Minsky interface:

1. The menus.

   File Edit Insert Options Runge Kutta Help

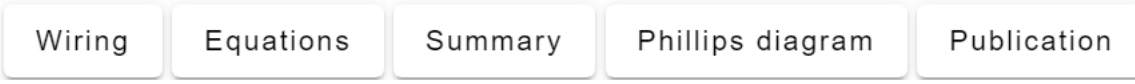
2. The Run buttons

3. The simulation speed slider



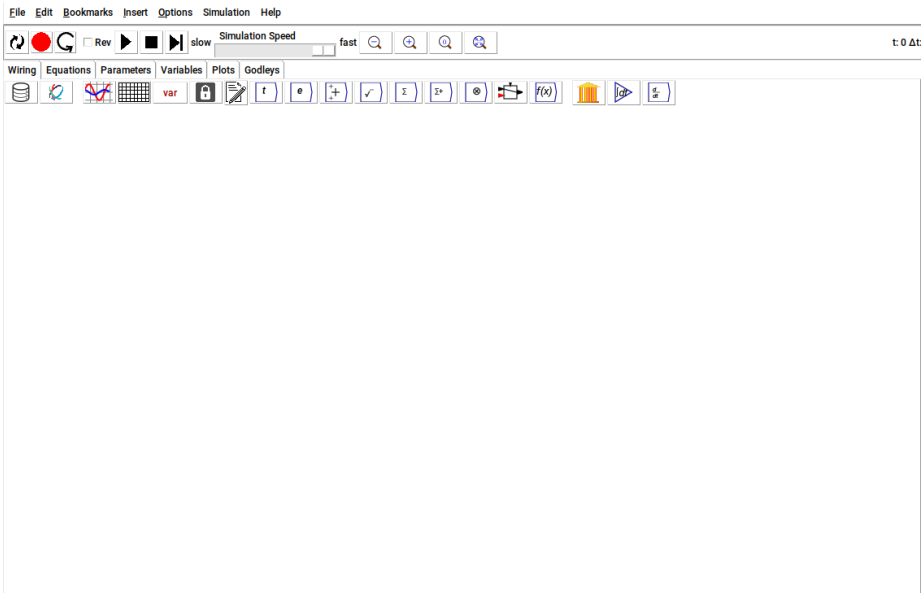
4. The Zoom buttons

5. The Wiring and Equation tabs



6. The design icons

7. And finally the Design Canvas–the large drawing area beneath the buttons and icons.



7.3.1 Menu

File Edit Insert Options Runge Kutta Help

The menu controls the basic functions of saving and loading files, default settings for the program, etc. These will alter as the program is developed; the current menu items are:

File

About Minsky Tells you the version of Minsky that you are using.

New System Clear the design canvas.

Open Open an existing Minsky file (Minsky files have the suffix of "mky").

Recent Files Provides a shortcut to some of your previously opened Minsky files.

Library Opens a 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 Minsky file directly into the current model as a group

Export Canvas Export the current canvas into *svg, *pdf, *eps, *tex, or *m format. If using LaTeX (*tex), produce the set of equations that define the current system for use in documenting the model, for use in LaTeX compatible typesetting systems. If your LaTeX implemention doesn't support breqn, untick the wrap long equations option, which can be found in the preferences panel under the options menu. If using a MatLab function this can be used to simulate the system in a MatLab compatible system, such as MatLab[1] or Octave[2].

Log simulation Outputs the results of the integration variables into a CSV data file for later use in spreadsheets or plotting applications.

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.

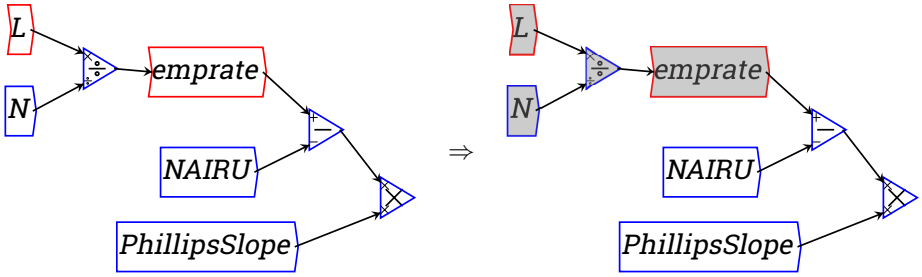
Quit Exit the program. Minsky will check to see whether you have saved your changes. If you have, you will exit the program; if not, you will get a reminder to save your changes.

Debugging use Items under the line are intended for developer use, and will not be documented here. Redraw may be useful if the screen gets messed up because of a bug.

[1] https://en.wikipedia.org/wiki/MATLAB
[2] http://www.gnu.org/software/octave/

Edit

- Undo and Redo allow you to step back and forward in your editing history. If you step back a few steps, and then edit the model, all subsequent model states will be erased from the history.
- 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. At the time of writing, copy-pasting between different instances of Minsky, or into other applications, may not work on certain systems. Pasting the clipboard into a text-based application will be a Minsky schema XML document.



- Create a group using the contents of the selection. Groups allow you to organise more complicated systems specification into higher level modules that make the overall system more comprehensible.

Insert

This menu contains a set of mathematical operator blocks for placement on the Canvas. You can get the same effect by clicking on the Design Icons. Also present are entries for Godley table items and Plots.

Options

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.

Simulation

- Time unit allows one to specify units for the time dimension for dimensional analysis. eg "year", "s" etc.
- Controls aspect of the adaptive Runge-Kutta equation solver, which trade off performance and accuracy of the model.
- Note a first order explicit solver is the classic Jacobi method, which is the fastest, but least accurate solver.
- 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. Screen refresh is the period between screen updates, in ms. If an iteration takes less than this time, the screen refresh is postponed until the time has expired. 100ms is fast enough for a smooth animation of the simulation - increasing this value will improve simulation performance at the cost of a jerky animation of the simulation.
- 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 ince t reaches a certain value. Setting this to "Infinity" causes the simulation to run indefinitely, or until some arithmetic error occurs.

Help

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

7.3.2 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 changes 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.

7.3.3 Recalculate button

The recalculate button computes the values of all variables at the start of simulation. It is particularly useful for recalculating the state of the model with tensor valued data.

7.3.4 Run Buttons

The Run buttons respectively:

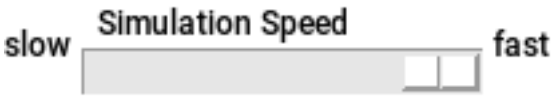
1. Start a simulation–when started the button changes to a pause icon, allowing you to pause the simulation  .

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.

7.3.5 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.

7.3.6 Zoom buttons

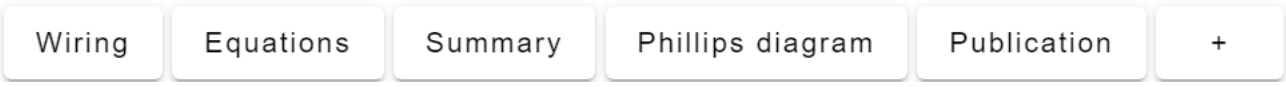
The Zoom buttons zoom in and out on the wiring canvas. The same functionality is accessed via the mouse scroll wheel. The reset zoom button resets the zoom level to 1, and also recentres the canvas. It can also be used to recentre the equation view.

The zoom to fit button zooms the model so that it just fits in the current canvas window.

7.3.7 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.

7.3.8 Wiring and Equations Tabs



This allows you to switch between the visual block diagram wiring view and the more mathematical equations view.

7.3.9 Design Icons

These are the "nuts and bolts" of any system dynamics program. The number of icons will grow over time, but the key ones are implemented now:

Import data Opens an import CSV file dialog, allowing the creation of a parameter from a CSV data file. See Importing CSV files.

Ravel See Ravel.

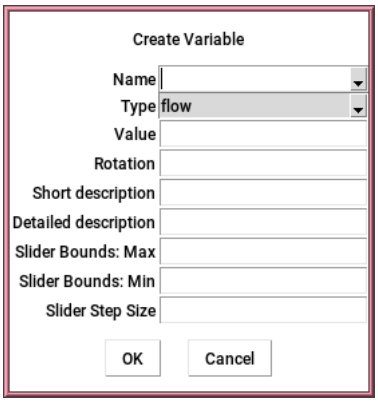
Plot widget Add plots to the canvas.

Sheet widget Add a sheet to the canvas.

Variable .

This is a pull down menu, giving access to creating variables, constants and parameters.

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:



The only essential step here is providing a name for the Variable. You can also enter a value for it (and a rotation in degrees), but these can be omitted. In a dynamic model, the value will be generated by the model itself, provided its input is wired.

When you click on OK (or press Enter), the newly named variable will appear in the top left hand corner of the Canvas. Move the mouse cursor to where you want to place the variable on the Canvas, click, and it will be placed in that location.

Constants are entities whose value is unaffected by the simulation or other entities in the model. Click on it and a constant definition window will appear:

The only essential element here is its value. You can also specify its rotation on the Canvas in degrees. This lets you vary a parameter while a simulation is running—which is useful if you wish to explore a range of policy options while a model is running.

A constant is just a type of variable, which also include parameters (named constants), flow variables, stock variables and integration variables. In fact there is no real conceptual difference between creating a constant or creating a variable, as you can switch the type using the type field.

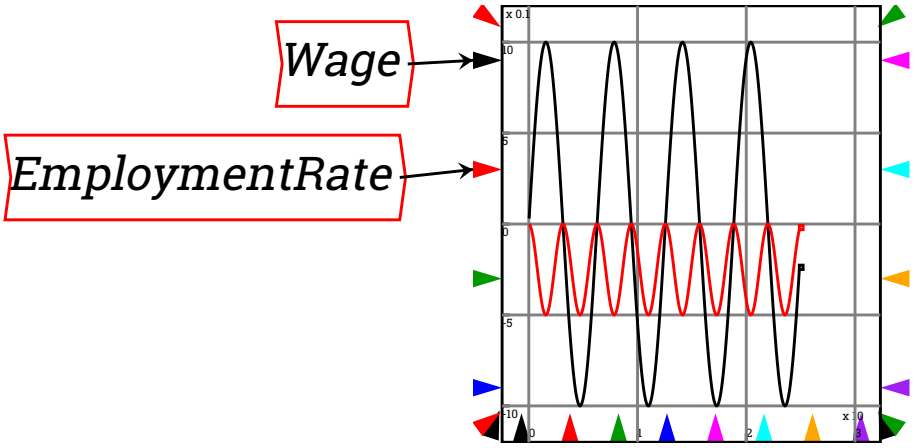
Like the variable and constant button, the parameter button creates a variable defaulting to the parameter type. Parameters differ from flow variables in not having an input port, and differ from constants in having a name and being controllable by a slider during simulation.

Lock Lock widgets are used with Ravels.

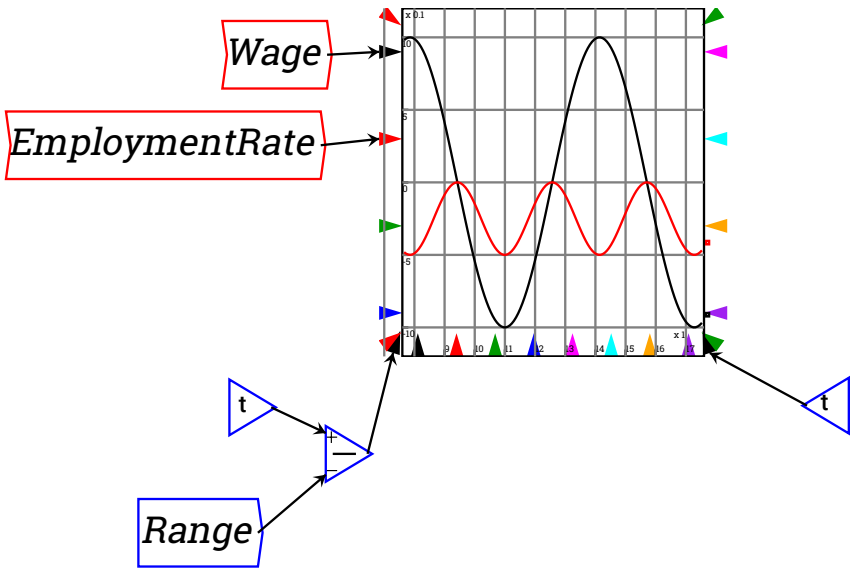
Notes Add textual annotations

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:



Unary functions  These are a fairly standard complement of mathematical functions.

Binary operations  . These execute the stated binary mathematical operations. Each input can take multiple wires as well—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.

Min & Max Functions These take the minimum and maximum values, respectively. These also allow multiple wires per input.

Pow and log. These are binary operations (taking two arguments). In the case of the power operation, the exponent is the top port, and the argument to be raised to that exponent is the bottom port. This is indicated by the x and y labels on the ports. In the case of logarithm, the bottom port (labelled b) is the base of the logarithm.

Logical Operators $<$ $\leq$, $=$, $\wedge$ $\vee$ $\neg$ (and, or, not)] These return 0 for false and 1 for true.

Reduction operations This menu contains operations that reduce a vector to a scalar, or reduce the rank of a tensor. Typically sum, product, any, all etc.

Scans These are running sums and the difference operator

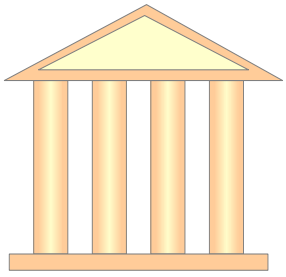
Miscellaneous tensor operations Any other tensor function not covered elsewhere.

Switch Add a piecewise-defined function block to the canvas. Also known as a hybrid function.

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

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

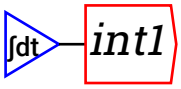
Clicking on it and placing the resulting Bank Icon on the Canvas enters a Godley table into your model:



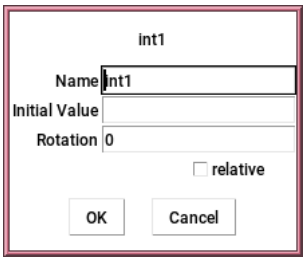
Double-click on the Bank Icon (or right-click and choose “Open Godley Table” from the context menu) and you get a double-entry bookkeeping table we call a Godley Table, which looks like the following onscreen:

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

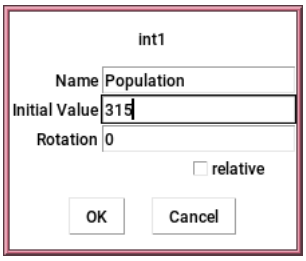
Integration . This inserts a variable whose value depends on the integral of other variables in the system. This is the essential element for defining a dynamic model. Click on it and the following entity will appear at the top left hand side of the canvas (and move with your mouse until you click to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

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

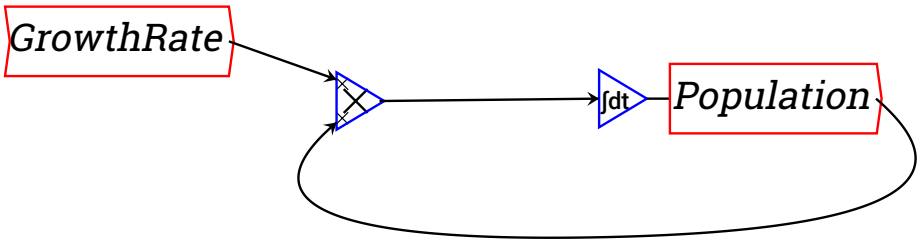
To express this as an integral equation, firstly we multiply both sides of this equation by Population to get:

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

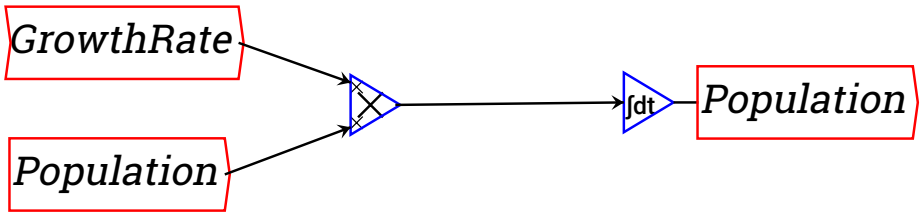
Then we integrate both sides to get an equation that estimates what the population will be T years into the future as:

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

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 Var". Then you will get another copy of the Population variable, and you can wire up the equation this way:



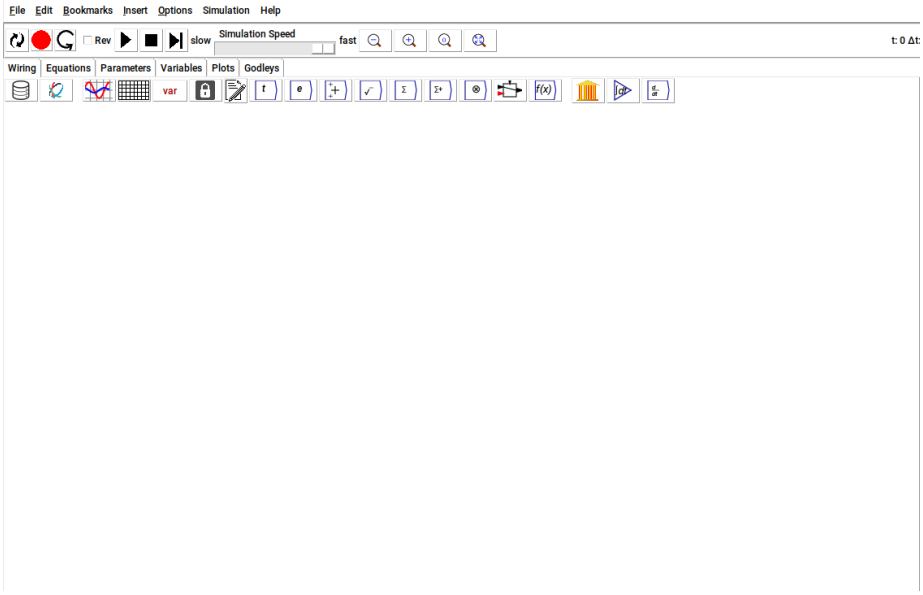
Either method can be used. I prefer the latter because it's neater, and it emphasizes the link between the simple formula for a percentage rate of change and a differential equation.

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

7.3.10 Design Canvas

The Design Canvas is where you develop your model. A model consists of a number of blocks—variables, constants and mathematical operators—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 first mouse button together. The canvas is effectively unlimited, however the scroll bars treat the canvas as a 10000 pixels in size.



7.3.11 Equations tab

This displays the mathematical representation of the model

7.3.12 Summary Tab

This tab provides a summary table of all variables in the system, in a heirarchical fashion that can be navigated by expanding or hiding sections by toggling the caret. Each variable shows its name, it definition, dimensions (for tensor-valued variables), initial expression, units and current value.

Most of these fields are editable, and usually do the obvious thing. 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.

7.3.13 Phillips diagram tab

This tab implement's Minsky's take on a Phillips diagram showing the stocks and flows in a monetary economy. The Phillips tab shows all the information contained in the Godley tables of the model - if there aren't any, this tab will be blank.

Initially, all the stocks will be arranged around a circle, with the flows shown as connecting arrows showing the current direction of the flow. You can move and rotate the stocks, and bend the flows to make a pleasing layout. The stocks will be coloured as though filled with a fluid like Bill Phillip's original analogue computer, and dynamically updated as the simulation proceeds.

Publication tabs allow the creation of dashboards to emphasise certain aspects of a simulation. For example, you may wish to focus on a particular plot or Godley table when running the simulation.

Multiple publication tabs can be created by clicking the '+' tab.

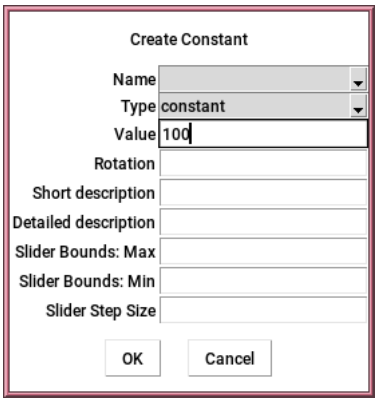
Any item from the wiring tab can be added to a publication tab, and then moved, resized or rotated independently of the item on the wiring tab. For items that dynamically update, the publication tab will be updated dynamically during the simulation.

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

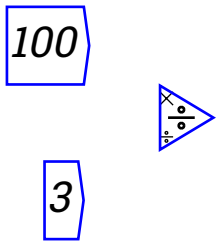
Wires cannot be added to the publication tab, however you can insert arrows (eg $\rightarrow$) by typing the LaTeX text `\rightarrow`, and then rotate and scale the arrow to connect parts of the dashboard together.

7.3.14 Wires

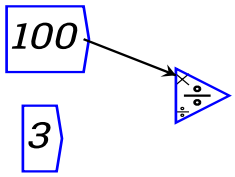
The wires in a model connect blocks together to define equations. For example, to write an equation for 100/33, you would place a  on the canvas, and give it the value of 100:



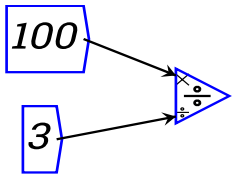
Then do the same for 33, and place a divide block on the canvas:



Then click on the right hand edge of  and drag to extend the wire to the numerator ($\times$) port of the divide operation.



Finally, add the other wire.



7.4 Working with Minsky

7.4.1 Components in Minsky

There are a number of types of components in Minsky

1. Mathematical operators such as plus (+), minus (-)

2. Constants (or parameters, which are named constants) which are given a value by the user

3. Variables whose values are calculated by the program during a simulation and depend on the values of constants and other variables; and

4. Godley Tables, which define both financial accounts and the flows between them. In the language of stock and flow modelling, the columns of a Godley table are the stocks, which are computed by integrating over a linear combination of flow variables.

5. Integrals — represent a variable computed by integrating a function forward in time.

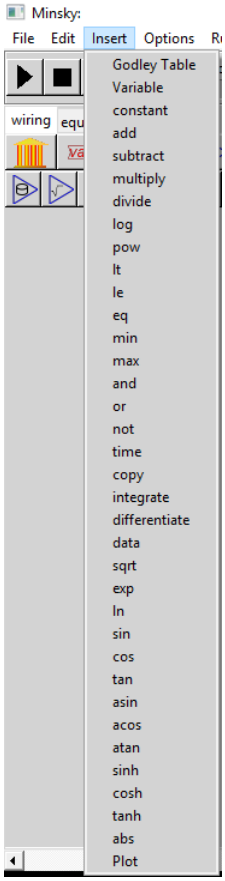
6. Groups, which allow components to be grouped into modules that can be used to construct more complex models.

7.4.2 Inserting a model component

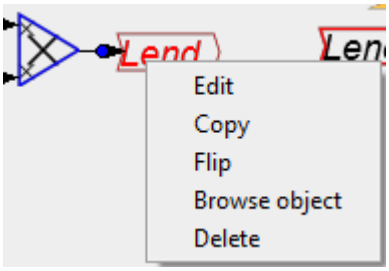
There are five ways to insert a component of a model onto the Canvas:

1. Click on the desired Icon on the Icon Palette, drag the block onto the Canvas and release 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 enetered by typing the numerical value. 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.

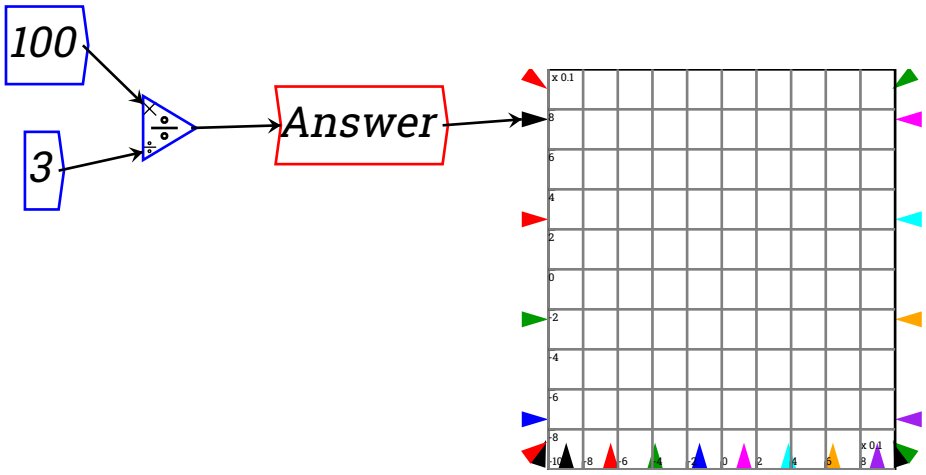
7.4.3 Creating an equation

Equations are entered in Minsky 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

$$100/3 = 33.3$$

is performed in Minsky by defining a constant block with a value of 100, defining another with a value of 3, and wiring them up to a divide-by block. Then attach the output of the divide block to a variable, and run the model by clicking on  :



If you click on the equation tab, you will see that it is:

$$\text{Answer} = \frac{100}{3}$$

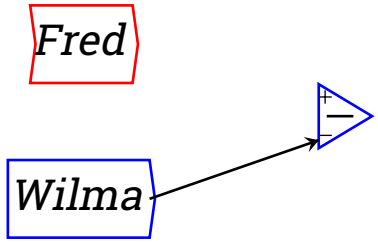
Very complex equations—including dynamic elements like integral blocks and Godley Tables—are designed by wiring up lots of components, with the output of one being the input of the next. See the tutorial for examples.

7.4.4 Wiring components together

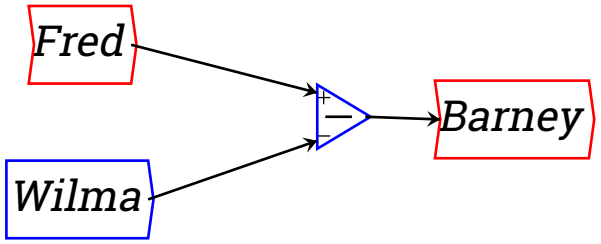
A model is constructed by wiring one component to another in a way that defines an equation. Wires are drawn from the output port of one block to the input port of another. Ports are circles on the blocks to which wires can be attached, which can be seen when hovering the pointer over the block. Variables 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.

To construct an equation, such as Fred - Wilma = Barney:

Click the mouse near the output port of one block and drag the cursor to the input port of another while holding the mouse button down. An arrow extends out from the output port. Release the mouse button near the required input port of the operator. A connection will be made.



The equation is completed by wiring up the other components in the same way.



7.4.5 Creating a banking model

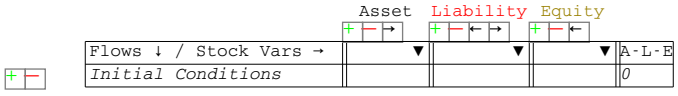
Creating a bank

The first step in creating a model with a banking sector is to click on the Godley Table Icon in the Icon Palette, and place the block somewhere on the Canvas.

Entering accounts

Double click or right click on the Godley table block to bring up the Godley Table. The table is divided up into sections representing the different accounting asset classes: Asset, Liability and Equity. Assets represent what you have to hand at any point in time, and should always be the sum of liabilities and equity. Liabilities represent amounts that are owed to other parties, and equity the amount of capital owned. The column `A-L-E` represents the *accounting equation* (Assets−Liabilities−Equity), and a properly formatted Godley table adhering to *double entry accounting conventions* will have this column zero for all rows.

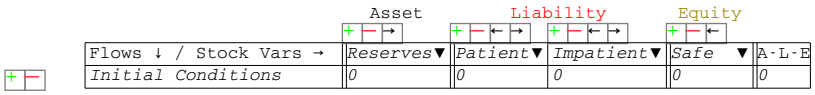
When a Godley Table is first loaded, each accounting class has room for one account (also known as a *stock*) to be defined. To create an additional accounts, click on the '+' button above the first account. One click then adds another column in which an additional account can be defined. Note that the table will delete excess blank accounts, so you should name them as you go. You can chancge the asset class of an account by moving it into the appropriate sector using the ← and → buttons, or by clicking and dragging the column variable name (the first row of the column).



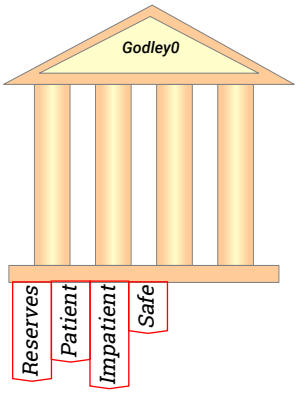
A column can be deleted by clicking on the '–' button above the column.

To define bank accounts in the system you enter a name into the row labeled "Flows ↓ / Stock Variables →". For example, if you were going to define a banking sector that operated simply as an intermediary between "Patient" people

and "Impatient" people—as in the Neoclassical "Loanable Funds" model–you might define the following accounts:

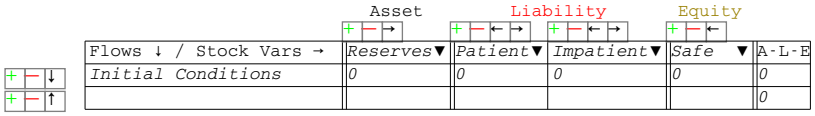


As you enter the accounts, they appear at the bottom of the Bank block on the canvas:

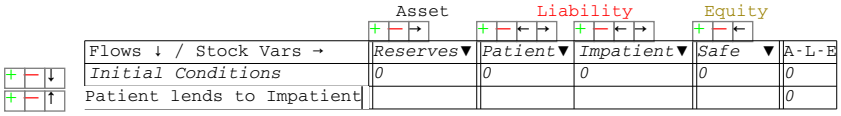


Entering flows between accounts

Flows between accounts are entered by typing text labels in the accounts involved. The source label is entered as a simple name—for example, if Patient is lending money to Impatient, the word "Lend" could be used to describe this action. Firstly you need to create a row beneath the "Initial Conditions" row (which records the amount of money in each account when the simulation begins). You do this by clicking on the '+' key on the Initial Conditions row. This creates a blank row for recording a flow between accounts.



The cell below "Initial Conditions" is used to give a verbal description of what the flow is:



The flows between accounts are then recorded in the relevant cells underneath the columns. Here we will start with putting the label "-Lend" into the Patient column. It is negative, because Patient is lending to Impatient.

		Asset	Liability		Equity	
	Flows ↓ / Stock Vars →	Reserves▼	Patient▼	Impatient▼	Safe ▼	A-L-E
+ — ↓	Initial Conditions	0	0	0	0	0
+ — ↑	Patient lends to Impatient		-Lend			Lend

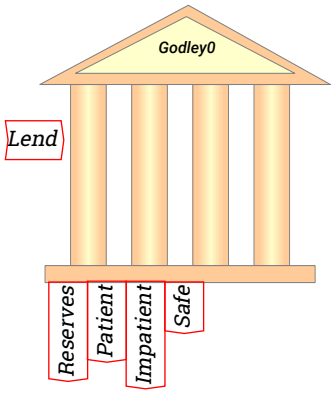
Notice that the program shows that the Row Sum for this transaction is currently "Lend", when it should be zero to obey the double-entry bookkeeping rule that all rows must balance. This is because a destination for "Lend" has not yet been specified. Please note that different asset class columns follow different +ve and -ve rules, so an asset and a liability with the same value might need to both be +ve or both -ve to sum to zero. The destination is Impatient's account, and to balance the row to zero this part of the transaction must be entered as "Lend":

		Asset	Liability		Equity	
	Flows ↓ / Stock Vars →	Reserves▼	Patient▼	Impatient▼	Safe ▼	A-L-E
+ — ↓	Initial Conditions	0	0	0	0	0
+ — ↑	Patient lends to Impatient		-Lend	Lend		0

The accounting equation also applies to the Initial Conditions (the amount of money in each of the accounts prior to the flows between accounts): the Initial Conditions must balance. This requires that there are entries on the Asset side of the Banking ledger that exactly match the sum of Liabilities and Equity:

		Asset	Liability		Equity	
	Flows ↓ / Stock Vars →	Reserves▼	Patient▼	Impatient▼	Safe ▼	A-L-E
+ — ↓	Initial Conditions	120	100	0	20	0
+ — ↑	Patient lends to Impatient		-Lend	Lend		0

As you enter flows, these appear on the left hand side of the bank block:



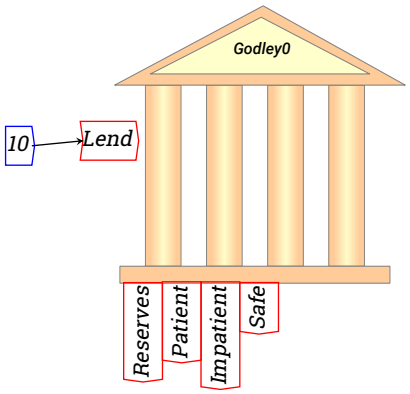
Defining flows

The entries in the Godley Table represent flows of money, which are denominated in money units per unit of time. The relevant time dimension for an

economic simulation is a year (whereas in engineering applications, the relevant time dimension is a second), so whatever you enter there represents a flow of money per year.

You define the value of flows by attaching a constant or variable to the input side of the flow into the bank as shown on the Canvas. For example, you could assign Lend a value of 10 (which would represent a loan of $10 per year by Patient to Impatient) by:

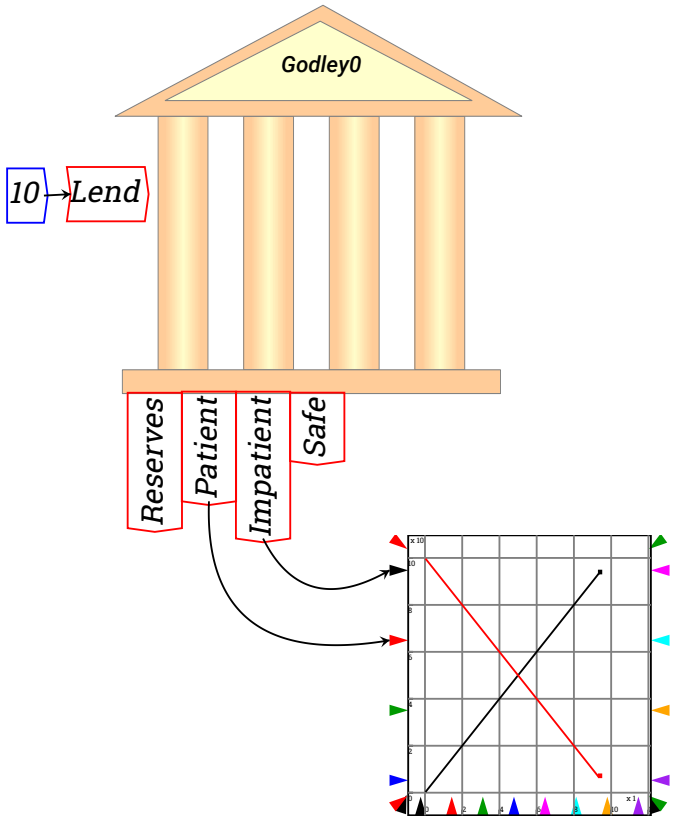
Create a constant with a value of 10, and attaching this to the input side of Lend:



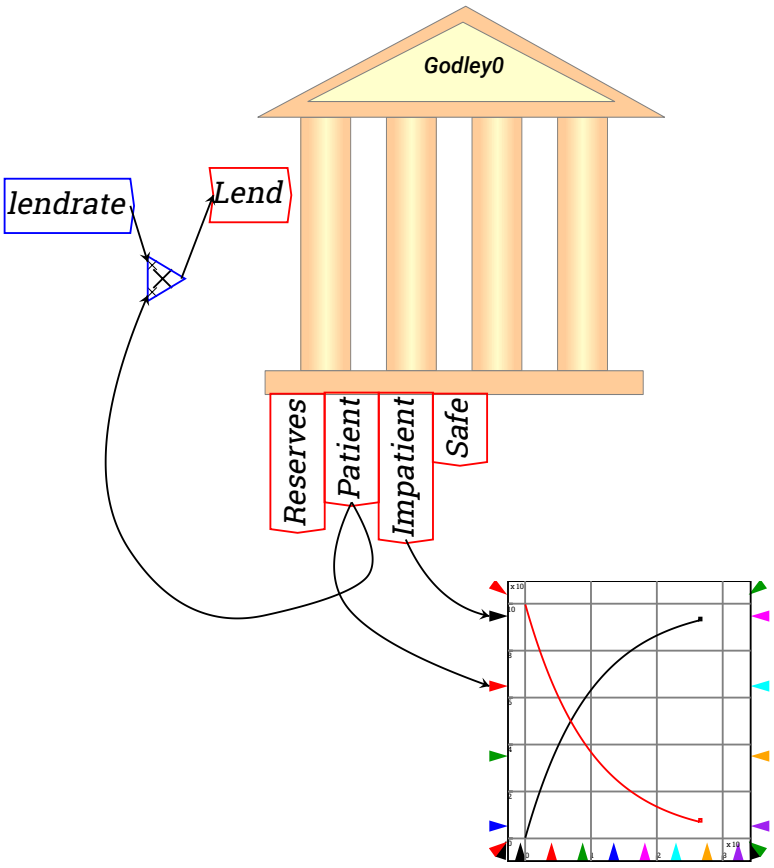
What you have now defined is an annual flow from Patient to Impatient of $10. In the dynamic equations this model generates, Minsky converts all amounts in accounts to positive sums—it shows the financial system from the point of the overall economy, rather than from the point of view of the bank:

$$\begin{aligned} \text{Lend} &= 10 \\ \frac{d\text{Impatient}}{dt} &= \text{Lend} \\ \frac{d\text{Patient}}{dt} &= -\text{Lend} \\ \frac{d\text{Reserves}}{dt} &= \\ \frac{d\text{Safe}}{dt} &= \end{aligned}$$

If you attach a graph to the accounts at the bottom of the bank block, you will see the impact of this flow over time on the balances of the two accounts. Patient's account begins at $100 and falls at $10 per year, while Impatient's account begins at $0 and rises by $10 per year.



Obviously this will result in a negative total worth for Patient after 10 years, so it is not a realistic model. A more sensible simple model would relate lending to the amount left in Patient's account (and a more complex model would relate this to many other variables in the model). That is done in the next example, where a constant "lendrate" has been defined and given the value of 0.1, and Lend is now defined as 0.1 times the balance in Patient's account. This now results in a smooth exponential decay of the amount in the Patient account, matched by a rise in the amount in Impatient account.



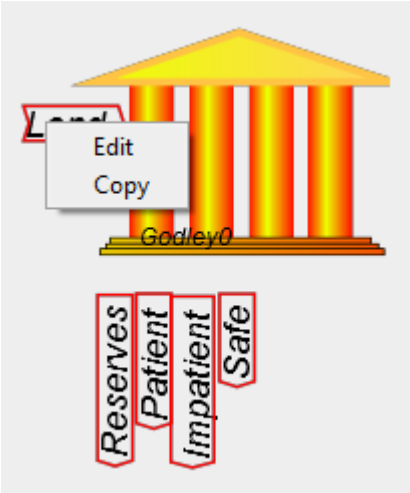
This is because the equation you have defined is identical to a radioactive decay equation, with the amount in the Patient account falling at 10 percent per year:

$$\begin{aligned} \text{Lend} &= \text{lendrate} \times \text{Patient} \\ \frac{d\text{Impatient}}{dt} &= \text{Lend} \\ \frac{d\text{Patient}}{dt} &= -\text{Lend} \end{aligned}$$

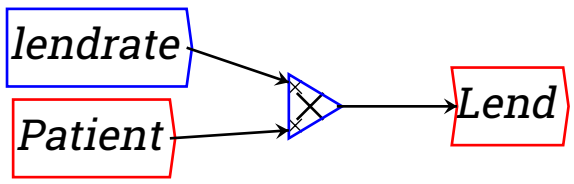
Note however that there are now wires crossing over other wires? There is a neater way to define flows.

Copying Godley Table input & outputs

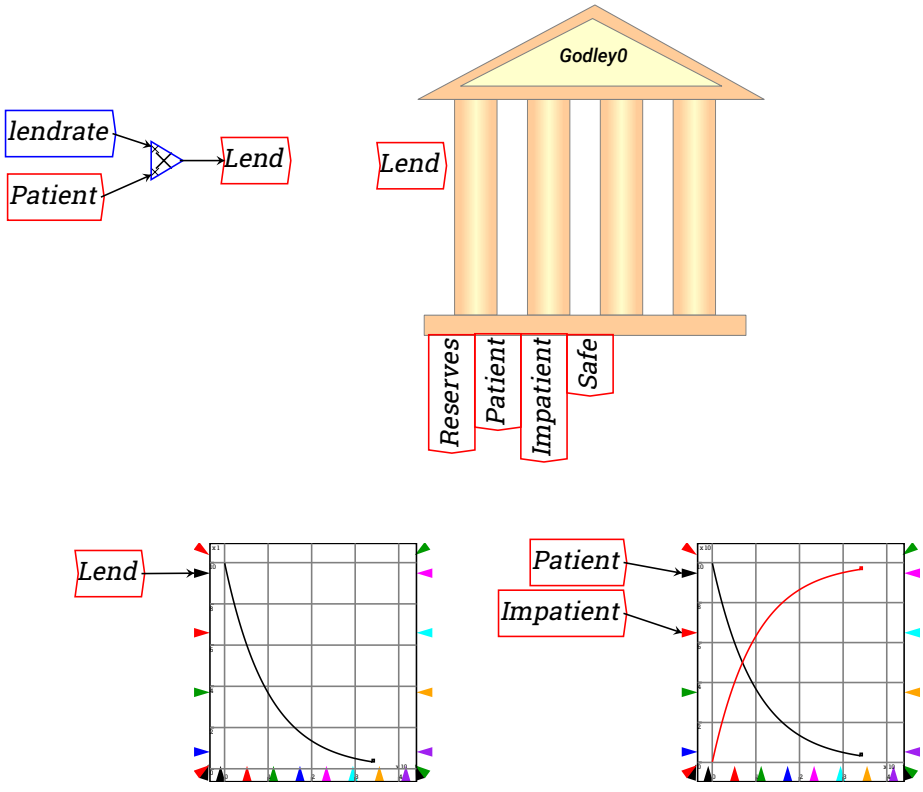
Right-click on the inputs and outputs of a Godley Table and choose “copy” from the drop-down menu:



Place the copied flows and accounts and place them away from the table. Then wire up your definition there:



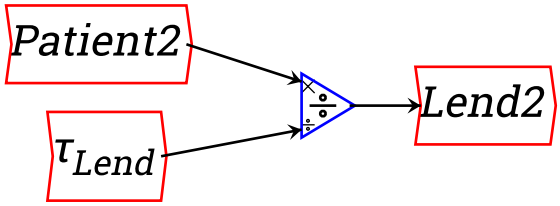
This now results in a much neater model. The same process can be used to tidy up graphs as well:



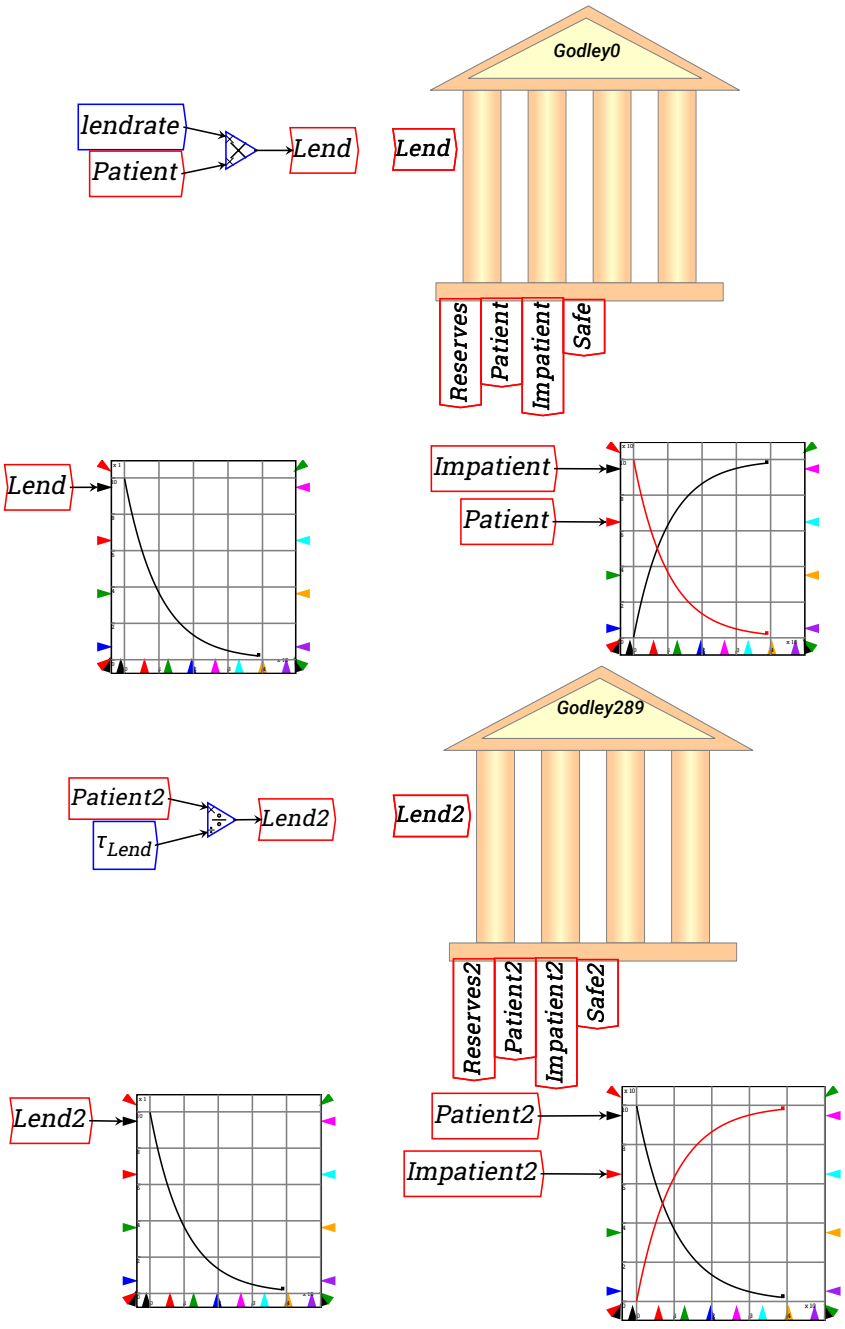
A more complex model would have many more flows, and these in turn would depend on other entities in the model, and be time-varying rather than using a constant "lendrate" as in this example—see the Tutorial on a Basic Banking Model for an example. This example uses the engineering concept of a "time constant", which is explained in the next section. Please note that right-clicking godley table variables and selecting "copy flow variables" creates a new group, which, when clicked and selecting "open in canvas", changes the canvas to show just that group. The normal canvas can be brought back by right-clicking and selecting "open master group".

Using "Time Constants"

The value of 0.1 means that the amount of money in the Patient account falls by one tenth every year (and therefore tapers towards zero). An equivalent way to express this is that the "time constant" for lending is the inverse of 1/10, or ten years. The next model uses a variable called τ_{Lend}, and gives it a value of 10:



As the simulation shows, the two models have precisely the same result numerically:

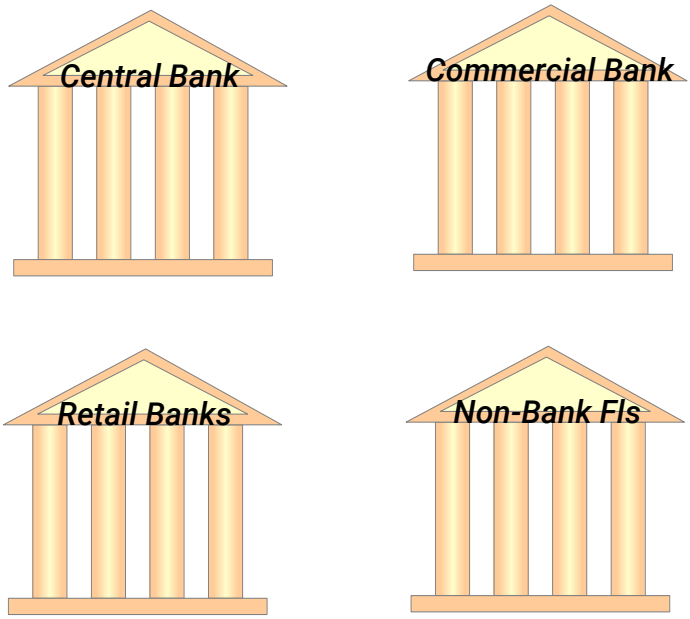


The advantage of the time constant approach is that it is defined in terms of the time that a process takes. A time constant of 10 says that, if this rate of lending was sustained (rather than declining as the account falls), then in ***precisely*** 10 years, the Patient account would be empty. The advantages of

this formulation will be more obvious in the tutorial.

Multiple banks

There can be any number of Godley Tables—each representing a different financial institution or sector in an economy—in the one diagram. The name of the institution can be altered by clicking on the default name ("Godley0" in the first one created) and altering it. Here is an example with 4 such institutions/sectors defined:



If there are interlocking accounts in these banks—if one lends to another for example—then what is an asset for one must be shown as a liability for the other.

Godley tables may be further placed in *groups*, which allows scoping of the flow variables and their defining equations, whilst still allowing the tables to be coupled via global variables.

Chapter 8

Tutorial

8.1 Basic System Dynamics model

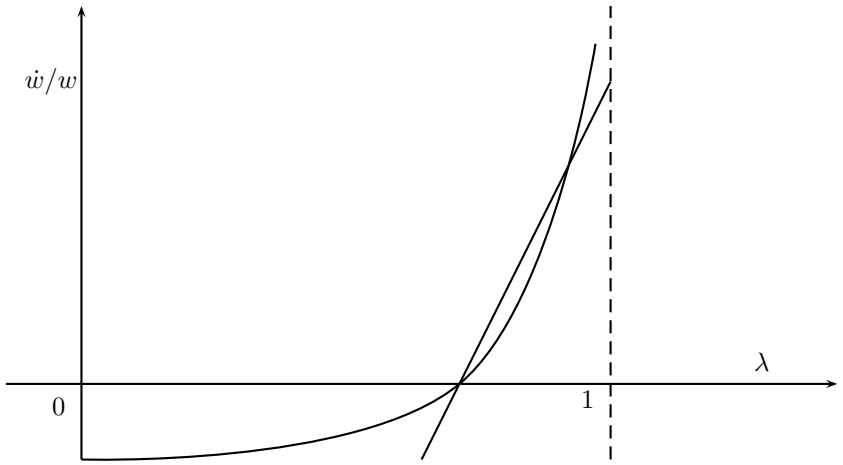
In 1965, Richard Goodwin, the great pioneer of complexity in economics, presented the paper "A Growth Cycle" to the First World Congress of the Econometric Society in Rome. It was later published in a book collection (Goodwin, Richard M. 1967. "A Growth Cycle," in C. H. Feinstein, *Socialism, Capitalism and Economic Growth.* Cambridge: Cambridge University Press, pp. 54–58.); to my knowledge it was never published in a journal.

Goodwin's model has been subjected to much critical literature about implying stable cycles, not matching empirical data, etc., but Goodwin himself emphasized that it was a "starkly schematized and hence quite unrealistic model of cycles in growth rates". He argued however that it was a better foundation for a more realistic model than "the more usual treatment of growth theory or of cycle theory, separately or in combination."

Goodwin emphasized the similarity of this model to the Lokta-Volterra model of interacting predator and prey, which can make it seem as if it was derived by analogy to the biological model. But in fact it can easily be derived from a highly simplified causal chain:

- The level of output (Y) determines the level of employment (L), with $L = Y/a$ where a is a measure of labor productivity;

- Given a population N, the employment rate $\lambda = L/N$ plays a role in determining the ***rate of change*** of the wage w: Goodwin used a linear approximation to a non-linear "Phillips Curve":



His linear approximation was:

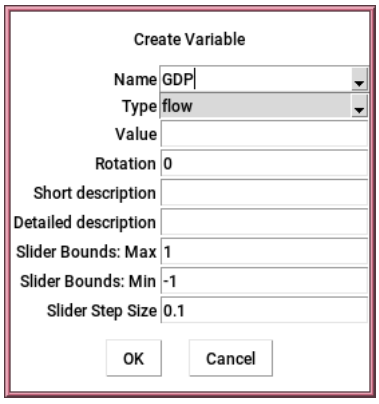
$$\frac{1}{w}\frac{d}{dt}w = -\gamma + \rho \cdot \lambda$$

- In a simple two-class model, profits Π equals the level of output Y minus the wage bill: $\Pi = Y - wL$
- For simplicity, Goodwin assumed that all profits were invested, so that Investment equals profits: $I = \Pi$.
- Investment is the rate of change of the capital stock K;
- The level of output is, to a first approximation, determined by the level of capital stock (K). A simple way of stating this is that Y is proportional to K: $Y = K/v$, where v is a constant (Goodwin notes that this relation "could be softened but it would mean a serious complicating of the structure of the model"); and finally
- Goodwin assumed that labor productivity grew at a constant rate α, while population grew at a constant rate β.

  Goodwin published the model as a reduced form equation in the two system states the employment rate (λ) and the workers' share of output (ω):

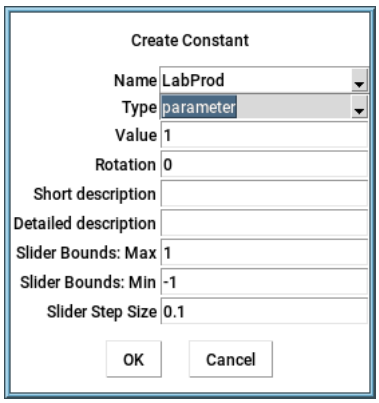
$$\begin{aligned}\frac{d}{dt}\lambda &= \lambda\left(\frac{1-\omega}{v} - \alpha - \beta\right)\\ \frac{d}{dt}\omega &= \omega \cdot (\rho \cdot \lambda - \gamma - \alpha)\end{aligned}$$

This form is useful for analytic reasons, but it obscures the causal chain that actually lies behind the model. With modern system dynamic software, this can be laid out explicitly, and we can also use much more meaningful names. We'll start with defining output (which is a variable). Click on var on the Icon Palette, or click on the Operations menu and choose "Variable". This will open up the "Specify Variable Name" window:



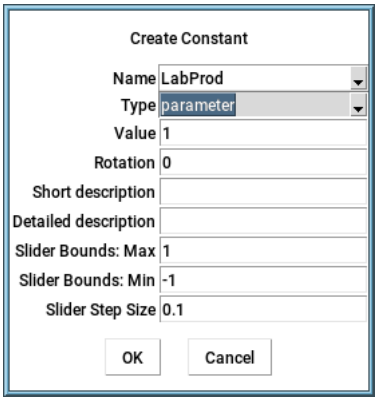
Enter "**GDP**" into the "Name" field, and leave the other fields blank—since **GDP** is a variable and we're defining a dynamic system, the value of **GDP** at any particular point in time will depend on the other entities in the model. Now Click OK (or press "Enter"). The variable will now appear, attached to the cursor. Move to a point near the top of the screen and click, which will place the variable at that location.

We are now going to write the first part of the model, that Labor (**Labor**) equals output (**GDP**) divided by labor productivity (**LabProd**). Just for the sake of illustration, we'll make **a** a parameter, which is a named constant (this can easily be modified later). For this we start by clicking on const on the Palette, or by choosing Insert/variable from the menu. This will pop-up the Edit Constant window:

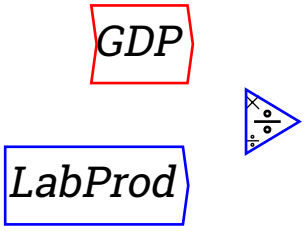


There is actually no real difference between the "Edit constant" dialog and the "Edit variable" dialog. The window's title differs, and the default value of Type is "constant" instead of "flow". We're going to select "parameter", allowing one to give the parameter a name.

Give the paramter the name "**LabProd**" and the value of 1 (i.e., one unit of output per worker). Click OK or press Enter and the constant LabProd will now be attached to the cursor. Place it below **GDP**:

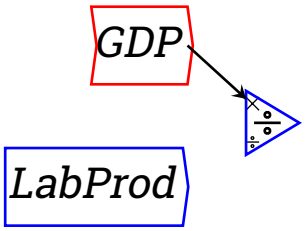


Now we need to divide **GDP** by **LabProd**. Click on the symbol on the palette and the symbol will be attached to the cursor. Drag it near the other two objects and click. Your Canvas will now look something like this:

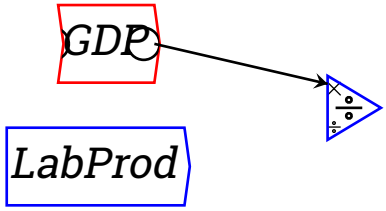


Now to complete the equation, you have to attach **GDP** to the top of the divide block and LabProd to the bottom.

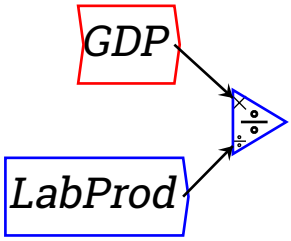
Now move your cursor to the right hand side of GDP and click, hold the mouse button down, and drag. An arrow will come out from GDP. Drag this arrow to the top of the divide block (where you'll see a tiny multiply sign) and release the mouse. You should then see this:



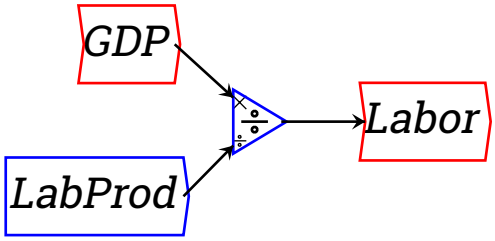
When the mouse hovers over a block, you will then see little circles that identify the input and output ports of the block:



Those are the connection points for wires, so start dragging from one and release on the other. Now wire LabProd to the bottom of the Divide block (where you'll see a miniature divide symbol (blown up below):



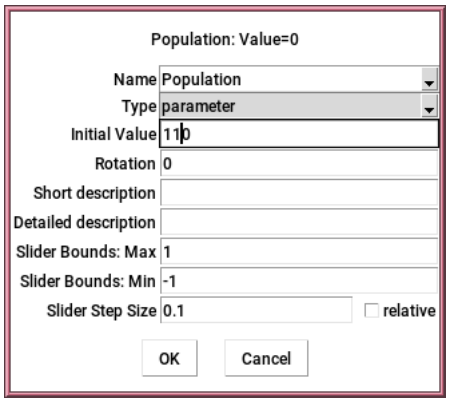
Then click on var in the Design Icons to create a new variable, call it Labor, place it the the right of the Divide block, and wire the output port from the Divide block to the input port for ***Labor***:



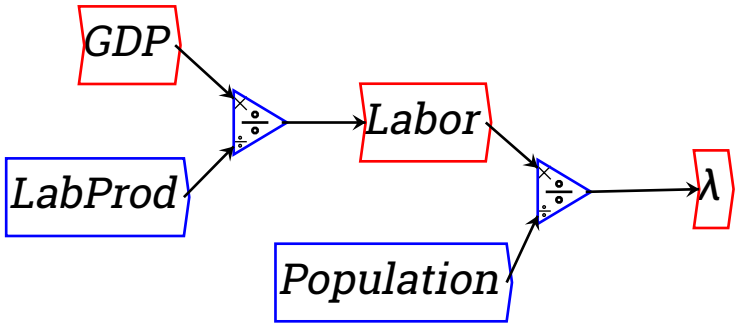
To show the correspondence between the flowchart above and standard modeling equations, click on the equations tab:

$$\begin{aligned} \text{GDP} &= \\ \text{Labor} &= \frac{\text{GDP}}{\text{LabProd}} \end{aligned}$$

Now let's keep going with the model. With ***Labor*** defined, the employment rate will be ***Labor*** divided by ***Population***. Define ***Population*** as a parameter (we'll later change it to a variable), and give it a value of 110.



Add it to the Canvas and you are now ready to define the employment rate—another variable. Click on var, give it the name "\lambda" (be sure to include the backslash symbol), put another Divide block on the canvas, choose Wire mode and wire this next part of the model up. You should now have:



Now switch to the equations tab, and you will see

$$\begin{aligned} \text{GDP} &= \\ \text{Labor} &= \frac{\text{GDP}}{\text{LabProd}} \\ \lambda &= \frac{\text{Labor}}{\text{Population}} \end{aligned}$$

Notice that Minsky outputs a Greek λ in the equation. You can input such characters directly, if your keyboard supports them as unicode characters, however you can also use a subset of the LaTeX language to give your variables more mathematial names.

With the employment rate defined, we are now ready to define a "Phillips Curve" relationship between the level of employment and the **rate of change** of wages. There was far more to Phillips than this (he actually tried to introduce economists to system dynamics back in the 1950s), and far more to his

employment-wage change relation too, and he insisted that the relationship was nonlinear (as in Goodwin's figure above). But again for simplicity we'll define a linear relationship between employment and the rate of change of wages.

Here we need to manipulate the basic linear equation that Goodwin used:

$$\frac{1}{w}\frac{d}{dt}w = -\gamma + \rho \cdot \lambda$$

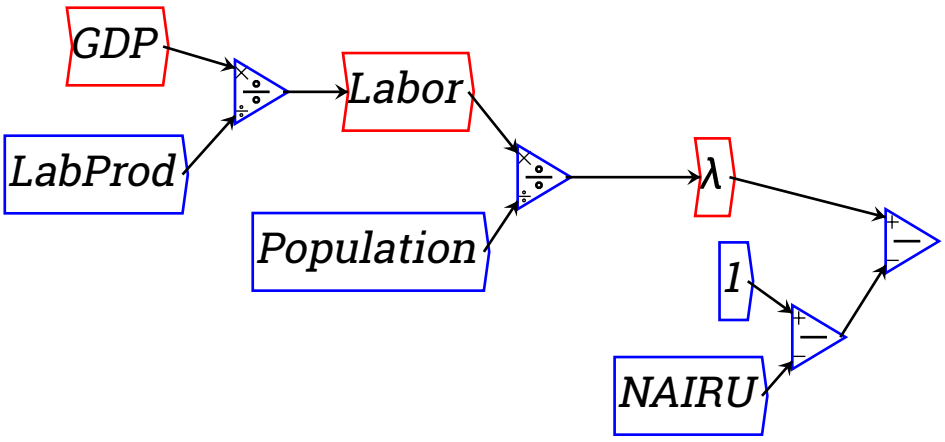
Firstly multiply both sides by w:

$$\frac{d}{dt}w = w \cdot (-\gamma + \rho \cdot \lambda)$$

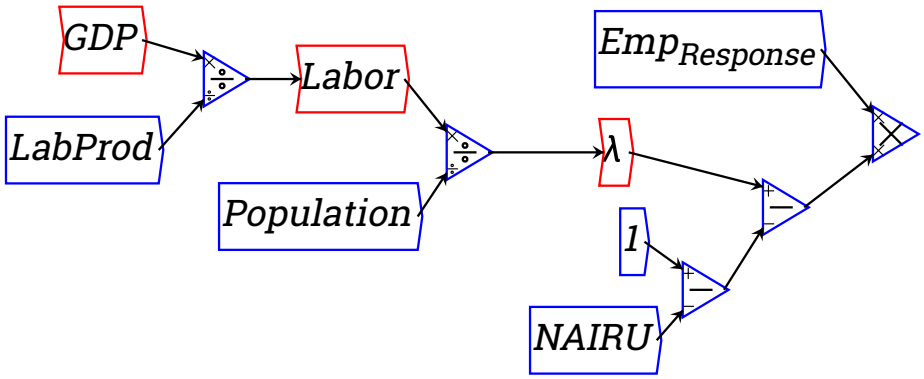
Then integrate both sides (because integration is a numerically much more stable process than differentiation, all system dynamics programs use integration rather than differentiation):

$$w = w_0 + \int w \cdot (-\gamma + \rho \cdot \lambda)$$

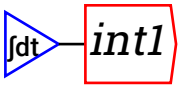
In English, this says that the wage now is the initial wage plus the integral of the wage multiplied by its rate of change function. That's what we now need to add to the Canvas, and the first step is to spell out the wage change function itself. Firstly, since we're using a linear wage response function, the rate of employment has to be referenced to a rate of employment at which the rate of changes is zero. I suggest using Milton Friedman's concept of a "Non-Accelerating-Inflation-Rate-of-Unemployment", or NAIRU. We need to define this constant, subtract it from 1, and subtract the result from the actual employment rate λ. To enter 1, click on const, define a constant and give it a value of 1. Then define another variable NAIRU, and give it a value of 0.05 (5% unemployment). Select "parameter" as the variable type. Subtract this from 1 and subtract the result from λ. You should have the following:



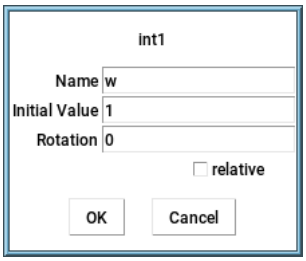
Now we need to multiply this gap between the actual employment rate and the "NAIRE" rate by a parameter that represents the response of wages to this gap. Let's call this parameter ***Emp_{Response}*** (remember to include the underscore and the braces). Define the parameter, give it a value of 10, and multiply (λ minus NAIRE) by it:



Now we are ready to add a crucial component of a dynamic model: the integral block, which takes a flow as its input and has the integral of that flow as the output. The wage rate ***w*** is such a variable, and we define it by clicking on the ▷ symbol in the Icon Palette (or by choosing Operations/Integrate from the *Insert* menu). This then attaches the following block to the cursor:



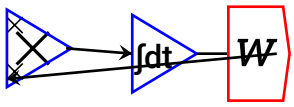
Now we need to rename this from the default name of "int1" to "**w**" for the wage rate. Either right click or double-click on "int1" and this will bring up the edit window . Rename it to "w" and give it a value of 1:



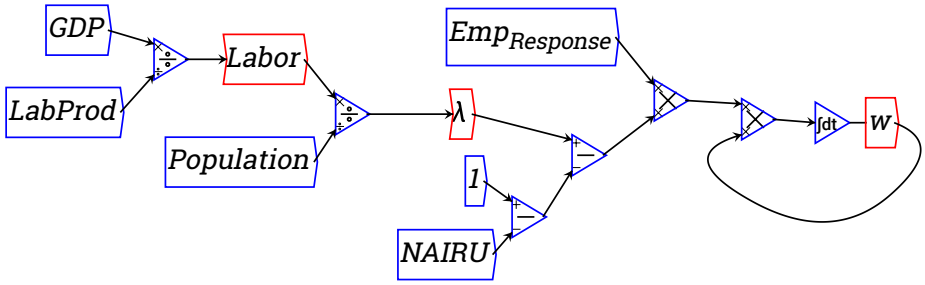
To compete the integral equation, we need to multiply the linear employment response function by the current wage before we integrate it (see the last equation above). There are two ways to do this. First, place a multiply block between the wage change function and the integral block, wire the function up to one input on the multiply block, and then either:

- wire the output of the ***w*** block back to the other input on multiply block; or
- Right-click on ***w***, choose "Copy Var", place that copy before the multiply block, and wire it up.

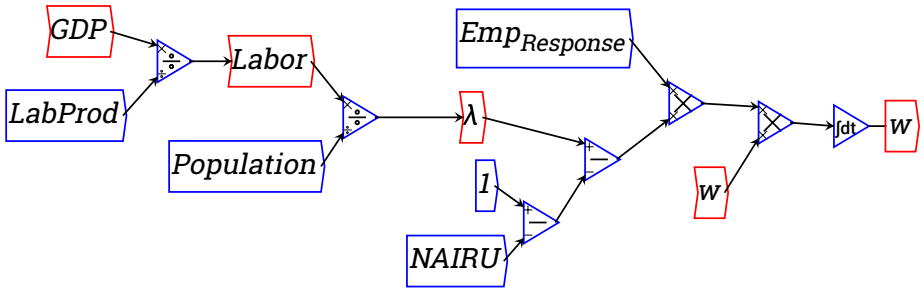
The first method gives you this initial result:



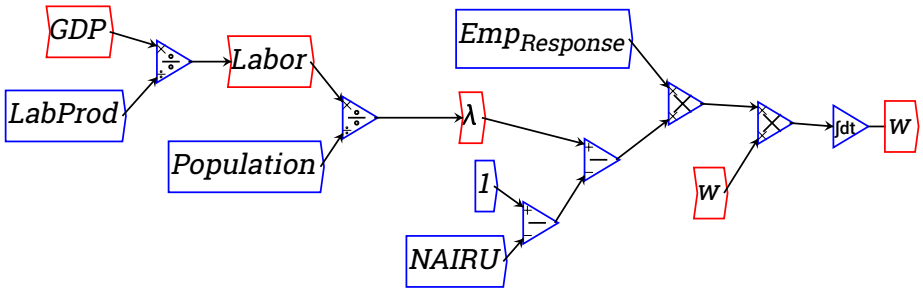
That looks messy, but notice the blue dot on the wire? Click and drag on that and you will turn the straight line connector into a curve:

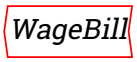


The second approach, which I personally prefer (it's neater, and it precisely emulates the integral equation), yields this result:



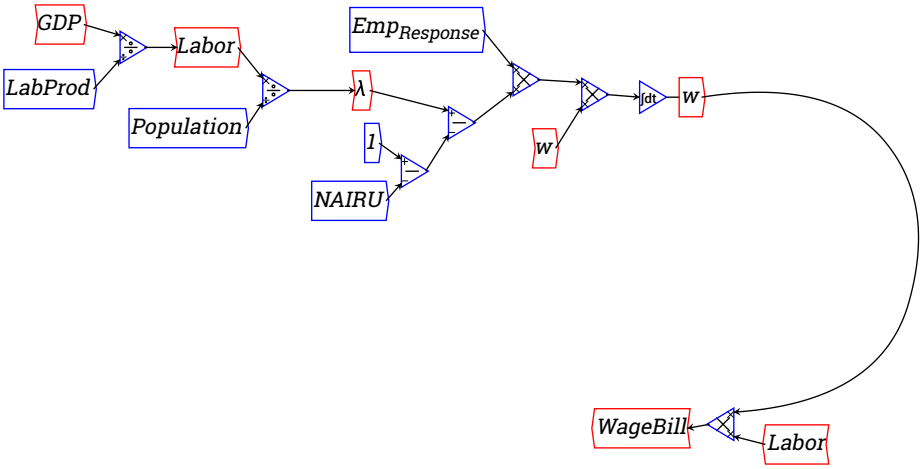
From this point on the model develops easily—"like money for old rope", as one of my maths lecturers used to say. Firstly if we multiply the wage rate ***w*** by ***Labor*** we get the **Wage Bill**. To do this, firstly create the variable Wage Bill, and put it well below where w currently is on your diagram:



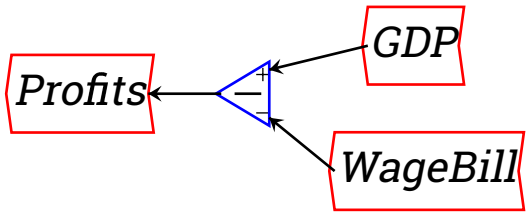


Now right-click on ***WageBill*** and choose "Flip". This rotates the block through 180 degrees (any arbitrary rotation can be applied from the variable definition window itself). Now right-click on ***Labor***, which you've already defined some time ago, and choose "Copy". Place the copy of ***Labor*** to the right of ***WageBill***:

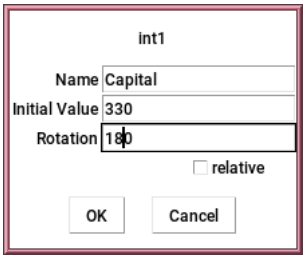
Now insert a multiply block before ***WageBill***, and wire ***w*** and ***Labor*** up to it. Curve the wire from w using the blue dots (you can do this multiple times to create a very curved path: each time you create a curve, another 2 curve points are added that you can also manipulate, as I have done below:



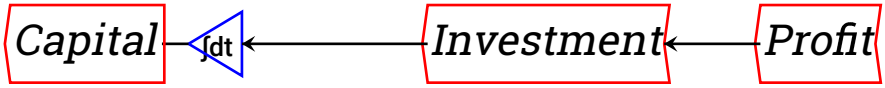
The next step is to subtract the ***WageBill*** from ***GDP*** to define ***Profits***. Take a copy of ***GDP***, insert it above ***WageBill***, insert a subtract block, and wire it up to define the variable ***Profits***:



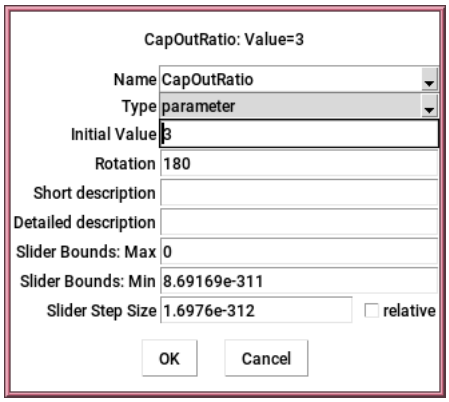
In the simple Goodwin model, all Profits are invested, and investment of course is the rate of change of the capital stock Capital. Create a variable called Investment, wire this up to Profits, and then create a new integral variable *Capital* using the ∫dt icon. Right-click or double-click on it to rename *int2* to *Capital*, and give it an initial value of 300:



Wire this up to Investment:

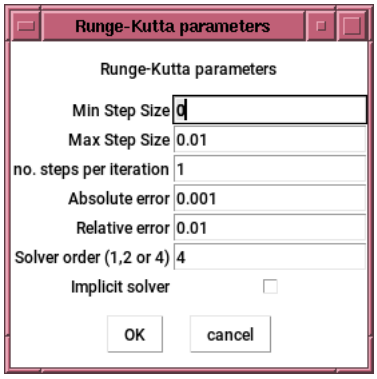


Now there's only one step left to complete the model: define a parameter CapOutputRatio and give it a value of 3:

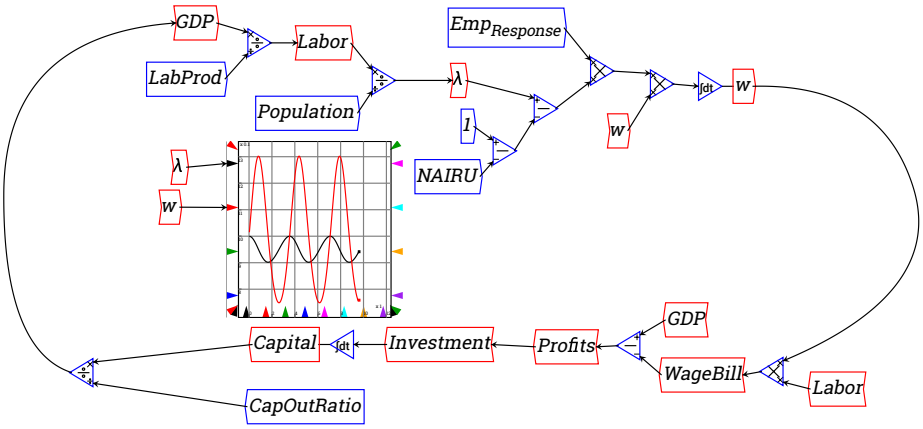


Divide Capital by this, and wire the result up to the input on GDP. You have now built your first dynamic model in Minsky:

Before you attempt to run it, do two things. Firstly from the *Runge Kutta* menu item, change the Max Step Size to 0.01—to get a smoother simulation.



Secondly, add some graphs by clicking on the  icon, placing the graph in the middle of the flowchart, and wiring up λ and w to two of the four inputs on the left hand side. You will now see that, rather than reaching equilibrium, the model cycles constantly:



If you click on the equations tab, you will see that you have defined the following system of equations:

$$
\begin{aligned}
\text{GDP} &= \frac{\text{Capital}}{\text{CapOutRatio}} \\
\text{Investment} &= \text{Profits} \\
\text{Labor} &= \frac{\text{GDP}}{\text{LabProd}} \\
\text{Profits} &= \text{GDP} - \text{WageBill} \\
\text{WageBill} &= w \times \text{Labor} \\
\lambda &= \frac{\text{Labor}}{\text{Population}} \\
\omega &= \frac{\text{WageBill}}{\text{GDP}} \\
\frac{dw}{dt} &= \text{Emp}_{\text{Response}} \times (\lambda - (1 - \text{NAIRU}) \times w \\
\frac{d\text{Capital}}{dt} &= \text{Investment}
\end{aligned}
$$

At this level of complexity, the equation form—if you're accustomed to working in equations—is as accessible as the block diagram model from which it was generated. But at much higher levels of complexity, the block diagram is far easier to understand since it displays the causal links in the model clearly, and can be structured in sub-groups that focus on particular parts of the system.

8.2 Basic Banking model

If you haven't yet read the section on Creating a Banking Model, do so now. This tutorial starts from the skeleton of the "Loanable Funds" model built in that section, and using time constants to specify how quickly lending occurs.

8.2.1 Loanable Funds

Our model begins with the single operation of Patient lending to Impatient at a rate that, if kept constant at its initial level of of $10 per annum, would empty the Patient account in 10 years. Because the rate of outflow declines as the Patient account declines, the money in the account decays towards zero but never quite reaches it.

		Asset	Liability		Equity	
	Flows ↓ / Stock Vars →	Reserves▼	Patient▼	Impatient▼	Safe ▼	A-L-E
+ — ↓	Initial Conditions	120	100	0	20	0
+ — ↑	Patient lends to Impatient		-Lend	Lend		0

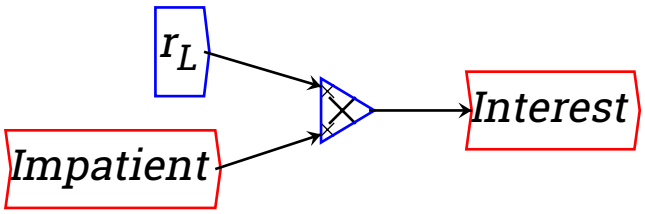
Many more actions need to be added to this model to complete it. For a start, Impatient should be paying interest to Patient on the amount lent. Add an additional row to the Godley Table by clicking on the '+' key next to "Patient lends to Impatient" to create a blank row:

		Asset	Liability		Equity	
	Flows ↓ / Stock Vars →	Reserves▼	Patient▼	Impatient▼	Safe ▼	A-L-E
+ — ↓	Initial Conditions	120	100	0	20	0
+ — ↑ ↓	Patient lends to Impatient		-Lend	Lend		0
+ — ↑						0

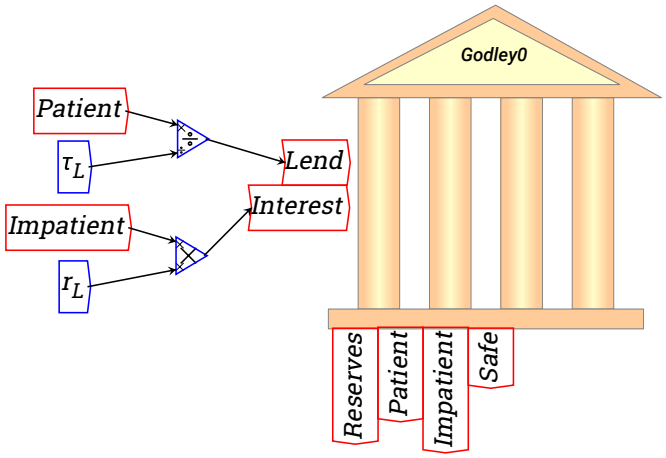
Then label this flow "Impatient pays interest" and make the entry "Interest" into the cell for Patient on that row. Make the matching entry "-Interest" in the cell for Impatient. The flow "Interest" now appears on the input side of the Godley Table on the Canvas:

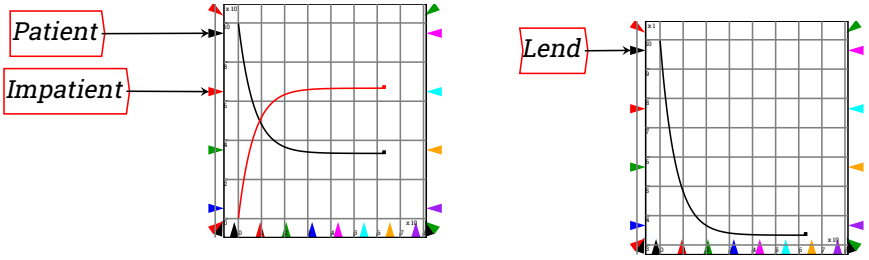
		Asset	Liability		Equity	
	Flows ↓ / Stock Vars →	Reserves▼	Patient▼	Impatient▼	Safe ▼	A-L-E
+ — ↓	Initial Conditions	120	100	0	20	0
+ — ↑ ↓	Patient lends to Impatient		-Lend	Lend		0
+ — ↑	Impatient pays interest		Interest	-Interest		0

Interest now has to be defined. It will be the amount in Impatient's account (since this began at zero) multiplied by the rate of interest charged by Patient:



With that definition, the dynamics of the model change: rather than the Patient account falling to zero and Impatient rising to 100, the two accounts stabilize once the outflow of new loans by Patient equals the inflow of interest payments by Impatient:



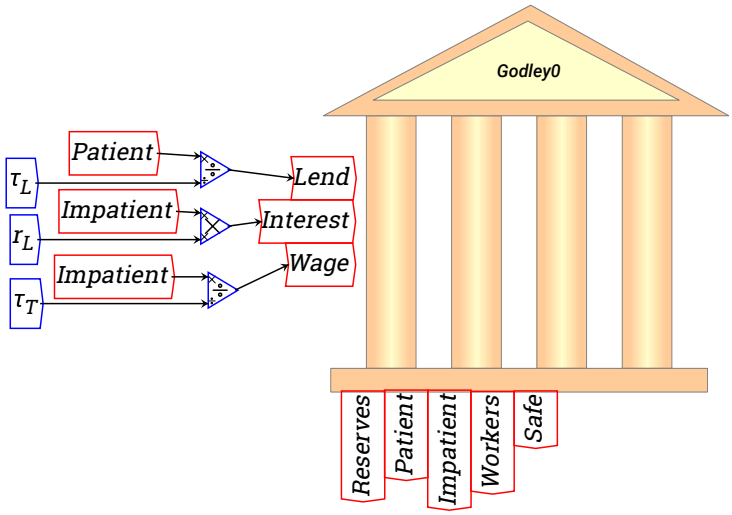


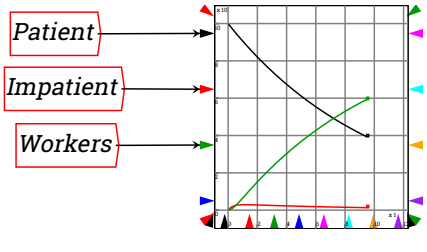
Though it stabilizes, this is is still a very incomplete model: neither Patient nor Impatient are doing anything with the money apart from lending it and paying interest. I am now going to assume that Impatient is borrowing the money in order to hire workers to work at a factory and produce output for sale. So we now need another account called Workers, and a payment from Impatient to Workers called Wage:

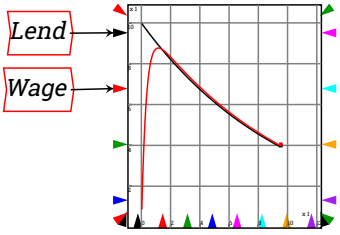
		Asset		Liability		Equity	
Flows ↓ / Stock Vars →	Reserves▼	Patient▼	Impatient▼	Workers▼	Safe ▼	A-L-E
Initial Conditions	120	100	0	0	20	0
Patient lends to Impatient		-Lend	Lend			0
Impatient pays interest		Interest	-Interest			0
			-Wage	Wage		0

In a more complex model, the Wage bill could be related to the current rate times the number of workers in employment. In this simple model I will regard the wage as a function of the amount of money in Impatient's account turning over several times a year in the payment of wages. Using a time constant, I will assume that the amount in Impatient's account turns over 3 times a year paying wages, so that the time constant τ_T is 1/3rd of a year:

The dynamics of this incomplete model are very different again: very little money turns up in the Impatient account, and all of the money ends up in the Workers account. However economic activity also ceases as both lending and the flow of wages falls towards zero:







This is because wages are being paid to workers, but they are doing nothing with it. So we need to include consumption by workers–and by Patient as well. Here the reason time constants are useful may be more obvious. The time constant for consumption by Workers is given the very low value of 0.05—or 1/20th of a year—which indicates that if their initial rate of consumption was maintained without any wage income, they would reduce their bank balances to zero in 1/20th of a year or about 2.5 weeks.