Chapter 1

Introduction

1.1 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.

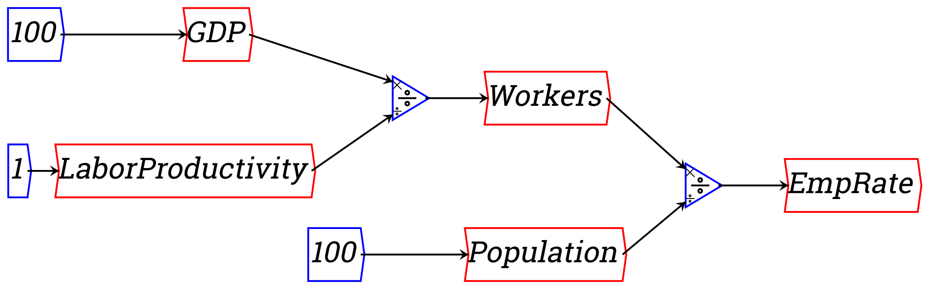
Minsky adds another means to create the dynamic equations that are needed to define monetary flows—the "Godley Table"—which is discussed in the next section for users who are experienced in system dynamics. In this section, we'll give you a quick overview of the generic system dynamics approach to building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

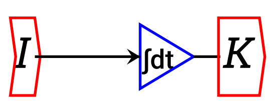
These causal chains always involve a relationship between stocks and flows. Economists normally model stocks and flows by adding an increment to a stock. For example, the level of capital K is defined as a difference equation, where capital in year t is shown as being capital in year $t-1$ plus the investment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

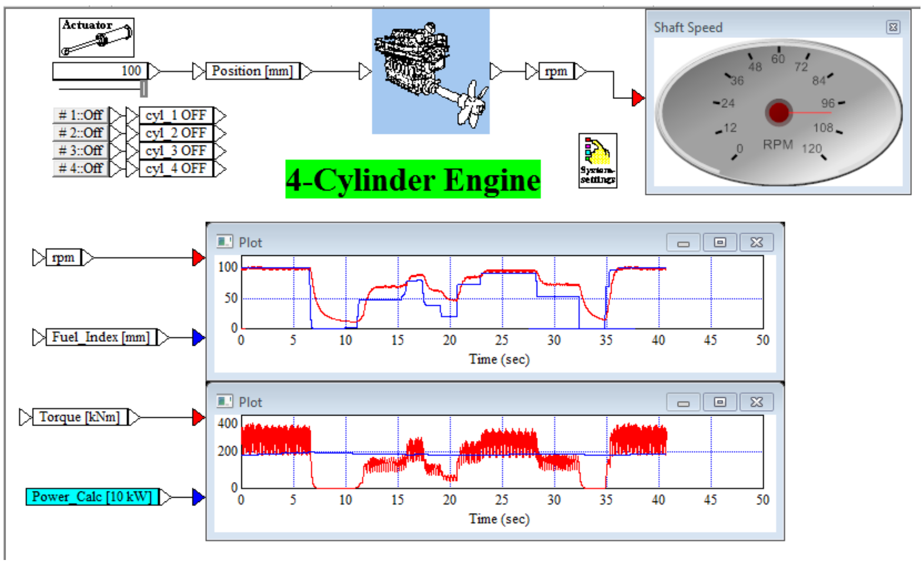


The advantages of the block diagram representation of dynamic equations over a list of equations are:

- 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:



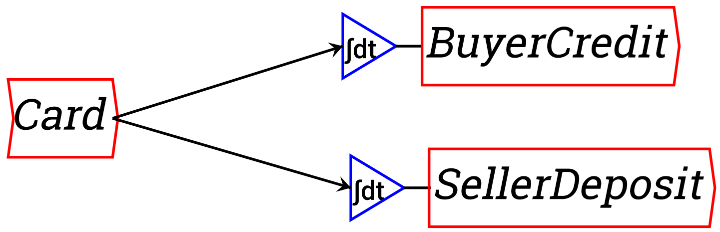
Programs like Vissim and Simulink have been in existence for almost 2 decades, and they are now mature products that provide everything their user-base of engineers want for modeling and analyzing complex dynamic systems. So why has Minsky been developed?

1.2 Experienced in system dynamics?

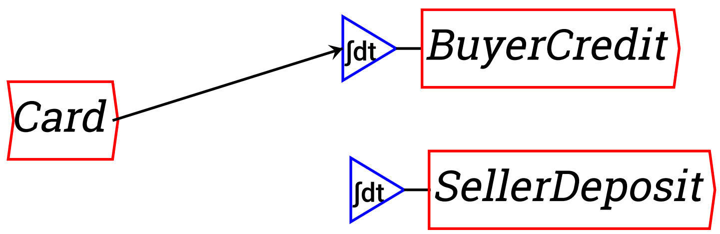
As an experienced system dynamics user (or if you've just read "New to system dynamics?"), what you need to know is what Minsky provides that other system dynamics programs don't. That boils down to one feature: The Godley Table. It enables a dynamic model of financial flows to be derived from a table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

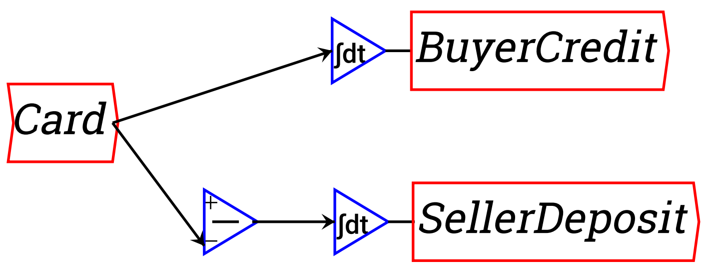
For example, if you go shopping and buy a new computer with your credit card, you increase your debt to a bank and simultaneously increase the deposit account of the retailer from whom you buy the computer. The two system states in this model—your credit card ("BuyerCredit") and the retailer's deposit 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 sum to zero. The source is shown as a "+", the sink is shown as a "−", and assets are shown as a positive sum while liabilities are shown as a negative. 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 2

Getting Started

2.1 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.

2.2 Getting help

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

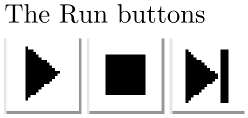
2.3 Components of the Program

There are 6 components to the Minsky interface:

1. The menus.

   File Edit Insert Options Runge Kutta Help

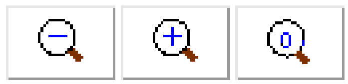
2. The Run buttons

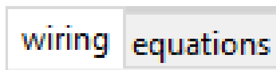
3. The simulation speed slider

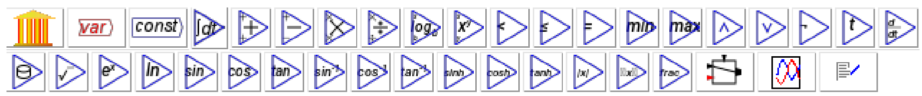
4. The Zoom buttons



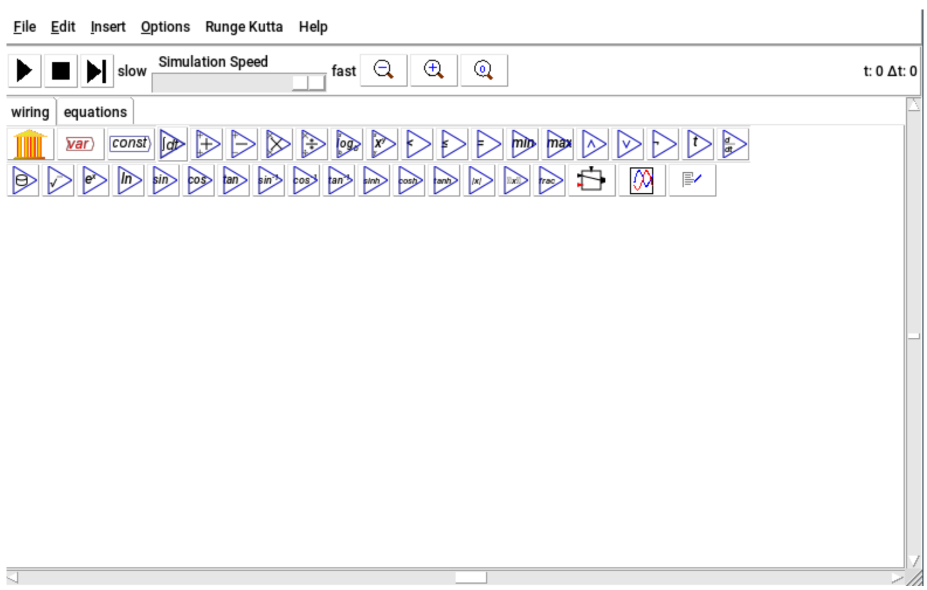
5. The Wiring and Equation tabs



6. The design icons



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



2.3.1 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 (as at the August 2016 Cantillon release) 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

Output LaTeX 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.

Output MatLab Output a MatLab function that 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.

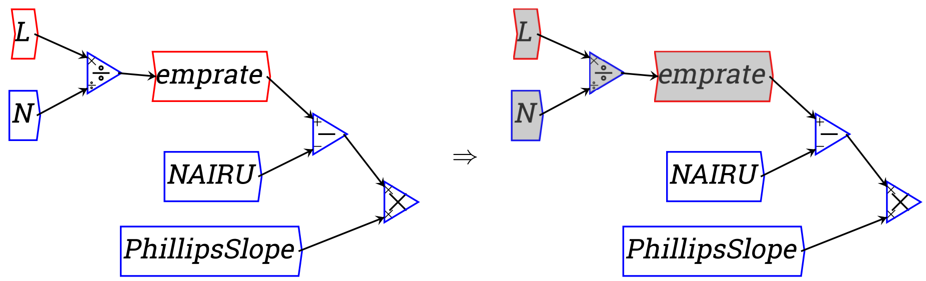
Debugging use Items under the line are intended for developer use, and will not be documented here. Redraw may be useful if the screen gets messed up because of a bug.

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.

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

- 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 double entry. Applies double entry book keeping semantics to the Godley table, where assets and liabilites have opposite mathematical meaning. See the Godley table section for more details. Unchecking this option reverts to a simpler mode where each stock (Godley table column) is treated the same.
- 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.

- 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.

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

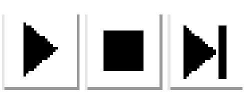
Runge Kutta

- 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 of the screen, increasing the number of solver steps per iteration decreases the overhead involved in updating the display, at the expense of smoothness of the plots.

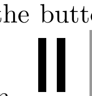
Help

Provides an in-program link to this manual.

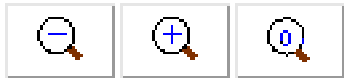
2.3.2 Run Buttons



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.

2.3.3 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.

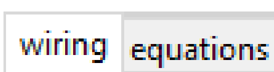
2.3.4 Speed slider

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.

2.3.5 Simulation time

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

2.3.6 Wiring and Equations Tabs



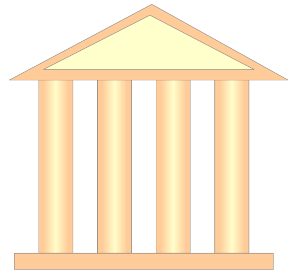
This allows you to switch between the visual block diagram wiring view and the more mathematical equations view.

2.3.7 Design Icons

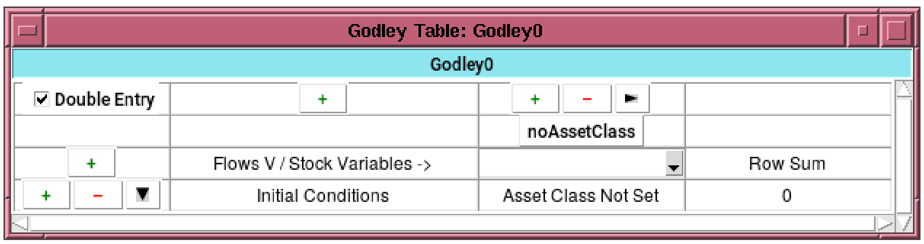
These are the "nuts and bolts" of any system dynamics program. The number of icons will grow over time, but the key ones are implemented now:

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:



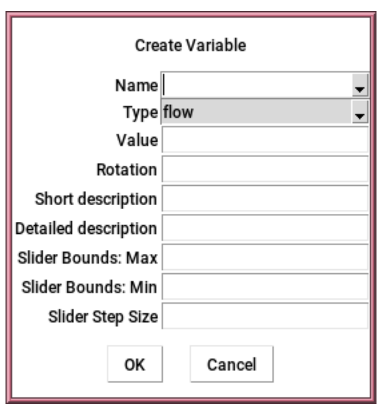
Double-click on the Bank Icon (or right-click and choose "Open Godley Table" from the context menu) and you get a double-entry bookkeeping table we call a Godley Table, which looks like the following onscreen:



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

Variable 

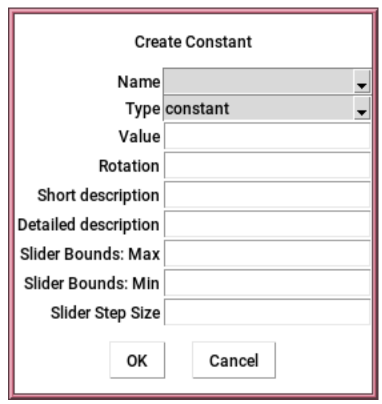
. This creates an entity whose value changes as a function of time and its relationship with other entities in your model. Click on it and a variable definition window will appear:



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.

Constant const creates an entity whose value is unaffected by the simulation or other entities in the model—but it can be varied during a simulation run by the user. Click on it and a constant definition window will appear:

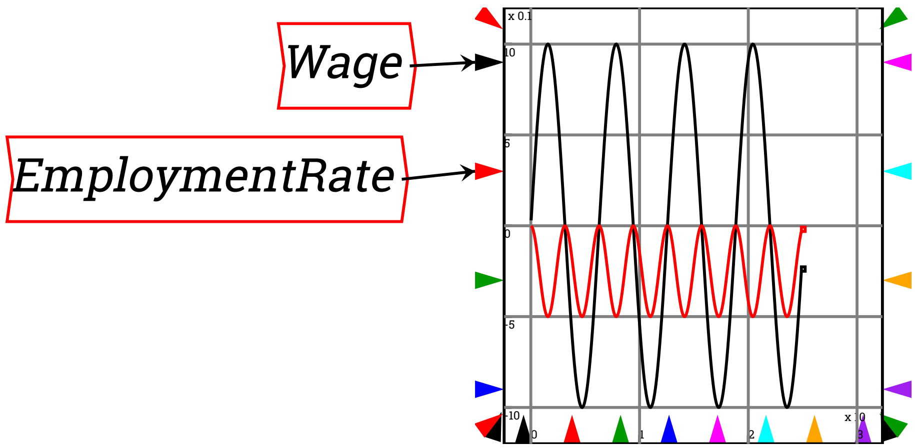


The only essential element here is its value. You can also specify its rotation on the Canvas in degrees, and its slider parameters if you make the slider active. This lets you vary a parameter while a simulation is running—which is useful if you wish to explore a range of policy options while a model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

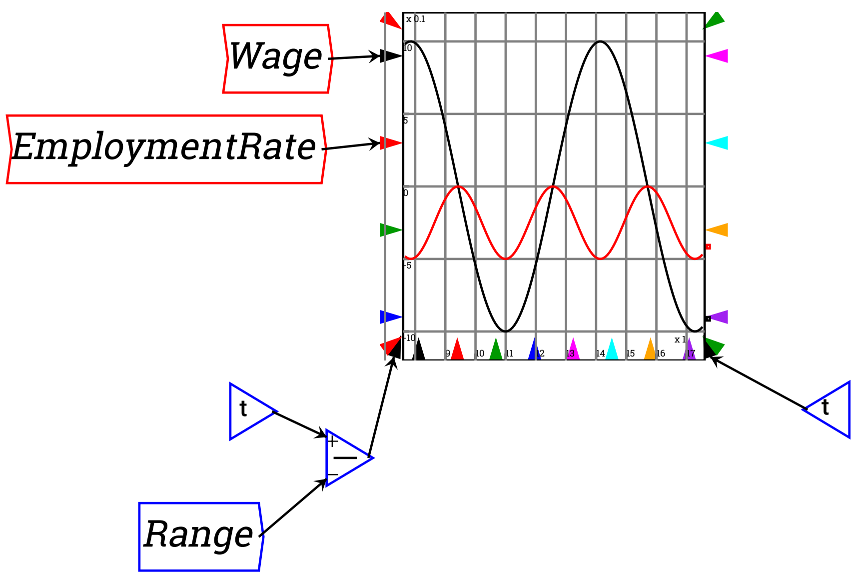
A constant is just a type of variable, which also include parameters (named constants), flow variables, stock variables and integration variables. In fact there is no real conceptual difference between creating a constant or creating a variable, as you can switch the type using the type field.

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.

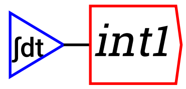
For example, by default a graph displays the simulation time on the horizontal axis, so that cycles get compressed as a simulation runs for a substantial period:



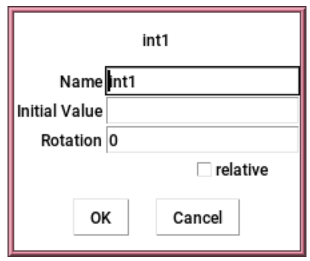
If a Time block is added to the marker for the x-axis range, you can control the number of years that are displayed. This graph is set up to show a ten year range of the model only:



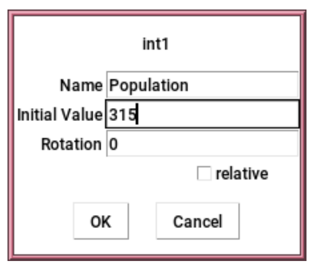
Integration . This inserts a variable whose value depends on the integral of other variables in the system. This is the essential element for defining a dynamic model. Click on it and the following entity will appear at the top left hand side of the canvas (and move with your mouse until you click to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

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

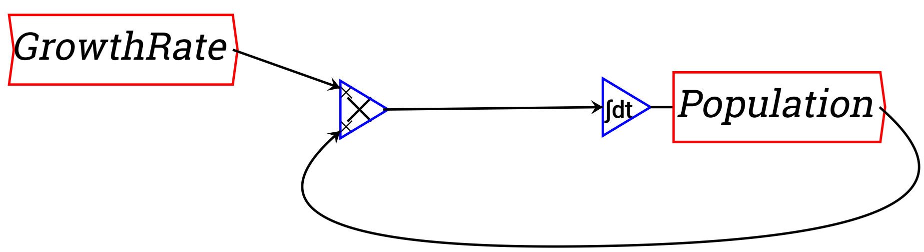
To express this as an integral equation, firstly we multiply both sides of this equation by Population to get:

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

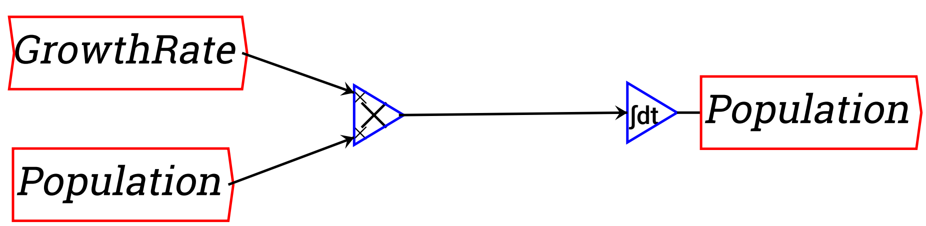
Then we integrate both sides to get an equation that estimates what the population will be T years into the future as:

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

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.

Plus, Minus, Multiply and Divide blocks . 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.

Power and Logarithm and 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)** , , , , and . These return 0 for false and 1 for true.

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

Data block A data block interpolates a sequence of empirical values, which may be generated outside of Minsky, and imported as a CSV file. This effectively defines a piecewise linear function.

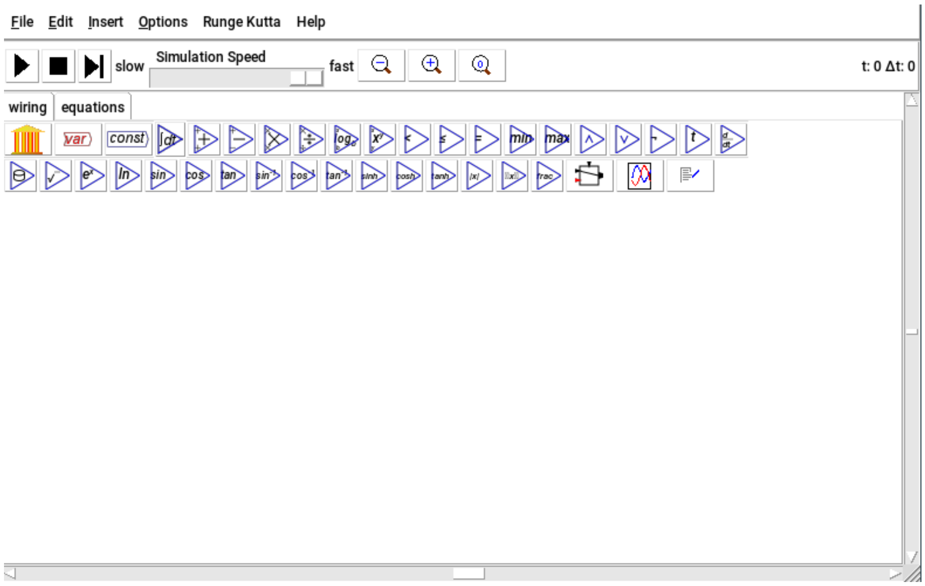
Plot widget Add plots to the canvas.

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

Notes Add textual annotations

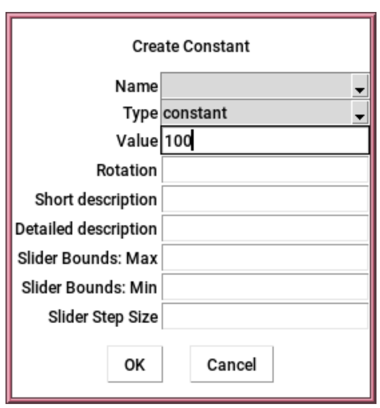
2.3.8 Design Canvas

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.

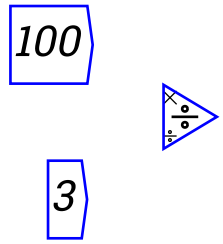


2.3.9 Wires

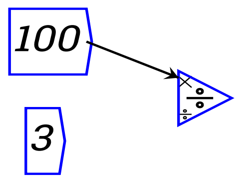
The wires in a model connect blocks together to define equations. For example, to write an equation for 100/33, you would place a  on the canvas, and give it the value of 100:



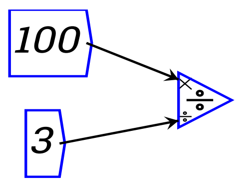
Then do the same for 33, and place a divide block on the canvas:



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



Finally, add the other wire.



2.4 Working with Minsky

2.4.1 Components in Minsky

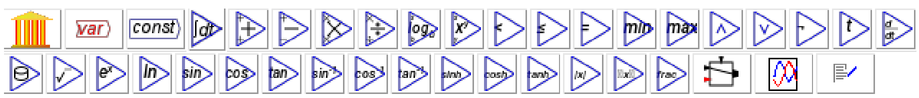
There are a number of types of components in Minsky

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.

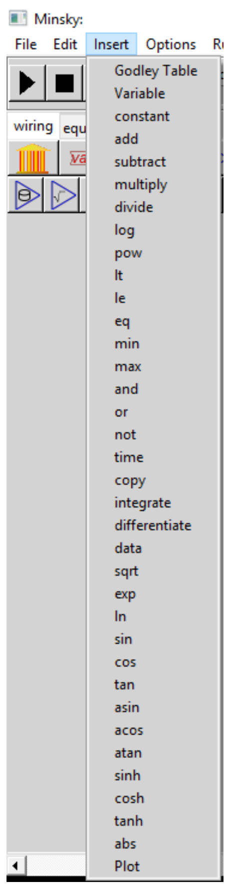
2.4.2 Inserting a model component

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

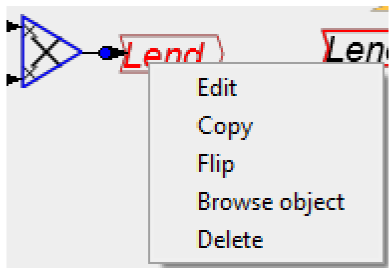
1. Click on the desired Icon on the Icon Palette, drag the block onto the Canvas and release the mouse where you want to insert it



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



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



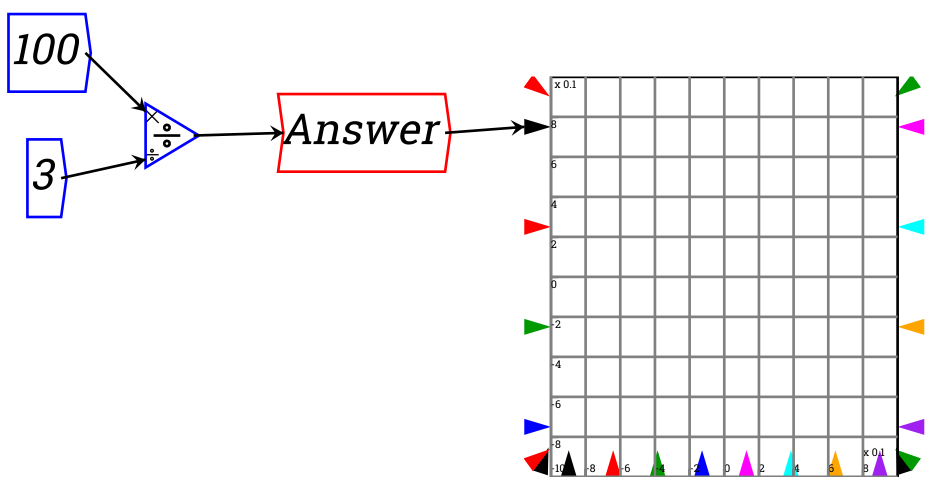
2.4.3 Creating an equation

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:

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

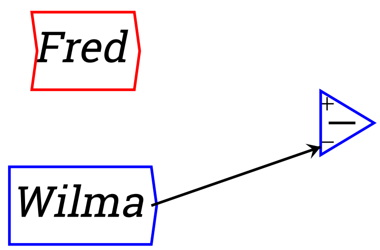
Very complex equations—including dynamic elements like integral blocks and Godley Tables—are designed by wiring up lots of components, with the output of one being the input of the next. See the tutorial for examples.

2.4.4 Wiring components together

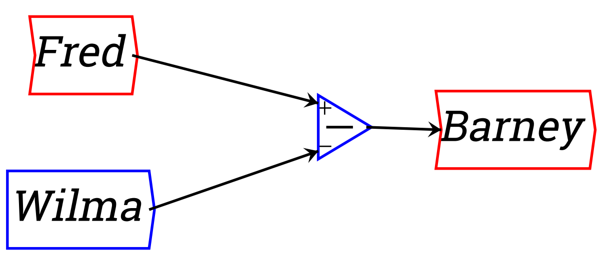
A model is constructed by wiring one component to another in a way that defines an equation. Wires are drawn from the output port of one block to the input port of another. Ports are circles on the blocks to which wires can be attached, which can be seen when hovering the pointer over the block. Variables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



2.4.5 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.

When a Godley Table is first loaded, it has room for one account 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.

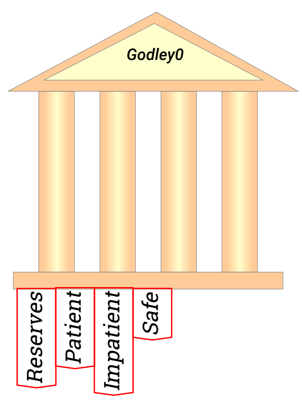
Flows ↓ / Stock Variables →		
	noAssetClass	
Initial Conditions	0	0

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 V / 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:

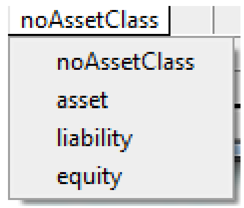
Flows ↓ / Stock Variables →	*Reserves*	*Patient*	*Impatient*	*Safe*
	noAssetClass			
Initial Conditions	0	0	0	0

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



2.4.6 Defining account types

Bank accounts must be classified as either an Asset, a Liability, or the Equity of the relevant Bank, using the drop-down menu currently labeled `noAssetClass` at the top of each account. In this model, Reserves are an asset of the banking sector, the accounts of "Patient" and "Impatient" are liabilities, and the "Safe" is the equity of the banking system. Click on the `noAssetClass` button and this drop-down menu will appear:



Choose the relevant entry for each column, and the accounts will be properly classified when the model is simulated:

Flows ↓ / Stock Variables →	*Reserves*	*Patient*	*Impatient*	*Safe*
	asset	liability		equity
Initial Conditions	0	0	0	0

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.

Flows ↓ / Stock Variables →	*Reserves*	*Patient*	*Impatient*	*Safe*
	asset	liability		equity
Initial Conditions	0	0	0	0

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

Flows ↓ / Stock Variables →	*Reserves*	*Patient*	*Impatient*	*Safe*
	asset	liability		equity
Initial Conditions	0	0	0	0
Patient lends to Impatient				

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.

Flows ↓ / Stock Variables →	*Reserves*	*Patient*	*Impatient*	*Safe*
	asset	liability		equity
Initial Conditions	0	0	0	0
Patient lends to Impatient		*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 sum to zero. This is because a destination for "Lend" has not yet been specified. The destination is Impatient's account, and to balance the row to zero this part of the transaction must be entered as "-Lend":

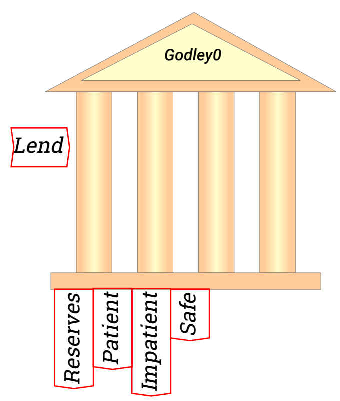
Flows ↓ / Stock Variables →	$Reserves$	$Patient$	$Impatient$	$Safe$
	asset	liability		equity
Initial Conditions	0	0	0	0
Patient lends to Impatient		$Lend$	$-Lend$	

This might appear strange if you are not used to accounting standards—"shouldn't the Patient account fall because of the loan, while the Impatient account should rise?"—but what is shown in the table makes sense, because all accounts are perceived from the Bank's point of view. Deposits at a bank are liabilities for the bank, and are shown as a negative amount, while assets are recorded as a positive amount. So a loan from Patient to Impatient decreases the Bank's liabilities to Patient, and increases the Bank's liabilities to Impatient.

The same rule applies to the Initial Conditions (the amount of money in each of the accounts prior to the flows between accounts): the Initial Conditions must sum to zero. This requires that there are entries on the Asset side of the Banking ledger that exactly match the sum of Liabilities and Equity (Equity is also shown as a negative in double-entry bookkeeping):

Flows ↓ / Stock Variables →	$Reserves$	$Patient$	$Impatient$	$Safe$
	asset	liability		equity
Initial Conditions	120	-100	0	-20
Patient lends to Impatient		$Lend$	$-Lend$	

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

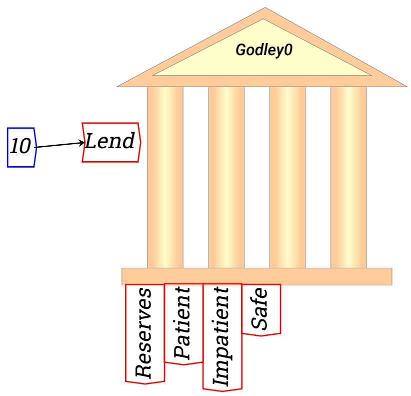


Defining flows

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:

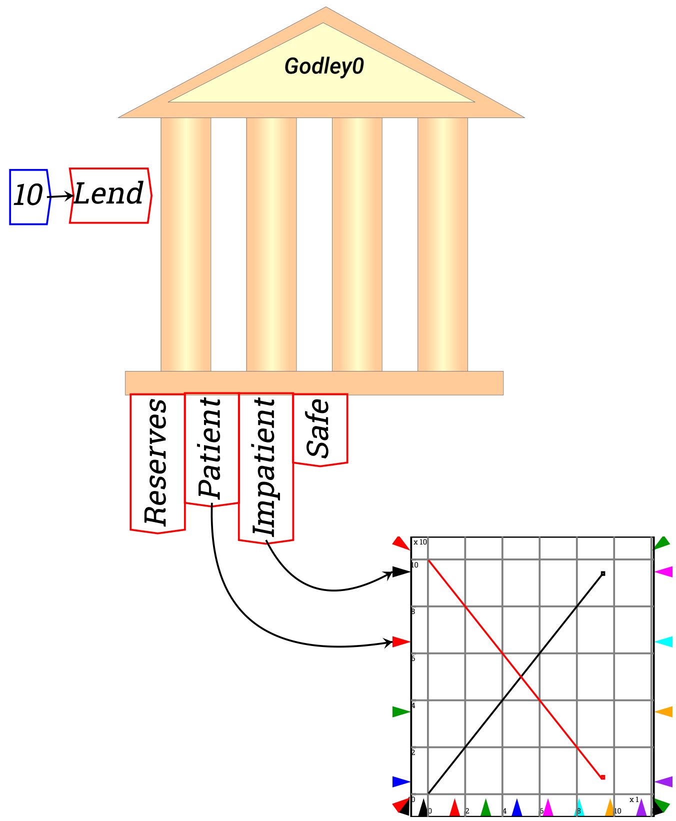
Create a constant with a value of 10, and attaching this to the input side of Lend:



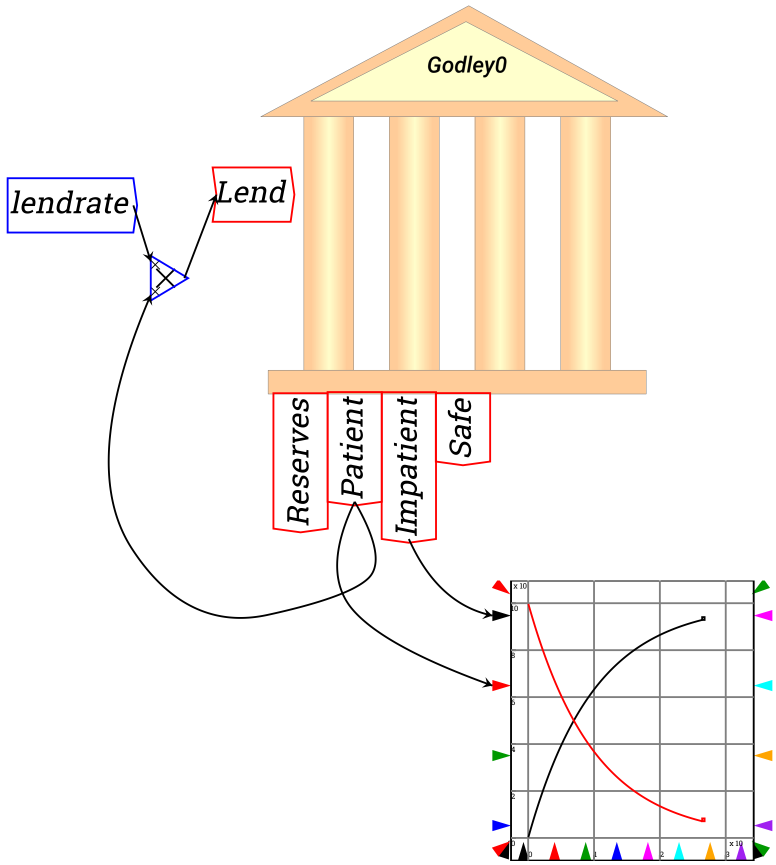
What you have now defined is an annual flow from Patient to Impatient of $10. In the dynamic equations this model generates, Minsky converts all amounts in accounts to positive sums—it shows the financial system from the point of the overall economy, rather than from the point of view of the bank:

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

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 Patient'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.



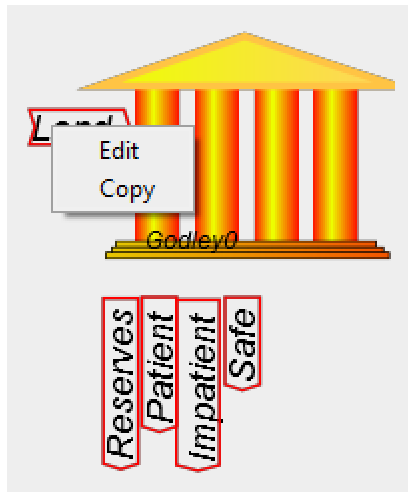
This is because the equation you have defined is identical to a radioactive decay equation, with the amount in the Patient account falling at 10 percent per year:

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

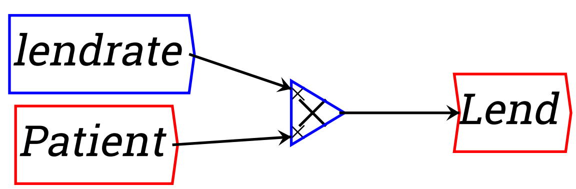
Note however that there are now wires crossing over other wires? There is a neater way to define flows.

Copying Godley Table input & outputs

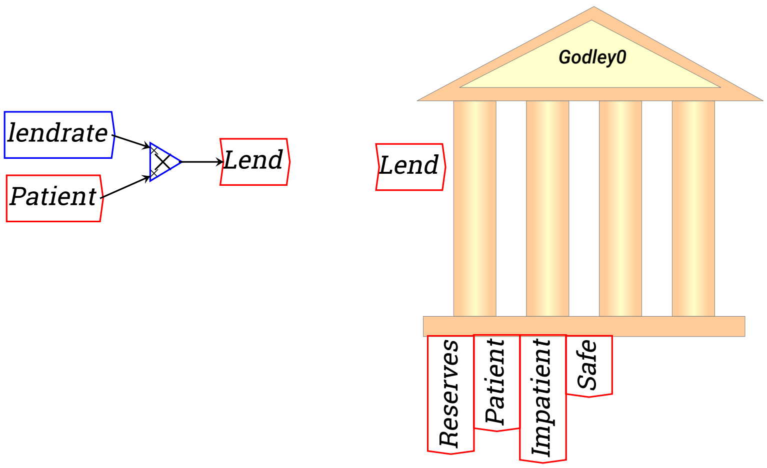
Right-click on the inputs and outputs of a Godley Table and choose “copy” from the drop-down menu:

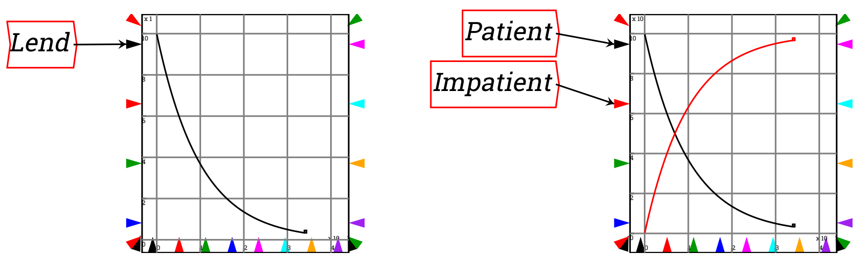


Place the copied flows and accounts and place them away from the table. Then wire up your definition there:



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

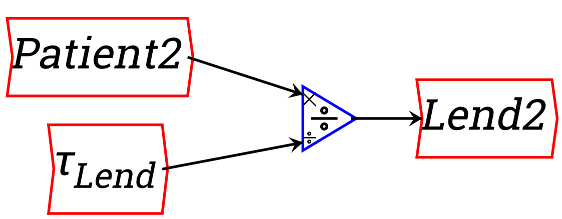




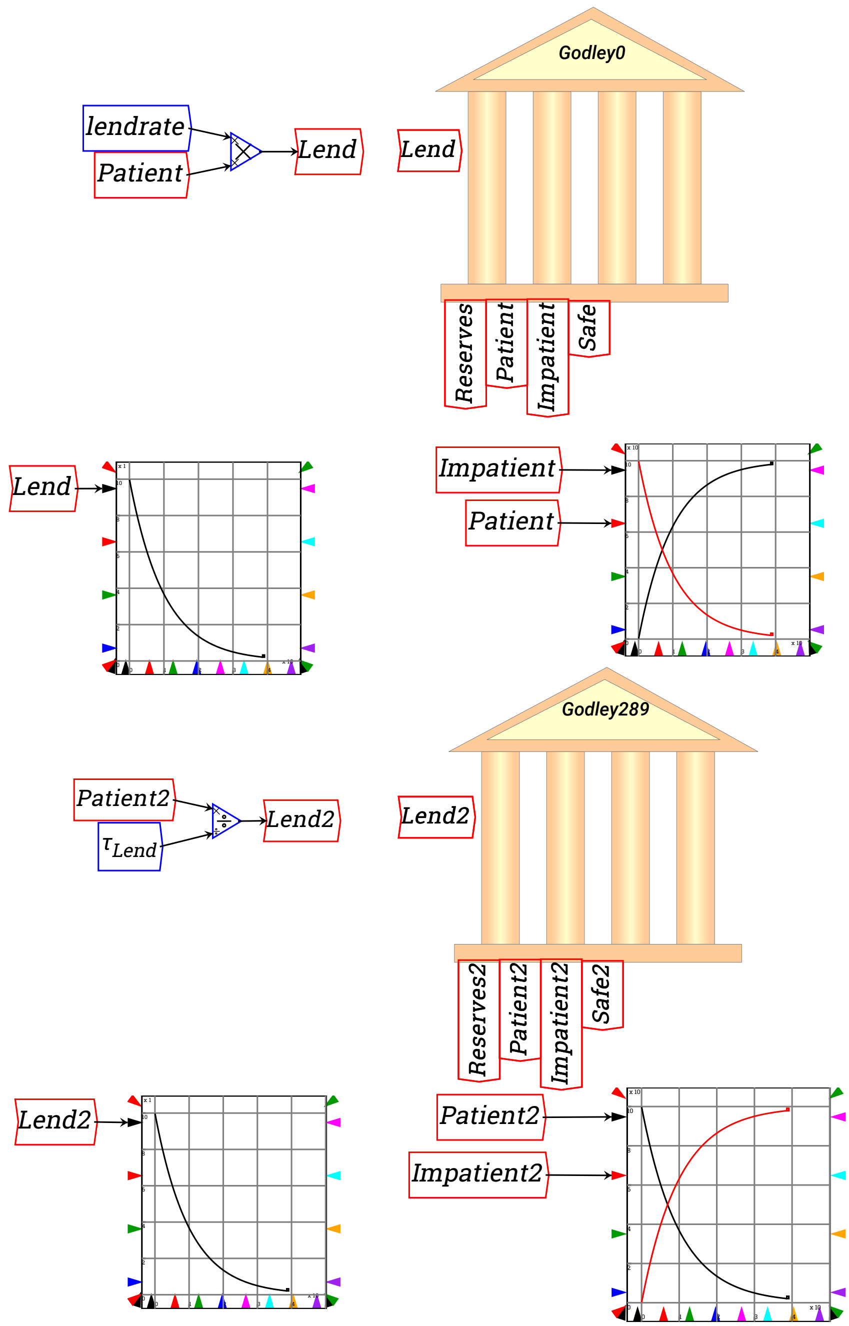
A more complex model would have many more flows, and these in turn would depend on other entities in the model, and be time-varying rather than using a constant “lendrate” as in this example—see the Tutorial on a Basic Banking Model for an example. This example uses the engineering concept of a “time constant”, which is explained in the next section.

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:



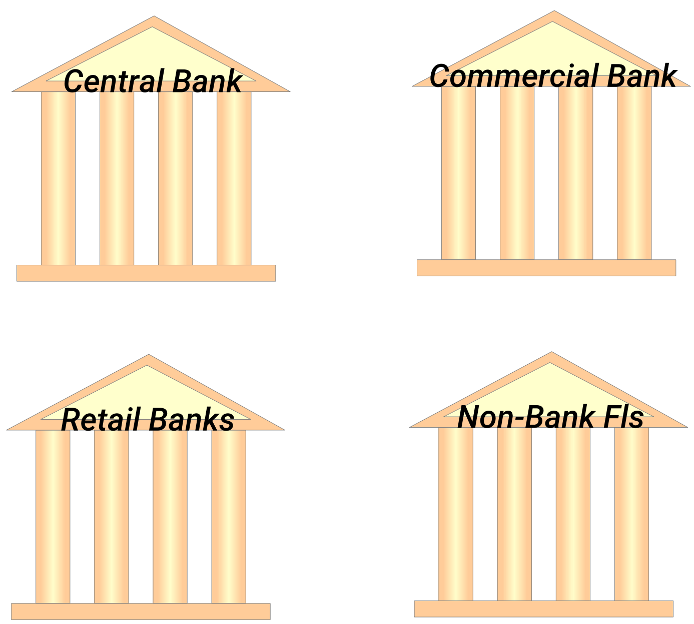
As the simulation shows, the two models have precisely the same result numerically:



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.

Chapter 3

Tutorial

3.1 Basic System Dynamics model

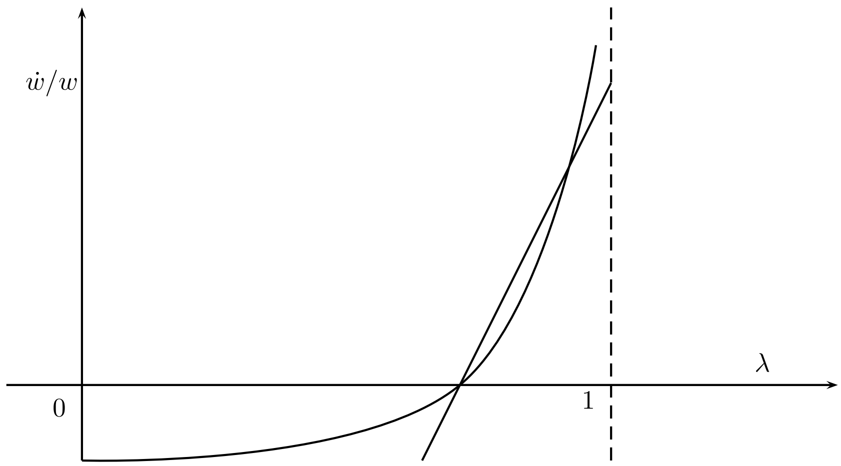
In 1965, Richard Goodwin, the great pioneer of complexity in economics, presented the paper “A Growth Cycle” to the First World Congress of the Econometric Society in Rome. It was later published in a book collection (Goodwin, Richard M. 1967. ”A Growth Cycle,” in C. H. Feinstein, *Socialism, Capitalism 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:

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

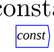
- In a simple two-class model, profits Π equals the level of output Y minus the wage bill: $\Pi = Y - wL$
- For simplicity, Goodwin assumed that all profits were invested, so that Investment equals profits: $I = \Pi$.
- Investment is the rate of change of the capital stock K;
- The level of output is, 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.

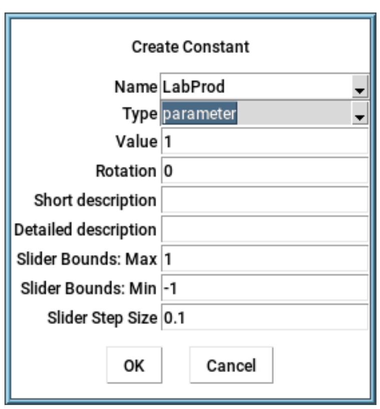
  Goodwin published the model as a reduced form equation in the two system states the employment rate (λ) and the workers' share of output (ω):

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

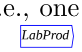
This form is useful for analytic reasons, but it obscures the causal chain that actually lies behind the model. With modern system dynamic software, this can be laid out explicitly, and we can also use much more meaningful names. We'll start with defining output (which is a variable). Click on  on the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

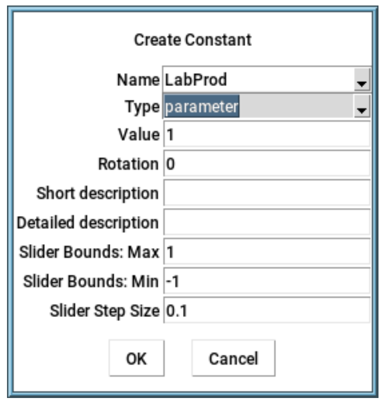
Enter "**GDP**" into the "Name" field, and leave the other fields blank—since **GDP** is a variable and we're defining a dynamic system, the value of **GDP** at any particular point in time will depend on the other entities in the model. Now Click OK (or press "Enter"). The variable will now appear, 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  on the Palette, or by choosing Insert/variable from the menu. This will pop-up the Edit Constant window:

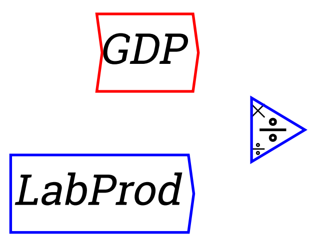


There is actually no real difference between the "Edit constant" dialog and the "Edit variable" dialog. The window's title differs, and the default value of Type is "constant" instead of "flow". We're going to select "parameter", allowing one to give the parameter a name.

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  will now be attached to the cursor. Place it below **GDP**:

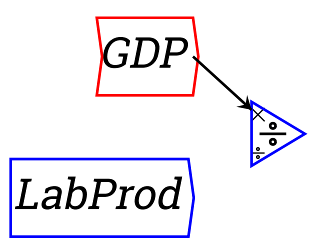


Now we need to divide **GDP** by **LabProd**. Click on the  symbol on the palette and the symbol will be attached to the cursor. Drag it near the other two objects and click. Your Canvas will now look something like this:

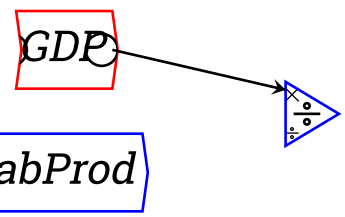


Now to complete the equation, you have to attach **GDP** to the top of the divide block and LabProd to the bottom.

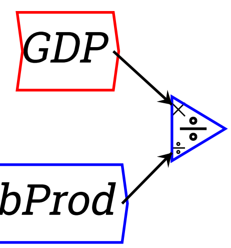
Now move your cursor to the right hand side of  and click, hold the mouse button down, and drag. An arrow will come out from . Drag this arrow to the top of the divide block (where you'll see a tiny multiply sign) and release the mouse. You should then see this:



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

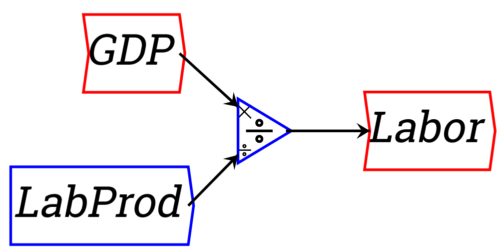


Those are the connection points for wires, so start dragging from one and release on the other. Now wire LabProd to the bottom of the Divide block (where you'll see a miniature divide symbol (blown up below):



Then click on 

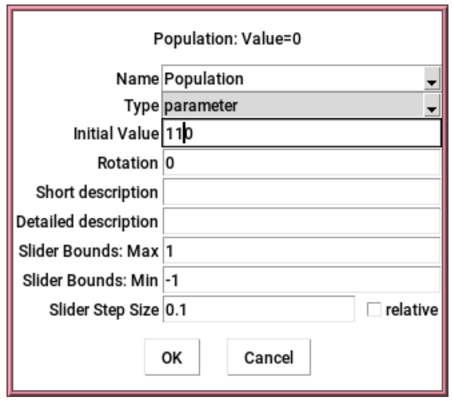
in the Design Icons to create a new variable, call it Labor, place it the the right of the Divide block, and wire the output port from the Divide block to the input port for ***Labor***:



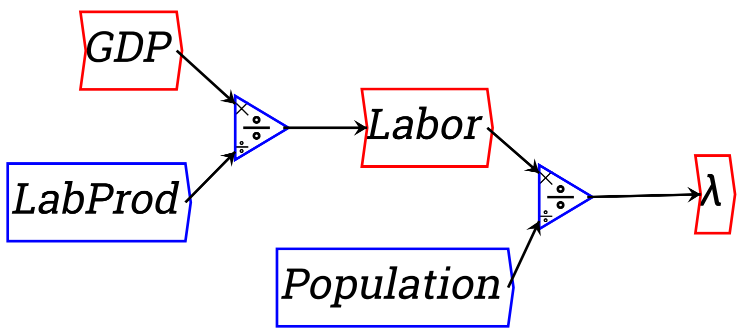
To show the correspondence between the flowchart above and standard modeling equations, click on the equations tab:

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

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  , 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

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

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:

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

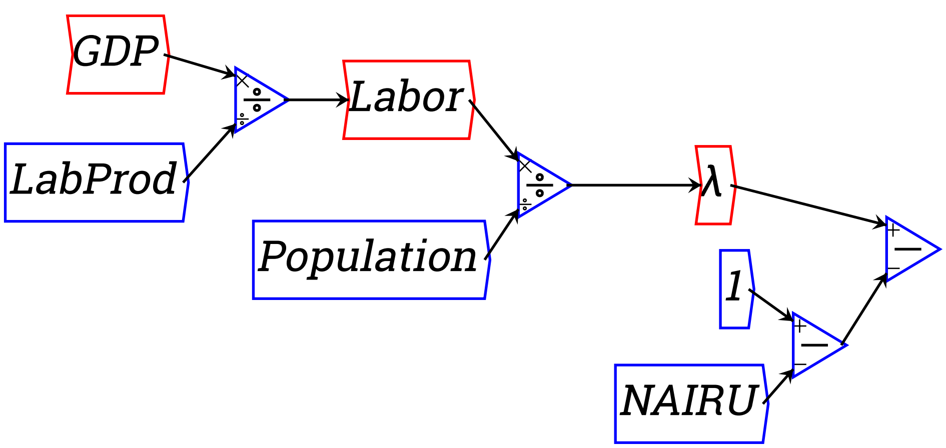
Firstly multiply both sides by w:

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

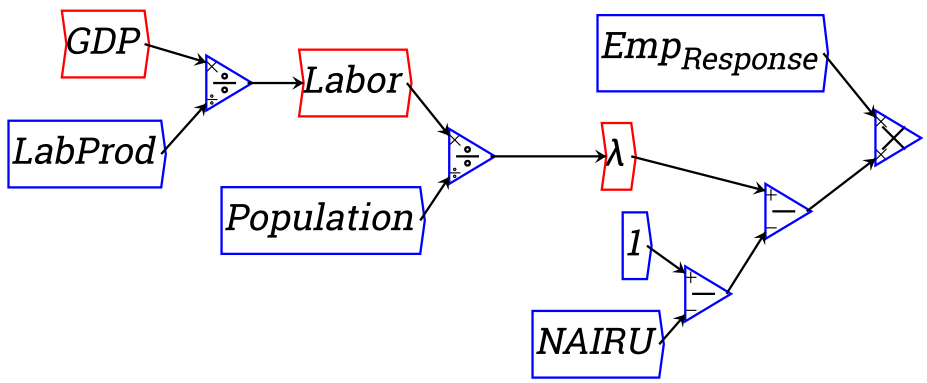
Then integrate both sides (because integration is a numerically much more stable process than differentiation, all system dynamics programs use integration rather than differentiation):

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

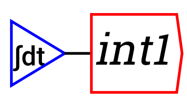
In English, this says that the wage now is the initial wage plus the integral of the wage multiplied by its rate of change function. That's what we now need to add to the Canvas, and the first step is to spell out the wage change function itself. Firstly, since we're using a linear wage response function, 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:



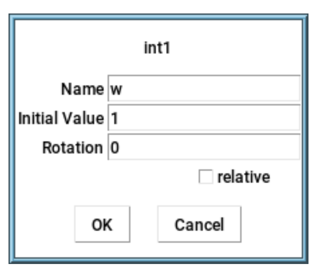
Now we need to multiply this gap between the actual employment rate and the “NAIRE” rate by a parameter that represents the response of wages to this gap. Let’s call this parameter ***Emp_{Response}*** (remember to include the underscore and the braces). Define the parameter, give it a value of 10, and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

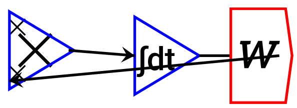


To compete the integral equation, we need to multiply the linear employment response function by the current wage before we integrate it (see the last equation above). There are two ways to do this. First, place a multiply block between the wage change function and the integral block, wire the function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

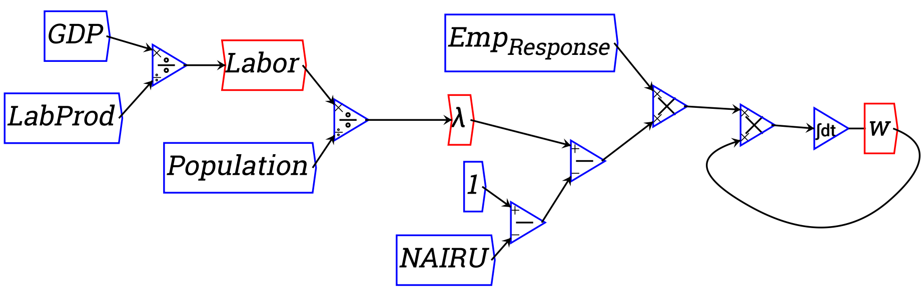
- wire the output of the $\boldsymbol{w}$ block back to the other input on multiply block; or

- Right-click on $\boldsymbol{w}$, choose "Copy Var", place that copy before the multiply block, and wire it up.

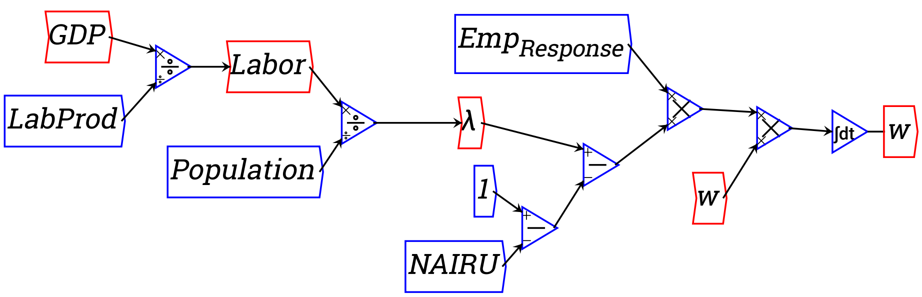
The first method gives you this initial result:



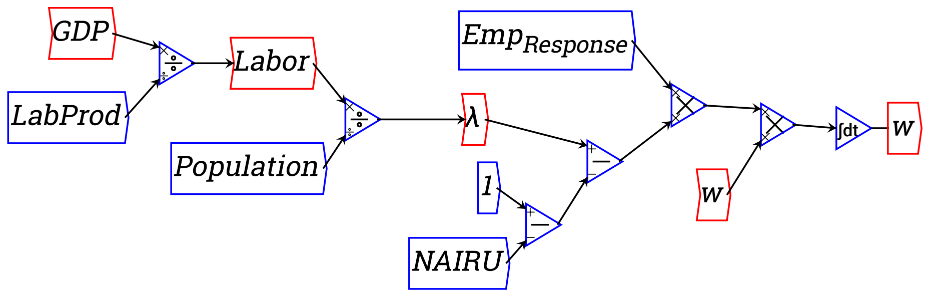
That looks messy, but notice the blue dot on the wire? Click and drag on that and you will turn the straight line connector into a curve:



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

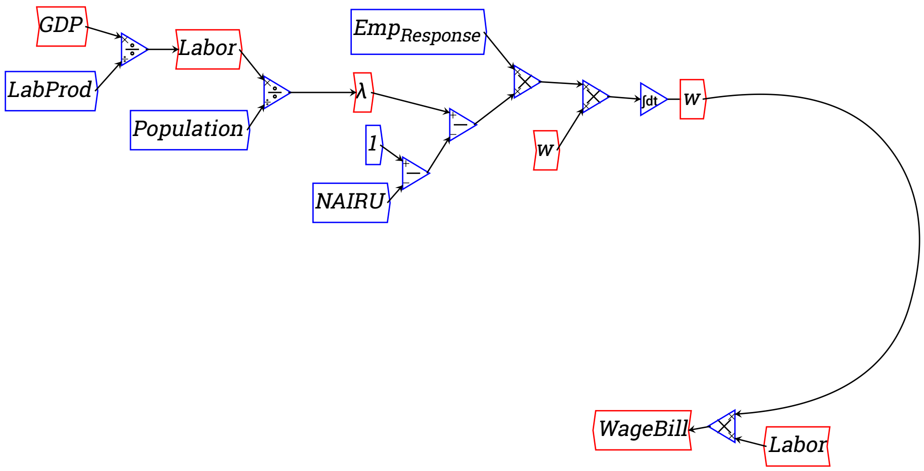


From this point on the model develops easily—"like money for old rope", as one of my maths lecturers used to say. Firstly if we multiply the wage rate $\boldsymbol{w}$ by ***Labor*** we get the **Wage Bill**. To do this, firstly create the variable Wage Bill, and put it well below where w currently is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

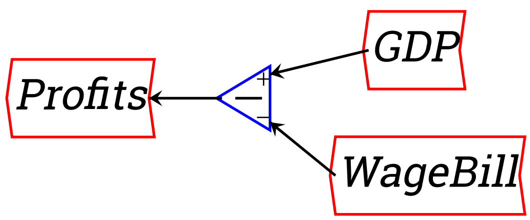


Now right-click on ***WageBill*** and choose "Flip". This rotates the block through 180 degrees (any arbitrary rotation can be applied from the variable definition window itself). Now right-click on ***Labor***, which you've already defined some time ago, and choose "Copy". Place the copy of ***Labor***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

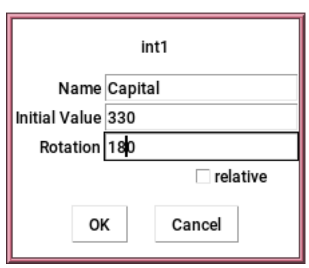
Now insert a multiply block before ***WageBill***, and wire ***w*** and ***Labor*** up to it. Curve the wire from w using the blue dots (you can do this multiple times to create a very curved path: each time you create a curve, another 2 curve points are added that you can also manipulate, as I have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



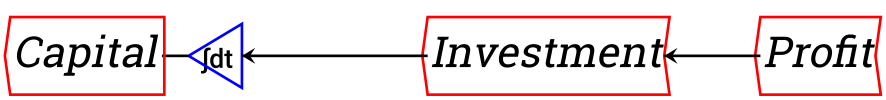
The next step is to subtract the ***WageBill*** from ***GDP*** to define ***Profits***. Take a copy of ***GDP***, insert it above ***WageBill***, insert a subtract block, and wire it up to define the variable ***Profits***:



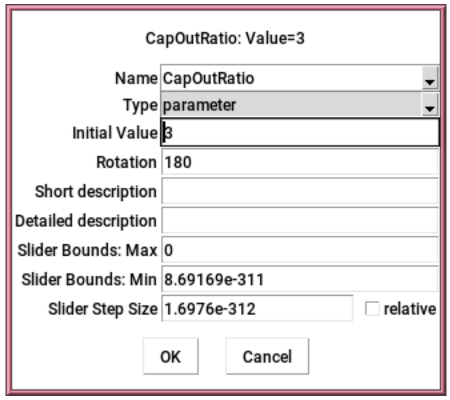
In the simple Goodwin model, all Profits are invested, and investment of course is the rate of change of the capital stock Capital. Create a variable called Investment, wire this up to Profits, and then create a new integral variable *Capital* using the icon. Right-click or double-click on it to rename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



Wire this up to Investment:

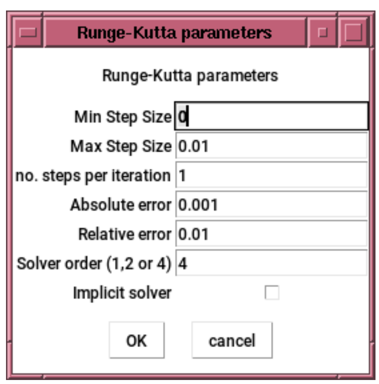


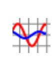
Now there's only one step left to complete the model: define a parameter CapOutputRatio and give it a value of 3:

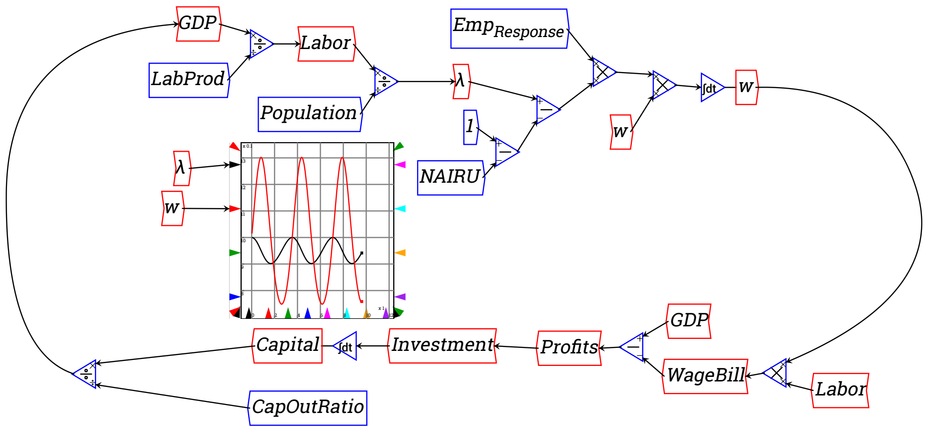


Divide Capital by this, and wire the result up to the input on GDP. You have now built your first dynamic model in Minsky:

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.

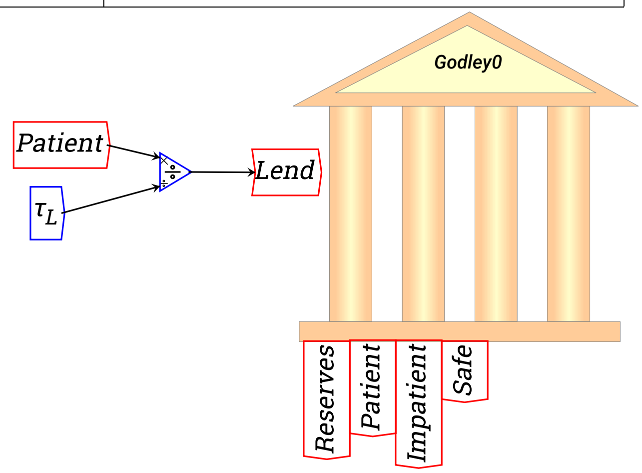
3.2 Basic Banking model

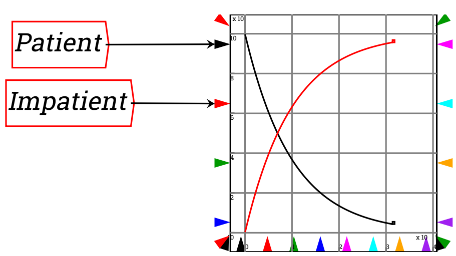
If you haven't yet read the section on Creating a Banking Model, do so now. This tutorial starts from the skeleton of the "Loanable Funds" model built in that section, and using time constants to specify how quickly lending occurs.

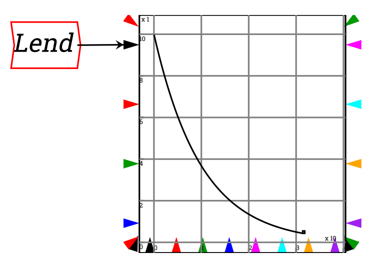
3.2.1 Loanable Funds

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.

Flows ↓ / Stock Variables →	*Reserves*	*Patient*	*Impatient*	*Safe*
	asset	liability		equity
Initial Conditions	120	−100	0	−20
Patient lends to Impatient		*Lend*	−*Lend*	







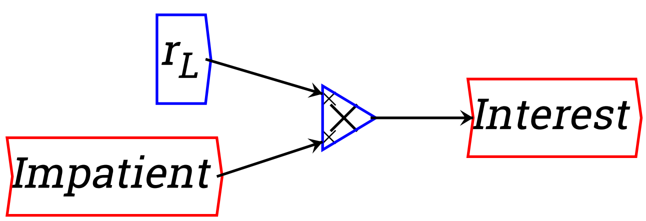
Many more actions need to be added to this model to complete it. For a start, Impatient should be paying interest to Patient on the amount lent. Add an additional row to the Godley Table by clicking on the '+' key next to "Patient lends to Impatient" to create a blank row:

Flows ↓ / Stock Variables →	*Reserves*	*Patient*	*Impatient*	*Safe*
	asset	liability		equity
Initial Conditions	120	-100	0	-20
Patient lends to Impatient		*Lend*	$-$*Lend*	

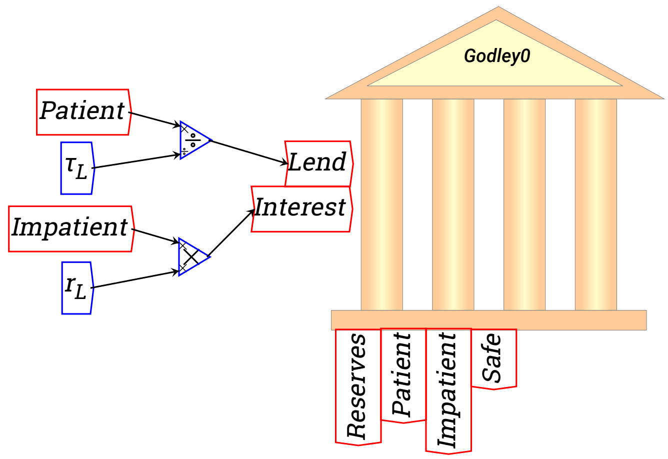
Then label this flow "Impatient pays interest" and make the entry "Interest" into the cell for Impatient on that row. Make the matching entry "-Interest" in the cell for Patient. The flow "Interest" now appears on the input side of the Godley Table on the Canvas:

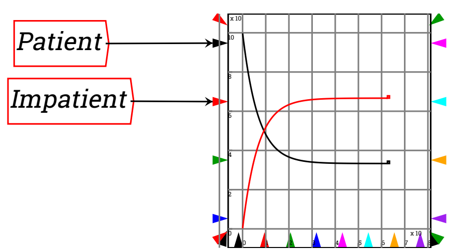
Flows ↓ / Stock Variables →	*Reserves*	*Patient*	*Impatient*	*Safe*
	asset	liability		equity
Initial Conditions	120	-100	0	-20
Patient lends to Impatient		*Lend*	$-$*Lend*	
Impatient pays interest		$-$*Interest*	*Interest*	

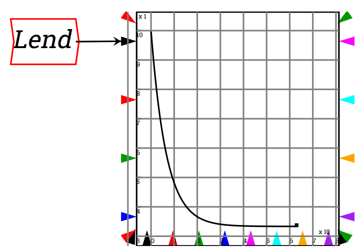
Interest now has to be defined. It will be the amount in Impatient's account (since this began at zero) multiplied by the rate of interest charged by Patient:



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:





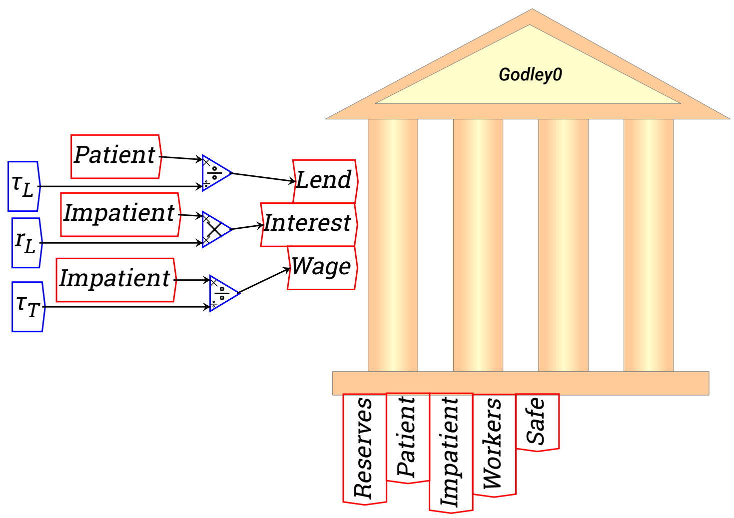


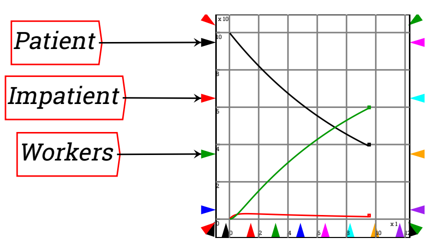
Though it stabilizes, this is is still a very incomplete model: neither Patient nor Impatient are doing anything with the money apart from lending it and paying interest. I am now going to assume that Impatient is borrowing the money in order to hire workers to work at a factory and produce output for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

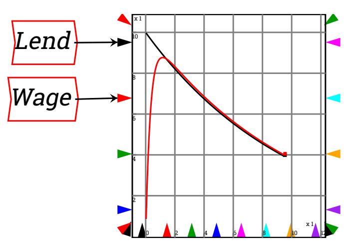
Flows ↓ / Stock Variables →	$Reserves$	$Patient$	$Impatient$	$Workers$	$Safe$
	asset	liability			equity
Initial Conditions	120	−100	0	−0	−20
Patient lends to Impatient		$Lend$	$-Lend$		
Impatient pays interest		$-Interest$	$Interest$		
Impatient pays Workers			$Wage$	$-Wage$	

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.

Chapter 4

Reference

4.1 Operations

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.

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.

multiply $\times$ 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.

divide $\div$ 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 '$\times$' and '$\div$' signs indicating which port refers to the numerator and which the denominator.

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.

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

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

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

eq $=$ Returns 0 or 1, depending on whether $x = y$ is true or false.

min Returns the minimum of x and y.

max Returns the maximum of x and y.

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 or false respectively.

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 or false respectively.

not $\neg$ The output is 1 or 0, depending on whether $x \leq 0.5$ is true or false respectively.

time Returns the current value of system time.

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

integrate Creates an integration (or stock) variable. Editable attributes include the variable's name and its initial value at $t = 0$.

differentiate Symbolically differentiates its input.

data A data interpolation widget. Currently, the data must be imported from a file containing two values on each line, eg:

0.1	0.3
0.5	0.7
0.9	1

If the input is less than the minimum key value (0.1 here), then the operation outputs the corresponding value (0.3). Similarly if the input is greater than the maximum (0.9), the corresponding value (1) is output. If it lies in between two keys (eg 0.2), the the output is linearly interpolated (0.4).

More formally, a data block is an empirical function, based on a table of pairs of values $(x_i, y_i, i = 1 \ldots n, x_{i+1} > x_i)$ read in from a file. The function's output is linearly interpolated from the data, ie:

$$f(x) = \begin{cases} y_1 & x < x_1 \\ y_n & x \geq x_n \\ \frac{y_i(x_{i+1}-x)+y_{i+1}(x-x_i)}{x_{i+1}-x_i} & x_i \leq x < x_{i+1} \end{cases}$$

sqrt $\surd$ Square root of the input

exp Exponential of the input

ln Natural logarithm

sin sine function

cos cosine function

tan tangent function

asin Arc sine, inverse of sine

acos Arc cosine, inverse of cosine

atan Arc tangent, inverse of tangent

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

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

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

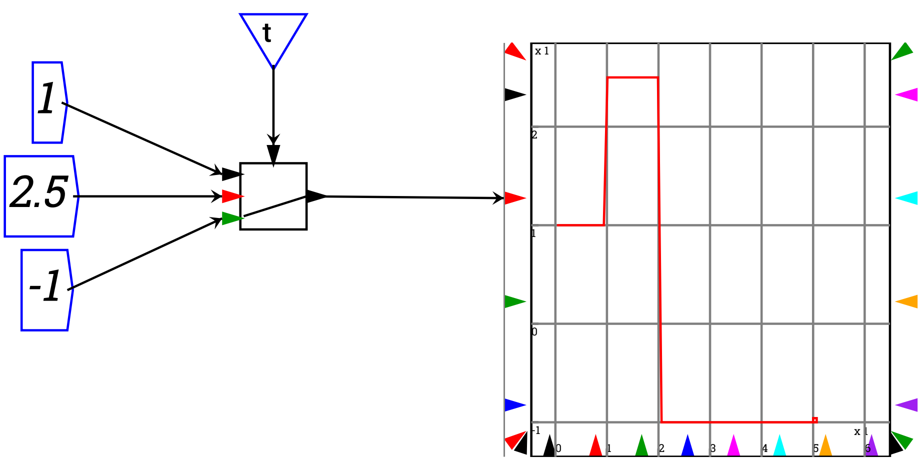
abs $|x|$ absolute value function

floor $\lfloor x \rfloor$ The greatest integer less than or equal to x.

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

4.2 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.

4.3 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.

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

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

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

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

4.3.1 Variable names

Variable names uniquely identify variables. Multiple icons on the canvas may have the same name — they all refer to the same variable. Variable names have scope, which is either global (indicated by a leading ':', or the numerical id of a group. 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.

4.3.2 Initial conditions

Variable initial conditions can be defined through the "init value" field of the variable edit menu, or in the case of Godley table stock variables, through the initial condition row of the Godley table. An initial value can be a simple number, or it can be a multiple of another named 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.

4.3.3 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.

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 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.

4.5 Groups

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

- by lassoing a number of items to select them, then selecting "group" from the canvas context menu, or the edit menu.

- by pasting the selection. You may "ungroup" the group from the context menu if you don't desire the result of the paste to be a group.

- by copying another group

- by inserting a Minsky file as a group

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.

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.

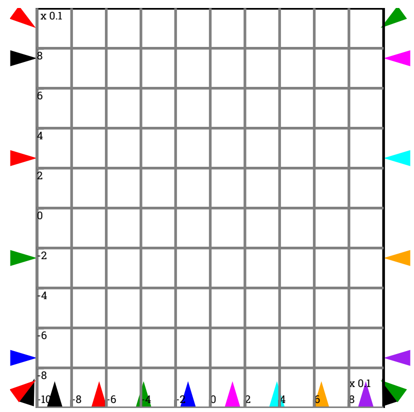
When creating a group, or dragging a variable or operation into or out of a group, if a wire ends up crossing the group boundary, a new temporary variable is added as an I/O variable.

Variable names within groups are locally scoped to that group. That means that a variable of the same name outside the group refers to a different entity completely. One can refer to variables outside the current scope by qualifying the variable name. The simplest qualification to understand is global scope, which is specified by prepending a ':' to the variable name. To refer to a variable in another group, the notation is `<group id>:<name>`, where `<group id>` is the integer id of the group in question. An alternative equivalent notation is `<description>[<group id>]:<name>`, where `<description>` is the textual name given to the group. The description string is ignored by Minsky on input—only the group id is important, but this mechanism makes it easier to see what variable belongs to which group in the drop down lists in the name field. Please try to keep the group names distinctive in their first five characters, as Minsky will truncate the descriptive string to fit in the drop down menu.

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

4.6 Plot widget

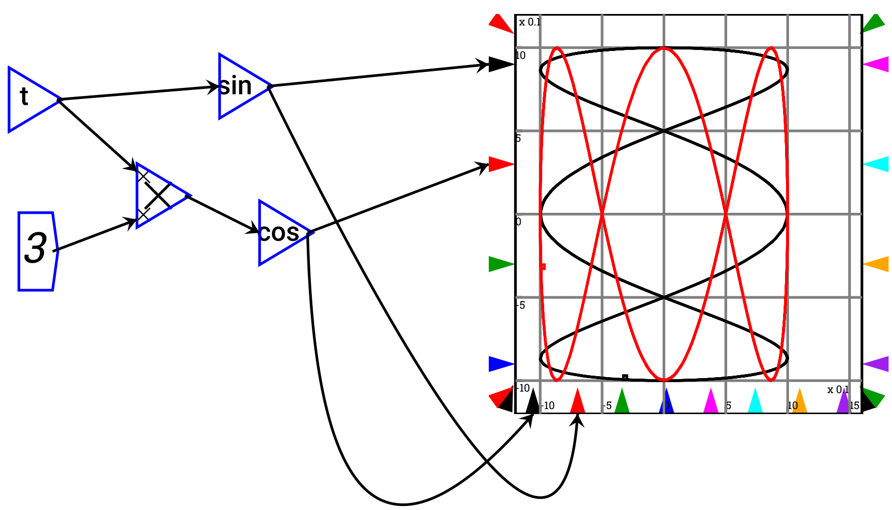
A plot widget embeds a dynamic plot into the canvas. Around the outside of the plot are a number of input ports that can be wired.



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

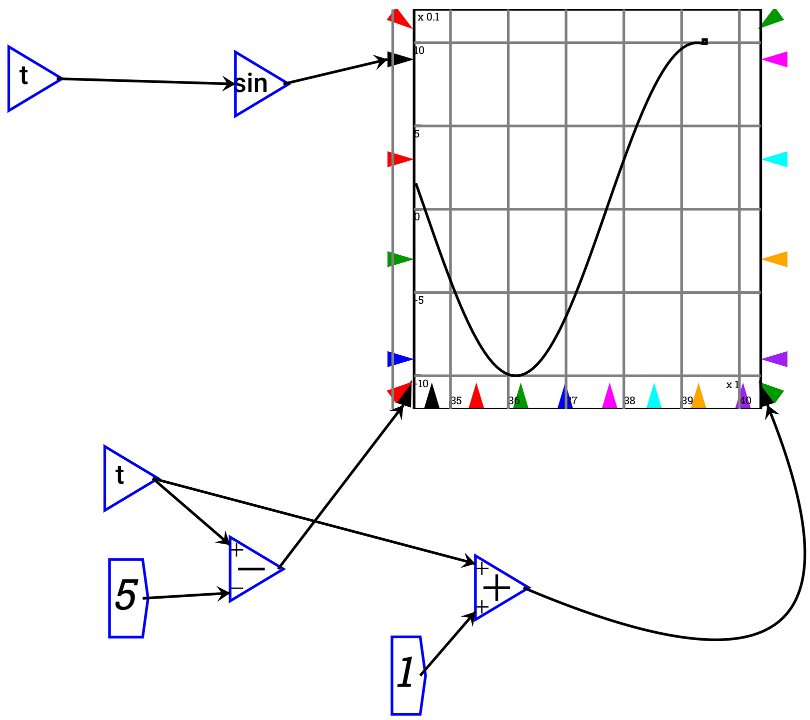
right hand edge Another 4 quantities can be added to the plot. These are shown on a different scale to the left hand inputs, allowing very different magnitudes to be compared on the one plot.

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



4.7 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, but LaTeX formatting is not currently 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 visible through the edit menu, or as a tooltip.

4.8 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. In "double entry" mode, 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 positive flow increasing the asset of one entity will appear as a negative flow, increasing the value of a liability). Instead of positive or negative flows, one can optionally use CR and DR prefixes, as specified in the options panel.

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 multipler times a flow variable. Unscoped variables are treated as global at present, however, in the future, Godley tables will be allowed to be part of groups, which will then define the context of the unqualified variable names.

The final column displays the row sum. A correctly functioning Godley table should have each row sum to zero — this ensures everything is accounted for, with no hidden sinks or sources.

4.9 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.

Raise/Lower Raise and lower the canvas items relative to each other. You may need to do this if a large item such as a Godley table or plot is obscuring a wire, making it hard to access the wire's context menu or handles,

Browse object gives a low level drilldown of the internal C++ object this canvas item represents. It is perhaps more of interest to developers.

Delete delete the object.

Item specific context menu items:

variables, parameters and constants

- **Slider** add a slider control to a variable. This is most effective for controlling parameters and constants, but can also be used to control inputless variables.
- **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.

4.10 Canvas background

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.

The following keystrokes insert an operation

`+`	add
`-`	subtract
`*`	multiply
/	divide
`^`	pow
`&`	integral
`=`	Godley table
`@`	plot
`%` or `#`	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.