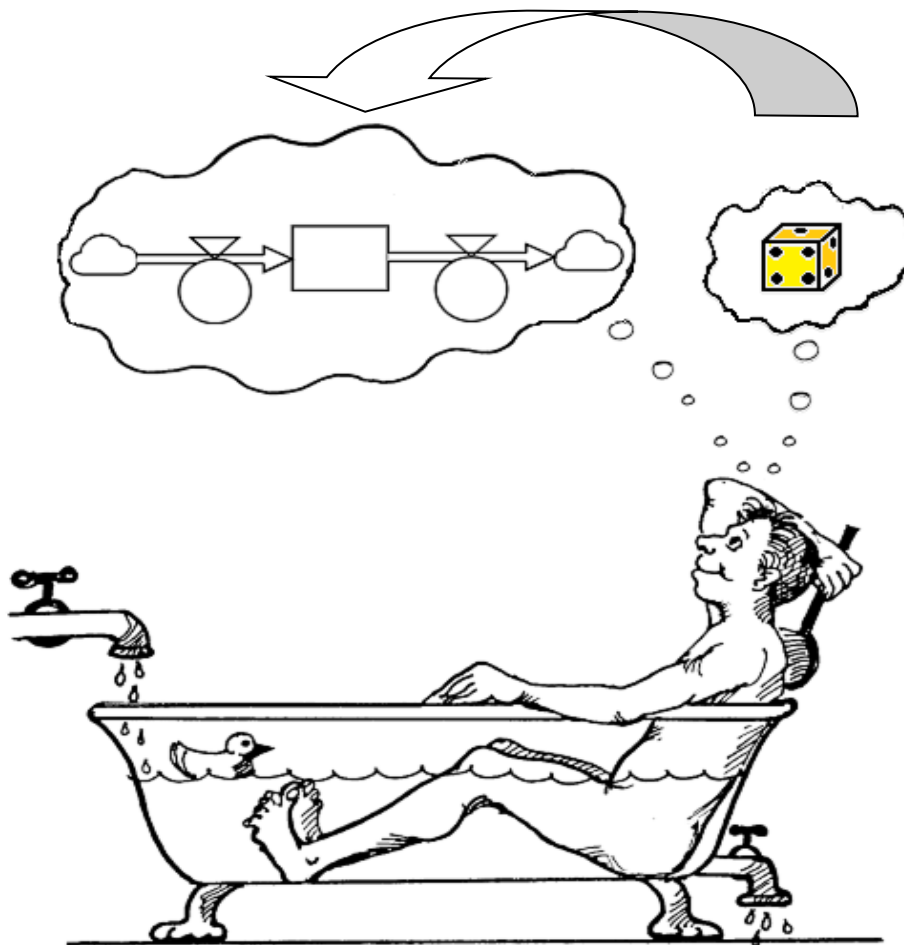




StochSD User's Manual and Tutorial



Part I. *Deterministic* modelling and simulation with StochSD

Part II. *Stochastic* modelling and simulation with StochSD

© Leif Gustafsson, Uppsala University, Uppsala, Sweden
File: StochSD_User's_Manual_2024-01-24.pdf

Home Page: <https://stochsd.sourceforge.io>

Contents

Introduction	5
1. Installation of StochSD	7
1.1 StochSD Desktop	7
1.2 StochSD Web	7
Part I. Deterministic modelling with StochSD	9
2. The building blocks of a model	10
3. Starting up StochSD	13
3.1 Menus	13
3.2 Buttons	16
3.3 Time display, Time Unit button and Inspection field	18
4. Model building	19
4.1 Primitives (Building blocks)	19
4.2 Styling buttons	21
4.3 Naming and defining relations of the primitives	21
5. Equations	24
5.1 Defining the equation	24
5.2 Error checking	24
6. Specifications of the simulation	26
7. Simulation	29
8. Results presentation	30
8.1 The Number Box	30
8.2 The Table	31
8.3 The Time Plot	31
8.4 The Compare Simulations Plot	32
8.5 The XY Plot	33
8.6 The Histogram	34
9. StochSD model examples	35
10. Functions	38
10.1 Programming Functions	40
10.2 Mathematical Functions	42
10.3 Historical Functions	46
10.4 Random Number Functions	49
10.5 Statistical Distributions	52
10.6 Converter (Table look-up function)	52
11. Macro Functions	54

12. Practical tips	56
13. References to deterministic CSS modelling	58
Part II. Stochastic modelling with StochSD	59
14. Stochastic modelling	60
14.1 A historic note and the motivation for StochSD	60
14.2 What is stochasticity?	61
14.3 Five types of uncertainty about the system under study	61
14.4 Why including uncertainties in the model?	62
14.5 How to model uncertainties	62
14.6 What can happen if you ignore stochasticity?	67
14.7 A warning example – Comparing the results from a deterministic and a stochastic SIR model	68
14.8 Making a stochastic model reproducible	68
15. A stochastic StochSD model	70
16. Tools in StochSD	71
16.1 The StatRes tool	71
16.2 The Optim tool	71
16.3 The ParmVar tool	72
16.4 The Sensi tool	72
16.5 Hiding the tool	72
16.6 Clearing a tool	72
17. References to stochastic CSS modelling	73
Appendix. The StochSD package, Licenses and Responsibility	74
A1. The structure of the StochSD package	74
A2. Short explanations of licenses for StochSD and third-party components	74
A3. Responsibility	75
Index	77

Introduction

StochSD (Stochastic System Dynamics) is an open-source tool for *stochastic* (and deterministic) modelling and simulation based on the System Dynamics approach to Continuous System Simulation (CSS).

Continuous System Simulation (CSS) is a macro description of a dynamic system by differential equation (numerically described as difference equations) and algebraic equations. During the simulation, the equations are stepwise updated timestep by timestep. Classical CSS only handled continuous matter, but **Full Potential CSS** can also handle discrete entities. In this latter case, randomness usually plays an important role – see Part II of this manual.

System dynamics (SD) is a pedagogic approach to structure and model complex systems based on stocks (compartments), flows and causal links. The pedagogic advantages are several: First, the model is structured in a click-and-play manner based on a few types of building blocks to recreate the main structure of a system under study. Second, SD's graphical stock-and-flow approach gives an intuitive understanding of how the model is updated over time. Third, the graphical structure of stocks, flows and causal links displays the feedback loops that explains the model's behaviour in terms of growth (from positive loops), stabilising factors (from negative loops) and oscillations. Fourth, the SD models are very easy to program; you focus on one symbol at a time and define its relation to those symbols pointing at it. Fifth, the model and its behaviour are easy to understand and communicate, also to persons less experienced in mathematics or programming.

A unique feature in StochSD is that it also supports *stochastic* modelling; i.e. modelling where random numbers are included. A *deterministic* model assumes that you have complete knowledge that you correctly implement in your model, which then produce categorical results. In a *stochastic* model, on the other hand, you describe the uncertainties with the help of random numbers drawn from a statistical distribution that describes your uncertainties. Each simulation of the model then will produce a different behaviour. By making many simulations, you obtain statistical distributions of results from which you can calculate max, min, averages, confidence intervals, correlations, and other statistics that displays the confidence you should have for different outcomes.

StochSD has the random number functions for stochastic model building. Further, StochSD also has tools for specifying the number of simulations, provide controllable seeds (if desired), collect data of selected quantities, make statistical analyses of the outcomes, and to present these results in various ways.

StochSD is based on *Insight Maker*¹ version 5 from which we have taken open parts: simulation engine, function library, error detection facility, macro facility etc., and replaced the non-open parts with open-source packages and our own code and design.

¹ Insight Maker is a simulation package developed by Scot Fortmann-Roe. We have taken the open System Dynamics part: simulation engine, function library, error detection facility, macro facility etc. from Insight Maker. However, the non-open parts based on mxGraph (for model flowchart and result diagrams) and Ext JS (for the model window and buttons etc.) have been replaced in order to make

That StochSD is open-source means that you get it and use it for free. It also gives you the right to modify the StochSD source code under open-source licences, see Appendix A2. StochSD is written in JavaScript.

StochSD is developed to build *deterministic* and *stochastic* models according to the System Dynamics approach, simulate them and analyse the results. It is also supplemented with tools for sensitivity analysis, optimisation, statistical analysis and parameter estimation.

Part I of this manual focuses on how you build and simulate *deterministic* models in StochSD, while **Part II** focuses on *stochastic* modelling, simulation and statistical analysis from multiple simulation runs.

StochSD can be used to study complex systems and their behaviours in a large number of fields; for example, biology, ecology, medicine, epidemiology, economics, physics, technology or social sciences. A model is constructed by placing and connecting predefined graphical symbols at suitable places on the screen. StochSD then transforms the graphical model into executable code that will display the model's behaviour.

StochSD is available in two versions: **StochSD Desktop** and **StochSD Web**. Both versions of StochSD work on Windows, Mac OS X and Linux.

The Desktop version can be downloaded from StochSD's home page and runs as a normal program (using NW.js which stands for Node-Webkit-JavaScript), while the web version requires a web-browser such as Google Chrome, Mozilla Firefox or Microsoft Edge to load StochSD and run your models. StochSD does not support Internet Explorer.

For installation of **StochSD Desktop** or running of **StochSD Web**, see **next section**.

StochSD Home Page: <https://stochsd.sourceforge.io>.

StochSD was developed by Leif Gustafsson, Erik Gustafsson and Magnus Gustafsson, Uppsala University, Uppsala, Sweden.

Mail: leif.gunnar.gustafsson@gmail.com

StochSD is fully open-source. Furthermore, the file handling has been rewritten. Finally, we have made some modifications to conform to the System Dynamics conventions. (Insight Maker is available at: <http://InsightMaker.com>.)

1. Installation of StochSD



StochSD is available in two versions: **StochSD Desktop**, which runs under Windows, macOS and Linux, and **StochSD Web**, which is a web application that can be run under other operative systems.

Although, the Web version can be directly used for testing StochSD without an installation, the Desktop version is recommended because your models can then be locally stored and loaded on your computer. Here you can also arrange your files in directories of your choice.

Furthermore, with the Desktop version you can use a Recent List for direct access to your latest model files.

1.1 StochSD Desktop

System requirements


StochSD Desktop can be used under the operative systems: Windows, Mac OS X and Linux.

Installation instructions

The Desktop version is downloaded from StochSD's home page: <https://stochsd.sourceforge.io>.

To the left of the Home Page you find and click: '[StochSD Software](#)'. Then choose the '[Download StochSD](#)', which takes you to the proper page in SourceForge where you download a zip-file for the proper version for Windows, Mac OS X or Linux.

Find the downloaded *StochSD.zip* file on your computer and unzip it into a folder of your choice.

Then find the *stochsd.exe* file . You can now start StochSD by clicking this file, but it is more convenient to first create a shortcut and place it on your desktop, from where you can start StochSD.

For Mac OS X and Linux there are minor differences from the installation instruction. For example, the postfix *.exe* is only used in Windows.

Note that StochSD is an unidentified developer to Apple. Therefore, Mac OS X requires some settings to be changed. For instructions, see "How To Open & Allow Unidentified Developer Apps & Allow Downloads From Anywhere On Apple Mac." A Step-By-Step Video Tutorial: <https://www.youtube.com/watch?v=WiYmUXVOIgl>.

1.2 StochSD Web

If you want to test StochSD you can try StochSD Web. Further, on a computer where StochSD Desktop is not possible to use, for example Chromebook, the Web application is a way to run StochSD.

System requirements

StochSD Web can be used under any operative system that can run a modern web browser. We recommend Mozilla Firefox, Google Chrome or Microsoft Edge. (Internet Explorer is not supported.)

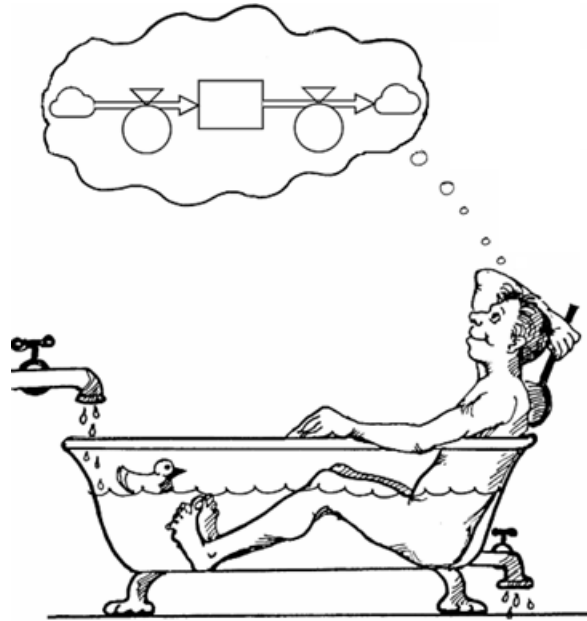
Running instruction:

Write: <https://stochsd.sourceforge.io/software> in Google Chrome or Mozilla Firefox and press Enter, (or Ctrl-click on this link from here). Alternatively, from the StochSD Home Page, you can click on “Try StochSD online”.

There are some drawbacks with StochSD Web.

- The browser steals a part of the screen.
- With the Web version you can't use a Recent List for direct access to your latest model files.
- Although you can open a file from any directory with the Web version, you can only save it in your Download directory (from where you, of course, can manually move the file to any directory). This limitation is because of security arrangements in the web browser. However, with new versions of Google Chrome, Microsoft Edge and other Chromium based browsers a file can be saved to any location.

Part I. Deterministic modelling with StochSD



Purpose: To build *dynamic* compartment models and simulate their behaviours.


Worldview: The world consists mainly of two kinds of fundamental elements: **Stocks** and **Flows**.

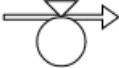
Deterministic modelling teaches the student the fundamentals of model building and simulation. It is also an ideal starting point for understanding the relation between the structure and the behaviour of a real system or a model. Furthermore, deterministic modelling can be used to study systems, which we completely understand.

However, most systems under study contain uncertainties of different kinds due to limited knowledge. This is handled by drawing random numbers from appropriate statistical distributions, and makes the model *stochastic*. How stochastic modelling and simulation are done is treated in Part II.

2. The building blocks of a model

In this section the six building blocks denoted *primitives*: **Stock**, **Flow**, **Link**, **Auxiliary**, **Parameter** and **Converter** by which you construct your model is StochSD are introduced. A seventh primitive, the **Ghost**, is introduced in Section 4, but it just duplicates an existing primitive of cosmetic reasons. (As soon as you have specified the time unit to use in the StochSD model, you can start building!)

 **Stock** is a container for something, e.g. water, people, cars, or money. (Synonyms: *State variable, Compartment, Level.*) A new Stock is assigned a start value and a proper name after double-clicking the symbol.

 **Flow** is the active primitive that generates changes. (Synonyms: *Transition, Rate.*) A flow is always a *rate* - something *per time unit*. It may be water per second, Immigrants per year, Deaths per month, Cars produced per week, etc. Flows fill up or drain Stocks. The new Flow should be defined and assigned a proper name after double-clicking the symbol. Note that the arrow shows the *positive direction* of a flow. If the flow has a negative value, it flows in the direction opposite to the arrow.

A cloud symbol appears in the end if it is not attached to a stock. This tells that the Flow starts or ends outside the studied system, see Figure 2.1.

A **bathtub analogy** is a useful metaphor. The Stock is a *bathtub* where e.g. water is added by an Inflow and drained by an Outflow. The **dynamic behaviour** of a model is created by flows accumulated in and/or drained from Stocks

A **stock** can only be changed by *physical flows* (of water, money, rabbits, etc.), but a **flow rate** can change because of *influence* from other primitives. This influence (transmitted through an *information link*) is represented by a single-lined arrow and may *not* be confused with a physical flow.

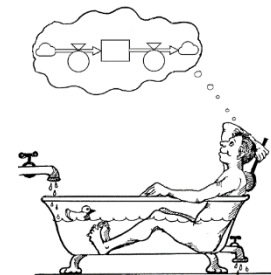





Figure 2.1. The bathtub analogy.

 **Link** transfers information from one primitive to another. (Synonyms, *Information, Signal.*) A link has no name, and the information it transfers is defined by the primitive at the sending end and its impact is defined by the primitive at the receiving end.

For practical reasons, three more graphical symbols are added.

 **Auxiliary** contains a calculation based on functions and the values of other primitives linked to it. The auxiliary can be defined and assigned a proper name after double-clicking the symbol.

 **Parameter** is a special case of an auxiliary that *must not have any incoming link*.² Of pedagogical reasons, it is good to distinguish between an

² In System Dynamics languages, the Auxiliary and the Parameter are defined a little differently. In some languages \diamond is restricted to a constant value, in some the \diamond changes its shape to \circ when it holds a function or is dependent of other quantities, and in some there is just one common symbol for Auxiliaries and Parameters.

unexplained Parameter (i.e. whose value is defined *outside* the system boundaries) and an Auxiliary (that is calculated *within* the model based on other model primitives linked to the Auxiliary). A Parameter can have a constant value, a time function or a random function. The Parameter is defined and named after double-clicking the symbol.



Converter is a table look-up function where you can describe the relation between two primitives like X and Y in tabular form. See Section 10.8.

A preview of a StochSD model

A *System Under Study*, which we want to describe, understand or predict its behaviour, is a delimited piece of the universe. The *system boundaries* divide the universe into two parts: The *System Under Study* and its *Environment*.

To give a short preview of a StochSD model, we have to take a wider view including a specific *System Under Study* and its *Environment* to be described in a *Simulation Model*, and also including some necessary specifications. So, first we have to dismiss the common view that a *Model only* contains a description of the System Under Study.

The System Under Study is defined by *space*, *time period*, what *subjects* or *substance* to be regarded and the *level of details* when describing it in form of a model.

However, how cautious and clever you were when defining the *System Under Study*, you will usually also have to consider important impacts from the Environment onto the System. In StochSD we describe such impacts by parameters (\diamond), which by links affect Auxiliaries or Flows in the model. Further, we are not interested in explaining the origins of objects or matter flowing into the model (by births, immigration, import, etc.) or where used objects or matter will go or decompose after transferred away from the model through an outflow. Therefore, a model flow can origin from an unexplained source or end in a sink outside our scope. In those cases, the flow starts or ends in a cloud symbol to show that the cloud belongs to the Environment rather than to the System Under Study.

Furthermore, the model must contain *specifications* about when in time the model study starts and for how long. Also, the *artificial* updating in time-steps, to resemble an almost continuous time, must be defined together with the chosen algorithm that performs the stepwise updating.

A simple example

To make this concrete, assume that you want to study the epidemic of an infectious disease in a population using a so-called SIR model. This model has the three stages *S* (Susceptibles), *I* (Infectious) and *R* (Recovered). When a Susceptible meets an Infectious there is a risk of infecting the Susceptible. The Infectious will recover after on average *T* time units, but a small fraction of the them will die. In Figure 2.2, the System Under Study is described in the *kernel* of the model. However, there are a number of issues outside the System boundaries that have to be considered. The risk for a Susceptible to be infected per time unit, *p*, depends on the infectiousness of the disease, on the structure of the society, on the behaviours of the individuals, etc. Since

we have no ambition or possibility to describe all such factors, we may settle with an (unexplained) average value placed in the parameter p . The same is valid for the sojourn time T in the Infectious stage which may be a complex function of treatment, medication, age, sex, etc.

In epidemic modelling, we are also interested in the so-called reproduction number, $R_t = S(t) \cdot p \cdot T$, that tells how many persons an infectious one will on average infect before he or she becomes immune or die³. If R_t is smaller than one, the epidemic will decrease. (In particular, R_0 tells whether there will be an epidemic or not.) In this model, we have included R_t as an Auxiliary explained by S , p and T .

In Figure 2.2, we have described the epidemic system with a structure of Stocks, Flows, Auxiliary and Links in the kernel of the model. Outside the kernel we have placed all Parameters to show that they describe how the Environment affects the System Under Study. The cloud symbol at the end of the death flow, although only cosmetic, tells that what happens after death is outside the scope of the system study, and thus the cloud belongs outside the kernel of the model.

Outside the ‘Environmental frame’ in Figure 2.2, you find the *Specifications* necessary to run the model. These are not displayed in the graphical representation, but are still an important part of the model.

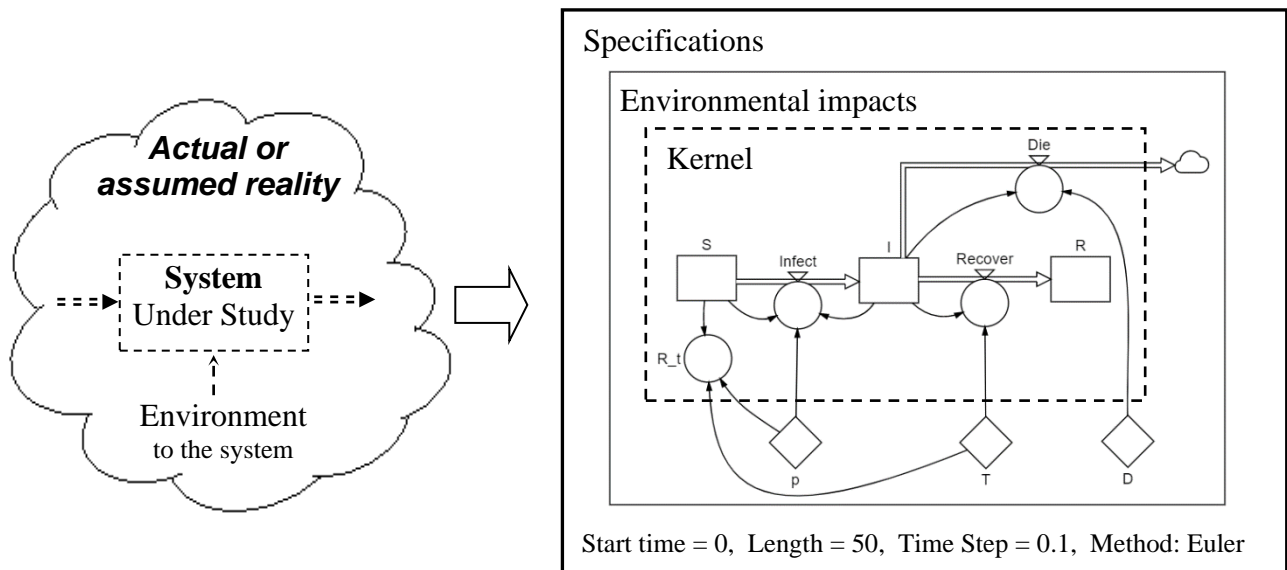


Figure 2.2. A simulation model consists of: 1) The model kernel representing the System Under Study, 2) The parameters representing influences from the environment. Further, in- and outflows through the boundaries of the system start or end with a cloud, 3) Specifications of Start time, Length of simulation, Time Step and Integration method.

However, *in practical modelling*, we place Parameters and Clouds where it is convenient and tidy. It is sufficient that the symbols tell what is within and outside the system boundaries.

Finally, a StochSD model uses Plots, Tables and Number Boxes to display the results.

³ To be exact, the infectious time constant, T , should also consider the dying persons ($T \rightarrow T+D$).

3. Starting up StochSD

StochSD is available in two versions: **StochSD Desktop** and **StochSD Web**. For installation and use, see Section 1. This manual applies for both versions.

When you open StochSD, the StochSD workspace is shown. On the upper part of the screen you see rows of Menus and Buttons, and below is the Model window (which is empty at start). As soon as you have specified the Time Unit to be used, you can begin the model building in a click-and draw manner.

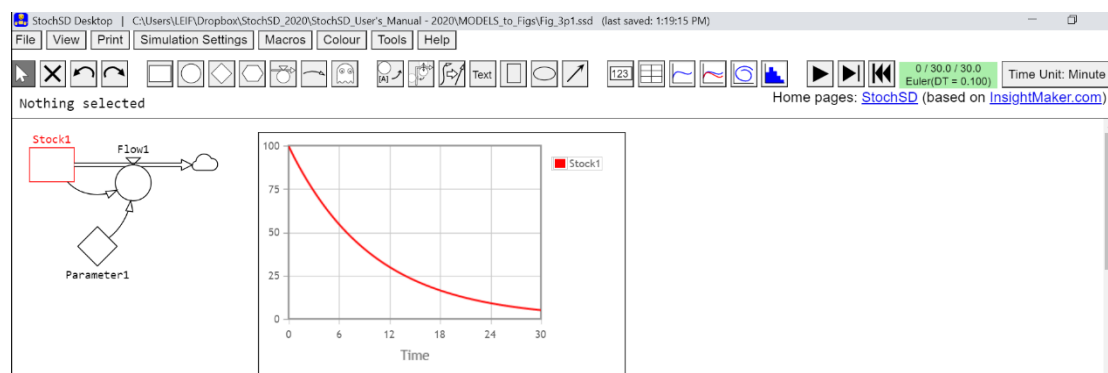


Figure 3.1. The StochSD screen consists of a Title bar, rows of Menus and Buttons, and the Model window. The Model window is here shown with a few building blocks and a Time Plot.

The **Title bar** shows the file name (with its path) and the time for last saving.

The row of **Menus** consists of pull-down and pop-up menus, and the row of **Buttons** is used for construction, result presentation and running the model.

In the right end of this row is a field showing Start Time, Current Time and End Time, and below the Integration Method and the Time Step used.

Further right is a button for specification of the Time Unit used for the model.

Finally, to the left under the buttons, there is an **Inspection field** where you can see the definitions of a marked primitive without opening it, and to the right there are links to the *home pages of StochSD* and *Insight Maker*.

3.1 Menus

In the menus you find: **File, View, Print, Simulation Settings, Macros, Colour, Tools** and **Help**.

File menu

The file handling in StochSD is managed from the File menu in a similar way to most programs, e.g. Microsoft Word.

In StochSD, a model constitutes a file. You can *create* a new model, *save* the model and *open* it again.

When using **StochSD - Desktop** you can choose the folder to save the model in. When using **StochSD – Webb** the files are always stored in your *Downloads* folder because of security arrangements in the web browser.

The File menu contains:

- New
- Open (Ctrl O)
- Save (Ctrl S)
- Save As ...
- List of Recent Files

To create a new model, you click **New** in the **File** menu.

A model can be opened by clicking **Open** in the **File** menu (or directly: Ctrl O).

By pressing the **Save as ...** option in the **File** menu, you open a form for saving your model. Here you can give the file a proper name. In the Desktop version of StochSD you can also browse to a dictionary of your choice, while in the Web version the model is always stored in your *Download* library.

When the model has a name and location, you can use the **Save** option in the File menu (or directly; Ctrl S).

Up to ten **Recent Files** are directly accessible. (Only for the Desktop version.)

View menu

With the View menu you can customize the size of StochSD on your screen.

- Zoom In (Ctrl +)
- Zoom Out (Ctrl -)
- Reset Zoom (Ctrl 0)

Print menu

Printing the model diagram or its underlying equations is performed from the Print menu. This menu contains:

- Print Diagram (Ctrl P)
- Print Equations

The model diagram can be printed on paper with the **Print** button or Ctrl P. However, the **Print Diagram** is a screen dump. This means that you first may have to reduce the size by zooming in (Ctrl –) or take several screen dumps to cover the whole model. You may also zoom in (Ctrl +) to make the model larger before printing it out.

The model equations can be viewed by opening the **Print menu** choosing **Print Equations**. You will then see the **Equation List**. From here you may also print the equations. See Figure 3.2.

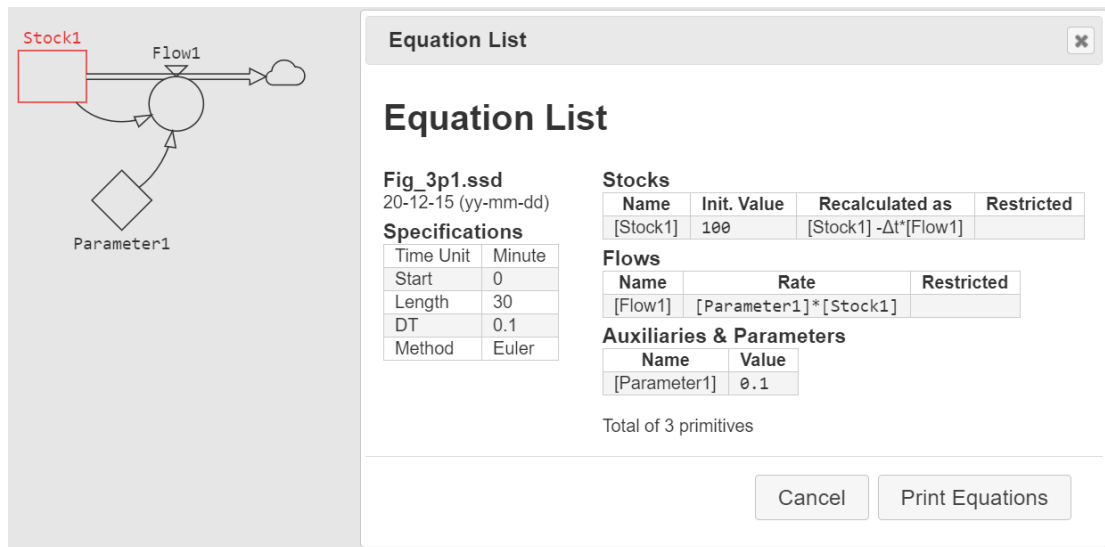


Figure 3.2. The Equation List for the small model shown to the left.

As seen in the figure, in addition to the equations, sorted into Stocks, Flows, etc., there are also information about File name, Date, Time Unit chosen, Start Time, Length, DT and updating Method (Euler or RK4).

Simulation Settings menu

Before you execute the model, you should open the **Simulation Settings menu** to specify what *time period* to study and the *size of the Time Step* (DT). The Time Step sets the time interval between recalculations of the model. Finally, Method stands for the updating algorithm ('integration') where you can choose between Euler and Runge-Kutta fourth order (RK4). Euler is *faster* and the *default*, while RK4 is *more accurate* for a given DT. For stochastic models the advantage of RK4 is lost; it will only make the simulation time longer.

- Start Time
- Length
- Time Step
- Method (Euler or RK4)

Macros menu

His menu enables the introduction of macros to StochSD, see Section 11.

Here you can define your own functions.

In stochastic modelling, it is often important to make the stochastic simulation reproducible. This can be accomplished by locking the seed to the random number generators. In the macro form, this can be done by selecting a seed and clicking the SetRandSeed button. See Section 11.

Colour menu

After marking one or several elements (primitives, plots, etc.), you can open the Colour menu and select one out of 12 colours.

Note that you can couple the colour of a primitive and the colour of its line in a Time Plot.

Tools menu

In the Tools menu, you find tools for *Optimisation*, *Sensitivity analysis*, *Statistical calculations and presentation*, and *Parameter estimation*. Selecting a tool will make it appear in a window on the right-hand side of the screen. With the *Hide* option, or the ‘→’ button at the top-right of the tool, you can remove the tool. The Tools menu has the following options:

- Optim
- Sensi
- StatRes
- ParmVar
- Hide

The tools will be treated in Section 16.

Help menu

The Help menu contains:

- About StochSD
- StochSD Licence
- Third-party Licences
- StochSD User’s Manual
- Optim Manual
- Sensi Manual
- StatRes Manual
- ParmVar Manual
- What is Full Potential CSS?
- Restart and Clear Model
- Restart and Keep Model

‘About StochSD’ very briefly describes StochSD. The ‘StochSD licence’ and ‘Third-party licences’ are found here. The ‘User’s manual’, which you are now reading, is also available here. You can also find more detailed manuals about the tools for *Optimization* (Optim), *Sensitivity analysis* (Sensi), *Statistical calculations and presentation* (StatRes) and *Parameter estimation* (ParmVar). In ‘What is Full Potential CSS? This concept is briefly explained.’ Finally, if StochSD is malfunctioning, you can restart it with or without the current model loaded.

3.2. Buttons

The buttons from left to right are discussed below.

Manipulation buttons

These buttons are for recapturing the ‘unloaded’ mouse state, deleting a block, undoing, and redoing an operation.

- Mouse
- Delete
- Undo
- Redo

Up to 10 undo steps are remembered and can be performed with the **Undo** and **Redo** buttons.

Building block buttons (Primitives)

Here are the building blocks of a StochSD model:

- Stock
- Auxiliary
- Parameter
- Converter (a table look-up function)
- Flow
- Link
- Ghost

More about the Building blocks in Section 4.

Styling buttons

The styling buttons provides blocks for styling the model by moving the names of the building blocks, moving the valve symbol of the flow, straighten links and for inserting texts, Rectangles, ellipses, lines, and arrows.

- Move label around the primitive
- Move valve symbol
- Straighten a link
- Text
- Rectangle (press Shift to get a square)
- Ellipse (press Shift to get a circle)
- Arrow / Line

The label can be moved to a position North, West, South or East of its primitive.

The valve symbol can be flipped, and in a *broken* flow the valve symbol can be placed at the different segments, See Figure 3.3.

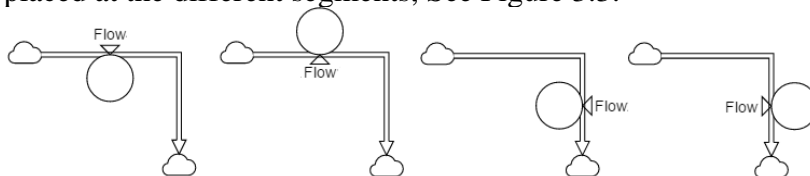


Figure 3.3. The Flow valve can be flipped and moved to different flow segments.

Marking a Link and pressing the Straightening button will make it straight.

Result buttons

Results can be displayed in *Number boxes*, *Tables*, *Time Plots*, *Compare (Simulations) Plots*, *XY Plots* and *Histogram*. The Compare Plot enables plots from two or more simulations in the same diagram.

- Number box
- Table
- Time Plot
- Compare (Simulations) Plot
- XY Plot (Plots Y versus X)
- Histogram

Execution buttons

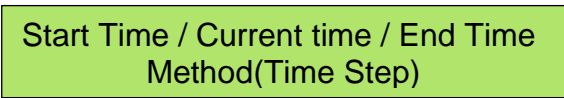
The execution buttons control the simulation (see Section 7). These are:

- Run/Pause
- Step
- Reset

3.3. Time display, Time Unit button and Inspection field

Time display

To the right of the Execution buttons, you find a display, which shows:



Start Time / Current time / End Time
Method(Time Step)

You will also see an orange progress bar expand over the grey field during execution. When the simulation is ended, the bar turns green.

Time Unit button

To the right of the Time display there is a button for the Time Unit, which must be specified before the model building can start. It is crucial to be consistent and choose one, and only one, time unit across the model. The chosen time unit is then displayed on the button.

The time unit can e.g. be second, minute, hour, day, week, month, year, century, or whatever you choose. For a *generic* model you can specify it as e.g. 'Time Unit', 't.u.' or 'tu'.

The Time Unit must contain at least one of the characters A-Z or a-z.

Inspection field

Finally, to the left under the buttons, there is an **Inspection field** where you can see the definitions of a marked primitive. For example, if 'Flow' is defined as 'parameter·STOCK' this will show up as: [Flow] = [Parameter]*[STOCK] when the Flow symbol is marked.

To the far right, you find links to the *home pages* of StochSD and insight Maker.

4. Model building

One of the most common and severe errors in model building is the confusion of the selected time. *There must be one, and only one, time unit consistently used throughout the model.* **Therefore, in StochSD you cannot start the modelling until you have specified what Time Unit you will use!**

The time unit can e.g. be second, minute, hour, day, week, month, year, century. For a generic model where you want to demonstrate a principle that is not related to a specific time duration, you can specify it e.g. as: 'Time Unit', 't.u.', 'tu' or whatever.

4.1 Primitives (Building blocks)

The primitives for building a model are: *Stock*, *Auxiliary*, *Parameter*, *Converter*, *Flow*, *Link* and *Ghost* are denoted **primitives**.



Figure 4.1. The buttons for StochSD building blocks: *Stock*, *Auxiliary*, *Parameter*, *Converter*, *Flow*, *Link* and *Ghost*.

The six first mentioned primitives are described in Chapter 2. With the Ghost button you can duplicate any of the first four primitives (unfortunately not the Flow) to be displayed twice or more.

The building blocks are attached to the Model window and connected to each other in a click-and-draw manner. Before a *Stock*, *Flow*, *Auxiliary*, *Parameter* or *Converter* is defined by a value or formula, it will show a question mark '?'.

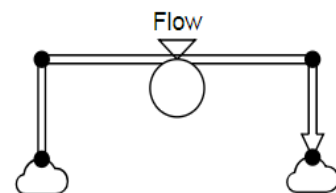
Stock: It is usually practical to start by placing and naming the *Stocks* – which are the primitives you usually (but not always) are most interested in. Then draw and name the *Flows* into, out from or between *Stocks*. Thereafter, place and name *Auxiliaries*, *Parameters* and *Connectors*. Finally, place the *Links* to connect primitives.

To place a **Stock**, **Auxiliary**, **Parameter** or **Converter**, click its button, place the mouse cursor at an appropriate place and drop it by clicking again.

To create a **Flow**, click the Flow button, place the mouse cursor, press the *left* mouse button to anchor the starting point, hold the button down and draw the mouse in horizontal or vertical direction until it reaches its destination where you release the mouse button.

A Flow has a start and an end point that may connect to a *Stock* or it will start and/or end with a cloud symbol to show that the flow content comes from or goes to somewhere outside the scope of the model. If the start or end point is over a *Stock*, it will connect to this *Stock* and the cloud disappears.

A *Flow* can also be *broken* in x and y directions. This is obtained by clicking the *right mouse button* during the drawing operation.



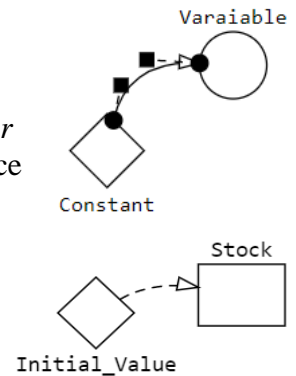
The flow may afterwards be adjusted by clicking the *Flow* and using the anchors (●) at start, end and points where the Flow is broken. The anchors will show up when the flow is marked.

Further, you can remove break points by dragging the end point (the point at the arrow) backwards and right clicking at previous points.

The *name of the flow* can be moved *around* its valve symbol, and the *valve symbol* can be *flipped or moved to another segment of a broken flow* as described in ‘Styling buttons’ below.

=

A **Link** is used to connect two primitives. It means that one primitive affects the other. The *Link* is obtained by clicking its button. Then place the mouse cursor over a *Stock*, *Auxiliary*, *Parameter*, *Converter* or *Flow* and draw it to another of these building blocks – but not *to* the *Parameter* (a *Parameter* may not have incoming links). A *Link* can be bent into a nice Bézier curve. When you click on a link two anchors (●) and two handles (■) will be visible. Then you can grip a handle to bend the link.



A link to a stock only gives the initial value of the stock, i.e. it only acts at ‘time zero’. In System Dynamics the convention is that a link *to* a stock is *dashed*.

To create a **Ghost** of a *Stock*, *Auxiliary*, *Parameter* or *Converter*, first click on the actual building block in the Model window, then click on the Ghost button and finally click on an appropriate place in the Model window. The Ghost is then just a graphical copy of the original primitive with a ghost symbol inserted. (Both the primitive and its Ghost are identical sharing the same definition).

The *Ghost* means that the same *Stock*, *Auxiliary*, *Parameter* or *Converter* may be represented at two or more places in the model. A *Ghost* cannot have any flows or incoming links - but it can have outgoing links!

Using a Ghosts can eliminate long connecting links that cross over the model making the model look ugly and unstructured. You may also use Ghosts to assemble information from e.g. Stocks and Auxiliaries from different parts of a model into an ‘instrument panel’ of Ghosts and Number boxes.

All Primitives can afterwards be adjusted by moving them to new positions.

The *name* of a primitive (*Stock*, *Auxiliary*, *Parameter*, *Converter* and *Flow*) is always starting with ‘[’ and ending with ‘]’. For example [Stock1], [Flow2], [Parameter1]. However, if you write the name without brackets, StochSD will include them. A Link has no name and a Ghost will automatically get the same name as the primitive it ghosts. More about naming in Section 4.3.

The default names should be renamed to be descriptive, e.g. [Rabbits], [Births per month], [Fertility]. All names can be rotated around its symbol with the ‘Rotate name’ button, see Section 4.2.

4.2 Styling buttons

There are eight *Styling buttons*: To move the name label around a primitive. To move the valve symbol of a flow. To straighten a curved Link. Four cosmetic symbols to include *Text*, *Rectangle*, *Ellipse*, and *Line/Arrow* into the model.

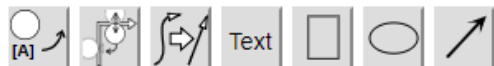


Figure 4.2. The Styling buttons: *Move label*. *Move valve*. *Straighten link*. *Text*. *Rectangle*. *Ellipse*. *Arrow or Line*.

With the ‘*Rotate name*’, the *label* of one or several *marked* primitives can be relocated to North, West, South and East of its primitive.

The ‘*Valve symbol*’ of a marked flow can be flipped upside-down, and in a ‘broken’ flow also be relocated to different segments of the flow. (See Figure 3.3, above.)

The third button *straightens a marked link*. This can sometimes be more convenient than using Anchors or Handles.

Text boxes, Rectangles, Ellipses, Straight lines and Arrows are created by clicking respective button, placing it and drawing it to a proper extension at the Model window. By pressing the *Shift* key when drawing a Rectangle or Ellipse you will obtain a Square or a Circle. The size of these figures can also be adjusted afterwards.

4.3 Naming and defining relations of the primitives

When a primitive is placed, it is time to give it an appropriate *name* and to *define* its value or relation to other building blocks affecting it.

Double-clicking a primitive opens its **Dialog box** where you can specify the *name* and define it in terms of numbers, functions and other primitives linked to it.

Figure 4.3 shows the Dialog box for a flow named [Flow] that is affected by a [STOCK] and a [Parameter].

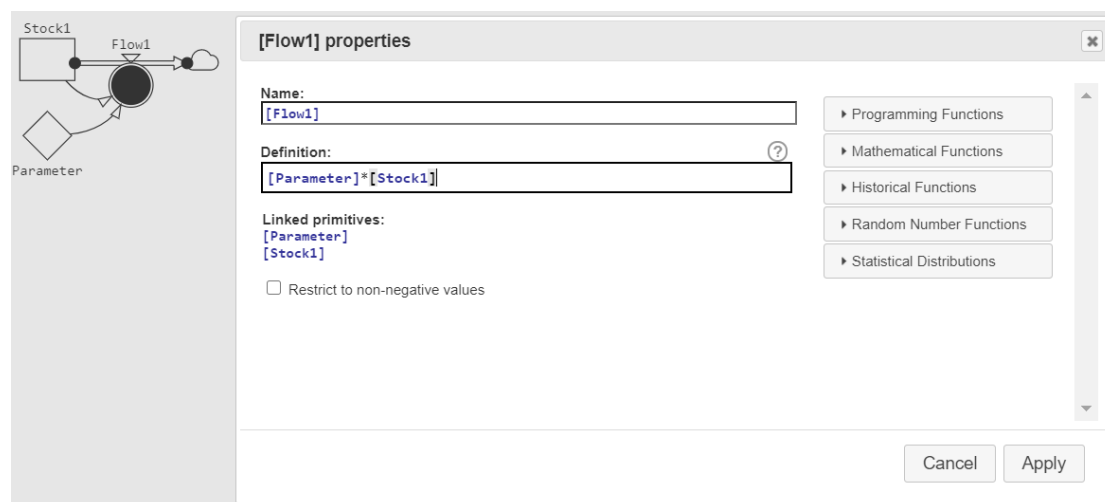


Figure 4.3. To the *left* a small model and to the *right* a Dialog box of [Flow] with: 1) **Name field** where the name can be changed. 2) **Definition field** where the value or function is specified. A comment after # can also be written here. 3) **Linked primitives**: show the primitives that must be included in the Definition field. 4) **Library functions** which can be

used in the Definition field. 5) Check box for **Restrict to non-negative values**. 6) **Cancel** and **Apply** buttons.

For the *Stock*, *Auxiliary*, *Parameter* and *Flow* the Dialog boxes are similar. (A *Ghost* has *the same* (not just a copy) dialog box as the primitive it ghosts.)

The **Dialog box** contains a **Name** field where the name (e.g. [Stock3] or [Auxiliary5]) can be changed to a suitable name in upper- or lower-case characters. *The Name includes brackets, e.g. [Stock1], [Water_Flow], [c].* However, if you write the name without brackets, StochSD will automatically insert them. *However, in the graphical display of the model and in tables and diagrams, the brackets are removed of aesthetic reasons.*

A **Name of a primitive** may *only* contain the characters **A** to **Z**, **a** to **z** and **_** (anywhere) and **0** to **9** (if not the first character) between the brackets.

Below the Name field is the **Definition** field where you define a formula or a value. Here, linked primitives and library functions can be used together with numbers and the arithmetic operators **+** **-** ***** **/** **^** and **()**. Also **{ }** are used for lists in e.g. Converters.

Further below, **Linked primitives** to be used in the Definition are listed. They enter the Definition field when you click them. *It is best to enter the primitives from here with a click*, because then you don't miss the brackets. Without the brackets, StochSD don't understand the definition.

For Stock and Flow building blocks a check box '**Restrict to non-negative values**' is found at the bottom-left. (This option should *seldom* be used.)

To the right you find **Library of functions** subdivided into groups named: *Programming Functions, Mathematical Functions, Historical Functions, Random Number Functions, and Statistical Distributions*. Clicking on such a group menu will give you a list of functions in the chosen category. These functions will be treated in Section 10.

StochSD knows how to calculate the value of a **Stock** from inflows to and outflows from the Stock defined by the graphical structure without further specification. Therefore, a Stock only requires an **initial value** at '*Time Zero*' (i.e. the time when the simulation starts) to be entered into the definition field. Usually, the initial value for a stock is a number, but it is also possible to initiate the Stock to the value of an Auxiliary or Parameter linked to the Stock. The initial value may also be a random number.

In the Definition Box you may also include a **comment** after a hash (#) sign.

Leave the dialog box by clicking the **Apply** button to realize your definition or by clicking the **Cancel** button to leave the dialog box without a new definition.

If there is a mismatch between the list of **Linked primitives** and the **primitives** in the formula (say that only [X] is used, or that you have written $c*[X]$ instead of $[c]*[X]$, or you have written $[c]*[X]*[Y]$, where [Y] is not linked), then you get a *Modelling Error message* when you click the **Apply** button. If the definition passes some tests of this kind, then the question mark '?' in the symbol of the primitive will disappear. More about this in Section 5.2 Error checking.

The **Converter** is a table look-up function that has its own Dialog box. Here you define pairs: x_1, y_1 ; x_2, y_2 ; ..., x_n, y_n of values defining an empirical function of $y = \text{function}(x)$. See Section 10.8.

The **Link** has no dialog box. It just links one primitive to another.

Commenting a relation in a primitive can be done in the definition field of the primitive by, after the relation, writing a hash (#) followed by an explaining text as shown in Figure 4.4. (However, this does not apply for the converter.)

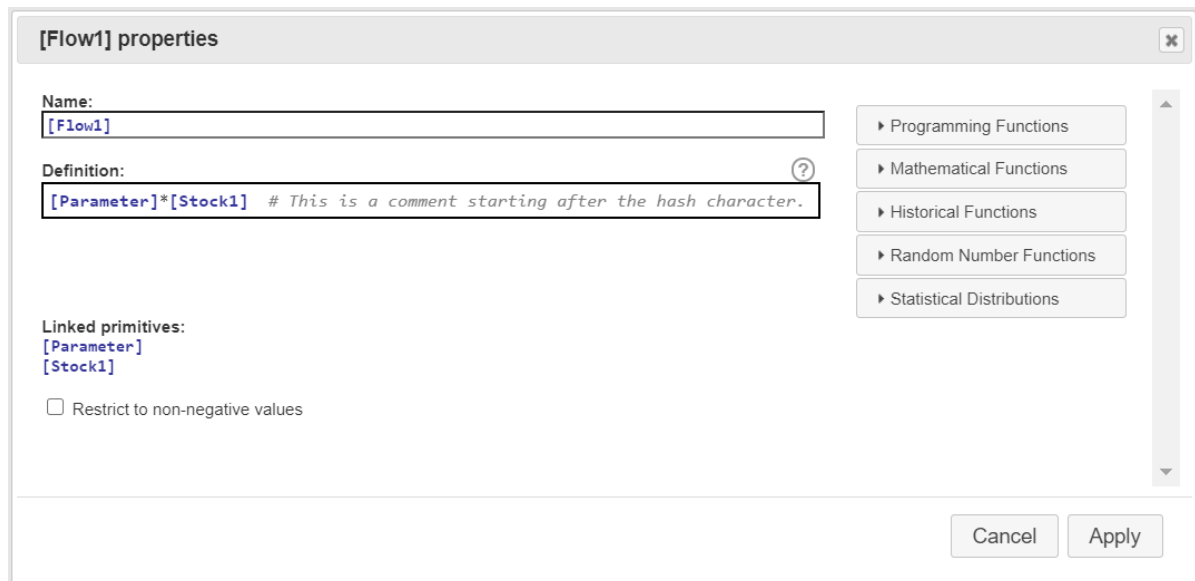


Figure 4.4. Documentation of a relation by free text can be included after a hash (#).

Shift-Enter and Enter

There are three forms where you may want to continue the input on a new line:

- Definition fields of Stock, Flow, Auxiliary, Parameter and Converter
- Macro form
- Text box

In all cases, **Shift-Enter** gives you a new line. However, for the Definition field it is usually better just to continue on the same line letting the field expand. However, the simulation will not work if you break a line in the middle of a Primitive name or a Function. In the Text box there are no such restrictions.

Enter has the same effect as pressing the **Apply** button.

5. Equations

5.1 Defining the equation

The equation (or more correctly ‘algorithm in an assignment statement’⁴) is defined in the Dialog box of a Stock, Flow, Auxiliary, Parameter or Converter.

The left-hand side of the equation is the Name of the primitive. This Name always starts with a *bracket*, ‘[’, followed by a string that may *only* contain the characters **A** to **Z**, **a** to **z** and **_** (anywhere) and **0** to **9** (if not the first character), and ending with a bracket, ‘]’, e.g. [Stock1], [Water_Flow], [c]. However, if you write the name without brackets, StochSD will include them.

The right-hand side of the equation is defined by other Primitives, Functions (always ending with parentheses ‘()’, with or without parameters) and Numbers (which may include a decimal point). Also, the e-format e.g. 1.7e-12 (meaning $1.7 \cdot 10^{-12}$) is allowed.

Primitives, Functions and Numbers are combined by the **arithmetic operators**: **+ – * / ^ ()** for addition, subtraction, multiplication, division, power (x^n) and parentheses. As always, the calculation order is: power, then multiplication and division, and last addition and subtraction. Parentheses ‘()’ can be used to alter this order. Also **{ }** are used for lists in e.g. Converters.

The equations are used exactly as you defined them with one exception. A Stock is only *directly defined* by the initial value you specify, but also *indirectly defined* by the in- and out-flows you graphically attached to it. If you modelled [Stock] with one inflow [F1] and one outflow [F2], then [Stock] will be defined as: $[Stock] := [Stock] + DT() \cdot [F1] - DT() \cdot [F2]$, where [Stock](at Start Time) = Initial value. This algorithm is called the **Euler** method. You can also choose a more advanced algorithm called **RK4**, see the **Simulation Settings** menu in Section 3.1 and Section 6.

When you have defined all primitives, StochSD will automatically sort the model equations in a proper order for updating (described in Figure 7.2, below). When all primitives are defined in a grammatically correct way – no ?-mark in any symbol of a primitive – you can try to run the simulation model. (However, more syntactic errors may now be displayed when a more complete syntax check is made.)

The model equations can be listed and printed. This is described under the **Print menu** in Section 3.1. See also Figure 3.2.

5.2 Error checking

Error checking is made at three occasions:

1. **Definition check.** When you have defined an equation in a dialog box and press the Apply button, a check that all primitives having an incoming link are

⁴ An assignment, like $X := X+1$, is not an equation because X can never be one unit larger than itself. This is an assignment where the assignment sign is $:=$ or \leftarrow . However, we still use the word ‘equation’, because the model is a numerical form of a system of differential equations.

included in the equation and that no primitive without an incoming link is included here.

Further, there is a check of unmatched bracket pairs ((...), [...], and {...}).

Furthermore, there is a check that the number of arguments for some functions are correct.

This is not a full syntax check, but if these conditions are not fulfilled, the ?-signs in a primitive's symbol will remain, otherwise the question mark will disappear.

2. **Syntax check.** When you press the **Run** or **Step** button a full syntax check is performed that tells you about remaining syntax errors.
3. **Run-time check.** Even when the model has passed the syntax check, a run-time error may occur during the execution. For example, if you have defined an Auxiliary with [A]/[B] and [B] gets the value zero during the execution, an overflow will occur because of division by zero.

6. Specification of the simulation

Before you execute the model, you should *specify* what *time period* to study and *how frequently recalculations of the model should be performed*.

The specification of the time handling is done in the **Simulation Settings** menu. Here you specify **Start Time**, **Length**, and **Time Step** for the simulation. If you don't want the default **Euler** method for the updating of Stocks, you may under specific conditions change to the **RK4** method in this menu.

The simulation then proceeds as a stepwise updating of the dynamic model from **Start Time** to **End = Start Time + Length** as shown in Figure 6.1.

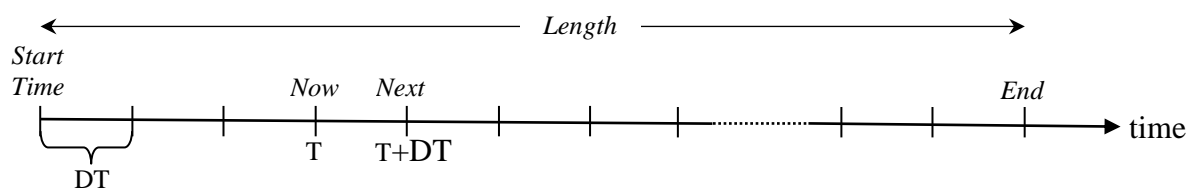


Figure 6.1. The time-handling relating *Start Time*, *Length* and *Time Step* (DT).

For each time step, all **Stocks**, **Auxiliaries**, **Parameters** and **Flows** are recalculated, which will be further described in Section 7. While **Auxiliaries**, **Parameters** and **Flows** are defined as *algebraic* expressions (based on + - * / etc.) without any dynamic complications, updating of the **Stocks** are done by *integration*.

The *integration* of the **Stocks** is as default performed by Euler's algorithm: $X(\text{Next}) := X(\text{Now}) + \text{Inflows}(\text{during Now to Next}) - \text{Outflows}(\text{during Now to Next})$. In StochSD this is expressed as: $X = X + \text{DT} * \text{InFlow} - \text{DT} * \text{OutFlow}$. However, when the model has a smooth development, there are more exact and efficient ways to update the **Stocks**. In StochSD, such an option is the classical 4:th order Runge-Kutta algorithm, usually abbreviated to RK4.

While Euler's method only estimates the change (derivative) at the beginning of a time-step (i.e. at time T), the RK4 method estimates the derivatives at T , $T + \frac{1}{2}\text{DT}$ and $T + \text{DT}$.

Choosing a proper integration method

Because Euler's integration method will work for all kind of models it is the default method to use. However, there are smooth deterministic models where the RK4 method is more accurate and efficient to use. For example, in Figure 6.2 a simple oscillator model is shown that produces $X = \text{Cos}(\text{time})$ and $Y = \text{Sin}(\text{time})$. In an XY Plot that should give a perfect circle. However, this system is at the verge of stable. Using Euler, this model requires a step size, DT, of about 0.001, while RK4 makes a good job with $\text{DT} = 0.1$ or even smaller. Figure 6.2.

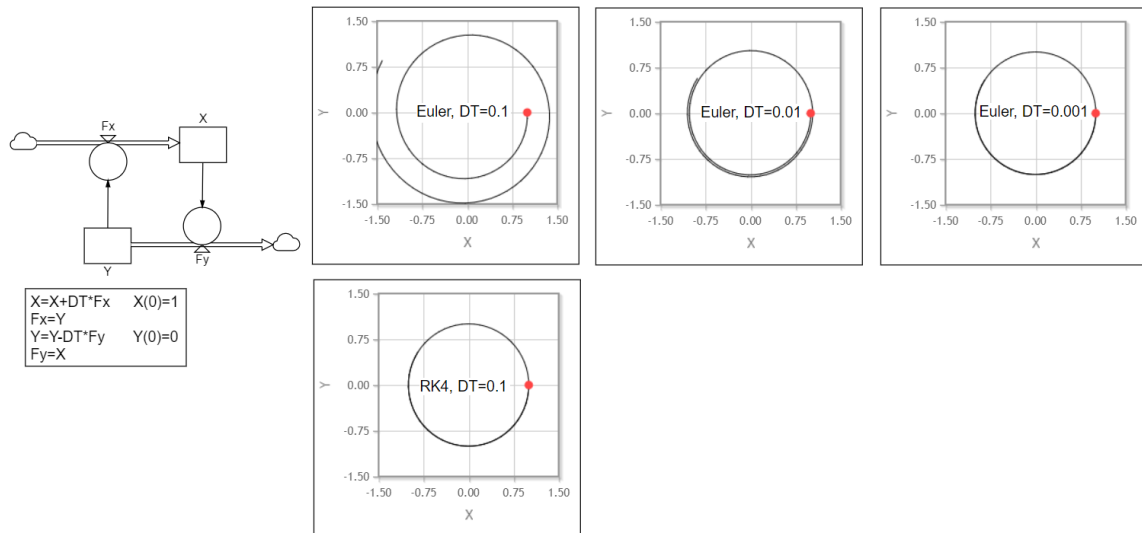


Figure 6.2. For some well-behaved models RK4 will be a good choice.

However, RK4 assumes continuity in F and its derivatives to not be fooled. Discontinuities in InFlows or OutFlows or their time derivatives to or from a Stock, when e.g. a Pulse, Step, Ramp, Table function etc. is involved will distort the RK4 calculations. In Figure 6.1, a Pulse delivering a *unity* content to an empty Stock will give the content of 1, which it does when using Euler's integration. However, with the classical RK4, the Pulse may erroneously only deliver e.g. $2/3$ of the content.⁵

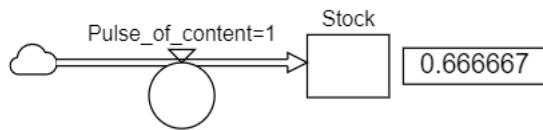


Figure 6.3. An example of the fatal mistake of using RK4 for models containing discontinuities.

For *stochastic models* (treated in Part II), which change randomly at each time-step, neither Flows nor its derivatives are smooth. Then the classical RK4 will then severely distort the results and should absolutely NOT be used.

WARNING: Do not use RK4 without a good reason, and NEVER if the model contains discontinuities (e.g. Pulse, Step or Random numbers)!

This warning is also displayed in StochSD when you select RK4 as the integration method.

Choosing a proper time-step

A short time-step (DT) increases the accuracy of the calculations but it also increases the number of time-steps between **Start Time** and **End**, and thus also prolongs the

⁵ When RK4 is used in this Pulse example, it will work in some cases but not in other depending on the synchronisation between actual time (calculated as Start Time+ $DT + DT \dots + DT$) and specified Pulse time.

execution time. You should, therefore, try to find a compromise where the time-step is sufficiently short to give good accuracy - but not much shorter than this.

Theoretically, the time-step should be considerably shorter than the shortest time-constant in the model. However, these time-constants are often complicated to theoretically calculate. Therefore, finding an appropriate time-step is a practical task that should be done for every new model before using it for a study.

A practical way to find an appropriate time-step of a deterministic model is the following:

Choose a time-step and perform a simulation. Then increase or decrease the time-step (by a factor of e.g. two or ten) and investigate if it has any important effect on the simulation results. If the results change significantly, then continue to decrease the time-step. (For well-behaved deterministic models, you may also consider a more efficient integration algorithm, e.g. RK4.) If the time-step is unnecessarily short, increase it.

For this purpose, you may use the Compare Simulations Plot described in Section 8.4, where this is exemplified.

For a stochastic model, such a comparison is more complicated, because the outcomes (with the same or with different DTs) will vary randomly. A heuristic way is to use a deterministic version of the model to find a proper time-step (using the Euler method), and then use half that time-step for the stochastic model.

Another way is to make, say 100 or 1000, replications for each time-step and then compare the average results for the different time-steps.

Limitation of the number of time-steps

For each time-step the values of all primitives are calculated and saved. This means that a simulation over, say, 1000 time units with $DT=0.01$ time unit requires 10^5 time-steps to calculate and save for the primitive in the model. This produces a number of consequences for saving, plotting and garbage collection that will slow down the performance significantly when the number of time-steps are very large. It also means that the simulation can require a considerable time between the calculations and the presentation of the results in Plots and Tables.

Of these reasons, StochSD issues a note (warning) when you in 'Simulation Settings' specify more than 10^4 time-steps (calculated as $Length/DT$), and it refuses to accept more than 10^5 time-steps per simulation.

7. Simulation

A simulation of the model is performed by pressing the **Run/Pause** button. It is also possible to test the model behaviour with the **Step** button. You can also reset the calculations with the **Reset** button.



Figure 7.1. The execution buttons: **Run/Pause**, **Step** and **Reset**. (To the right of the buttons the **Start Time / Actual simulation time / End Time** (before the run) and below the **Method** and **Time Step** are displayed).

Pressing the **Run** button (▶) starts the simulation. The results of the simulation is then displayed in Number boxes, Tables, Time Plots, Compare Simulations Plots, XY Plots, or Histograms see Chapter 8.

With the **Step** button (▶|) you can run a single time-step at a time. Finally, (◀◀) resets the model to *Time Zero* (i.e. your chosen Start Time).

The calculations behind the scene during a simulation

What happens during a simulation is that StochSD, time-step by time-step, recalculates the equation system. The working procedure used is the following (DT is the time-step used and End Time is the Start Time + Length of simulation):

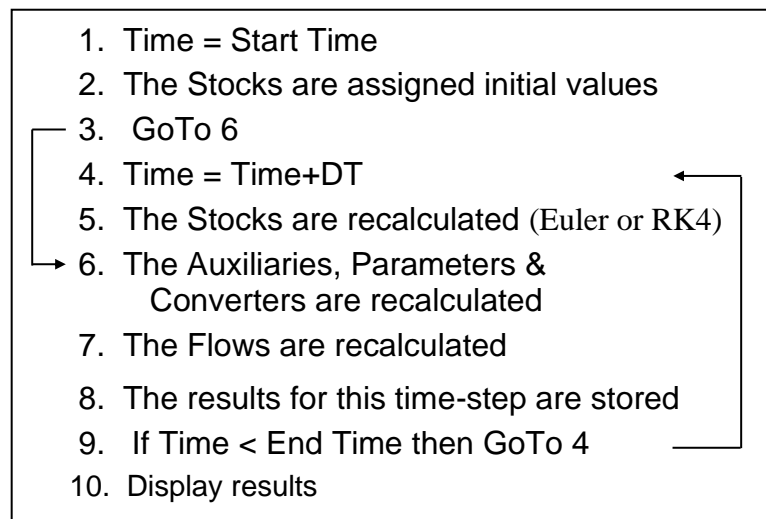


Figure 7.2. The calculations performed behind the scene.

8. Results presentation

To present the results of a simulation run you can use **Number Box**, **Table**, **Time Plot**, **Compare Simulations Plot**, **XY Plots** and **Histogram**.

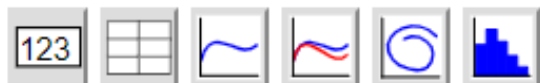


Figure 8.1. Result presentation buttons for **Number box**, **Table**, **Time Plot**, **Compare Simulations Plot**, **XY Plot** and **Histogram**.

Some common features

A **Table**, **Time Plot**, **Compare Simulations Plot**, **XY Plot** or **Histogram** is obtained by clicking respective result button. Then you place the cursor on a proper place on the screen and click-and-draw the item to a proper size. Tables and plots can also afterwards be drawn to a proper size by using the handles.

By double-clicking the **Table**, **Time Plot**, **Compare Simulations Plot** or **XY Plot**, you can choose the values of the primitives to be displayed over the simulation run. You can also choose to only display what happens during a selected period of the simulation run and how often the results should be tabulated or plotted. The plots are automatically scaled after a simulation, but you can select your own scaling. In the plots, you can hover over a line with the mouse cursor to read the coordinates of the points of the line. Part of the above also applies to the **Histogram**.

8.1 The Number Box

The **Number Box** is particularly useful to show the values of Stocks, Parameters and Auxiliaries. This means that you in the Model diagram can directly see the value of a parameter, the initial value of a Stock before the simulation or its final value after the run. See Figure 8.2.

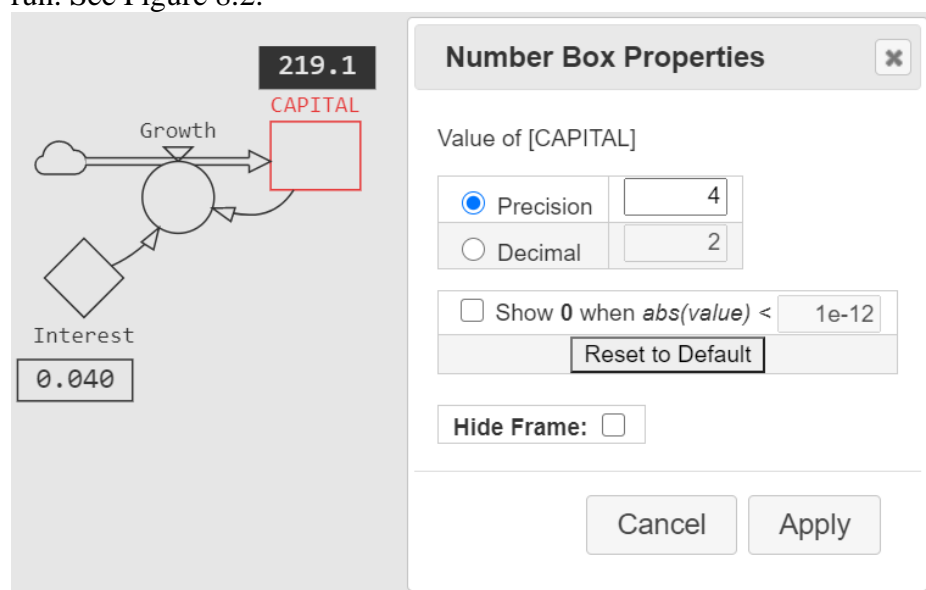


Figure 8.2. Use of Number Boxes. Here capital growth of 100 Euro at 4% interest after 20 years is shown. (DT=1 to create annual interest in the way the bank will calculate it.)

To use the Number Box, you must first mark the primitive you want to watch. Then click and place the Number Box.

By double-clicking the Number Box, you can select Precision and Number of Decimals. You can also choose to show zero for very small numbers, e.g. for $< 10^{-12}$.

8.2 The Table

By double-clicking on a Table, you open its **Table Properties** form to specify what primitives to tabulate. Here, you can also choose to show only a part of the data and select the time intervals between the rows.

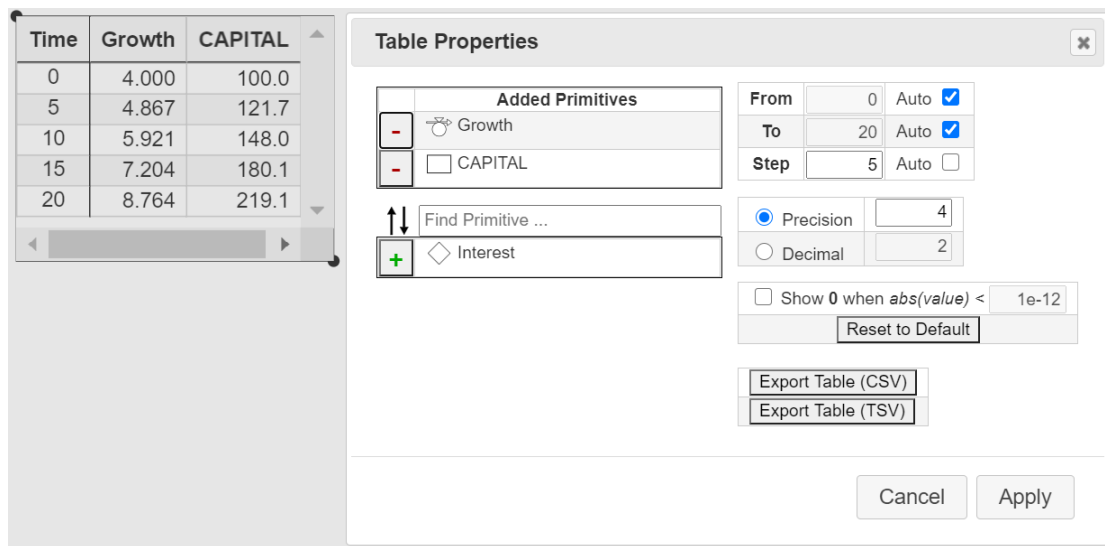


Figure 8.3. The **Table Properties** form and a **Table** presenting the results of Interest, CAPITAL and Growth of 100 Euro at 4% interest (from the model in Figure 8.2). Here we have chosen to show each second year.

By double-clicking the Table you can select Precision and Number of decimals. You can also choose to show zero for very small numbers, e.g. for $< 10^{-12}$.

From the **Table Properties** form, you can also export your values as Comma-separated values (CSV) or Tab-separated values (TSV) for use or analysis in e.g. a spreadsheet.

8.3 The Time Plot

By double-clicking on a Time Plot, you open its **Time Plot Properties** form, see Figure 8.4.

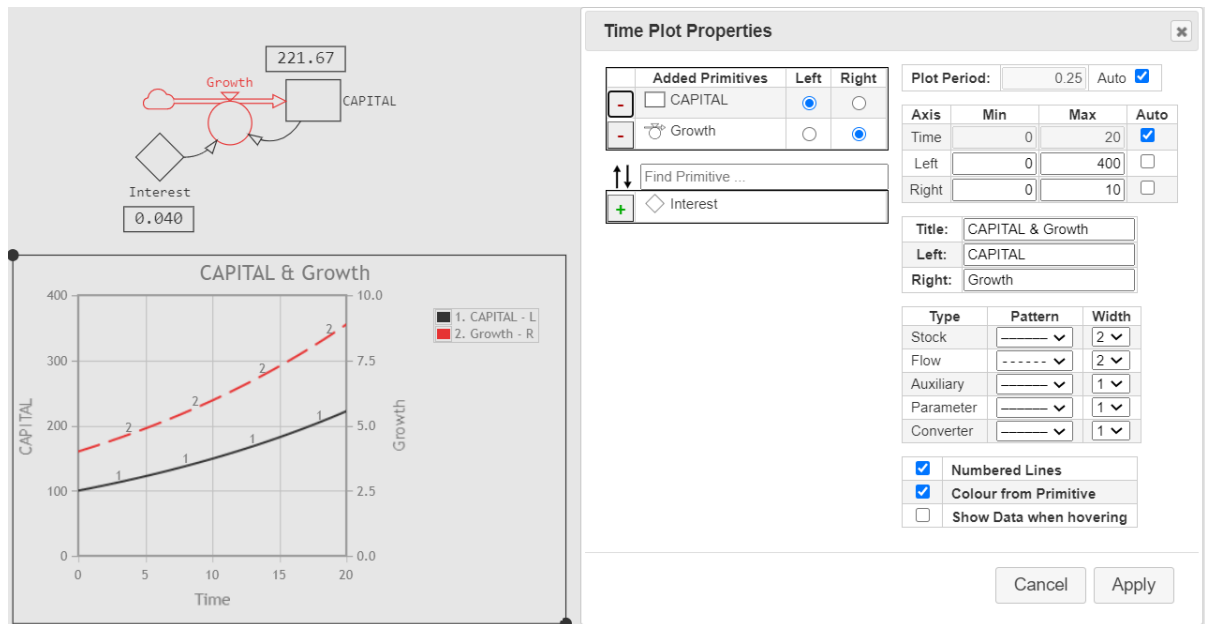


Figure 8.4. The **Time Plot Properties** form and a **Time Plot** presenting the results of **CAPITAL** and **Growth** (from the model in Figure 8.2). Here we have related **CAPITAL** to the left y-axis and **Growth** to the right one. We have also added a **Title**.

In the **Time Plot Properties** form you mark what quantities to present and whether to relate them to the left axis or to the right one. We can also add a **Title** and **Labels** for the axes. Further, we can let **StochSD** to give the lines different colours, or let each line have the same colour as its primitive. Having the same colour of the primitive and its line in a plot reduces the risk of misunderstanding.

In a **Time Plot**, a **Stock** will be drawn with a *solid line*, a **Flow** with a *dashed line* and other types of primitives with a *dotted line*. Finally, we can use **Numbered Lines**, and choose between **Thick** and **Thin** lines.

“Show data when hovering:” means that you can read time and value of the point of the curve your mouse hovers over.

8.4 The Compare Simulations Plot

The **Compare Simulations Plot** (or just **Compare Plot**) can display results from different simulations in the same plot. This is especially helpful in two cases. First, you can use it to find a proper time-step (see Section 6). Second, you can use it to see the effect on an important output when changing e.g. a parameter or initial value.

In Figure 8.5, we show the behaviours of a logistic model ($dX/dt = a \cdot X - b \cdot X^2$; $X(0)=2$, $a=1$, $b=0.02$) for 5 simulations with different values of **DT**.

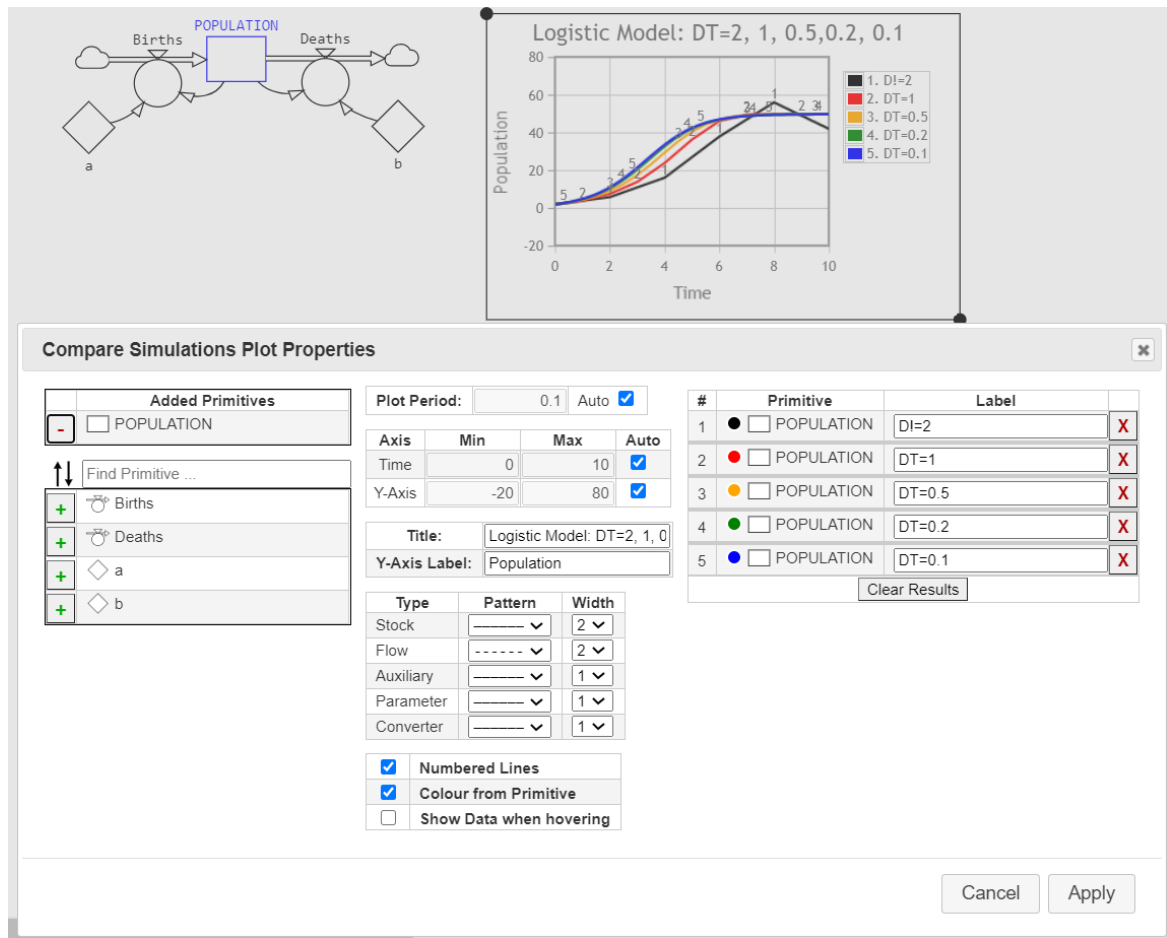


Figure 8.5. The **Compare Simulations Plot** presenting the logistic growth of a population. This device can help you to find a proper time-step, DT. Here you see that DT=2, 1 & 0.5 give a bad accuracy. DT=0.2 & 0.1 are almost similar. So, choose a time-step ≤ 0.2 (depending on the accuracy required).

The **Compare Simulations Plot Properties** form is similar to that of the Time Plot, but can display results from different simulations and it has only one y-axis. Further, you can remove one, several or all results.

8.5 The XY Plot

A **XY Plot** displays one quantity (y-axis) versus another quantity other than Time (on the x-axis). In an XY Plot each point in time is displayed with its coordinates (x,y), which makes a line when times progresses – but the time is implicit and is not shown in the plot.

In a **XY Plot** you can choose between *Line*, *Markers* or *both*. The colour is black.

A Volterra model describes the prey-predation relation between species. In Figure 8.6 the behaviour of a simple Volterra model of Rabbits and Foxes is shown in an XY Plot.

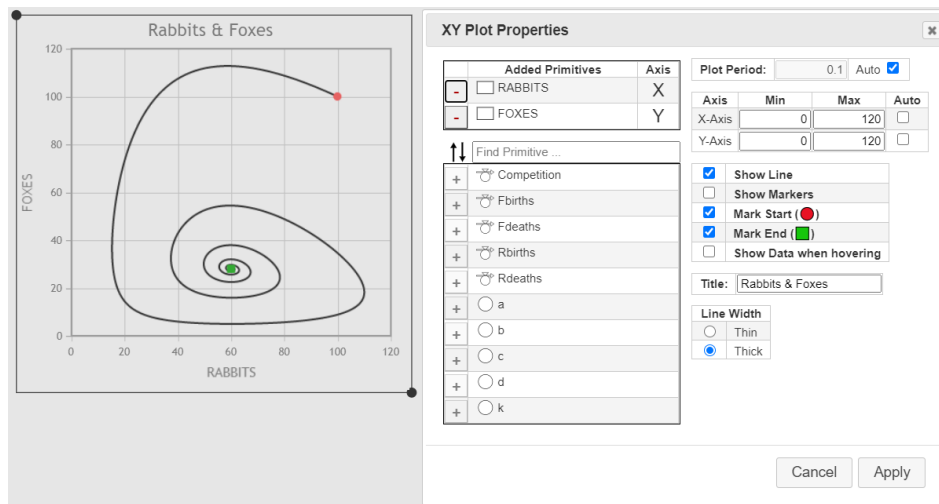


Figure 8.6. A XY Plot of Rabbits versus Foxes in a prey-predator model. The start- '●' and/or end-points '■' was indicated.

In the XY Plot all combinations between Stock, Flow, Auxiliary, Parameter and Converter will be drawn with a *solid line*.

In the XY Plot, *markers* can be used instead of a line to provide a **Scatter Plot**. You can also use a line together with *markers* to indicate the time. You get one mark per Plot Period, which you can specify – but if you use a Plot Period longer than DT, you will have straight lines between the plot times. Alternatively, you can hover over the line with the mouse cursor to display the coordinates as: (x, y, Time).

8.6 The Histogram

With a **Histogram** the frequency of different outcomes can be displayed. This is particularly useful for stochastic models. For example, a student may want to study the outcomes of a random number generator based on a given distribution and parameter values. A more important use can be to study how often a wanted or feared situation or event will happen. For example, how often will a water dam reach dangerous or catastrophic levels.

In Figure 8.7 a Histogram over 10 000 outcomes of a Poisson distributed random number generator with the parameter value 3.5 is shown.

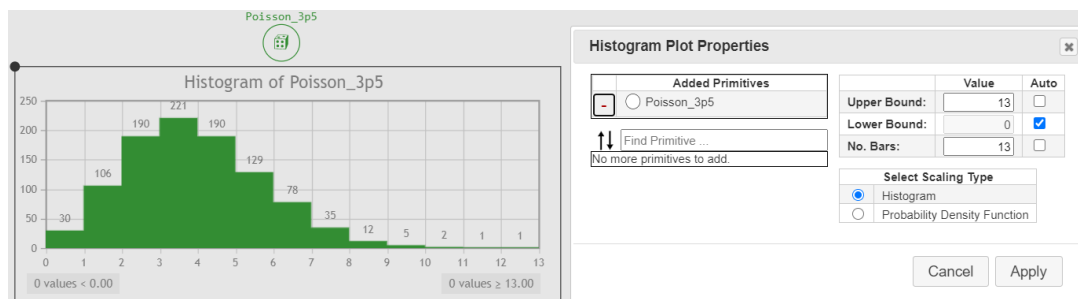


Figure 8.7. A Histogram of 10 000 outcomes of a Poisson(3.5) random number generator.

You have also the choice to transfer this to a probability density/distribution (a pdf) that (approximately) shows the statistical distribution *normalized* to an area of one. (You can also display a fraction of the outcomes between the chosen limits.)

9. StochSD model examples

Example 9.1. Filling a bathtub with an open outlet

Construct the following model to see what happens when water flows into and out from a bathtub.

We assume the following:

- 1) A BATHTUB is empty at Start Time.
- 2) The InFlow is regulated from the Valve.
- 3) The Valve of the InFlow is opened to give 30 litres/minute for 5 minutes and then closes.
- 4) The OutFlow is open, so 15% of the amount of water in the BATHTUB will leave the tub each minute.

What will happen during the next half hour?

Specify the time unit. **Time Unit:** *Minute*

Write the equations of the model:

[BATHTUB] = [BATHTUB]+DT()*([InFlow] – [OutFlow])
 [BATHTUB] = 0 (* Initial value *)
 [InFlow] = IfThenElse(T(<5, [Valve], 0) (* See Section 10.1 *)
 [OutFlow] = [c]*[BATHTUB]
 [Valve] = 30
 [c] = 0.15

Set **Start Time** to 0 and **Length** to 30 (minutes).

Set the **Time Step** (DT) to e.g. 0.1 (minute), so the equation system becomes updated for each 6 seconds until the simulation is complete.

You could have chosen the **Time Unit** to be 1 second, 1 minute or 1 hour. But you have to be *consistent* and *recalculate* all time data to the selected time unit!

The content of the bathtub is simulated for 30 minutes as shown in a **Time Plot** in Figure 9.1.

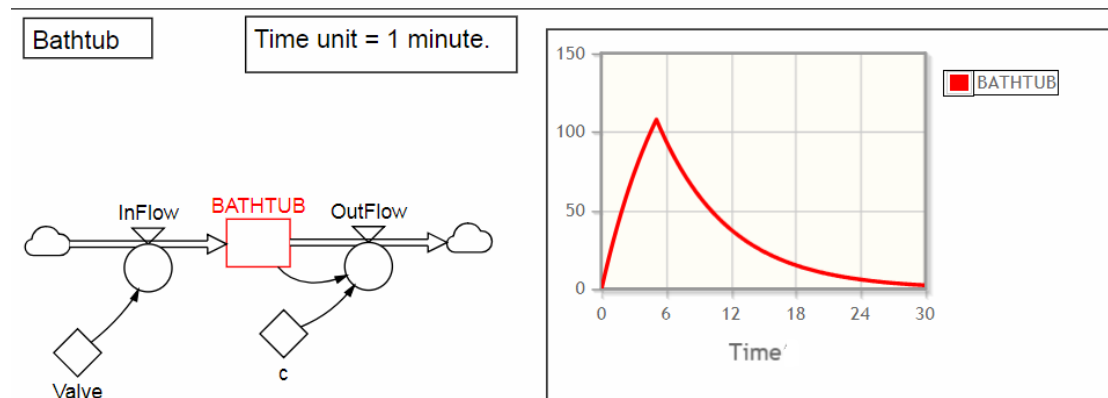


Figure 9.1. A simple bathtub model and its behaviour. ■

Example 9.2. Rabbits on an isolated island

We wish to study the progress of a rabbit population on an isolated island in the sea where 10 rabbits are introduced. They eat, breed and die of old age. The food is supplied at a constant rate of 100 kg of carrots per month. Further, the number of rabbits born per month is proportional to the size of the population and to the *square root* of the amount of food per rabbit and month. The fertility parameter has been calculated to 0.2. Furthermore, the average length of life is 20 months, which is about the same as 5% of the population dying each month.

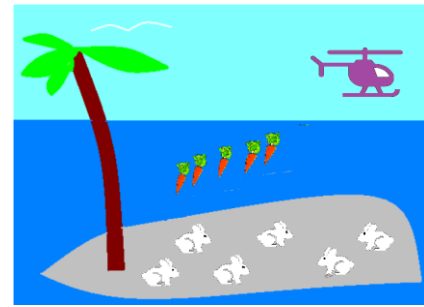


Figure 9.2. Picture of the system under study when dropping the carrots.

A so-called causal-loop diagram for the model is shown in the figure below. This is a type of diagram often used in System Dynamics that focuses on the structure of components and how they causally affect each other. From this diagram, you can see positive and negative loops that generates regulation and growth, respectively. The causal-loop diagram can then serve as a blueprint to the so-called Forrester diagram based on Stocks, Flows, Auxiliaries, Parameters and Links.

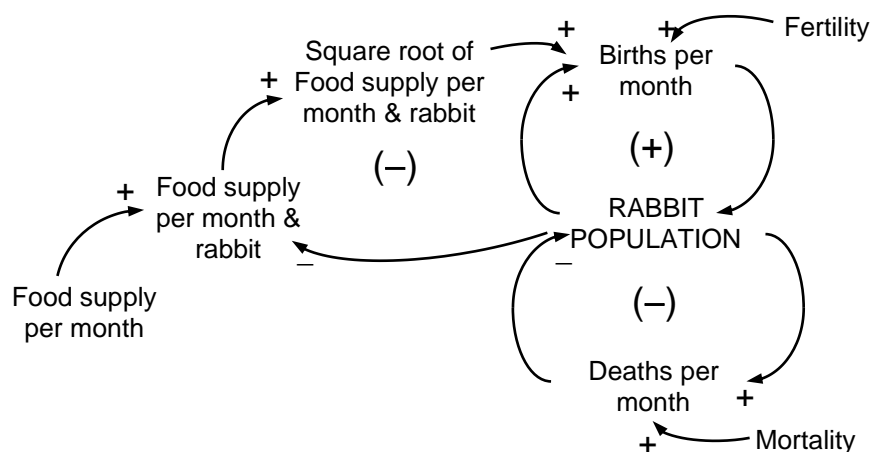


Figure 9.3. A causal-loop diagram of the rabbit system.

From the causal-loop diagram, we can start the StochSD modelling process based on the primitives of this language.

First, specify the time unit.

Time Unit: *Month.*

Then, build the model structure and define the equations of the primitives to create the StochSD model shown to the left in Figure 9.4.

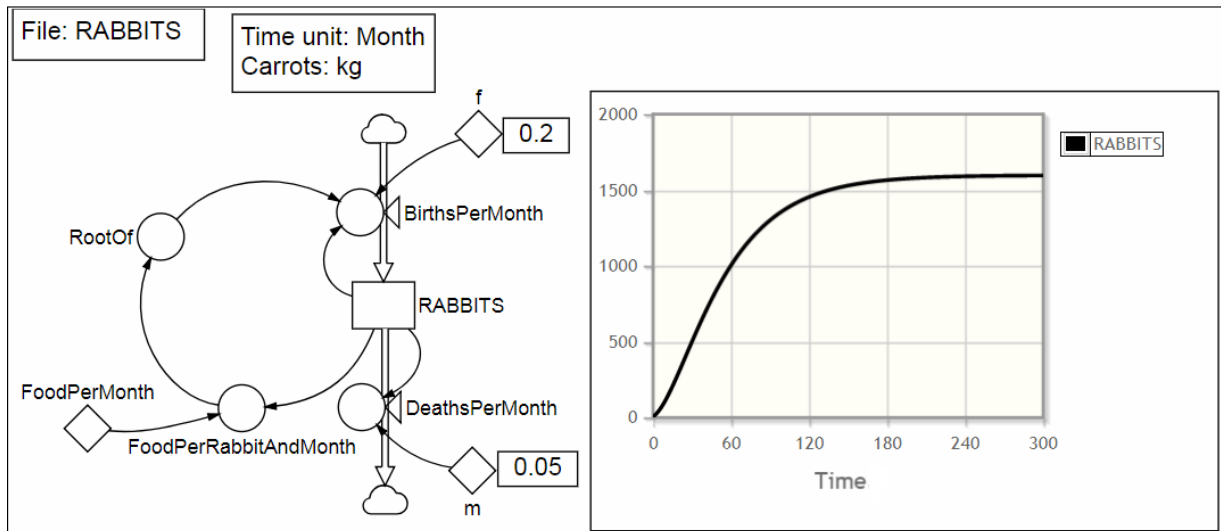


Figure 9.4. A StochSD model of the rabbit system and a Time Plot showing the RABBIT population over 300 (months).

The definitions of the primitives make the equation system of the StochSD model.

$$[RABBITS] = [RABBITS] + DT() * ([BirthsPerMonth] - [DeathsPerMonths])$$

$$[RABBITS] = 0$$

$$[BirthsPerMonth] = [RABBITS] * [RootOf] * [f]$$

$$[DeathsPerMonths] = [RABBITS] * [m]$$

$$[RootOf] = \text{Sqrt}([FoodPerRabbitAndMonth])$$

$$[FoodPerRabbitAndMonth] = [FoodPerMonth] / [RABBITS]$$

$$[FoodPerMonth] = 100$$

$$[f] = 0.2$$

$$[m] = 0.05$$

Set **Start Time** to 0 and **Length** to 300 (months) and the **Time Step** to e.g. 0.5 (months).

The simulation shows the growth of the RABBIT population, seen to the right in Figure 9.4.

We could also have used a **Table** and selected e.g. *RABBITS*, *BirthsPerMonths*, *DeathsPerMonths*, and *FoodPerRabbitAndMonth*. ■

10. Functions

In StochSD there are a large number of library functions. The functions are sorted into different sub-libraries according to categories. You find them to the right in the Dialog boxes of a **Stock**, **Flow**, **Auxiliary** or **Parameter**. Here, the functions are first listed as an overview. Then a detailed presentation is then given.

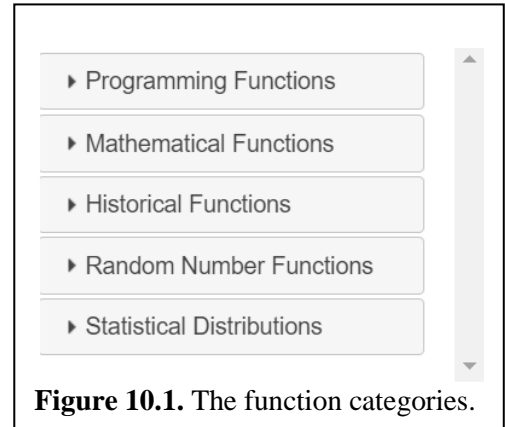
The categories of functions in StochSD

1. Programming Functions

- IfThenElse (Two-way choice)
- If-Then-Else (Structured If-Then-Else function. More flexible.)
- Max (Returns the largest of e.g. [X], [Y], ...)
- Min (Return the smallest of e.g. [X], [Y], ...)
- StopIf (Terminates the run at a defined event)
- Function (Define a new function)
- Throwing Error (Here you can specify your own message)

2. Mathematical Functions

- Current Time
- Time Start
- Time Step
- Time Length
- Time End
- Round (Rounds to nearest integer)
- Round Up (Rounds up to nearest integer)
- Round Down (Rounds down to nearest integer)
- Pulse
- Step
- Ramp
- Sin (Argument in radians)
- Cos (Argument in radians)
- Tan (Argument in radians)
- ArcSin (Finds the arc-sine of a value. Return value in radians.)
- ArcCos (Finds the arc-cosine of a value. Return value in radians.)
- ArcTan (Finds the arc-tangent a value. Return value in radians.)
- Log (Logarithm base 10)
- Ln (Logarithm base e)
- Exp (Exponential)
- Abs (Absolute number)
- Mod (Gives remainder of the division between two numbers)
- Sqrt (Square root)
- Sign (Greater, Equal or Less than zero gives +1, 0 or -1.)
- pi 3.1415926...
- e 2.7182892...
- eps $2.22... \cdot 10^{-16}$. Machine epsilon: the smallest number > 0 .



3. Historical Functions

- Delay (Static delay)
- Delay1 (Dynamic delay of order 1)
- Delay3 (Dynamic delay of order 3)
- Smooth
- PastMax
- PastMin
- PastMedian
- PastMean
- PastStdev
- PastCorrelation
- Fix (Samples and holds a value for specified time periods.)

4. Random Number Functions

- PoFlow (Simplified random values function for a Poisson flow)
- Rand (From a Uniform distribution)
- RandNormal (From a Normal distribution)
- RandLognormal (From a Log-Normal distribution)
- RandBernoulli (From a Boolean distribution)
- RandBinomial (From a Binomial distribution)
- RandNegativeBinomial (From a Negative Binomial distribution)
- RandPoisson (From a Poisson distribution)
- RandExp (From a Exponential distribution)
- RandBeta (From a Beta distribution)
- RandGamma (From a Gamma distribution)
- RandTriangular (From a Triangular distribution)
- RandDist (From a Customized distribution)

5. Statistical Distributions

In 'Statistical Distributions' you find Pdf:s, Cdf:s and Inversions of *Normal*, *Lognormal*, *t*, *F*, *Chi square*, *Exponential* and *Poisson* distributions. These distributions can be used in some advanced statistical studies. However, these distributions will not be further discussed in this manual.

6. Converter (A primitive)

- The *Converter* is an often-useful empirical graph function (sometimes called table-look-up function). The *Converter* is implemented as a primitive with its own button located in the row of building blocks (see Section 3.2). Here you can enter x-values to obtain y-values. Although implemented as a primitive, it behaves as a function why it is discussed (last) in this chapter.

Note that functions have a pair of parenthesis – function(...) – to tell StochSD that they are functions. This applies even the function has no argument (e.g. T() and DT() for Time and Time Step). The constants Pi, e and Eps takes no parentheses.

Below you find a detailed description of these functions.

10.1 Programming Functions

IfThenElse – Two-way IfThenElse function

Syntax: IfThenElse(Test Condition, Expression if True, Expression if False)

Result: If the logical *Condition* is True then the first *Expression* is evaluated and its *Value* is returned, if it is False then the second *Expression* is evaluated and its *Value* is returned.

Example: If(A < B, A, B) returns the smallest of A and B.

Example: If(A < B, 5, 2*sqrt(25)) returns 5 if A < B, otherwise 10.

Logical operators are: <, >, <=, >=, = and <> ('not equal to'), AND, OR.

Syntax: A < B, A > B, A <= B, A >= B, A = B, A <> B, X>=A AND X>B

Example: A <> B will return True if A ≠ B and False if A = B.

If-Then-Else – Structured If-Then-Else function

Syntax: If Condition Then
Expression
Else If Condition Then
Expression
Else
Expression
End If

Result: Tests one or more conditions and selectively executes code based on these tests.

Example: In a model the initial number of Stock (S) at time zero is unknown. In 60% it is expected to be $S(0)=1$, in 30% to be $S(0)=2$ and in 10% to be $S(0)=3$ persons. Then you can draw a uniformly distributed random number between 0 and 1 and use this to initialize the Stock at time zero.

```
If Rand() < 0.6 Then
  1
Else If Rand() < (0.6+0.3) Then
  2
Else
  3
End If
```

Compare with the simple IfThenElse function, above, which only has two options.

Max – Maximum function

Syntax: Max([X], [Y], ...)

Result: Returns the largest value in the list.

Min – Minimum function

Syntax: Min([X], [Y], ...)

Result: Returns the smallest value in the list.

StopIF – StopIf function

Syntax: StopIf(Condition)

Result: If the Condition becomes True, then the simulation run is terminated after the current time-step.

Example: StopIf([X] < 0.5)
 [When the [X] goes below a certain value (e.g. 0.5) the simulation run terminates.]

Function – Define a new function

Syntax: Function Name()
 Expression
 End Function

Result: Creates a reusable function. Especially useful as a Macro – see Chapter 11.

Example: Cube(x) # defined as a Macro ◇ [In]: T() # in the Model
 x*x*x ↓
 End Function ○ [T3]: Cube([In])

 Gives: 0, 1, 8, 27, 64, ... for DT=1

Throw Error – Error message function

Syntax: Throw "YOUR MESSAGE"

Result: If the foreseen error occurs, then the simulation run is terminated and YOUR MESSAGE is shown.

Example: See Figure. Note, Run (▶) is pressed the simulation will be aborted and your message is shown. Here the simulation was run stepwise (▶|) why you also see the table values before the abortion. After 4 time units X was smaller than Out why, in the next step X becomes negative and Out = Sqrt(Negative number) causes the abortion.

(DT=1 was used.) Here the Table values was displayed DT by DT using the step button (▶ |) to display the timesteps before the crash.

The screenshot shows a simulation interface. On the left is a model diagram with a box labeled 'X' and a circle labeled 'Out'. Below the diagram is a code editor with the following text:

```

MODEL:
=====
[X]:
-----
10 # Initial value

[Out]:
-----
If [X] >= 0 Then
  Sqrt([X])
Else
  Throw "Square root of a NEGATIVE NUMBER"
End If

```

To the right of the code editor is a table with the following data:

Time	X	Out
0	10.00	3.162
1	6.838	2.615
2	4.223	2.055
3	2.168	1.472
4	0.6955	0.8340

Overlaid on the bottom right of the table is an alert dialog box with the title 'Alert' and the message 'Message from engine: Square root of a NEGATIVE NUMBER'. An 'OK' button is visible at the bottom right of the dialog.

Note: Alternatively, this could have been prevented by checking: Restrict to non-negative values” in the Definition box for [X].

10.2 Mathematical Functions

T – Current time

Syntax: T() or T

Result: Gives the current simulation time.

TS – Start time for the simulation

Syntax: TS() or TS

Result: Gives the start time for the simulation. This value is specified in the **Simulation Settings** form.

DT – Step-size

Syntax: DT() or DT

Result: Gives the step-size used for the simulation. This value is specified in the **Simulation Settings** form.

TL – Time length of simulation

Syntax: TL() or TL

Result: Gives the length in time for the simulation. This value is specified in the **Simulation Settings** form.

TE – End time for simulation

Syntax: TE() or TE

Result: Gives the End time for the simulation.
 $TE() = TS() + TL()$.

Round - Rounding function

Round([X]) Rounds [X] to its nearest integer.

Round Up - Rounding up function

Ceiling([X]) Rounds [X] **up** to its nearest integer.

Round Down – Rounding down function

Floor([X]) Rounds [X] **down** to its nearest integer.

Pulse – Pulse function

Syntax: Pulse(Time, Volume, Repeat)

Result: Creates a pulse input at the specified *Time* with the specified *Volume*. *Repeat* is optional and will create a pulse train with the specified *Repeat* as interval if positive. With a negative *Repeat* value you get only a single pulse at *Start*.

Example: Pulse(2, 1, 4) generates repeated pulses with the *Volume*: 1, starting at time=2 and then repeating each 4 time units, see Figure 10.2.

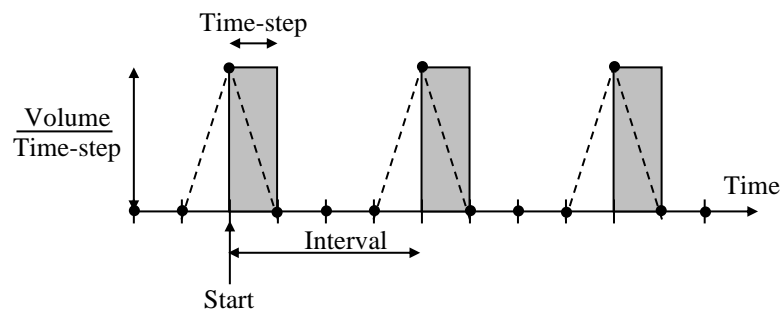


Figure 10.2. A Pulse.

An *ideal* pulse is infinitely high and infinitesimally wide, so that the Height*width equals the specified Volume. The smaller time-step - the better the approximation of an ideal pulse.

The graph in the figure is shown for the time-step = 1 time unit. (For e.g. time-step = 0.1 the pulse will be 10 times higher and have a tenth of the width. The interval between the pulses will still be 4 time units – which now is 40 time-steps.)

Note the difference between what happens mathematically (grey rectangles) and what it looks like in a graph where the lines (dashed) are connected to the values (dots) at each time-step.

[The Pulse replaces the awkward Pulse in Insight Maker where the pulse-content (volume) changes when you adjusted the time-step.]

Step - Step function

Syntax: Step(Start, Height)

Result: Generates a step at time *Start* with amplitude *Height*. That is, it returns 0 if $\text{TIME} < \text{Start}$, and *Height* otherwise. See Figure 10.3.

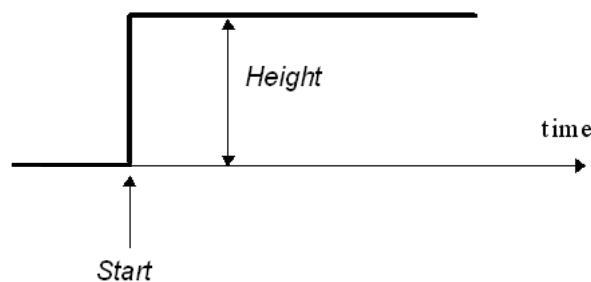


Figure 10.3. The Step function.

Ramp - Ramp function

Syntax: Ramp(Start, Finish, Height)

Result: Generates a ramp, which at *Start* goes linearly from 0 to *Height* between the *Start* and *Finish* times. Before *Start* the value is 0; after *Finish* the value is *Height*.

Sin - Sinus function

Sin([X]) Note: [X] in radians. (Sin([X]*180/Pi) for [X] in degrees.)

Cos - Cosinus function

Cos([X]) Note: [X] in radians. (Cos([X]*180/Pi) for [X] in degrees.)

Tan - Tangent function

Tan([X]) Note: [X] in radians. (Tan([X]*180/Pi) for [X] in degrees.)

ArcSin – Arcus Sinus function

ArcSin([X]) Note: Return value is given in radians.
(180/pi*ArcSin(..) for degrees.)

10.3 Historical Functions

Functions that depend on what has happened *earlier* are called Historical Functions.

Short Introduction to delays

A delay is, for example, the time from which a product is ordered until it is delivered, or the time from which a person becomes infected until he becomes sick. Also, information can be delayed, for example, the time to deliver a message. A delay can be *static* or *dynamic*.

A *static delay* just delays the time from an input to an output *without changing the time variations* of the input. StochSD has the static delay: `Delay([Primitive], Delay Length, Default Value)`. Because the `Delay` looks `Delay Length` time units back, it needs a `Default Value` as long as it looks at a time before the Start Time of the simulation.

A *dynamic delay* is a *dynamic subsystem* that contains a series of stocks and flows. The number of stocks in that subsystem is the *order* of the delay. The main delays in StochSD are: `Delay1([Primitive], Delay Length, Initial Value)` and `Delay3([Primitive], Delay Length, Initial Value)`. See the Figure 10.4 where `Delay3([Pulse_In], 6, 0)` is used.

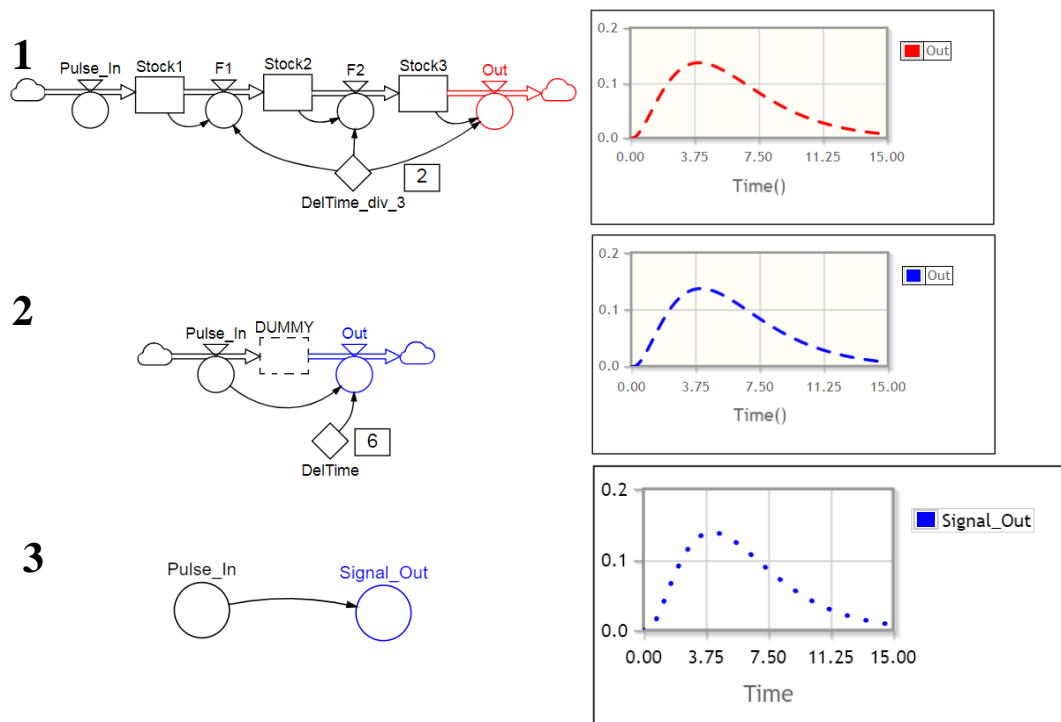


Figure 10.4. A delay of order 3 (with a total delay time of 6 time units) can be obtained in different ways:

1) By connecting three stocks in a series, each of the three stocks delays by 2 time units.

2) By using the `Delay3` function, which here connects ‘`Pulse_In`’ to the outflow ‘`Out`’:

`[Out]= Delay3([Pulse_In], [DelTime], 0)`. (A stock, here called `DUMMY`, because it is not involved in the delaying process, may or may not be included. It is merely placed between the

inflow and the outflow, so that it is filled by *Pulse_In* and drained by *Out*. *DUMMY* will then have the same content as $Stock1+Stock2+Stock3$.)

3) Also, signals can be delayed in the same way: $[Signal_Out]=Delay3([Pulse_In], 6, 0)$.

List of Historical functions

Delay – Static Delay function

Syntax: $Delay([Primitive], Delay\ Length, Default\ Value)$

Example: $Delay([Income], 6, 1000)$

Returns the value of a primitive for the specified length of time ago. Default Value stands in for the primitive's value in the case of negative times. (E.g. at time=0, this function looks back at time=-6, which in this example is defaulted to 1000. *Skipping to define a proper Default Value may cause problems that are hard to trace!*)

Delay1 – Delay function of order 1

Syntax: $Delay1([Primitive], Delay\ Length, Default\ Initial\ Value)$

Example: $Delay1([Inflow], 6, 0)$

Returns a smoothed, first-order exponential delay of the value of the primitive. The Initial Value is optional.

Delay3 – Delay function of order 3

Syntax: $Delay3([Primitive], Delay\ Length, Default\ Initial\ Value)$

Example: $Delay3([Inflow], 6, 0)$

Returns a smoothed, third-order exponential delay of the value of the primitive. The Initial Value is optional.

Smooth – Smoothing function

Syntax: $Smooth([Primitive], Length, Default\ Initial\ Value)$

Example: $Smooth([Inflow], 6, 0)$

Returns a smoothing of a primitive's past values. Results in an average curve fit. Length affects the weight of past values. The Initial Value is optional.

PastMax – Maximal value over a past period. – For statistics over all or a part of a simulation

Syntax: $PastMax([Primitive], Period = All\ Time)$

Results: Calculates the maximal value of a primitive over the specified past *Period*. If the *Period* is omitted, it shows the maximum sofar over the whole replication.

Example: `PastMax([X])` will create a function that displays the first value of X after one time-step, the largest value of X in the past two time-steps, and so on until the replication is over when the largest value of X is shown.

Example: `PastMax([X], 5)` will create a function that starts as in the example above, but after 5 time units or more displays the largest value during the last 5 time units.

PastMin – Minimal value over a past period.
– For statistics over all or a part of a simulation

Syntax: `PastMin([Primitive], Period = All Time)`

Results: Calculates the minimal value of a primitive over the specified *Period*.

PastMedian – Median value over a past period.
– For statistics over all or a part of a simulation

Syntax: `PastMedian([Primitive], Period = All Time)`

Results: Calculates the median value of a primitive over the specified *Period*.

PastMean – Mean value over a past period.
– For statistics over all or a part of a simulation

Syntax: `PastMean([Primitive], Period = All Time)`

Results: Calculates the mean value of a primitive over the specified *Period*.

PastStdev – Standar deviation over a past period.
– For statistics over all or a part of a simulation

Syntax: `PastStdev([Primitive], Period = All Time)`

Results: Calculates the standard deviation of a primitive over the specified *Period*.

PastCorrelation – Correlation between two quantities over a past period.
– For statistics over all or a part of a simulation

Syntax: `PastCorrelation([Primitive1], [Primitive2], Period = All Time)`

Results: Calculates the correlation between two primitives over the specified Period. The last argument is optional.

Fix – A device to sample and hold values for regular periods of time.

Syntax: Fix(Value, Period)

Results: A device to sample and hold values for regular periods of time. The last argument is optional. If last argument is missing or -1, then the first value is fixed.

Example: Fix(Rand(0,10), 5) will draw a random number between 0 and 10 that remains fixed for 5 time units at a time.

10.4 Random Number Functions

Here the most common random number functions are presented. In many simulation languages, random number functions include a *seed* that defines the initial value of the random number sequence. In StochSD the seed is set globally, which can be done by a macro. See Example 11.2.

Stocks, Flows, Auxiliaries and Parameters can all contain randomness. When this is the case, a dice will be shown in the primitive.

PoFlow - Poisson distributed flow

(This function is an addition to the Insight Maker functions.)

Syntax: PoFlow(FlowRate)

Result: Generates a Poisson distributed random flow with a mean of *FlowRate*. The result is always an integer ≥ 0 .

Example: PoFlow([c]*[X]) is a shorter, more practical and more readable form for RandPoisson(DT()*[c]*[X])/DT().

DT()*PoFlow() is used to produce a random number of events (e.g. arriving customers, people getting infected, cars passing a bridge) in a flow during a time-step.

Rand - Uniformly distributed random numbers

Syntax: Rand(Min, Max)

Result: Generates uniformly distributed random numbers between *Min* and *Max*. Rand() gives uniformly distributed random numbers between 0 and 1.

Example: Rand(5, 17) gives uniformly distributed random numbers between 5 and 17.

RandNormal - Normally distributed random numbers

Syntax: RandNormal(Mean, Standard Deviation)

Result: Generates normally distributed random numbers with a specified *Mean* and *Standard Deviation*.

Example: RandNormal(3, 0.5) gives normally distributed random numbers with a mean of 3 and a standard deviation of 0.5.

RandLogNormal - Log-normally distributed random numbers

Syntax: RandLognormal(Mean, Standard Deviation)

Result: Generates log-normally distributed random numbers with a specified *Mean* and *Standard Deviation*.

RandBernoulli - Binary distributed random numbers

Syntax: RandBernoulli(Probability)

Result: Generates exponentially distributed random numbers that returns *1* with the specified *Probability*, and *0* otherwise.

Example: RandBernoulli(0.5) can be used for the flipping of a symmetric coin.

RandBinomial - Binomially distributed random numbers

Syntax: RandBinomial(Count, Probability)

Result: Generates a binomially distributed random number, which is the number of successes from *Count* trials, each with the specified *Probability* of success. The result is always an integer ≥ 0 .

Example: RandBinomial(10, 0.3) gives a binomially distributed random number of successes from 10 trials, each with the expected value of 0.3.

RandNegativeBinomial - Negative Binomially distributed random numbers

Syntax: RandNegativeBinomial(Successes, Probability)

Result: Generates a negative binomially distributed random number. The number of random events, each with **Probability** of success required to generate the specified **Successes**.

RandPoisson - Poisson distributed random numbers

Syntax: RandPoisson(Lambda)

Result: Generates a Poisson distributed random number with a mean of *Lambda*. The result is always an integer ≥ 0 .

Example: RandPoisson(5) gives Poisson distributed random numbers with a mean of 5.

RandPoisson can be used to obtain the number of events during a time interval.

RandExp - Exponentially distributed random numbers

Syntax: RandExp(Lambda)

Result: Generates exponentially distributed random numbers with a *rate*: *Lambda*. The result is always an integer ≥ 0 .

Example: RandExp(5) gives exponentially distributed random numbers between 0 and *infinity* with a mean of $1/Lambda=1/5=0.2$. (If there arrive 5 customers per hour, then the distance in time between the customers is $1/5 = 0.2$ hours on average.)

RandExp can be used to obtain the distance in time between successive events when the rate of events is *Lambda*. Rate *Lambda* implies that the *expected distance* between events is $1/Lambda$.

RandBeta - Beta distributed random numbers

Syntax: RandBeta(Alpha, Beta)

Result: Generates Beta distributed random numbers.

RandGamma - Gamma distributed random numbers

Syntax: RandGamma(Alpha, Beta)

Result: Generates Gamma distributed random numbers.

RandTriangular - Triangularly distributed random numbers

Syntax: RandTriangular(Minimum, Maximum, Peak)

Result: Generates triangularly distributed random numbers between Minimum and Maximum with the largest value at Peak.

Example: RandTrangular(5, 12, 10) gives triangularly distributed random numbers between 5 and 12 with peak value at 10.

RandDist - Custom distributed random numbers

Syntax: `RandDist(X, Y)`; where X and Y are two vectors of x- and y-values.

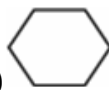
Result: Generates random numbers according to a custom distribution (an empirical distribution, usually obtained from real measurements). The customized distribution is specified by a sequence of x-values $\{x_1, x_2, \dots, x_n\}$ and a sequence of the same number of y-values $\{y_1, y_2, \dots, y_n\}$ where the sequences are separated by a comma. The Points between the defined coordinates are linearly interpolated. The distribution does not have to be normalized such that the area under the curve is 1, but the points must be sorted from the smallest to the largest value of x_i ; $i=1, 2, \dots, n$.

Example: `RandDist({0, 1, 2, 3, 4}, {0, 2, 1, 7, 3})`

10.5 Statistical Distributions

In 'Statistical Distributions' you find Pdfs, Cdfs and Inversions of *Normal*, *Lognormal*, *t*, *F*, *Chi square*, *Exponential* and *Poisson* distributions. These distributions can be used in some advanced statistical studies. However, these distributions will not be further discussed in this manual.

10.6 Converter (Table look-up function)



The **Converter** is a *Table look-up function* of the type $y = f(x)$. (Of course, you can use any names for x and y.)

The Converter is implemented as a primitive with its own icon. Click on the converter button, which gives you a six-sided figure. Double-click the *Converter* to *Define the data pairs x_i, y_i* . The data pairs must be separated by semicolon.

(Space between each pair can be inserted to make the table more readable.)

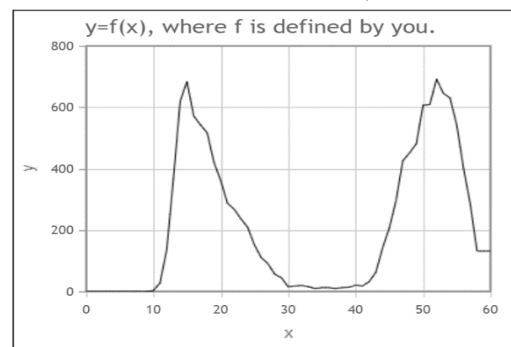
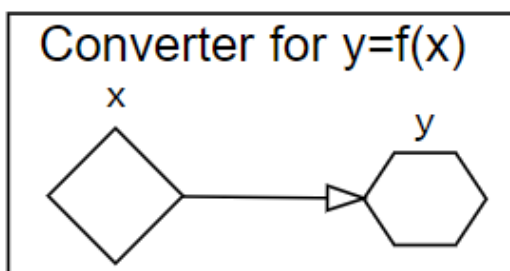
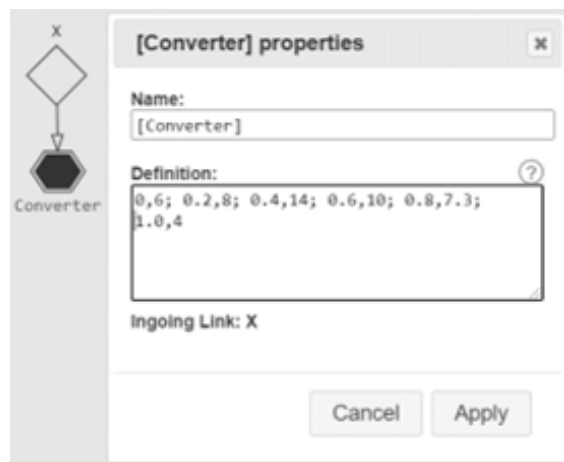


Figure 10.5. Weekly statistics of Covid19 in Sweden. x in weeks and y in deaths/week.

This means that you can use empirical data in your model, taken from statistics or measurements, which you can't describe by a function in the function library. For example, you may want to use the empirically known Temperature as function of Time, or Fertility as function of Food supply in your model expressed in tabular form.

The **input** to the Converter (x, time, temperature, amount of water, or whatever) can be taken from a Parameter, Auxiliary, Flow, Stock, or even from another Converter.

The **output** is the empirically defined $y=f(x)$. The function f is defined by coordinate pairs x,y separated by ';' (and for readability, you can also include a space), e.g.: 0,2.7; 0.5,3.4; ...; 9.5,18.3; 10,14.1. The x-values don't have to be equidistant, but the xy-pairs must be written in an ascending order of the x-values. The more precisely you want to describe the function, the more xy-pairs you need.

- For x-values *below* the first one, the output will keep the y-value of the first pair.
- For x-values *above* the last one, the output will keep the y-value of the last pair.
- For x-values *in between* the table values, interpolation is used.

Example: *Easy way to include statistics with another time unit than your model's.*
 In a StochSD study the number of deaths per day in Covid19 in Sweden during the first 406 days (58 weeks) was modelled. The model thus used the time unit: Day.

To fit the model, real statistics in deaths per **Week** was available and included in a Converter. Because the time unit for the model was **Day**, and the statistics (implemented in the Converter) was per **Week**, it was easiest to drive it by an Auxiliary $x=T()/7$, requiring 7 days to go to the next week. Also, the Converter output was cases per week – but since the model was defined for days, the Converter's output is rescaled to $y=f/7$. (Although, x was denoted Week, f was denoted MortStat, and y was denoted DeathsPerDay.) See Figures 10.6.

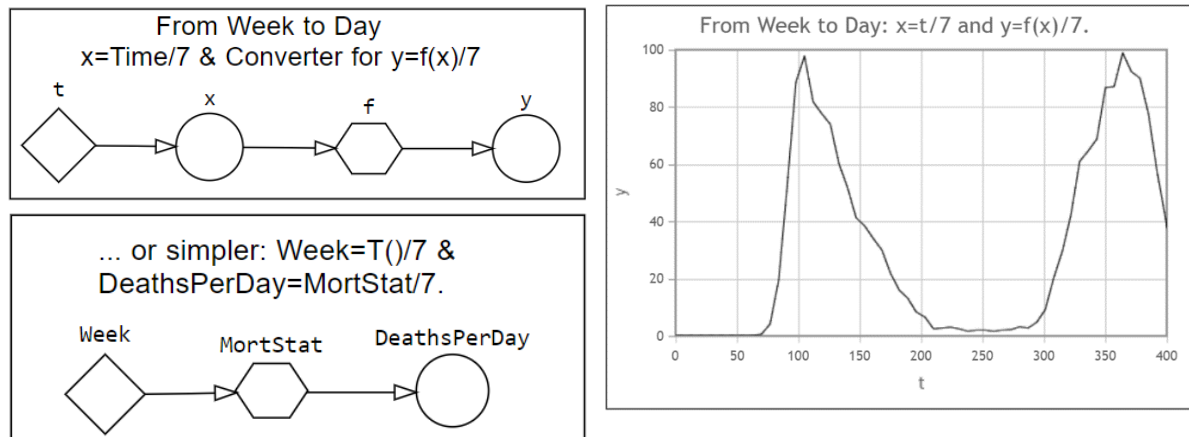


Figure 10.6. Daily statistics of Covid19 in Sweden using the weekly statistics from the Converter in Figure 10.5. ■

11. Macro Functions

Macro allows you to define your own functions that can be used in the model. Click the **Macros** menu to open a form to write the code.



Figure 11.1. In the Macro form, you can define a macro function of your choice. It can also be used to make a stochastic simulation reproducible by selecting a **Seed** (a number that initiate the random number generators) and click the button *SetRandSeed*.

Macros can be used in the definition of a primitive, just like a function. A macro can be created in two forms (Example from the Insight Maker manual).

A **single-line macro** has the syntax: *Myfcn* <- *functional expression*.

```
myFcn(a, b, c) <- sin((a+b+c)/(a*b*c))
```

Example 11.1 Defining a Poisson flow in a simplified way

In Section 10.45 we added `PoFlow(Lambda)` as a simpler to write function for `Poisson(Dt()*Lambda)/DT()`. If this function had not been available, you could have introduced it as a macro:

```
PoFlow(Lambda) <- RandPoisson(Dt()*Lambda)/DT().
```

Then you can use it in a flow as e.g.: `PoFlow(4)` or `PoFlow([X])` or `PoFlow([p]*[X])`.

■

Example 11.2 Defining an expression

A **multi-line macro** with parameters *a*, *b* and *c* has the syntax:

```
Function myFcn(a, b, c)
  x <- (a+b+c)
  y <- (a*b*c)
  return sin(x/y)
End Function
```

This can then be used in e.g. an Auxiliary as: myFcn(4, [X], [Y]). ■

Example 11.3 Making a stochastic simulation run reproducible

A macro can also be used to lock random number sequences so that each simulation of a stochastic model will have the same behaviour.

To make a stochastic simulation model *reproducible*, you have to lock the *seed* for the random number generators in the model. Then the same sequences of random numbers will be generated for each simulation run. Just click the **Macros button**, define the Seed and click the *SetRandSeed* button, or write e.g.:

```
SetRandSeed(13)
```

By changing the argument, you will get another (reproducible) simulation run.

To unlock the Seed you must manually delete the **SetRandSeed(Seed)** statement.

Note: You don't find **SetRandSeed(Seed)** in the Function library where you define a primitive, because this function does not belong to any primitive but is a *global* function of the model. Instead **SetRandSeed(Seed)** is defined in a macro by specifying the argument, **Seed**, of the **SetRandSeed** function.

Note: Setting the global parameter **Seed** will only make your model instance reproducible. If you build the same model with the primitives created in a different order, the same Seed would produce another (but for this new model reproducible) result.

The reproducible result may also be altered (but still reproducible) if you make changes in your model or changes the step size, DT. ■

12. Practical tips

Read these tips carefully. It will spare you a lot of trouble and make your modelling better.

StochSD-Desktop or StochSD-Web

If you do more than just test StochSD or want to use it quickly without an installation, try StochSD-Web. Otherwise download the StochSD-Desktop version so that you more easily can work with your files locally.

Adjust your screen

- You can zoom in and out with Ctrl+ and Ctrl- to obtain a proper size of your model.

Model building

- Perhaps the most common trouble in simulation originates from *confusion about the units used. In particular this applies to time.* Therefore, StochSD will force you to **define a one-and-only time unit** to be used in all equations of the actual model before you can start the model building. So chose the time unit with care (second, minute, day, year or even Martian fortnight) and then be consistent and use only this time unit!
- If you want to place many primitives of the same kind, e.g. Stocks, then you can right-click the Stock button and place a new Stock for each left-click. You leave this mode by a new right-click.
- You can fine-adjust the position of a primitive by high-lighting it and use the arrow keys.
- When you connect a Flow to a Stock, be sure that the connecting flow will be attached. The *Cloud* at the connected end will then disappear.
- *Give each primitive a proper name.* This will support your thinking and make it easier to communicate the model and its results to others.
- The arrow of a flow points in the *positive direction*. This means that when a flow has a positive value it goes in the direction of the arrow, and when its value is negative it goes in the opposite direction. For example, the flow direction of $[\text{Flow}] = \text{Sin}(T())$ will alternate over time.
- A Flow can only be drawn in x- and y-directions. However, you can break the flow in right angles by clicking the *right* mouse button while drawing the flow. This is particularly useful when you have several flows between stocks.
- A link is represented by a so-called Bézier-curve that can be bent into a smooth curve. You should use this of aesthetic reasons – for example, you don't want to cross over other symbols. To do so, click on the link and use its handles (■) or anchors (●) to form the link.
- A primitive will show a question mark '?' if there is no 'equation' in the definition field, or if all links are not used in the formula, or if unlinked primitives are used in the definition. It also keeps the question mark if there are unmatched brackets in the defining formula. However, the absence of a question mark does not guarantee that the formula is syntactically correct.

- Sometimes the displayed model will look messy because of links crossing the model because the primitives to be connected are far away. In such a case you can create a ‘*Ghost*’ of a primitives Stock, Auxiliary, Parameter, Converter (but not of a Flow) at a proper place and draw a link from this ghost instead. (Remember that a Ghost can only have *outgoing* links.) To use a ghost, *mark* the primitive you want to ghost before clicking the **Ghost button**.

Simulation Settings

StochSD is based on JavaScript and runs in a web browser. This browser administrates garbage collection (restoring used memory space) outside your control which requires time. The following point is therefore important:

- Don’t use smaller time-step (DT) than necessary (and of course not too large). The time of the execution depends mainly on the Length, DT and the size of the model. This dependency is about the square of $\text{ModelSize} \times \text{LENGTH} / \text{DT}$. Of this reason $\text{LENGTH} / \text{DT}$ gives a warning when $\geq 10\,000$ and is limited to 100 000 time-steps.
- The Compare Simulation Plot is a powerful device to study the effect of different step-lengths. Use this to find a proper value of DT.

Running the model

- If running the simulation ends with an Error Message and you don’t know where or why the error happened, then you can step (▶|) DT by DT while you display the values of the interesting primitives in a Table. (For a small DT it may be many steps!)
- For large models with many time-steps, the time after the calculations until Plots and Tables are drawn can be considerable (even larger than the calculation time that you can follow as a horizontal bar above the canvas) and even take minutes. Do not intervene with the model during this time.
- It may happen that StochSD gets stuck when you reopen the PC after having closed it or put it to rest. Clicking the StochSD icon at the bottom of the screen will then minimize StochSD and the next click on the icon will restore it.
- If the model becomes slow, open the **Help menu** and click **Restart and Keep Model**, and run the model again.

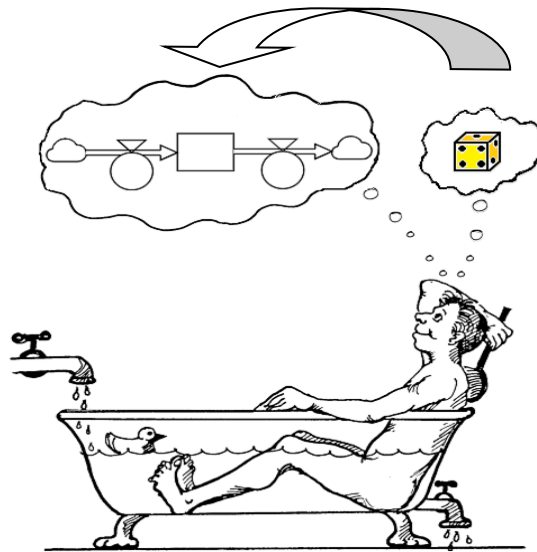
Result presentation

- Remember that the colour of a primitive will be used for the line of this primitive in a Time Plot, Compare Plot and area of Histogram when the ‘Colour from Primitive:’ box of the Plot or Histogram is checked.
- For documenting a smaller model, you may first select Print/Print Equations and place these equations at a proper place on the screen (perhaps after reducing the size by Ctrl-). Then select Print/Print Diagram so you can print both Diagram and Equations on the same sheet of paper.

13. References to deterministic CSS modelling

- [1] [Forrester, J.W. (1961) *Industrial Dynamics*. Cambridge, MIT Press, MA. (The book where System Dynamics was introduced.)
- [2] Meadows, D.H. (2008) *Thinking in systems: a primer*. White River Junction, Vt: Chelsea Green Pub. (An excellent first introduction to System Thinking and System Dynamics.)
- [3] Fortmann-Roe, Scott. (2014) Insight Maker: A General-purpose tool for web-based modeling & simulation. *Simulation Modelling Practice and Theory*, 47, 28-45. <http://www.sciencedirect.com/science/article/pii/S1569190X14000513>
- [4] The Manual for Insight Maker. (Here you find more features (e.g. functions) that are supported but not described in StochSD. Note however that only the System Dynamics part of Insight Maker is supported in StochSD.) <https://insightmaker.com/book/export/html/40>
- [5] Home page for Insight Maker: <https://insightmaker.com/>

Part II. Stochastic modelling with StochSD



Extended purpose: To build dynamic and *stochastic* models and simulate their behaviours.

Extended worldview: The world consists mainly of two kinds of fundamental primitives: **Stocks** and **Flows**. *However, lack of information requires stochasticities to be included!*

Stochastic modelling is an extension of deterministic modelling where you should handle the uncertainties in a statistically correct way. This extension has several important consequences:

Stochastic modelling makes Compartment Based Modelling (in e.g. StochSD) consistent with other types of simulation if you consider the following:

- Continuous matter should be described as a continuous and infinitely divisible amount, and discrete objects should usually⁶ be modelled as a number of indivisible entities.
- Uncertainties must be included in a statistically correct way, rather than using average estimates as in a deterministic model.
- A stochastic model will produce different results for each run, which reflects the underlying uncertainties of the model's behaviour. Therefore, many simulations of the model are required followed by a statistical analysis of the results. StochSD has the tool, **StatRes**, which does this.
- With a stochastic model you can estimate variations in the results, and also calculate confidence intervals and other statistics.
- Never use the RK4 method in a stochastic model.

⁶ Modelling is a pragmatic activity. If the number of objects, e.g. water molecules in a river, is very large then the water can be regarded as a continuous and 'infinitely' divisible matter. In particular, the law of large numbers then tells that a standard deviation is negligible compared to the average number (of water molecules) why transition stochasticity will disappear.

14. Stochastic modelling

14.1 A historic note and the motivation for StochSD

Simulation languages for digital computers were introduced from around 1960. There are two main forms **Discrete Event Simulation** (DES) that is based on a micro approach where each individual or entity is to be represented, and **Continuous System Simulation** (CSS) that is based on a macro approach (of which StochSD is an example). In *classical* CSS both continuous matter and individual objects were described as a continuous and infinitely divisible matter, which was lumped into compartments (stocks). Only the amount in each compartment is then represented.

However, when modelling *the same system under study*, a DES and a CSS model often produced inconsistent results (i.e. results that contradicted each other). During the remaining of the 20ies century this was an issue of concern and discussion.

In a series of papers between 2000 and 2017 the reasons for these inconsistencies were analysed and resolved (see references [6] to [12] in Section 17). In particular, the conditions for a *deterministic* CSS model to produce unbiased results were investigated [10]. In these papers, the importance of modelling continuous matter as continuous amounts and discrete objects as discrete numbers was demonstrated. Further, it was shown how to handle attributes, how to achieve adequate sojourn-time distributions in stages, how to handle different types of stochasticity, and how to model queues in CSS. It was also shown that following some basic rules for these issues in CSS modelling will make CSS models consistent with DES models. Furthermore, this approach opened the possible for combined discrete and continuous modelling and simulation within CSS.

It was also shown that different types of simulation (Agent-based, Entity-based, Compartment-based (i.e. CSS) and Situation-based (e.g. Markov models) could all produce mutually consistent results. Furthermore, it was demonstrated *how these model types are related and how they can be translated into each other*. Finally, the pros and cons of choosing one of these types instead of another were investigated.

In “The full potential of Continuous System Simulation modelling” [12], the rules for correct CSS modelling were presented and exemplified.

In short, **Full Potential CSS** modelling extends classical CSS by:

- Including discrete entities and giving the algorithm for transition between compartments,
- Preserving the sojourn time distribution of a stage by representing it with a structure of compartments in series or parallel,
- Correctly implementing the attributes of the entities by parallel but interconnected sub-models,
- Proper handling of the different types of uncertainties.

This knowledge can be applied using other CSS languages where proper random number generators are included. However, stochasticity will require multiple simulation runs followed by a statistical analysis and various forms of result presentation.

In order to apply Full Potential CSS, **StochSD** was developed.

StochSD contains several modelling tools, see [13] to [16]. In particular, the tool **StatRes** (see Section 16.1) will order multiple runs of a stochastic model, collect the results, analyse and present statistics from the multiple runs.

14.2 What is stochasticity?

Every modelling study must have a well-defined *purpose*. The art of modelling is based on *only* describing what is relevant for this purpose. However, in modelling you often have to deal with *incomplete information*.

Lack of information (*uncertainty*) can be handled in two ways:

1) You can ignore that you have incomplete information. For example, you can replace unknown variations by an average value, and hope that not too much damage is done, and that no one will notice. For example, if there is a dice involved, the possible outcomes 1, 2, 3, 4, 5 and 6 are always replaced by the average value (3.5). By including the statistical knowledge, you can make better models.

2) You can handle lack of information by at least *including the information you have*, often in form of a probability density (or distribution) function (pdf). Then random numbers can be drawn from this pdf to produce stochasticity in the model. For example, a dice is described by a probability distribution function that has equal probabilities for the outcomes 1, 2, 3, 4, 5 and 6. A random number is then used to decide the outcome for each throw. This is the correct way of handling incomplete information. Then the model includes the information you actually have!

14.3 Five types of uncertainty about the system under study

Five types of uncertainty about a *system under study* may have to be handled in a model building study. These are:

- *Structural uncertainty* – incomplete knowledge about the structure of the system under study and about how the components interact.
- *Initial value uncertainty* – incomplete information about the initial conditions at ‘time zero’.
- *Transition uncertainty* – incomplete information about when events will happen (e.g. customers will arrive, vehicles will pass a bridge, persons get sick).
- *Parameter uncertainty* – incomplete information about the impact from the Studied system’s *environment* and how this impact will vary over time (e.g. when and how much it will rain).
(*Environmental uncert.*)
- *Signal uncertainty* – incomplete knowledge about transferred information. (A signal can be delayed or distorted.)

Each of these uncertainties will affect the results of a model study. The more uncertainty about structure, initial conditions, transitions, parameter values and signals, the more uncertain the results will become.

14.4 Why including uncertainties in the model?

The short answer is that uncertainties contain more information than just averages, and this information can be important for the model's behaviour!!

The uncertainties must be incorporated in the model to produce realistic estimates of min values, max values, averages, standard deviations, confidence intervals, etc. A deterministic model will just ignore this information. Furthermore, the excluded information may in some cases cause biased results, or eliminate important phenomena that might occur in the studied system such as extinction of a species or that a weaker force defeats a stronger one.

Also, when you choose to use a deterministic model, you must be sure that the important results of a deterministic model agree with those of a stochastic one. But even then, you will lose all statistic information.

While a deterministic model only has to be run once since it always will produce the same behaviour, a stochastic model will produce a different behaviour for each replication (simulation run). Therefore, a stochastic model must be executed a large number of times so that the uncertainties of the model behaviour can be studied.

Multiple replications of the model will generate a probability density/distribution functions (pdf) for each outcome. From these pdfs, all kinds of statistics can be deduced, e.g. estimates of Min and Max values, Median values, Percentiles, Averages and Standard deviations, Confidence intervals, etc., and joint pdfs provide Correlation between outcomes. In StochSD this is swiftly obtained by the **StatRes** tool.

14.5 How to model uncertainties

A compartment-based model is constructed by *stocks, flows, auxiliaries & parameters*⁷, and *links*. There can be uncertainty related to each of these primitives, but also uncertainty about the structure of the model.

Now, we use an example to demonstrate how and where different types of stochasticity can enter into a compartment-based model.

Example 14.1 Uncertainties in a SIR model

A classical *SIR model* is an epidemic model composed of three stages: S (for Susceptible), I (for Infectious), and R (for Recovered). The following is assumed:

⁷ Auxiliaries and parameters are only distinguished in order to show if they mirror a part of the system under study or its environment – see footnote 2 in Section 2. From the *uncertainty* point of view, there is no distinction.

A Susceptible person has the probability p to become infected by an Infectious one at each time unit. Furthermore, an Infectious person will recover and become immune after on average T time units.

A classical SIR model, where each stage is represented by a single compartment, is shown as ‘*Structure 1*’ in Figure 14.1. This (oversimplified) model is used to discuss the five types of uncertainty mentioned above.

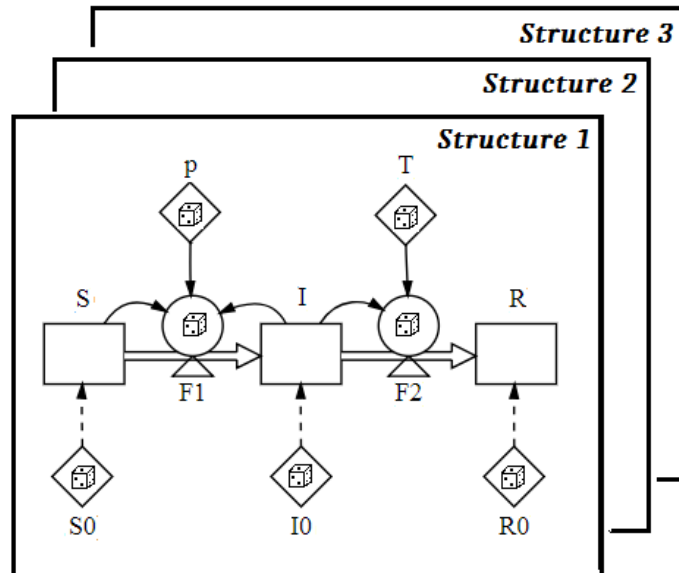


Figure 14.1. A classical SIR model where the stages S , I and R are represented by single compartments. The stocks are initiated to S_0 , I_0 and R_0 . The transition flows are denoted F_1 and F_2 , and the parameters are denoted p and T . Dice show where randomness to describe uncertainty can be located.

The five possible types of uncertainty, *Structural*, *Initial value*, *Transition*, *Parameter*, and *Signal uncertainties*, will all be recognized for this example. The *structural uncertainty* refers to the description of the system under study in terms of an appropriate *model structure* of stocks, flows, parameters and links, while the other four uncertainty types refer to the different primitives *within* the chosen model. *Initial uncertainty* relates to stocks, *Transition uncertainty* relates to flows, *Parameter uncertainty* relates to auxiliaries or parameters, and *Signal uncertainty* relates to (information) links.

1. Structural uncertainty

The structural description of the SIR model can be questioned.

One structural possibility is that an infected person does not immediately become infectious. The exposed person may require a latent period before he or she enters the Infectious stage. This can be accomplished by including an Exposed (E) stage between the Susceptible and the Infectious stages (giving a so-called SEIR model).

Another possibility is that the Recovered (immune) persons lose their immunity after some time, which means that they return ‘in a flow’ from Recovered to the Susceptible stage (a so-called SIRS model).

A third possibility refers to the time in the Infectious stage (the so-called *sojourn time*), which on average is T time units. But some persons will recover sooner and others will require a longer time to recover. By modelling the Infectious stage by a

single compartment, we have implicitly assumed an exponential sojourn-time distribution. (Other assumptions about this distribution can be realized by using several compartments in parallel and/or series.)

Structural uncertainty is handled by exploring alternative models to find the one behaving most similar to the studied system.

2. Initial value uncertainty (*Stock*)

The information about the number of persons in the three compartments at time zero may be approximate. Say that we know that the population in a village is $N=1000$ persons. Three persons from this village returns from a trip abroad bringing home a new disease, but we don't know whether 1, 2 or all 3 of them were infectious. The *initial value* for the stock $I(0)$ is then 1, 2 or 3 according to some probability distribution.

Example:

In a SIR model the initial number of Infectious (I) at time zero is unknown. In 60% it is $I(0)=1$, in 30% it is $I(0)=2$ and in 10% it is $I(0)=3$ infectious persons. Then you can draw a uniformly distributed random number between 0 and 1 and use this distribution to initialize the I-stage at time zero. (See If-Then-Else in Section 10.4.)

```
If Rand() < 0.6 Then
  1
Else If Rand() < (0.6+0.3) Then
  2
Else
  3
End If □
```

Note: Because initial stochasticity only affects the model at time zero, it will not be influenced by the choice of the time-step DT.

3. Transition uncertainty (*Flow of discrete entities*)

Stochastic CSS can handle both *continuous matter* and *discrete objects*! It is crucial to model continuous matter as continuous amounts and discrete objects as discrete numbers (unless the number of objects is very large, see Section 14.1).

In the discrete case, a stock will hold an integer number of indivisible entities that can be transferred by a flow. The number of entities transferred per time unit must then also be integer. However, there is almost never enough information to decide the exact event times for the transitions of entities into, out from or between stocks. Transition uncertainties must then be included in the description of the flows, using available statistical information.

Technically, this is handled by a Poisson distributed random number function. Transition uncertainty is often the most fundamental and important uncertainty to include in a model with discrete entities. It is easy and straightforward to implement and stochasticity may drastically change the nature of the model's behaviour (described in Section 14.7).

In the continuous case, any fraction of the amount in a stock can enter or leave by a flow during a time unit. However, no transition stochasticity is here included because

of the ‘infinite divisibility’ according to the law of large numbers. (Of course, a continuous flow may still vary stochastically when it is affected by parameter or signal stochasticity.)

In a *deterministic* SIR model, with the same structure as that in Figure 14.1, the flows F1 and F2 would have the forms: $F1 := p \cdot S \cdot I$ and $F2 := I/T$. To understand how F1 and F2 change when transition stochasticity is implemented, we multiply both hand sides by DT.

<u>Deterministic model</u>		<u>Stochastic model</u>
$DT \cdot F1 := DT \cdot p \cdot S \cdot I$	→	$DT \cdot F1 := \text{Poisson}(DT \cdot p \cdot S \cdot I)$
$DT \cdot F2 := DT \cdot I/T$	→	$DT \cdot F2 := \text{Poisson}(DT \cdot I/T)$

However, $DT \cdot F1$ and $DT \cdot F2$ is not allowed to the left of an assignment sign ($:=$), why we have to divide both sides with DT. Applying the syntax of StochSD then gives:

$$\begin{aligned}
 [F1] &:= \text{RandPoisson}(DT() \cdot [p] \cdot [S] \cdot [I]) / DT() & (1) \\
 [F2] &:= \text{RandPoisson}(DT() \cdot [I]/[T]) / DT() & (2)
 \end{aligned}$$

$DT \cdot F$ is the *expected* number of transferred entities during a time-step DT . The Poisson Random Number Generator will always produce an integer number of transitions that on average amounts to $DT \cdot p \cdot S \cdot I$ or $DT \cdot I/T$. Thus, if you start with integer numbers in the stocks, and the transitions add or subtract integer numbers, then the stocks will remain integer.

To simplify (1) and (2), a new function PoFlow has been introduced (see Section 10.5). This function also illuminates the close relationship between the deterministic and stochastic cases – just include the deterministic case as an argument in PoFlow():

<u>Deterministic model</u>		<u>Stochastic model</u>
$[F1] := [p] \cdot [S] \cdot [I]$	→	$[F1] := \text{PoFlow}([p] \cdot [S] \cdot [I])$ (1')
$[F2] := [I]/[T]$	→	$[F2] := \text{PoFlow}([I]/[T])$ (2')

The Poisson distribution has a unique property that makes it DT-independent

This can be formulated as: $\text{Po}(DT \cdot \lambda) \leftrightarrow \text{Po}(\frac{1}{2} \cdot DT \cdot \lambda) + \text{Po}(\frac{1}{2} \cdot DT \cdot \lambda)$ in the meaning that the outcomes of $\text{Po}(DT \cdot \lambda)$ is indistinguishable from the outcomes of $\text{Po}(\frac{1}{2} \cdot DT \cdot \lambda) + \text{Po}(\frac{1}{2} \cdot DT \cdot \lambda)$. This means that it is allowed to change the time-step DT without destroying the validity of transition stochasticity!

A WARNING: In a *deterministic* case, you may *add* or *subtract* flow equations to and from a stock:

Say that $Stock := DT \cdot F1 - DT \cdot F2$. Then you may write:

$$Stock := DT \cdot (F1 - F2) = DT \cdot \text{NetFlow}.$$

However, this is not valid for the corresponding *stochastic* case: Thus:

$$Stock := \text{PoFlow}(DT \cdot F1) - \text{PoFlow}(DT \cdot F2) \quad (3)$$

MAY NOT be written as:

$$Stock := \text{PoFlow}(DT \cdot F1 - DT \cdot F2) \quad (4)$$

To see why this is wrong, assume that $DT \cdot F1=2$ and $DT \cdot F2=k$ (so that $DT \cdot F1 - DT \cdot F2 = 0$). Then the random numbers from the two terms in (3) may sometimes generate a larger input than output to Stock and sometimes the opposite, while (4) will always give $Poisson(2-2) = Poisson(0)$ **which always is zero**. Then the Stock will remain constant over time.

4. Parameter uncertainty (*Parameter & Auxiliary*)

Impact on the studied system from unexplained, irregularly varying phenomena in the environment can be modelled by stochastic parameters described in statistical terms to generate realistic inputs. In this case there is no given form to describe such uncertainties. Only statistical knowledge about the variations can guide you here. In the SIR model in Figure 14.1, the infectious parameter, p , and the sojourn time as infectious, T , may vary with e.g. behaviour and weather in an unpredictable way.

A problem with parameter stochasticity is that its influence will change with the size of the time-step DT . If you reduce DT , more random numbers will be drawn per time unit. The effect of this is smoothing because of the law of large numbers.

One way to handle this problem is to pulse the changes of the parameter (using Pulse(Start, Volume, Repeat) in Section 10.1). This allows the time-step, DT , to be less or equal to the time interval (Repeat) between pulses, without affecting the parameter stochasticity if DT is changed.

5. Signal uncertainty (*Belongs to a Link, Implemented in connected primitives*)

Before we discuss signal uncertainty, we have to clarify that an information link in a model can be of two different kinds:

- a. *Artificial link* – a technical concept to communicate logic between artificially separated model parts. For example, when a radioactive atom decays, there is one radioactive atom less in the system under study. In a compartment-based model this is accomplished by a construction of a stock *and* an outflow. An *artificial link* from the stock to the valve regulating the outflow has here to be included. Its only function is to transfer information about the content (value) of the stock to the flow equation. There is no counterpart to such a link in the real system under study, and *no uncertainty or delay can be involved*.
- b. *Signal link* – a description of a real information link that communicates information in the system under study. This communicated information we call ‘*signal*’. A signal can be *distorted* and/or *delayed*.

To exemplify this, we include an Authority in Figure 14.1 as shown in Figure 14.2. Here we assume that this Authority collects information about the number of Infectious persons (which takes time and is never exact). Then the Authority may issue recommendations about using sanitary measures and avoiding contacts, which later on, when the message is received, may affect the infection rate, $F1$.

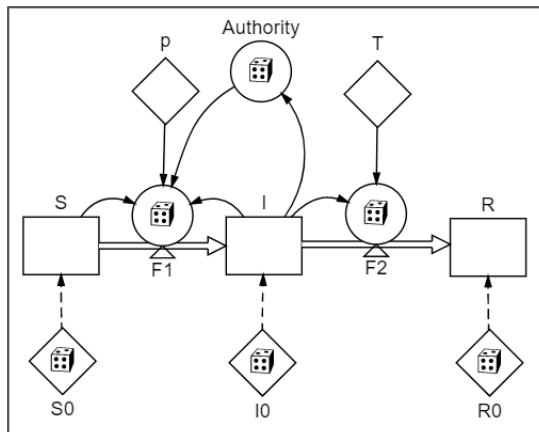


Figure 14.2. The SIR model supplemented with an Authority that collects information of the number of Infectious (I) and issues recommendations that later on will affect F1.

Signals require time to reach the receiver that may vary in an unpredictable way. (The information may be sent by post, it may arrive during the weekend but noticed first on the following Monday, etc.), and the information may be distorted because of various reasons (imperfect information, typing error, distortion of the signal, misinterpretation, not reaching all subjects, etc.).

The distortion of the number of Infectious may be modelled by a stochastic distribution function, and the length of a delay can be generated by drawing a random number from another proper distribution. In this way, uncertainty about signals can be handled in the sending or receiving primitive (usually Auxiliary, Parameter or Flow).

Initial value uncertainty, Transition uncertainty, Parameter uncertainty and Signal uncertainty are all treated in a similar way. First you describe the uncertainty by a statistical distribution. Then random numbers are drawn from this distribution.

14.6 What can happen if you ignore stochasticity?

There is a widespread misconception that you should ignore information you don't know exactly and e.g. replace it with an average value, ending up with a deterministic model. This is a stupid, dishonest and dangerous attitude that often leads to biased results, exclusion of phenomena, erroneous conclusions and no insight in the precision of the estimates.

A dynamic *and stochastic* model should be based on all relevant information – even though this information may not be complete!

When a population system of discrete objects is studied by a *deterministic* model, various types of distortions may occur. For example:

- Information about *irregular variations, risks, extremes* and *correlations* are eliminated.
- The possibility to calculate *confidence intervals*, etc. is lost.
- The elimination of transition stochasticity may result in *biased* estimates.

- Important *phenomena*, such as extinction of a population or *elimination of an epidemic*, which can happen in the system at study and is reflected in stochastic modelling, will be lost.
- A deterministic model produces *categorical answers*, e.g. that a force X will win over a force Y , whereas a stochastic model will give the probabilities of X and Y as winners.
- *Oscillations* might disappear in a deterministic model although they should be excited by transition stochasticity.
- In a deterministic model, the lack of transition stochasticity may erroneously *prevent queues* from building up.
- *The time* until a specific event occurs should vary between replications in a stochastic population model, but will be the same and often erroneous in a deterministic one. For example, in a deterministic model the time to all radioactivity is decayed will erroneously become infinitely long, even when it should be finite and short.

In short, using a deterministic model will also mean that we only are aiming for average estimates – losing all insight in variations and extreme behaviours. Unless this average will equal the average value obtained by a corresponding discrete and stochastic model, the use of a deterministic model will produce biased results and prevent interesting phenomena to be displayed.

14.7 A warning example – Comparing the results from a deterministic and a stochastic SIR model

Assume the following for the classical SIR model, presented in Example 14.1 in Section 14.4 above. The initial population $N=S+I+R$ consists of 1000 susceptible persons, a single infectious person and no recovered persons. The infection risk parameter is $p=0.0003$ per person and time unit and the sojourn time in the infectious stage is $T=4$ time units.

10 000 replications of the (transition-) stochastic SIR model to study the cumulated number of susceptible individuals being infected, give an ensemble of results with the average value of 53.1 persons that got the disease [95% confidence interval: 50.8 – 55.5 persons]. However, the corresponding deterministic model produces 318.5 persons getting the disease. In this case, an error of about 600 percent was introduced by dropping the transition stochasticity.

The reason why the deterministic model distorts the results is because the initial infectious subpopulation $I(0)=I$ is small. There is then a large chance that the single infectious person will recover before infecting another susceptible, so that no epidemic will occur.

14.8 Making a stochastic model reproducible

Making a stochastic model reproducible may seem to be a contradiction. However, this is a legitimate technique that has many advantages in various situations. The background is that stochasticity is generated by Random Number Generators (RNGs), where each RNG produces a sequence of random numbers from a specified statistical distribution. An RNG generates a very long sequence of random numbers that have

virtually all the statistical properties of the specified statistical distribution. When the sequence finally ends (which will not happen in your practice) it starts over and repeats the cycle. When you define a *Seed* you tell where in the cycle to start, which gives you the opportunity to exactly repeat a simulation. In StochSD the Seed is a ‘*super-seed*’ that controls all the seeds for the different RNGs, which makes it easier for you. (However, the possibility to assign a separate seed to every RNG is more powerful.)

Reproducibility can be used in the following cases:

- To demonstrate or reconstruct a simulation run because something of particular interest happened, e.g. a species died out, or a water dam was destroyed because of overflow of water and you want to understand why.
- To compare two similar models under *similar conditions*. For example, comparing different designs of a queuing system with several stations. The different designs should ideally be compared under the same stochastic conditions. If we can provide the same sequences of arriving customers for the two cases, then we would eliminate one source of variations in the results. This technique is called *variance reduction* using *common random numbers*. Unfortunately, to use variance reduction (except for very simple cases) requires the possibility to assign separate seeds to each RNG, which is not possible in StochSD

To make a stochastic simulation model *reproducible*, you have to lock the seed for the random number generators in the model. Just click the **Macros** menu, define a Seed and click the *SetRandSeed* button, or write e.g.: **SetRandSeed(17)** in the definition field of the macro. By changing the argument, you will get another (reproducible) simulation run. See Example 11.2.

To unlock the Seed you must manually delete the **SetRandSeed(Seed)** statement.

15. A stochastic StochSD model

Example 15.1 Radioactive decay

A specimen, containing $N=60$ radioactive atoms that decay with a time constant of $T=10$ minutes, is studied during 50 minutes.

A model of this can be described by the equations:

$$N(t+DT) = N(t) - DT \cdot F(t) \quad (\text{where } N(0) = 60)$$

$$F(t) = \text{Po}[DT \cdot N(t)/T]/DT \quad (\text{In a deterministic model: } F(t) = N(t)/T.)$$

Po[.] stands for a Poisson distributed random sample.

A StochSD model and a replication are shown in 15.1.

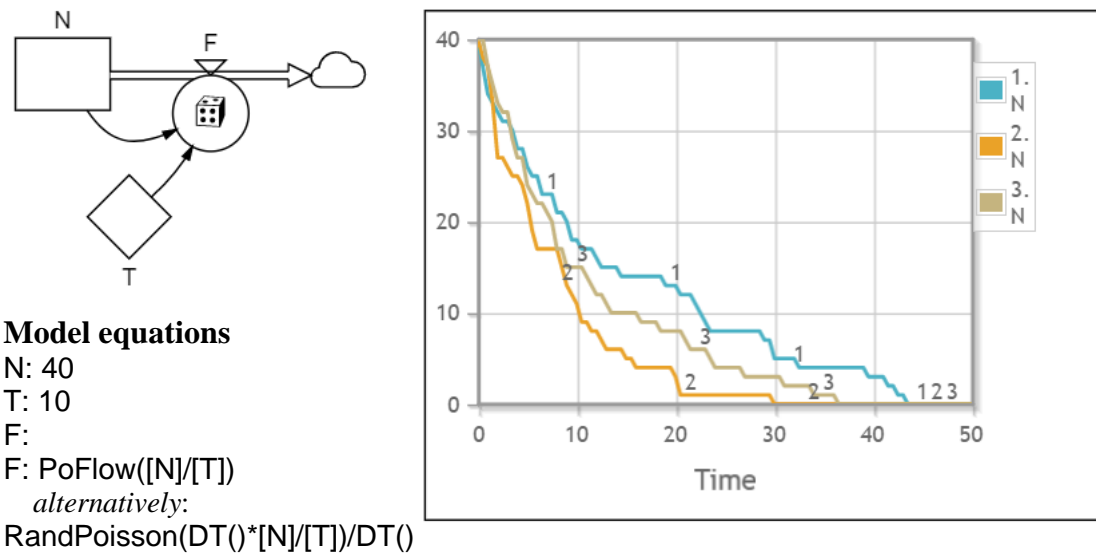


Figure 15.1. A stochastic model of radioactive decay and three possible outcomes.

Note that the definition of the Flow [F] as 'RandPoisson(DT()*[N]/[T]) / DT()' can be simplified by using the function PoFlow() as: 'PoFlow([N]/[T])'.

Also, note that in this stochastic model, there might be no radioactive atoms left after half an hour, while a corresponding deterministic model will never reach zero radioactive atoms. ■

Of the same reason a stochastic SIR model can reach zero infectious persons and the epidemic is over (unless more infectious persons are imported). However, a corresponding deterministic model only asymptotically approaches zero.

Further, in a stochastic ecological model a species can get extinct, while in a deterministic model it will recover from any fraction of an animal.

16. Tools in StochSD

StochSD contains four tools: **StatRes**, **Optim**, **ParmVar** and **Sensi** (for Statistical analysis and result presentation, Optimisation, Parameter estimation variations, and Sensitivity analysis). These tools are found in the **Tools** menu. The selected tool will show up in a separate window to the right on the screen, and it works on the model displayed in the Model window. Each of these four tools also has its own documentation. Here we only indicate what they are used for.

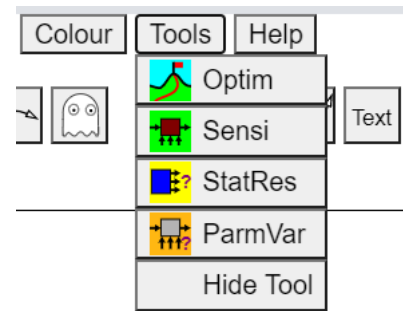


Figure 16.1. The Tools menu.

16.1 The StatRes tool

The price to pay for using a stochastic simulation model, is that many replications are required, which must be followed by a statistical analysis and presentation of the results.

StatRes (**Statistical Results**) orders StochRes to run a model a specified number of times. Then a statistical analysis of specified quantities (Stocks, Flows, Auxiliaries or Parameters) is performed. The *Average*, *Standard deviation*, *Confidence interval*, *Min*, *Max*, and *Percentiles* for the performed number of replications are presented for each specified quantity. You can also present the quantities from the replications in a tabular form, or as a histogram. Further you can plot two outcomes and also obtain an estimate of the *correlation* between the two quantities. For more information, see the StatRes manual [13].

16.2 The Optim tool

Optim (**Optimisation**) is a simplex optimizer for finding the optimum (maximum or minimum) of an objective function defined in a *deterministic* model with respect to a number of model parameters.

An optimizer has two main purposes in deterministic modelling.

First, an optimizer can be used for *parameter estimation*. Then you construct a function that measures *the difference* between the system and model behaviours. This function is then *minimized* with respect to a set of parameters. The set of parameter estimates obtained are those which produce the model behaviour that best reproduces the behaviour of the system under study. Parameter estimation is often a necessary part of the construction of a model.

Second, an optimizer can be used *to find the optimal strategy* for a model, e.g. minimize costs or maximize benefits. To do this, an *objective function* is defined in the model. This function depends on some model parameters. When the objective function and the parameters are specified, the optimizer will systematically vary the values of the specified parameters to find the optimal set of parameter values.

Optimisation is easy to perform on deterministic model – and much trickier on a stochastic one.

The simplex optimizer is stable and reasonable fast and has the advantage that you easy can include boundaries in the parameter space. For more information, see the Optim manual [14].

16.3 The ParmVar tool

ParmVar (**Par**ameter **Var**iations) is a tool for assessing the accuracy of previously estimated parameters. After parameter estimation (with e.g. **Optim**) an important question remains: How *accurately* are the set of parameters values estimated? This question can be addressed with ParmVar.

Parameter estimation is tricky to perform on a *stochastic* model. It requires a large number of replications for each tested set of parameter values (as compared to a single replication in the deterministic case). However, for two important classes of models the expected outcome from a stochastic model is the same as the outcome from a corresponding deterministic one. Then you can find the optimal set of parameter values with **Optim**.

This is a more advanced tool that usually will not be used in an introductory course. For more information, see the ParmVar manual [15].

16.4 The Sensi tool

Sensi (**Sens**itivity Analysis) is a tool for sensitivity analysis. With this tool you can study the effect on one or several outcomes from changing a parameter or initial value. For example, you may study how the price of a commodity will affect the revenue of a company. For more information, see the Sensi manual [16].

16.5 Hiding the Tool

The **Hide** option hides a displayed tool and lets the Model window occupy the whole width of the screen.

16.6 Clearing a tool

After using a toll, the content is protected. If you want to clear the tool, you first have to use the **Reset**-button in the actual tool. This is also important when you swap to a new model and want to use the tool again. The tool must then be reset and the variables specified for the old model must then be deleted.

17. References to stochastic CSS modelling

- [6] Gustafsson, L. (2000) Poisson Simulation – A Method for Generating Stochastic Variations in Continuous System Simulation. *Simulation*, 74, 264-274.
<https://www.researchgate.net/publication/220164720>
- [7] Gustafsson, L. (2003) Poisson Simulation as an Extension of Continuous System Simulation for the Modeling of Queuing Systems. *Simulation*, 79, 528-541.
<http://dx.doi.org/10.1177/003759703040234>
- [8] Gustafsson, L. and Sternad, M. (2007) Bringing Consistency to Simulation of Population Models-Poisson Simulation as a Bridge between Micro and Macro Simulation. *Mathematical Biosciences*, 209, 361-385.
<http://dx.doi.org/10.1016/j.mbs.2007.02.004>
- [9] Gustafsson, L. and Sternad, M. (2010) Consistent Micro, Macro and State-Based Modelling. *Mathematical Biosciences*, 225, 94-107.
<http://dx.doi.org/10.1016/j.mbs.2010.02.003>
- [10] Gustafsson, L. and Sternad, M. (2013) When Can a Deterministic Model of a Population System Reveal What Will Happen on Average? *Mathematical Biosciences*, 243, 28-45. <http://dx.doi.org/10.1016/j.mbs.2013.01.006>
- [11] Gustafsson, L. and Sternad, M. (2016), A guide to population modelling for simulation. *OJMSi*, 4, 55-92. http://file.scirp.org/pdf/OJMSi_2016042717425486.pdf
- [12] Gustafsson, L., Sternad, M. and Gustafsson E. (2017), The full potential of Continuous System Simulation modelling, *OJMSi*, 5, 253-299.
<http://www.scirp.org/JOURNAL/PaperInformation.aspx?PaperID=80104>
- [13] Gustafsson, L. (2018) StatRes – A tool for statistical analysis of stochastic StochSD models. https://stochsd.sourceforge.io/manuals/StochSD_StatRes.pdf
- [14] Gustafsson, L. (2018) Optim – An optimiser for deterministic StochSD models. https://stochsd.sourceforge.io/manuals/StochSD_Optim.pdf
- [15] Gustafsson, L. (2018) ParmVar – A tool for studying the variation of parameter estimates in StochSD models.
https://stochsd.sourceforge.io/manuals/StochSD_ParmVar.pdf
- [16] Gustafsson, L. (2018) Sensi – A sensitivity analyser for StochSD models.
https://stochsd.sourceforge.io/manuals/StochSD_Sensi.pdf

Appendix. The StochSD package, Licenses and Responsibility

A1. The structure of the StochSD package

StochSD is constructed from open-source software only. Figure A.1 shows the overall structure of the StochSD package.

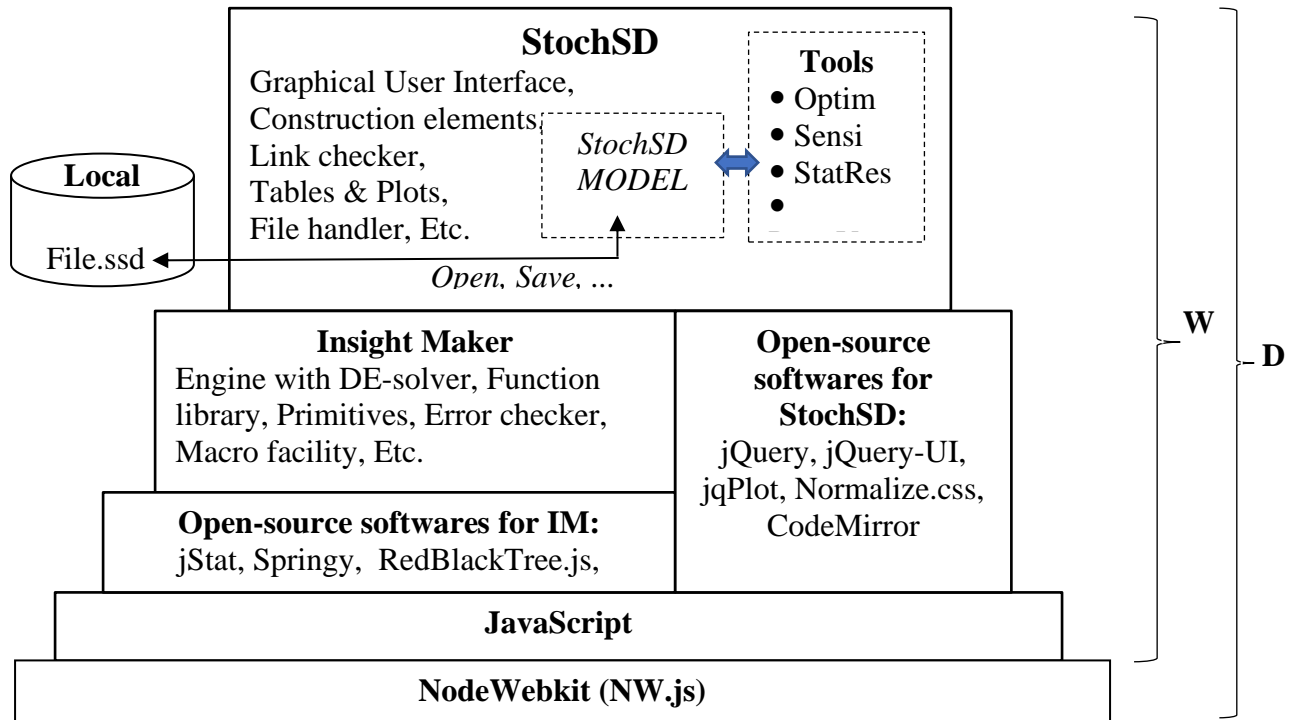


Figure A.1. The components of StochSD and its use of third-party software.

The Tools always connects to the actual model in the Model Window.

D: StochSD-Desktop includes NW.js.

W: StochSD-Web runs within a browser.

A2. Short explanation of licenses for StochSD and third-party components

StochSD is a stochastic CSS language based on the System Dynamics philosophy.

StochSD is released under Affero General Public License v3.

<https://www.gnu.org/licenses/agpl-3.0.html>

However, StochSD uses many open-source third-party components. The most notable one is Insight Maker, which is licensed under its own *Insight Maker Public License* that in turn is an extension of Affero GPL. All non-open source components of the original Insight Maker are removed in StochSD.

The third-party components used in StochSD are listed below.

Third-party components

- **Insight Maker (IM):** Primitives, Simulation engine, Function library, Macro function facility, DE solver, Error checker ,etc.
Insight Maker Public License thus covers all StochSD's JavaScript, HTML and CSS (Cascading Style Sheets) code for StochSD. (In the StochSD package the original non-open source code in Insight Maker, such as ExtJS and mxGraph are completely eliminated and replaced by jQuery-UI and jqPlot.)
Insight Maker Public License: <https://insightmaker.com/impl>.
- **jqPlot:** Diagram drawing tool.
MIT License: <https://github.com/jqPlot/jqPlot#legal-notice>
- **jQuery:** Library to simplify HTML, DOM tree transversal and manipulation, and event handling.
MIT License: <https://github.com/jquery/jquery/blob/master/LICENSE.txt>
- **jQuery-UI:** User Interface components. Handles pop-up windows.
MIT License: <https://github.com/jquery/jquery-ui/blob/master/LICENSE.txt>
- **jStat:** Statistical library.
MIT License: <https://github.com/jstat/jstat/blob/1.x/LICENSE>
- **Springy:** A force directed graph layout algorithm. (Used in IM.)
MIT License: <https://github.com/dhotson/springy/blob/master/LICENSE>
- **RedBlackTree.js:** Organizes a self-balancing binary search tree. (Used in IM.)
BSD 3 Clause License: <http://www.kevlindev.com/license.txt>
- **Normalize.css:** Making the User Interface look more similar across browsers.
MIT License: <https://www.npmjs.com/package/normalize.css/v/3.0.2>
- **NW.js:** JavaScript Desktop wrapper. (Runs JavaScript without a browser.)
MIT License: <https://github.com/nwjs/nw.js/blob/nw23/LICENSE>
- **Electron:** Alternative JavaScript Desktop wrapper. (Included but not used.)
MIT License: <https://github.com/electron/electron/blob/master/LICENSE>
- **ANTLR:** A parser generator. Interprets text into formulas.
BSD 3 Clause License: <https://www.antlr.org/license.html>
- **CodeMirror:** Text editor for editing code. Used for styling and brackets matching.
MIT License: <https://codemirror.net/LICENSE>

The complete licences are also found under **Help/Third-party Software** in StochSD.

A3. Responsibility

The user is fully responsible for the use of StochSD. The producer and the supplier of this code take *no* responsibility for the use or functioning of StochSD and its tools. See the AGPL license: <https://www.gnu.org/licenses/agpl-3.0.html>

The StochSD responsibility text in accordance to AGPL is cited here:

“15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. “

Index

Abs function.....	45	Histogram	34
ArcCos function	45	Historical Functions	46
ArcSin function.....	44	IfThenElse function	40
ArcTan function	45	If-Then-Else function / Structured....	40
Arithmetic operators	24	Initial value	22, 24, 29
Auxiliary	10, 19	Initial value uncertainty	61, 64
Bathtub analogy	10	Insight Maker.....	5
Building blocks	19	Installation of StochSD.....	7
Buttons	16	Integration method.....	26
Ceiling function (Round Up)	43	Length.....	26
Classical CSS	5	Licenses and Responsibility	74, 75
Comment.....	23	Link.....	10, 20
Compare Simulations Plot	32	Ln function.....	45
Continuous System Simulation.....	5	Log function.....	45
Converter.....	11, 52	Macro approach	60
Converter – Table look-up function..	52	Macro Functions	54
Cos function	44	Mathematical Functions.....	42
CSS	5	Max function.....	41
CSV.....	31	Menus	13
Delay function.....	47	Micro approach.....	60
Delay function of order 1	47	Min function	41
Delay function of order 3	47	Model building.....	19
Delays.....	46	Modulus function.....	45
Deterministic modelling.....	9	Number Box.....	30
Dialog box.....	21	Optim tool.....	71
DT – step-size	42	Parameter	10, 19
Environment.....	11, 12	Parameter uncertainty	61, 66
Equation List.....	15	ParmVar tool.....	72
Equations.....	24	PastCorrelation	48
Error checking.....	24	PastMax	47
Euler’s method.....	24, 26	PastMean.....	48
Exp function.....	45	PastMedian	48
eps (Smallest number).....	45	PastMin	48
Fix	49	PastStdev.....	48
Floor function (RoundDown)	43	PoFlow – Poisson flow rand. Func. ..	49
Flow	10, 19	Primitives.....	19
Full Potential CSS.....	5, 16, 60, 61, 73	Programming Fuctions.....	40
Function (defined by you).....	41	Pulse function	43
Functions.....	38	Ramp function	44
Ghost.....	19, 20	Rand - Uniform random function	49

RandBernoulli – 0 or 1 rand. Func....	49	Step function.....	44
RandBeta –Beta random Func.	51	Stochastic modelling.....	60
RandBinomial – Binomial rand func	49	Stochasticity.....	61
RandDist – Empirical random func. .	52	StochSD	5
RandExp – expo. rand. Func.....	51	StochSD Desktop.....	7, 74
RandGamma – Gamma rand. Func...51		StochSD Home Page.....	6
RandNormal - Normal rand. Func. ...	50	StochSD Web.....	7, 74
Random Number Functions	49	Stock	10, 19
RandPoisson – Poisson rand. Func. ..	51	StopIf function	41
RandTriangular - Triang. rand distr.	51	Structural uncertainty.....	61, 63
References.....	58 73	System boundaries	11
Restrict to non-negative values ...	21, 22	System dynamics	5
Results presentation	30	System Under Study	11, 12
RK4 method.....	24		
Round Down function.....	43	T – Current time.....	42
Round function.....	43	Table	31
Round Up function.....	43	Tan function.....	44
		TE – end time of simulation	43
SD	5	Throwing Error function.....	41
Seed.....	54, 55	Time Functions	42
Sensi tool.....	72	Time Plot	31
SetRandSeed	54, 55	Time Step.....	15, 26, 42
Signal uncertainty	61, 66	Time Unit.....	13, 18, 19, 56
Sign function.....	45	TL – length of simulation	42
Sin function.....	44	Tools in StochSD.....	71
Smooth function.....	47	Transition uncertainty	61, 64
Specification of the simulation	26	TS – Start time for simulation	42
Sqrt function.....	45	TSV.....	31
Start Time.....	26, 42		
Static Delay function.....	41	Uncertainty	61
Statistical Distributions.....	52		
StatRes tool.....	71	XY Plot.....	33