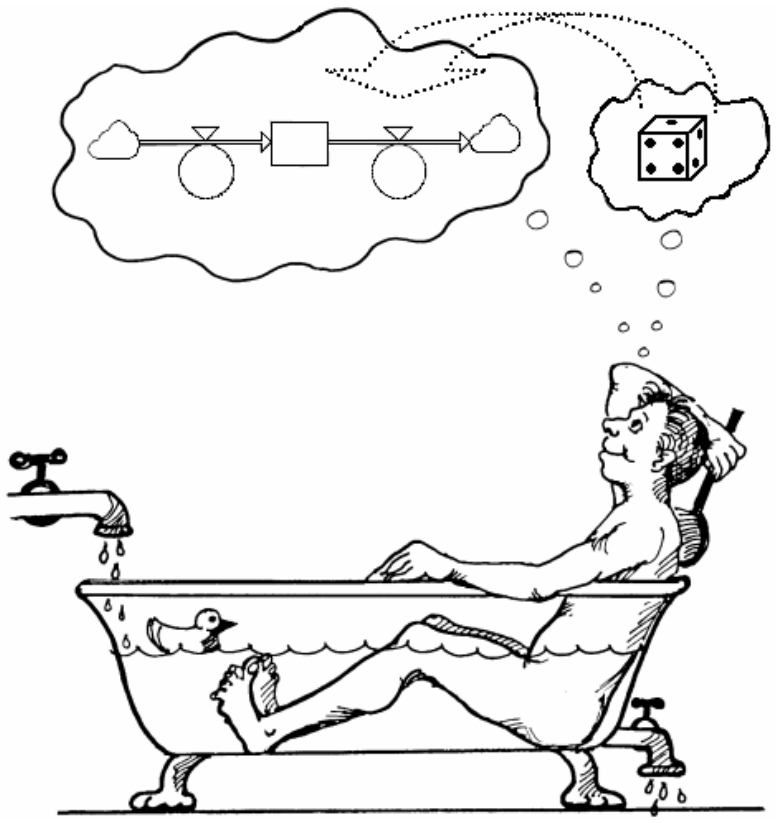
StochSD User's Manual



Part I. Deterministic modelling and simulation with StochSD

Part II. Stochastic modelling and simulation with StochSD

File: StochSD_User's_Manual.doc 170725

Home Page: https://stochsd.sourceforge.io/homepage/

Contents

Part I. Deterministic modelling with StochSD

Part II. Stochastic modelling with StochSD

Part I. Deterministic modelling with StochSD

1. Introduction

StochSD (Stochastic System Dynamics) is an open-source tool for deterministic and stochastic modelling and simulation based on Insight Maker[1] version 5. That StochSD is open-source means that you get it and use it for free. It even gives you the right to use and modify the StochSD source code under open-source licence. StochSD is written in JavaScript, see Appendix B.

StochSD is developed to build, simulate and analyse *deterministic* and *stochastic* modelling and simulation according to the System Dynamics approach. StochSD 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 time continuous progress in a great number of areas, for example biology, economics, physics or ecology. A model is constructed by placing and connecting predefined graphical symbols at suitable places on the screen. StochSD then transforms the graphical model into an executable code.

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

The Desktop version is downloaded from StochSD's homepage and runs as a normal program while the web version requires a web-browser such as Google Chrome or Mozilla Firefox to load StochSD and run your models. StochSD does not support Internet Explorer.

For installation of StochSD Desktop and running of StochSD Web see Appendix A.

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

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

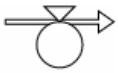
StochSD Home Page: https://stochsd.sourceforge.io/homepage.

---

[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 modelling window and buttons etc.) have been replaced in order to make StochSD completely open. 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.)

2. The fundamental building blocks of a StochSD model

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



Flow is the active element that generates changes. 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. (Synonyms: **Transition**, **Rate**). The new Flow should be assigned a proper name after double-clicking the symbol. Note that the arrow shows the *positive direction* of a flow. If the flow rate is negative, it flows at the opposite direction.

The **dynamic behaviour** of a model is created by flows accumulated in and/or drained from Stocks! *Flows into Stocks make the world go around!*

Often a ***bathtub analogy*** is used as a metaphor. The Stock is a *bathtub* where e.g. water is added by an Inflow and drained by an Outflow.

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

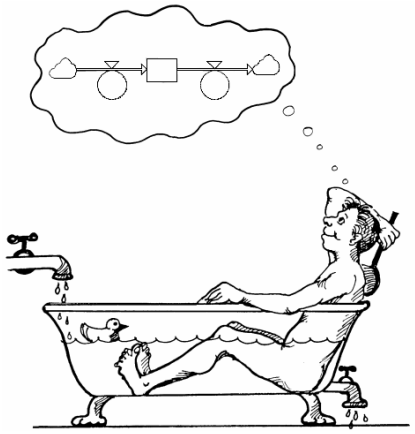


Figure 2.1. The bathtub analogy.

Information link: Influence from one element on another. (Synonymous: **Link**, **Information, Signal**.)

For practical reasons, we add two more graphical symbols.

Variable contains calculations based on the values of other elements. It can also be a constant that is fixed over time. (Synonym: **Auxiliary**). The variable should be assigned a proper name after double-clicking the symbol.

Converter is a table look-up function where you can implement any (in particular empirically described) function between two variables. See 11.2.6.

3. Starting up StochSD

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

Open StochSD. The StochSD screen is then shown. On the upper part of the screen you see the main menus and some buttons, and below is the model window (which is empty at start).

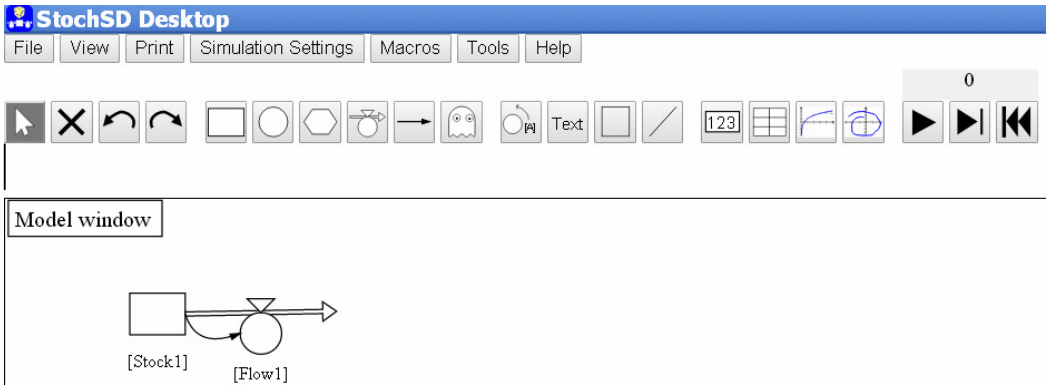


Figure 3.1. The main menu of StochSD located above the modelling window with a few primitives.

The main menu of StochSD consists of pull-down menus and a row of buttons for constructing and handling the model.

3.1 MENUS

In the pull-down menus you find: **File**, **View**, **Print, Simulation Settings**, **Macro**, **Tools** and **Help**.

The File menu

The file handling in SochSD is handled 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 run locally, you can choose the folder to save the model in.
When StochSD runs on a server the files are always stored in the *Downloads* folder.

The File menu contains the usual facilities:

- New
- Open
- Save
- Save as …

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

A model can be opened by clicking the **Open** option 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. Her 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 the *Download* library.

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

The View menu

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

- Zoom in (Ctrl +)
- Zoom out (Ctrl -)
- Reset zoom (Ctrl 0)

The Print menu

Printing model diagram or 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.
The equations can be listed and then printed on paper.

Simulation Settings menu

In the Simulation Settings dialog you specify the *Simulation start*¸ *Simulation* length, and *Simulation time- step* ('DT()'):

- Start
- Stop
- Step

Before you execute the model, you should specify what time period to study and how to handle it. The **Simulation Settings** menu opens a form for specifying the *Start*¸ *Length*, and *Time-Step* for the simulation.

Macros menu

Enables the introduction of macros to StochSD, see Section 11.

Tools menu
To the right of the modelling window, you find tools for *Optimisation* and *Sensitivity analysis, Statistical calculations and presentation* and for *Parameter estimation.* Pressing a tool option will make the tool appear in a window on the right-hand side of the screen. With the Hide option you can close the tool.

The Tools menu has the following options:
- Optim
- Sensi
- StatRes
- ParmVar
- Hide

The tools will be treated in Part II.

Help menu
The Help meny contains:
- About
- Manual
- Debug log
- Restart and clear model
- Restart and keep model

About briefly describes the current version of StochSD. A User's manual is also available. The 'Debug log' is only valuable for a developer of the software. Finally, if StochSD is malfunctioning, you can restart it with or without the current model loaded.

3.2. BUTTONS

The buttons are from left to right:

Manipulation buttons
These buttons are for recapturing the 'unloaded mouse', deleting a block, or undoing or redoing an operation.
- Mouse
- Delete
- Undo
- Redo

Building blocks (Primitives)
Here are the building blocks of a StochSD model.
- Stock
- Variable
- Converter (for a table look-up function)
- Flow

- Link
- Ghost

Style blocks

The style blocks provides blocks for styling the model by moving the names of the building blocks and for inserting frames, texts and lines.

- Moves label around the primitive
- Text
- Frame
- Line

Result blocks

These blocks provide Number boxes, Tables, Time Series and Scatter Plots, for presenting results.

- Number box
- Table
- Time diagram
- Scatter Plot (XY-plot)

Execution buttons

The execution buttons controls the simulation.

- Run/Halt
- Step
- Stop

4. Model building

The main primitives for building a model are: *Stock, Variable*, *Converter* (a table look-up function where you can enter any connection between two variables like *x* and *y* in a tabular form), *Flow* and *Link*.

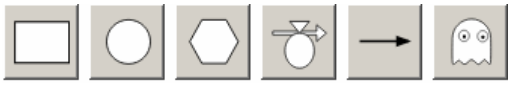


Figure 4.1. The StochSD primitives (building blocks): *Stock, Variable*, *Converter, Flow* and *Link*. A ghost symbol to duplicate the three first mentioned primitives is also available here.

The name of a the symbols: Stock, Variable, Converter and Flow are always starting with '[' and ending with ']'. For example [Stock1], [Variable3], [Flow2]. The default names should be renamed to be descriptive, e.g. [Rabbits], [Fertility], [Births per month].

The *Ghost Primitive* means that the same Stock, Variable or Converter may be represented on two or several places in the model. This can eliminate connecting links to cross over the model, making the model look ugly and unstructured. The Ghost is graphically represented as a copy of the original primitive with a ghost symbol inserted.

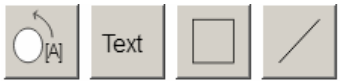


Figure 4.2. The Style elements: Move label, Text, Frame and Line.

Finally, the Style buttons contain features for replacing the name of a building block and for including texts, frames and lines into the model.

4.1 Placing the building and styling blocks at the model window

The model is built in a click-and-play manner. Use the mouse to select a symbol (Stock, Variable, Converter, Flow, Link or Ghost) from the **Primitives** by clicking its button. Position the mouse and click again.

A **Flow** has a start and an end point that may or may not connect to a Stock. When you click the Flow on the screen the endpoint is positioned. Then, with the mouse button still down, you draw the mouse to where you want the end point. If the start and/or end point is over a Stock, it will be connected to this Stock. An unconnected start point means that the flow comes from outside the defined model (i.e. from the models environment), and an unconnected end point means that the flow leaves the model.

A **Link** must always connect two primitives. It means that one primitive affects another.

Continue in this way until the structure of your model is finished and then give all the elements appropriate names.

The primitives on the screen can then be adjusted by drawing them to new positions. The default names of the primitives are: [Stock1], [Variable1], [Converter1] and [Flow1] for the first of its kind. A Link has no label and the Ghost gets the same name as the symbol it ghosts. All names can be rotated around its symbol with the Rotate button (the first of the Style buttons).

Also the other **Style blocks**: Text, Frame and Line can be moved to a new place. However, for these blocks there is no label and the size of these blocks can be adjusted.

4.2 Naming and defining relations or values

When the structure of primitives is ready, it is time to give the primitives appropriate names and to exactly specify relations and values.

Note, if a formula that includes another primitive is first defined and you then change the name of that other primitive, the formula will not function since it refers to a non-existing name. You must then also update the formula.

To open the form (dialog box) of a primitive, double-click the primitive.

For the Stock, Variable and Flow primitives the Dialog boxes are similar.

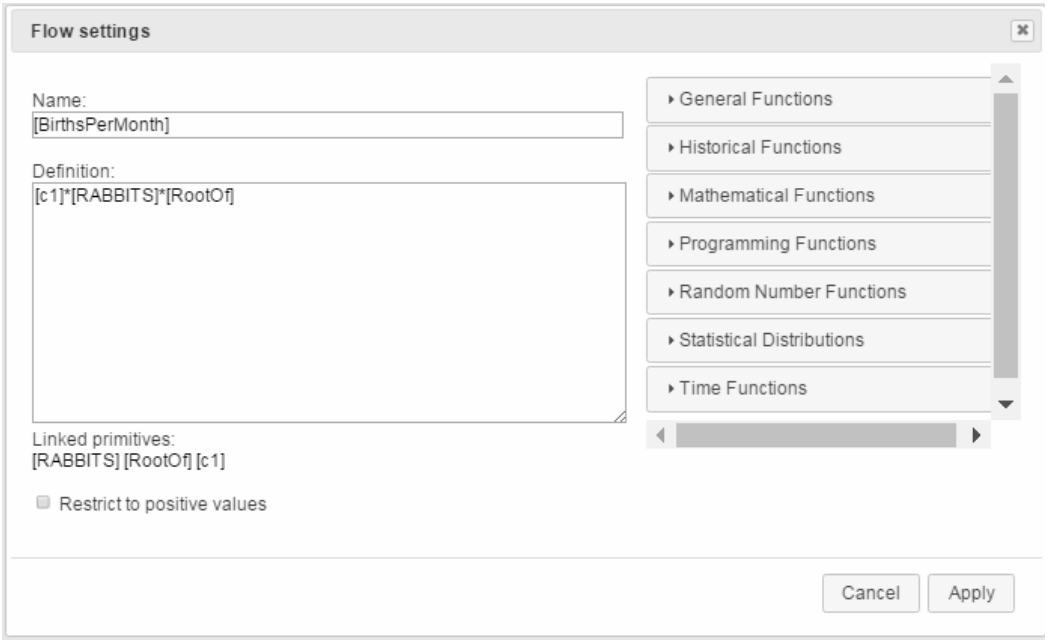


Figure 4.3. Dialog box of a Flow with: 1) Name field where the name can be changed, 2) Value field where the value or function is specified. 3) Linked primitives show the primitives that should be included in the Value field. 4) library functions which also could be used in the Value field. 5) Check box for Restriction to positive values. 6) Cancel or Apply buttons.

The **Dialog box** contains a **Name** field where the name (e.g. [Stock3], [Variable5], [Converter1] or [Flow2]) can be changed to a suitable name in upper or lower case characters.

Below is the **Definition** field of the primitive where you enter a formula or a value. Here, linked primitives and library functions can be used together with numbers and the arithmetic operators + - * / ^ and ( ).

Further below, **Linked primitives** to be used in the Definition are listed. They enter the Definition field when you click them.

For Stock and Flow primitives a check box for '**Restriction to positive values**' is found at the bottom-left. (This should *very seldom* be used).

To the right, a library of functions subdivided into categories named: *General Functions, Historical Functions*, *Mathematical Functions*, *Programming Functions*, *Random Number Functions*, *Statistical Distributions* and *Time Functions*. Clicking on such a group button will give you a list of functions in the category. These will be treated in Section 10.

StochSD knows every inflow and outflow to a stock from the graphical structure without further specification. Therefore, only the initial value at '*time zero*' (i.e. the time when the simulation starts, which may be 12

o'clock or January 1, 2017) is required in the definition field. Often you just define the initial value for the stock as a number. But it is also possible to initiate the stock to the value of a Variable linked to the stock. The initial value may also be a random number.

A Ghost has *the same* (not just a copy) dialog box as the primitive it ghosts.

The **Converter** is a table look-up function where you specify the pairs: x_1,y_1; x_2,y_2, …, x_n,y_n of values. It has its own dialog box.

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

Leave the dialog box by clicking the **Apply** button to realise your specifications or by clicking the **Cancel** button to leave the dialog box without the new specifications.

5. Equations

When you specify all relations and values related to the primitives, StochSD automatically generates the equations that mathematically describe the model.

The model equations can be viewed by opening the **Print** menu and choose **Print Equations**. (From there you may also print the equations.)

6. Specification of the simulation

Before you execute the model, you should *specify* what time period to study and how accurately to handle it. The **Simulation Settings** menu opens a form for specifying the *Start¸ Length*, and *Time-Step* for the simulation.

The *integration* (updating of the model over time) is performed by Euler's algorithm. This algorithm updates a stock X over time according to: X(next) = X(now) + Inflows(during now to next) – Outflows(during now to next). Mathematically this is expressed as: X(t+dt) = X(t) + dt*InFlowRate – dt*OutFlowRate.

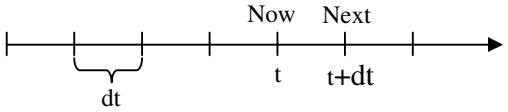


Figure 6.1. The time-handling in CSS.

For 'well-behaved' deterministic models there are more efficient algorithms than Euler's, but for stochastic models the gain from these algorithms are small or none.

Note that a short time-step increases the accuracy of the calculations but also increases the simulation time. You should therefore try to find a compromise where the time step is sufficiently short to give good accuracy (the results will then change very marginally), but not shorter than this.

7. Simulation

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

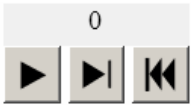


Figure 7.1. The execution buttons: Run/Halt, Step and Reset.

Pressing the **Run** button (►) starts the simulation and enables specified *Simulation Results* in Number boxes, Tables, Time diagrams or Scatter plots.

The calculations behind the scene of a simulation
What happens during a simulation is that StochSD, time-step by time-step, solves the equation system. The working procedure used is the following (DT is the simulation time-step used and StopTime is the StartTime + Simulation Length):

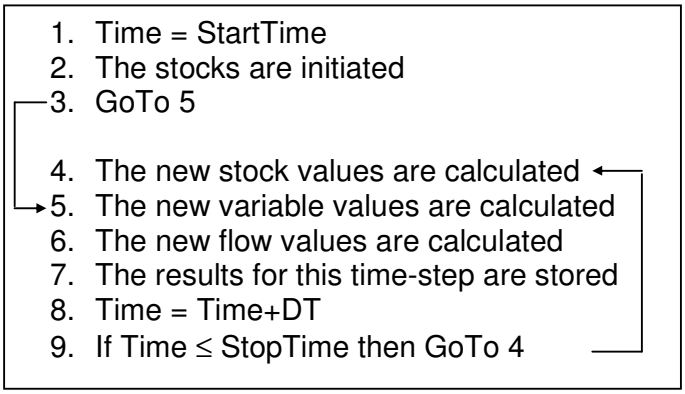


Figure 7.2. The calculations performed behind the scene.

The calculations of the stock variables are performed according to Euler's integration method.

8. Configuration and presentation of results

To see the results of a simulation run you can use **Number boxes**, **Tables**, **Time diagrams** and **Scatter plots.**

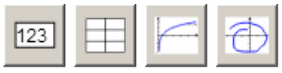


Figure 8.1. Result presentation buttons for **Number box**, **Table**, **Time diagram** and **Scatter plot**.

The result facilities are obtained by clicking a result button and then clicking at a proper place on the screen. Tables, and diagrams can then be drawn to a proper size by using the handles.

By double-clicking the Table, Time diagram or scatter 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 graphs are automatically scaled after a simulation.

For Tables you can also choose to print only a part of the data and the time between the printings.

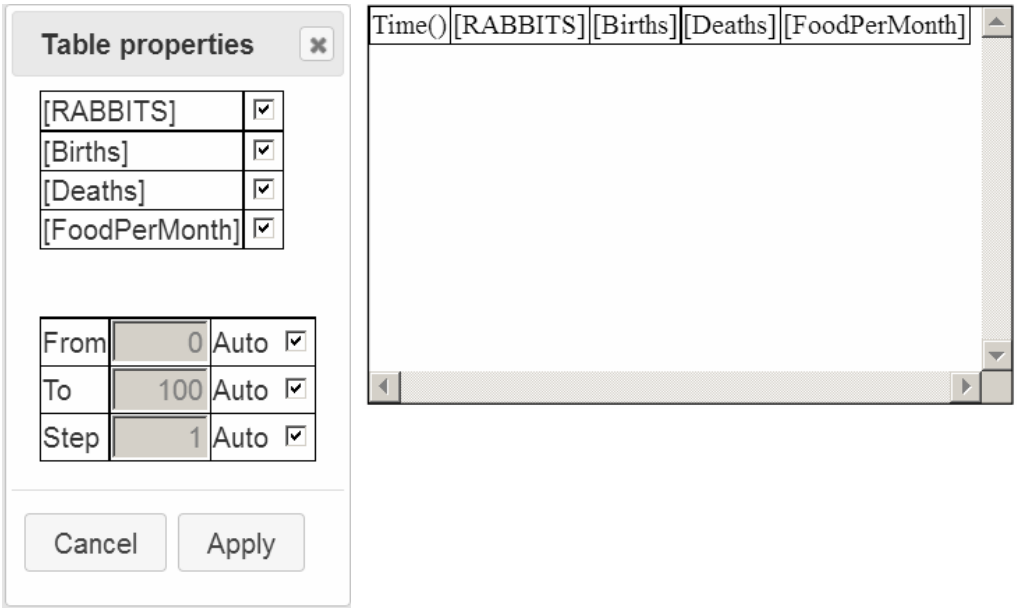


Figure 8.2. Dialog box for a Table, and the resulting Table *after* selecting all primitives to be displayed.

If you only want to display e.g. the final value of a Stock, Variable or Flow, a Number box is handy to use. It is also practical to include the Number box to display the value of an important constant in the model in this way.

9. 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.

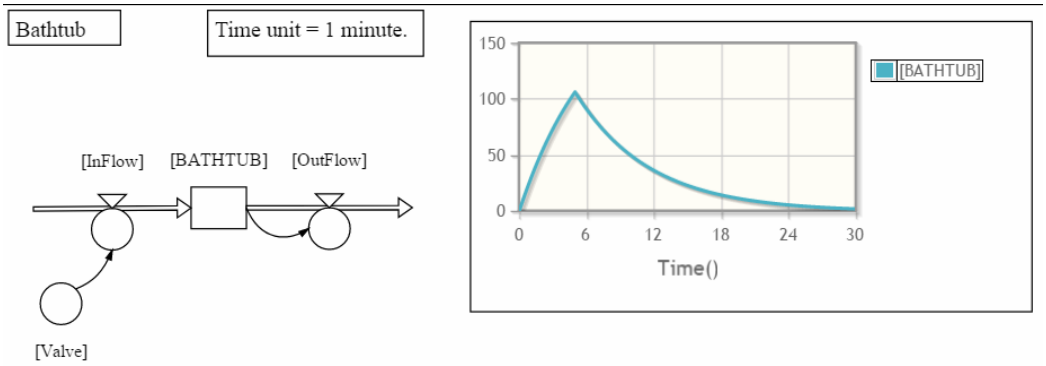


Figure 9.1. A simple bathtub model and its behaviour.

We here assume the following:
1) The BATHTUB is empty at Start Time.
2) The InFlow is regulated from the Valve.
3) The Valve handle is opened to give 0.5 litres/second for *5* minutes and then closes.
4) The OutFlow is open. For simplicities sake, we assume that 15% of the amount of water in the BATHTUB will leave the tub each minute through the OutFlow.

The equations generated from the model are:

```
[BATHTUB] = [BATHTUB]+DT()*([InFlow] – [OutFlow])
[BATHTUB] = 0                                    (* Initial value *)
[Valve] = IfThenElse(T()<5, 30, 0)               (* See Section 10.2.1 *)
[InFlow] = [Valve]
[OutFlow] = 0.15*[BATHTUB]
```

The model is repeatedly updated time-step by time-step until the simulation is complete.

Using *the time unit = 1 minute*, the time-step (DT) is set to *0.1* minute, so the equation system becomes updated for each *6* seconds.

You can choose the **time unit** to be e.g. *1 second*, *1 minute* or *1 hour*. But you have to be *consistent* and *recalculate* all time data to the selected one!

The content of the bathtub is shown for half an hour (30 minutes) in a Time Diagram in Figure 9.1 above. ■

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.

Figure 9.2. Picture of the system under study.

Studies of the studied system have shown that the food supply available is constant and is estimated to *100* kg per month. It has also been shown that 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 constant has been calculated to *0.2*. Further, it has been found that the average length of life is *20* months, which is about the same as *5%* of the population dying every month.

A so-called causal-loop diagram for the model is shown in the figure below.

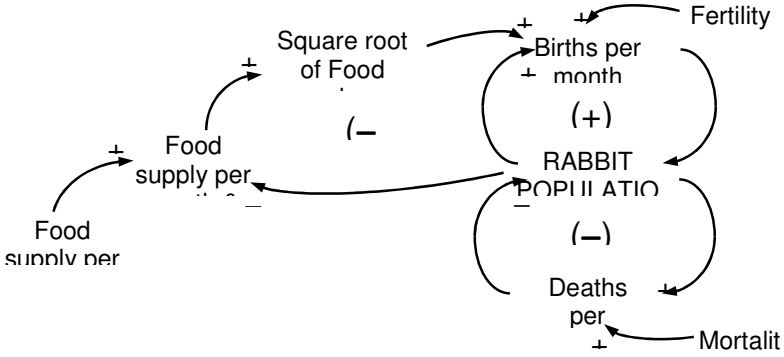


Figure 9.3. A causal-loop diagram of the rabbit system.

Building the model structure and filling in all relations and parameters (the fertility and mortality constants are c1 = *0.2* and c2 = *0.05*, respectively) gives the following StochSD model:

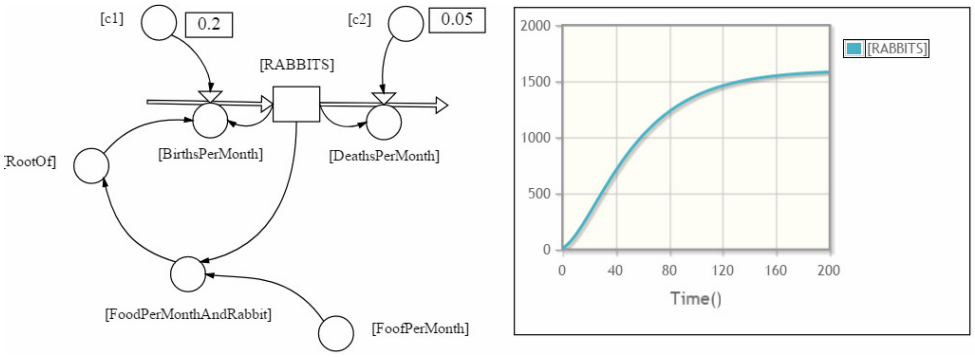


Figure 9.4. A StochSD diagram of the rabbit system and the result.

Select the **Simulation Settings** menu and set the *Simulation Length* to *200*, chose *Months* as *Time unit* and set the *Simulation Time-Step* to *0.1* (months).

When the model is complete, and the **Run** button is pressed, we choose the **Time Diagram,** here only showing RABBITS, see Figure 9.4, above. We could also choose **Table** and e.g. select *RABBITS*, *BirthsPerMonths*, *DeathsPerMonths*, and *FoodPerMonth*. ■

10. Functions

In StochSD there are a large number of library functions. You find them to the left of the Dialog box of a **Stock**, **Flow**, or **Variable**. The functions are first listed as an overview. A more detailed presentation is given below.

The arithmetic operators are: + − ∗ / ^ ( ) for addition, subtraction, multiplication, division, power (x^n) and parentheses. As always, the calculation order is: power, then comes multiplication and division, and last addition and subtraction. Parentheses '( )' can be used to alter this order.

10.1 The categories of functions in StochSD

Below, we will shortly present the most commonly used functions sorted into different categories.

Only the most useful functions will be discussed. Sorted by category, these are:

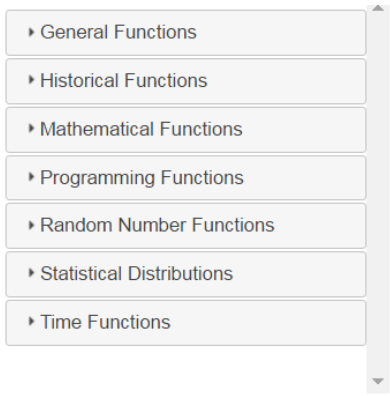


Figure 10.1. The categories of functions.

General Functions
- If Then Else
- PulseFcn
- Step
- Ramp
- Stop (Terminates the simulation run)

Historical Functions
- Delay1 (Dynamic delay of order 1)
- Delay3 (Dynamic delay of order 3)
- PastMax
- PastMin
- PastMean
- PastStdev
- PastCorrelation
- Fix (A device to sample and hold values for regular periods of time)

Mathematical Functions
- Abs (Absolute number)
- Sqrt (Square root)
- Sin (Argument in radians)
- Cos (Argument in radians)
- Tan (Argument in radians)
- Log (Logarithm base 10)
- Ln (Logarithm base e)
- Exp (Exponential)

Programming Functions

- If-Then-Else (More advanced version than that in *General Functions*.)

Random Number Functions

- PoFlow (Simplified expression for Poisson flows)
- Rand (Uniform distribution)
- RandNormal
- RandBinomial
- RandPoisson
- RandExp
- RandBoolean
- RandDist (Custom distribution)

Statistical distributions

■ 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.

Time Functions

- Current Time
- Time Start
- Time Step
- Time Length
- Time End

Converter

- The often-useful empirical graph function (sometimes called table-look-up function) where you can enter an x-value to obtain the y-value is named *Converter*, and is a **Primitive** with its own button. However, since it is a function it is discussed (last) in this chapter.

10.2 Detailed description of the functions

10.2.1 General Functions

IfThenElse – Simple 'If Then Else' function

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

Result: If the logical *Condition* is True then the first *Expression* (*Value*) is evaluated and returned, if it is False then the second *Expression* is evaluated and 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 $A \neq B$ and False if A = B.

PulseFcn – Pulse function

Syntax: PulseFcn(Start, 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.

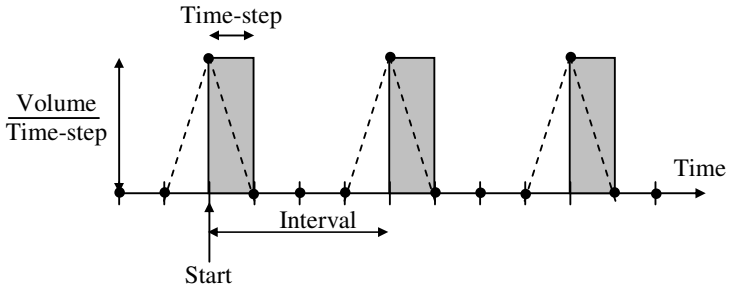


Figure 10.2. The PulseFcn.

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 PulseFcn 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 TIME < Start, and *Height* otherwise.

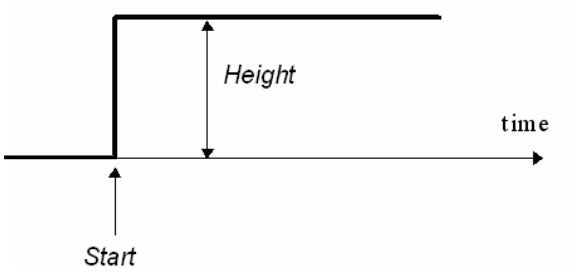


Figure 10.3. The Step function.

Ramp - Ramp function	
Syntax:	Ramp(Start, Finish, Height)
Result:	Generates a ramp, which moves linearly from *0* to *Height* between the *Start* and *Finish* times. Before *Start* the value is *0*; after *Finish* the value is *Height*.

Stop – Stop function	
Syntax:	Stop()
Result:	Immediately terminates the simulation run when executed. The Stop function can be used in an IfThenElse function.
Example:	IfThenElse([InfectiousPersons] < 0.5, Stop(), 0) [When the number of infectious persons goes below a certain value (e.g. 0.5) the simulation run terminates.]

10.2.2 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 letter.

A delay is a dynamic subsystem that contains a series of stocks and flows. The number of states in that subsystem is the *order* of the delay. See the Figure 10.4.

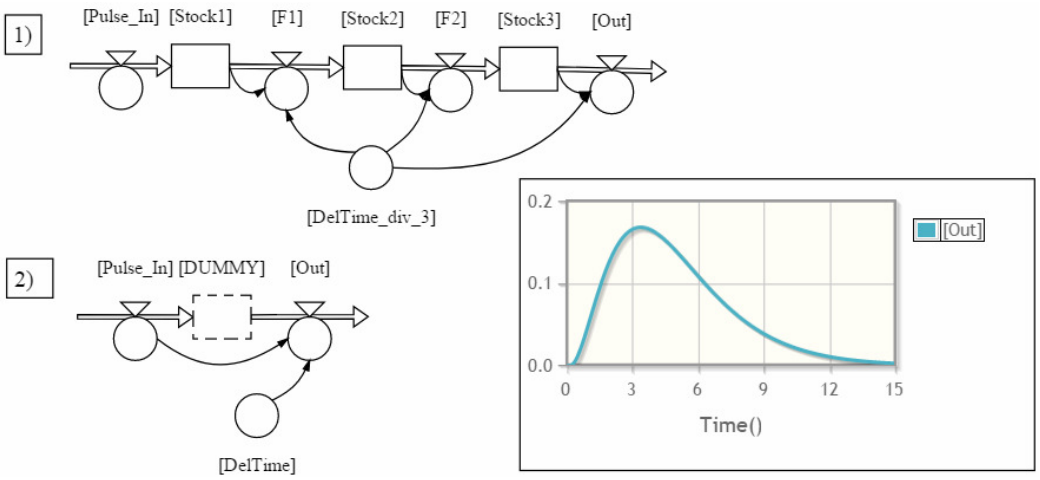


Figure 10.4. A delay of order 3 can be obtained in two ways: **1)** By connecting three stocks in a series, where each of the three stocks is delayed for Del/3 time units.
2) By using the *Delay3* function for the outflow '*Out*', so [O*ut*]= Delay3([Pulse_In], [Del], 0). This connects *Pulse_In* to *Out*. (A stock, here called *DUMMY* because it is not involved in the delaying, 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*.)

The order of the delay determines how quickly the output *starts to change* after a change in the input. Low-order delays *start to* respond more quickly than high-order delays having the same average delay time Del. See Figure 10.4, above.

Delay1 – Delay function of order 1	
Syntax:	Delay1([Primitive], Delay Length, Initial Value)
Example:	Delay1([Inflow], 6, 0)

Delay3 – Delay function of order 3	
Syntax:	Delay3([Primitive], Delay Length, Initial Value)
Example:	Delay3([Inflow], 6, 0)

PastMax – Maximal value over a past period. For statistics over (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 Days]) will create a function that starts as in the example above, but after 5 days and more displays the largest value during the last five days.

PastMin – Minimal value over a past period. For statistics over (a part of) a simulation

---

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

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

PastMean – Mean value over a past period. For statistics over (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 (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 (a part of) a simulation

---

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

Results: Calculates the correlation between two quantities over the specified *Period.*

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.

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

10.2.3 Mathematical functions

Syntax	Function	Comment
Round(X)		Rounds a number to its nearest integer
Abs(X)	abs(X)	The absolute value of X
Sqrt(X)	$\sqrt{X}$	
Sin(X)	sin X	Note: X in radians
Log(X)	$\log_{10} X$	The 10-logarithm
Ln(X)	ln X	The e-logarithm
Exp(X)	e^x	
(X) mod (Y)	X (mod Y)	Returns the remainder of X/Y **Example:** 13 (mod 5) = 3

10.2.4 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 in **Chapter 11. Macros & Variables**.

PoFlow - Poisson distributed flow
(This function is an addition to the Insight Maker functions.)

Syntax: PoFlow(FlowRate)

Result: Generates 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 mor readable form for RandPoisson(DT()*([c]*[X])/DT().

PoFlow() can be used to produce a random number of events during a time-step in a flow.

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*.

RandBinomial - Biomially distributed random numbers

Syntax: RandBinomial(Count, Probability)

Result: Generates binomially distributed random numbers from *Count* trials, each with the specified *Probability*. The result is always an integer ≥ 0.

Example: RandBinomial(10, 0.3) gives binomially distributed random numbers between *0* and *10* with an expected value of $10 \cdot 0.3 = 3$.

RandPoisson - Poisson distributed random numbers

Syntax: RandPoisson(Lambda)

Result: Generates Poisson distributed random numbers 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(Beta)

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

Example: RandExp(10) gives exponentially distributed random numbers between *0* and *infinity* with a mean of *Beta=10*.

RandExp can be used to obtain the distance in time (or space) between successive events when the rate of events is Beta.

RandBoolean - Binary distributed random numbers	
Syntax:	RandBoolean(Probability)
Result:	Generates exponentially distributed random numbers that returns *1* with the specified *Probability*, and *0* otherwise.
Example:	RandBoolean(0.5) can be used for the flipping of a symmetric coin.

RandDist - Custom distributed random numbers	
Syntax:	RandDist(x, y)
Result:	Generates random numbers according to a custom distribution. The custom distribution is specified by a sequence of (x,y)-coordinates. The Points between are linearly interpolated. The distribution does not have to be normalised such that the area under the curve is *1*, but the points must be sorted from the smallest to the largest x.
Example:	RandDist({0, 1.2}, {2, 2.3}, {3, 5.3}, {4, 3.3}).

10.2.5 Time functions

T – Current time	
Syntax:	T()
Result:	Gives the current time for the simulation.

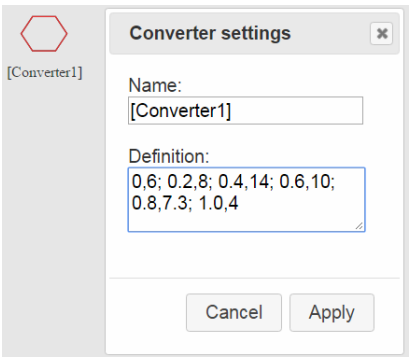
TS – Start time for the simulation	
Syntax:	TS()
Result:	Gives the start time for the simulation. This value is specified in the **Simulation Settings** form.

DT – Step-size	
Syntax:	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()
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()
Result:	Gives the End time for the simulation. TE() = TS() + TL().

10.2.6 Converter (Table look-up function)



The **Converter** is implemented as a primitive with its own button. Click on the converter button, which gives you a six-cornered figure. Double-click the *Converter to Define the data pairs* x_i, y_i. The data pairs should be separated by semicolon. (Space between each pair is inserted to make the table more readable.)

Converter – A table function (graph) that converts from input *X* to output *Y*	
Syntax:	Specify the X to Y conversions as a table in the Definition field.
Example:	You may want to use the empirically known Temperature as function of Time, Energy consumption as function of day of week or Fertility as function of Food supply in your model. Then you define the empirical function between an input X and an output Y. This relationship is expressed in tabular form. Consider for example the following relationship between X and Y: For example: 0,6; 0.2,8; 0.4,14; 0.6,10; 0.8,7.3; 1.0,4 (Shown in Figure 10.5.)

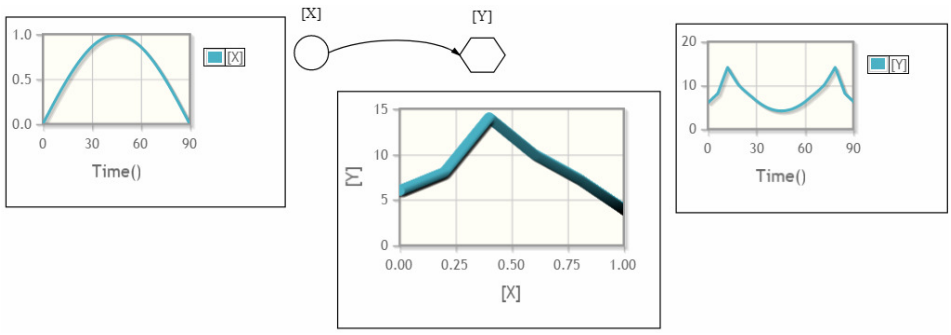


Figure 10.5. Empirical relation between input X and Output Y. Input to the converter is X(t)= Sin(Pi*T()/90), which is converted to Y(t) during 90 time units.

By using more points the table can describe the reality more accurately.

11. Macros

Macros allow you to define model code that can be included in the model. Click the **Macros** menu to open a form to write the code.

A macro can be created in two forms (Example from the Insight Maker manual).

A single-line macro has the syntax: *Myfcn <- functional expression.*
myFn(a, b, c) <- sin((a+b+c)/(a*b*c))

A multiline macro has the syntax:
Function myFcn(a, b, c)
x <- (a+b+c)
*y <- (a*b*c)*
return sin(x/y)
End Function

Macros can also be used to define a variable or a constant that will be available in the model.

Example 11.1 Making a stochastic simulation run reproducible
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. This can be done in the macro form by specifying the argument of the SetRandSeed function:

SetRandSeed(17)

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

Example 11.2 Defining a Poisson flow in a simplified way
In Section 10.2.4 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 introduce it as a macro:
*PoFlow(Lambda) <- RandPoisson(Dt()*Lambda)/DT()*

12. Practical tips

- Perhaps the most common trouble originates from *confusion about the units used.* Be very careful to specify your *time unit* as well as every other unit used. This can be done in plain text in the diagram. For example: 'Time in months, Rabbits in 1000's, Food in kg.'

- When you connect a Flow to a Stock, be sure that the connecting flow will be attached. If it is not it can create a problem that is hard to detect.

- *Try to name the primitives before you define the formulas.* Say that you have the linked variables [Variable3] and [Stock1] linked to a Flow. You define the Flow as: [Variable3]*[Stock1]. Then you change [Variable3] to [c] and [Stock1] to [X]. The 'Linked primitives:' will now be changed to [c] and [X] – but the 'Definition:' of the Flow will still be [Variable3]*[Stock1] and you will get an error message when running the model. (Of course you can then correct the Flow definition – but it is more convenient to name the primitives before you make the definitions.

- Note that a Flow can be drawn in any direction.

- 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 direction of the flow [F]=Sin(T()) will alternate over time.

- Sometimes you may want to draw several flows between two compartments. Unfortunately, this is not technically possible in the current version. However, you can make a NetFlow by adding or subtracting the flows. Say that [F1]=[c1]*[X1] and [F2]=[c2]*[X2]. Then [NetF]=[c1]*[X1] ± [c2]*[X2] in the deterministic case.

- A link can be bent into an appropriate curve of aesthetic reasons – for example you don't want to cross over other symbols or it makes the graph less messy. To do so, use the handles to form the link.

- Sometimes the model graph will suffer from links crossing the model because the elements to be connected are far away. In these cases you may create a '*Ghost*' of a symbol and place this ghost in another place and connect this instead of the original symbol. To use a ghost, mark the symbol you want to ghost, and click the **Ghost** button.

13. References to deterministic modelling

[1] [Forrester, J.W. (1961) *Industrial Dynamics*. Cambridge, MIT Press, MA. (The book where System Dynamics was introduced.)

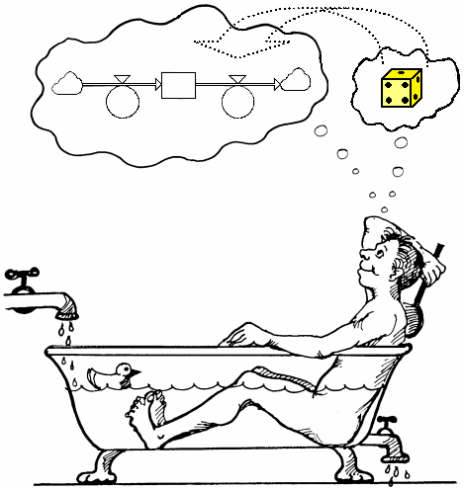
[2] Meadows, DH., and Wright D. (2008) *Thinking in systems: a primer*. White River Junction, Vt: Chelsea Green Pub. (An excellent first introduction to System Thinking and System Dynamics.)

[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 *stochastic* and dynamic models and simulate their behaviours.

Extended worldview: The world consists of two kinds of fundamental elements: Flows and States. *However, lack of detailed information requires a model to include stochasticities!*

14. 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. In modelling you often have to deal with *incomplete information*.

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

1) You ignore that you have incomplete variation. For example unknown variations are replaced by e.g. an average, and the hope that not to 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, or 6 are replaced by the average value (3.5). This is a common, stupid and dishonest way to operate.

2) You handle lack of information by at least including the information you have, often in form of a probability density/distribution function (pdf). Then random numbers can be drawn from this pdf producing 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!

A model with stochasticity is a *stochastic* model; without stochasticity it is a *deterministic* model.

14.1 The five types of uncertainty

There are five types of uncertainty about the system under study that have to be handled in the model building process. These are:

- *Structural uncertainty* – incomplete knowledge about how the system components interact.
- *Initial value uncertainty* – incomplete information about the initial conditions at 'time zero'.
- *Transition uncertainty* – incomplete information about when an event will happen (a person gets sick, a car will pass a bridge, a customer will arrive).
- *Parameter uncertainty* – incomplete information about the value of a parameter.
- *Signal uncertainty* – incomplete information about transferred information. The signal can be delayed or distorted.

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

The uncertainty of the model behaviour should manifest itself by running the model a large number of times. While a deterministic model always will produce the same behaviour, a stochastic model will produce a different behaviour for each replication (simulation run).

One way to measure uncertainty in results is to use *confidence intervals*, which is easy to obtain by simulation. For example, a two-sided *95%* confidence interval is estimated in the following way: Just run the model, say, *1000* times. Sort the *1000* outcomes of a quantity you are interested in. Remove the *25* (*2.5%*) lowest and the *25* highest replications. The remaining *950* replications will span a two-sided *95%* confidence interval.

14.2 Uncertainties in a compartment model

A compartment model is constructed by *stocks*, *flows*, *parameters*, and *information links*. There can be uncertainty related to each of these (in addition to the uncertainty of the structure used for arranging the just mentioned components).

We use an example to demonstrate how and where different types of stochasticity can enter in discrete modelling.

Example 14.1 *A classical 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: Each person meets everybody else, where a Susceptible person has the probability *p* to become infected by an Infectious one at each time unit. Furthermore, an Infectious person will recover after on average *T* time units.

A classical SIR model where the stages are represented by single compartments is shown in the Figure. This model is used to discuss the four types of uncertainty mentioned above.

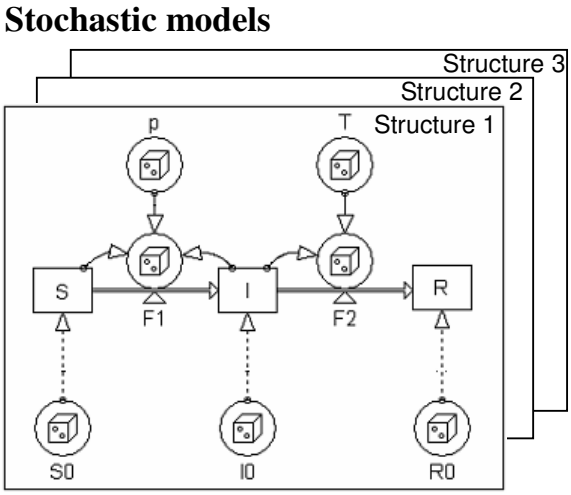


Figure 14.1. A Forrester diagram of a classical SIR model where the stages *S*, *I* and *R* are represented by single compartments. The compartments are initiated to *S0*, *I0* and *R0*. The transition flows are denoted *F1* and *F2*, and the parameters are denoted *p* and *T*.

Technically, a compartment-based model is composed of stages, transitions and parameters. Each of these composition elements can have its own type of stochasticity (symbolised by dice). It can also be questioned whether this model is a sufficiently adequate description of the system under study.

There are five possible types of uncertainty to handle: *Structural*, *Initial value*, *Transition*, *Parameter*, and *Signal uncertainties*. These will all be recognised for this example. In a compartment model the *structural uncertainty* affects how to describe the system under study in terms of an appropriate model structure of stocks, flows, parameters and information links. This is an overall uncertainty about the choice *between different model structures*, while the other four uncertainties appear at different elements *within* the chosen model. *Initial uncertainty* relates to stocks, *Transition uncertainty* relates to flows, *Parameter uncertainty* relates to parameters, and *Signal uncertainty* relates to information links.

1. Structural uncertainty

The description of the SIR model is not complete. Especially, the time in the Infectious stage (the so-called *sojourn time*) is on average *T* time units. But some persons will recover sooner and other will require a longer time. 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 realised by using several compartments in parallel and/or series.)

Another structural possibility is that an infected person does not immediately become infectious. The exposed person may require a latent period before he 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).

2. Initial value uncertainty

The information about the number of persons in the three compartments at time zero may be approximate. Say that we now 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 if 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. Further, *S(0)=N-I(0)* and *R(0)=0*.

3. Transition uncertainty

There is almost never enough information to decide the exact event times for a transition. Therefore, *transition uncertainties* in the flows *F1* and *F2* must be included in the description of the flows, using available statistical information for the transitions.

In a population model (e.g. a SIR model), transition uncertainty is usually the most fundamental and important to handle. It is easy and straightforward to implement and it will change the nature of the model from a continuous mass of a population to distinct individuals, which in turn has a number of consequences. After this introduction we will focus on how to handle this form of uncertainty.

4. Parameter uncertainty

Unexplained irregularly varying values of the parameters *p* and *T* may motivate the description of *parameter stochasticity* to generate realistic inputs from the parameters. In this case there is no given form to describe the uncertainty. Only statistical knowledge about the parameter variations or uncertainty of a parameter value can guide you here.

5. Signal uncertainty

Before we discuss signal uncertainty we have to clarify that information links in a model can be of two different types:

1. *Artificial information 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 compartment and a physical outflow. An artificial link from the compartment to the valve regulating the outflow has here to be included. Its only function is

to transfer information from the compartment equation to the flow equation. There is no counterpart to such a link in the system under study, and *no uncertainty or delay can be involved.* Artificial information links from a stage to its input or output flows are usually shown in SD languages – but not so in StochSD.

2. *Signal link* – a description of real information link that communicates information in the system under study. This communicated information we call '*signal*'. A signal can be *distorted* and *delayed.*

In this example there are no signals. However, we could assume that when the responsible authority for epidemics have collected information about the number of Susceptibles (which takes time and is never error free) it may issues recommendations about avoiding contacts and using sanitary measures, which will affect the infectious rate (when the message later on is received by some fraction of the Susceptibles).

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.).

Delays can be generated as sojourn times drawn from a proper distribution and distortion may be treated as parameter stochasticity operating at the information about the number of Infectious. In this way, uncertainty about signals can be handled by *signal stochasticity*. ■

Initial 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 you draw a random number from this distribution.

For example, assume that you estimate the probability of the *I* compartment in the SIR model initially being *I(0=1* infectious person to *60%*, *I(0)=2* to *30%* and *I(0)=3* to *10%*. Then you can draw a uniformly distributed random number between *0* and *1* and use this distribution to interpret the outcome. If *outcome* ≤ 0.6 then *I(0)=1*, else if *outcome* $\leq 0.6+0.3$ then *I(0)=2*, else *I(0)=3*.

14.3 Implementing transition uncertainty in a population model

A *population model* describes the development of a population of persons, cars, rabbits, molecules, radioactive atoms, etc. A main quality is that the number in a population or subpopulation is integer.

In a compartment-based description every compartment must contain an integer number. A continuous flow, where the flow rate is expressed as a fraction of the content in a compartment, as in a deterministic model is no longer valid. For example, in the system of decaying radioactive atoms will run out of radioactive atoms in a finite time, while a *deterministic* model of this system removing a certain fraction of remaining radioactive atoms per time unit never will reach zero.

The way to make the model correct is the following:

1) Initiate the compartments to integer values.
2) The in- and outflows to and from a compartment consist of sequences of events, where integer numbers of items arrive or depart during each time-step.

The second point requires *Transition stochasticity*.

We describe transition stochasticity with the simple example of $N=50$ radioactive atoms decaying over time with the time constant T.

A deterministic model is:

$N(t+dt) = N(t) - dt \cdot F(t)$ (where $N(0)=50$)
$F(t) = N(t)/T$

Here $dt \cdot F(t)$ is the number of decays during the time-step dt. This amount 'flows' out from the compartment N at each time-step.

In the stochastic case $dt \cdot F(t)$ is instead the number of *expected* decays during the time-step dt, and we know that this number must be integer. The probability theory tells us that the number of events (here decays) during a short time period (here dt) is *Poisson distributed*. This means that $dt \cdot F(t)$ is replaced by Po[$dt \cdot F(t)$]; where Po is the Poisson distributed random number function that is included in almost every simulation language.

However, there is a minor complication; the modelling syntax is: $F(t) = N(t)/T$ (without the dt). Therefore, $dt \cdot F(t) = dt \cdot N(t)/T$ is replaced by $dt \cdot F(t)$] = Po[$dt \cdot N(t)/T$]. Dividing left and right hands of the equation by dt gives: $F(t)$] = Po[$dt \cdot N(t)/T$]/dt.

The stochastic model thus becomes:

$N(t+dt) = N(t) - dt \cdot F(t)$ (where $N(0)=50$)
$F(t)$ = Po[$dt \cdot N(t)/T$]/dt

This is the general form for transition stochasticity.

Notice that as long as N is very large, the difference between the results from a deterministic and a stochastic model is negligible.

Sometimes you may want to draw several flows between two compartments. Unfortunately, this is not technically possible in the current version. However, you can make a NetFlow by adding or subtracting the flows. Say that [F1]=[c1]*[X1] and [F2]=[c2]*[X2]. Then [NetF]=[c1]*[X1] ± [c2]*[X2] in the deterministic case. For the corresponding stochastic case it is: [NetF]=PoFlow([c1]*[X1]) ± PoFlow([c2]*[X2]). (NOTE however that [NetF]=PoFlow([c1]*[X1] - [c2]*[X2]) is seriously wrong!)

14.4 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 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 under study is modelled as a deterministic population model, various types of distortions may occur. For example:

- The elimination of transition stochasticity may introduce *bias* in the average behaviour and result in biased estimates.
- Information about *irregular variations*, *risks*, *extremes*, *correlations*, the possibility to calculate *confidence intervals*, etc. are lost.
- 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 when they should be excited by transition stochasticity.
- In a continuous population model, the lack of transition stochasticity may *prevent queues* from building up.
- *The time* until a specific event occurs will (correctly) vary between replications in a stochastic population model, but will be the same in a continuous and deterministic one. A deterministic model may then introduce bias in the estimates. Furthermore, in a deterministic model this time can even erroneously become infinite, even when it should be finite.

Using a deterministic population model means that we only are aiming for average estimates. Unless this average will equal the average value obtained by a corresponding discrete and stochastic model, the use of a continuous model will produce biased results and prevent interesting phenomena to be displayed.

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

Assume the following for the classical SIR model, presented in Section 15.2 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 compartment is $T=4$ time units.

Simulations of the (transition) stochastic SIR model to study the number of susceptible individuals being infected, give an ensemble of results with the average value of *53.1* persons [*95% confidence interval: 50.8 – 55.5* persons] falling ill. However, the corresponding deterministic model produces *318.5* persons. An error of about *500* 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)* 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.

15. A stochastic StochSD model

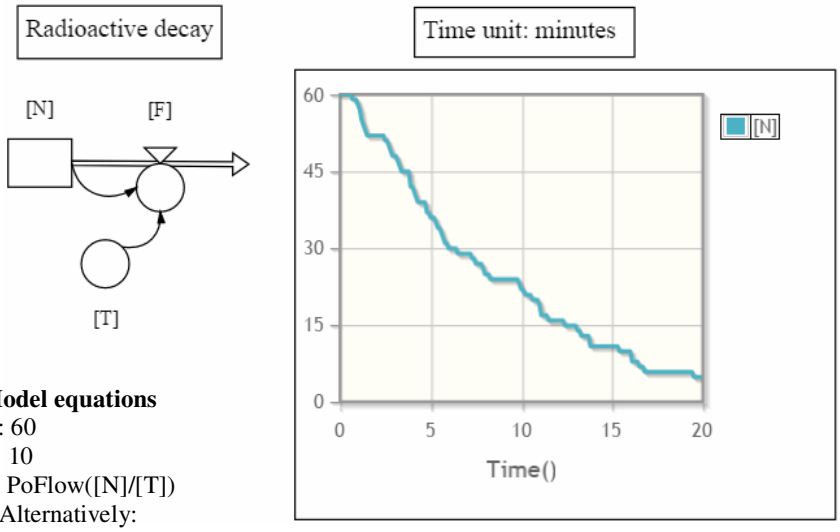
Example 15.1. Radioactive decay
A specimen contains *N=60* radioactive atoms will decay with a time constant *T=10* minutes is studied during *20* minutes.

A model can be mathematically described by the equations:

$N(t+dt) = N(t) - dt \cdot F(t)$ (where $N(0) = 60$)
$F(t) = Po[dt \cdot N(t)/T]/dt$ (In a deterministic model: $F(t) = N(t)/T$.)

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

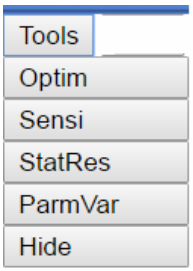
A StochSD model and a replication are shown below.



Model equations
N: 60
T: 10
F: PoFlow([N]/[T])
Alternatively:
F: RandPoisson(DT()*[N]/[T])/DT()

Figure 15.1. A stochastic model of radioactive decay and a possible outcome.

16. Tools in StochSD



StochSD contains four important tools:, **Optim**, **Sensi, StatRes** and **ParmVar** (for optimisation, sensitivity analysis, statistical analysis & result presentation, and parameter estimation). Each tool is activated by clicking on respective icon to the right. Each of these four tools also has its own documentation. Here we only indicate what they are used for. The selected tool will show up in a separate window to the right on the screen, and the selected tool works on the displayed model.

Figure 16.1. The Tools menu.

16.1. The Optim tool

Optim (Optimisation) is a simplex optimiser 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 optimiser has two main purposes in a *deterministic model*. First, an optimiser can be used *to find the optimal strategy* for a model, e.g. minimise costs or maximise benefits. To do this of an *objective function* is defined in the model as function of a number of model parameters. Then the optimiser systematically varies the values of the specified parameters to find the optimal set of parameter values.

Second, it can be used for *parameter estimation*. Then you construct a function that measures *the difference* between the system's and model's behaviour. This function is then *minimised* with respect to a set of parameters. The set of parameter estimates obtained are those which produce a 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.

The optimiser is easy to use on deterministic models – and much trickier on a stochastic one.

The simplex optimiser is reasonable fast (in number of required replications) and has the advantage that you easy can include boundaries in the parameter space. For more information, see the Optim manual [10].

16.2. The Sensi tool

Sensi (Sensitivity Analysis) is a tool for sensitivity analysis. With this tool you can study the effect of e.g. a parameter or initial value on an outcome. 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 [11].

16.3. The StatRes tool

The price to pay for using a stochastic simulation model is that many replications are required, followed by a statistical analysis.

StatRes (Statistical Results) orders a model to be run a specified number of times. Then a statistical analysis of specified outcomes (Stocks, Flows or Variables) is performed. Then *Average*, *Standard deviation*, *Min*, *Max*, and *Percentile*s for the performed number of replications are presented for each specified outcome. You can also present the outcomes from the replications in a tabular form, or as a histogram. Further you can plot two outcomes in an xy-plot and also obtain an estimate of the correlation between them. For more information, see the StatRes manual [12].

16.4. The ParmVar tool

ParmVar (Parameter Variations) is a tool for assessing the accuracy of previously estimated parameters.

Parameter estimation is tricky to perform on a *stochastic* model. It requires a large number of replication 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**.

But after this 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 the tool **ParmVar** (Parameter Variations). This is a more advanced tool that usually will not be used in an introductory course. For more information, see the ParmVar manual [13].

16.5. Hide tool

The Hide option hides a displayed tool.

17. References to stochastic modelling

[6] Gustafsson L. (2000) Poisson Simulation – A method for generating stochastic variations in Continuous System Simulation. Simulation 74/5, 264-274.

[7] 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.

[8] 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

[9] The full potential of Continuous System Simulation modelling, (2017) OJMSi, x, yy-zz. http://file.scirp.org/pdf/OJMSi_2016042717425486.pdf ???

[10] Gustafsson, L., and Gustafsson E. (2017) Optim – An optimiser for deterministic StochSD models. https://stochsd.sourceforge.io/homepage/docs/Optim.pdf

[11] Gustafsson, L., and Gustafsson E. (2017) Sensi – A sensitivity analyser for StochSD models. https://stochsd.sourceforge.io/homepage/docs/Sensi.pdf

[12 Gustafsson, L., and Gustafsson E. (2017) StatRes – A tool for statistical analysis of stochastic StochSD models.
https://stochsd.sourceforge.io/homepage/docs/StatRes.pdf

[13] Gustafsson, L., and Gustafsson E. (2017) ParmVar – A tool for studying the variation of parameter estimates in StochSD models, 2017.ParmVar.
https://stochsd.sourceforge.io/homepage/docs/ParmVar.pdf

Appendix A. Installation of StochSD

StochSD is available in two versions: **StochSD Desktop** and **StochSD Web**.

A.1 StochSD Desktop

System requirements
StochSD Desktop can be used under the operative systems Windows, Mac OS X or GNU/Linux.

Installation instruction
The Desktop version is downloaded from StochSD's homepage and runs as a normal program. Download it from StochSD's home page:
https://stochsd.sourceforge.io/homepage

Find the *StochSD.zip* file and unzip it into a folder of your choice on your computer. Then start StochSD by opening the file "*Start StochSD*".

A.2 StochSD Web

System requirements
StochSD Web can be used under any operative system that can run a modern web browser. We recommend Mozilla Firefox and Google Chrome. (StochSD does not work in Internet Explorer.)

Running instructions:
Go to https://stochsd.sourceforge.io/software in Firefox or Google Chrome.

Appendix B. License and Responsibility

B1. License

StochSD is based on the open source part of Insight Maker. StochSD's source code, including the open source part of Insight Maker, is released under a custom license based on the Affero GPL called Insight Maker Public license. Insight Maker's Public License is available at: https://insightmaker.com/impl.

This license covers all StochSD's JavaScript, HTML, and CSS code for the Insight Maker model builder and simulator. The non-open source code in Insight Maker, such as ExtJS and mxGraph are completely eliminated and replaced in StochSD.

B2. Responsibility

The user is fully responsible for the use of this product. The producer and the supplier of this code take no responsibility for the use or functioning of StochSD and its tools.