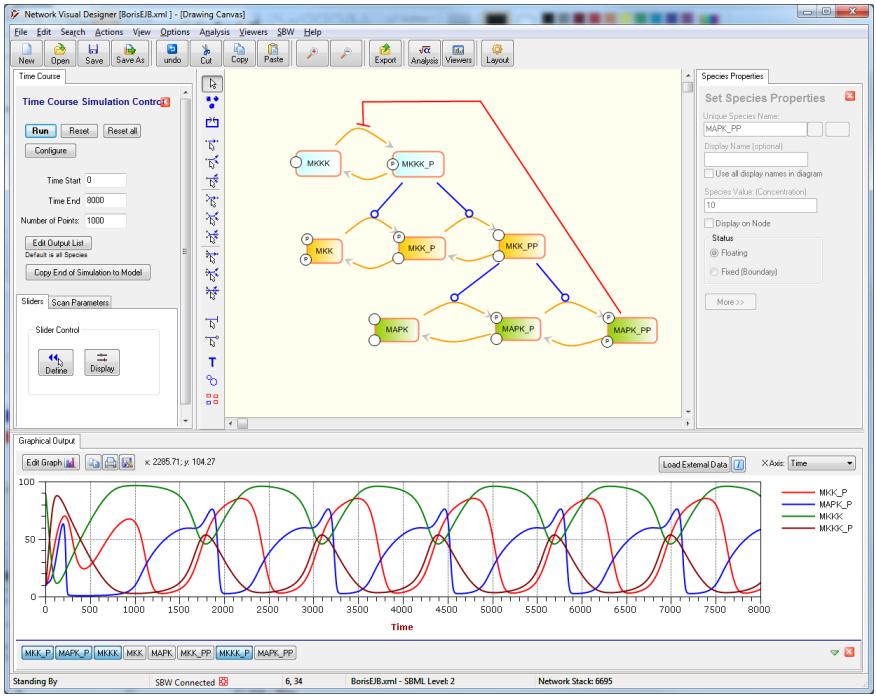
An Introduction to Biochemical Modeling using JDesigner and Jarnac



Herbert M. Sauro
July 2013, Revision 7.

An Introduction to Biochemical Modeling using JDesigner and Jarnac

By:

Herbert M. Sauro
Ambrosius Publishing
Seattle, WA

Published by Ambrosius Publishing

Version 2.2, July 2013

Funded by DARPA/IPTO BioCOMP program, contract number MIPR 03-M296-01 and the DOE GTL program, application number: DE-FG02-04ER63804. Currently funded by: NIGMS: Grant GM081070

Acknowledgements

A number of people have contributed to this booklet, most notably in chronological order:

Satish Vammi, Brian Gates, and Michael Allain for production assistance.

Corrections by Olivia Peters.

Currently supported by: NIGMS: Grant GM081070

Contents

1 Modeling **1**

1.1 Introduction 1

1.1.1 Biochemical Systems 1

1.1.2 Chemical Equations 4

2 The Systems Biology Workbench **11**

2.1 Systems Biology Workbench 11

2.1.1 Installation 11

2.1.2 How to Start JDesigner 12

2.1.3 How to Start Jarnac 12

2.1.4 Using JDesigner 12

3 JDesigner Tutorials **15**

3.1 Tutorial 1 – Basic Skills 15

3.1.1 Change Names, Concentrations and Status of Nodes 16

3.1.2 Saving and Loading Models 23

3.1.3 Setting Reaction Rate Laws 23

3.1.4 Obtaining Raw Data from a Simulation . 25

3.1.5 Setting Boundary Conditions 27
3.1.6 Steady State Simulations 28
3.1.7 Printing Models 29
3.2 Tutorial 2 – Modeling a Simple Branch 29
3.3 Tutorial 3 – Modeling a Simple Cycle 33
3.4 Tutorial 4 – Modeling True Bistable Systems . . 36
3.5 Tutorial 5 – Oscillators and Sliders 42

4 Jarnac Tutorials 49
4.1 Tutorial 1 49
4.1.1 Basic Pathway Model 49
4.1.2 Setting Reaction Rate Laws 52
4.1.3 Setting Boundary Conditions 53
4.1.4 Steady State Simulations 55
4.2 Tutorial 2 - Modeling a Simple Branch 55
4.3 Tutorial 3 - Modeling a Simple Cycle 58
4.4 Tutorial 4 - Modeling a Bistable System 63
4.5 Tutorial 5 - Reaction Stoichiometry and Oscillators 65
4.6 Tutorial 6 - Stochastic Simulation 68

1 Modeling

1.1 Introduction

This booklet introduces some of the basic ideas that are involved in modeling biochemical networks. The main purpose of this booklet is to introduce the software package JDesigner, a visual biochemical network simulation tool and Jarnac, a script based modeling package (www.sys-bio.org).

1.1.1 Biochemical Systems

A biochemical system consists of a network of coupled chemical reactions. These reactions may be enzyme-catalyzed reactions, transporters or simple spontaneous reactions. Such networks encompass metabolic, genetic, and signaling pathways.

Why Model?

This question is often asked by traditional biochemists and molecular biologists. However, what is not fully appreciated is that biochemistry and molecular biology are already full of models, though generally not quantitative models. An inspection of any of the popular textbooks reveals page after page of colorful diagrams, each a model (hypothesis) derived from thousands of painstaking experiments. Such models form the basis of a new generation of models, **quantitative** models.

The utility of quantitative modeling is that it offers an ability to evaluate the completeness and usefulness of a hypothesis. Such an undertaking will often reveal gaps in information or inconsistencies in evidence, as well as producing predictions for further testing. Quantitative models are more exact and demanding on the available evidence and are a natural progression from the current qualitative models.

The System

For the purposes of this discussion, we will define a system to be a collection of connected chemical reactions and the boundary of the system where the set of reactions intersects with its environment. In our discussions, we will assume that the boundary is made up of **fixed** species levels.

Inside the system, there is a set of reactions that connect the internal species. In addition, there will also be a set of reactions that connect the system to a set of boundary (fixed) species; it is only the internal species that evolve in time. External species may change but only under the control of the modeler. For example, a modeler might simulate a bolus injection of a substance in the form of a square wave pulse. However the pulse is imposed on the system and is not affected by changes occurring inside the system.

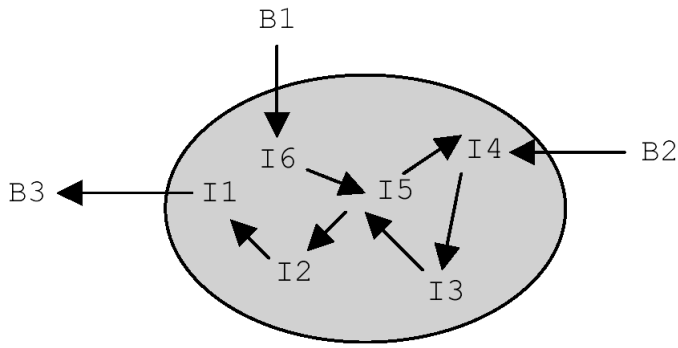


Figure 1.1 Internal and External Molecular Species. I_x represent internal molecular species and B_x external fixed molecular species.

Time Evolution and Steady States

It is almost without exception that dynamical systems exist in one of three possible states, these are:

- Thermodynamic Equilibrium
- Transient State
- Steady State

Thermodynamic equilibrium is of little interest in biology since the chief characteristic of biological systems is that they are out of equilibrium. Equilibrium is the state where no energy is dissipated and all energy or mass gradients within or between the system and the environment are nonexistent.

The remaining states, transient and steady state, are of much more interest. When the boundary of a system is fixed, it is often the case that the system will evolve to some stable unchanging state called

the steady-state. By unchanging we mean that all internal concentrations and all reaction rates are steady. However, even though the system appears quiescent, it is in fact dissipating energy across the system boundary and there will be a net movement of mass from one boundary to another. In mathematical terms, the steady-state is characterized by the rates of change of all internal species being zero,

$$\frac{dS_i}{dt} = 0 \text{ for all } i$$

where i represents all internal species.

Source and Sinks

The terms source and sink are sometimes used to refer to the boundary species and are meant to suggest that some boundary species supply mass to the system, while others provide mass exit points from the system.

Kinetic Rate Laws

One of the most important decisions when building a biochemical model is deciding on the form of each reaction rate law. Many readers will probably be familiar with the fundamental mass-action rate laws, these are the basic rate laws from which all others, including aggregated forms such as Michaelis-Menten rate laws, are built. The reader is referred to the text book, Enzyme Kinetics fort Systems Biology, Sauro HM (2011), ISBN-10: 0982477317. for more information on rate laws and their selection.

1.1.2 Chemical Equations

Chemical equations are commonly written in the following way:

$$A + B \rightarrow C + D$$

indicating that species A and B react together to form species C and D. From the chemical equation, we can easily write down rate laws based on mass-action kinetics.

Mass-Action Rate Laws

Consider the simple mass-action reaction shown below:

$$S \rightarrow P$$

This simple reaction is often described using a first-order rate equation of the form:

$$v = k_1 S$$

where v is the rate of conversion of substrate (S) into product (P) and k_1 is the rate constant. If the reaction is elementary then the order of the reaction can determined by the stoichiometry of the reaction. For example the reaction:

$$2S \rightarrow P$$

often has a rate equation described by:

$$v = k_1 S^2$$

where S is now second-order with respect to the rate. In the most general case where we have the following reaction:

$$\alpha_1 S_1 + \alpha_2 S_2 + \alpha_3 S_3 + \ldots \rightarrow$$

The generalized rate for this reaction is given by:

$$v = k_1 S_1^{\alpha_1} S_2^{\alpha_2} S_3^{\alpha_3}$$

Reversible rate laws are treated in a similar fashion, for example the rate law for the reaction:

$$S \rightleftharpoons P$$

can be written as

$$v = k_1 S - k_2 P$$

that is the rate of reaction is the forward rate minus the reverse rate.

Aggregate Rate Laws

Aggregate rate laws tend to be more useful in modeling metabolic pathways where enzymes catalyze the individual reaction. In the early days of modeling (50s and 60s), researchers often described enzyme catalyzed reactions explicitly in terms of individual mass-action rate laws. For example the simple irreversible Michaelis-Menten mechanism:

$$S + E \rightleftharpoons \mathrm{ES} \rightarrow E + P$$

would be described explicitly using the differential equations:

$$\begin{aligned}
\frac{dS}{dt} &= k_2 \mathrm{ES} - k_1 E S \\
\frac{dE}{dt} &= k_1 E S - k_2 \mathrm{ES} + k_3 \mathrm{ES} \\
\frac{dP}{dt} &= k_3 \mathrm{ES}
\end{aligned}$$

However, such a representation requires knowledge of the individual rate constants, k_1, k_2 and k_3, information that is rarely available.

Instead people realized that they could use aggregate equations derived using steady-state assumptions resulting in much simpler rate laws.

The most well-known aggregate rate law in this family is the single substrate **irreversible** Michaelis-Menten rate law:

$$v = \frac{V_m S}{K_m + S}$$

Aggregate equations proved much easier to use and although they are derived using the steady-state assumption there has been little evidence that this affects the model dynamics. In addition, the number of parameters required to describe an aggregate rate law is much fewer.

In biochemical modeling, probably the most commonly used rate law is the **reversible** Michaelis-Menten rate law described by:

$$v = \frac{V_m / K_{m_1} (S - P/Keq)}{1 + S/K_{m_1} + P/K_{m_2}}$$

Stoichiometry

Stoichiometry refers to the numbers of molecules that take part in a particular reaction. Thus the molecular species, S, in the reaction

$$S \rightarrow P$$

has a stoichiometry of unity. Where as in the reaction

$$2S \rightarrow P$$

has a stoichiometry of 2.

Mass-balance Equations

All physical systems must obey the law of conservation of mass, that is any disappearance of mass must be accompanied somewhere by an appearance of an equal amount of mass. Thus, given a molecular species, S_i, sometimes termed a species pool, we can describe how S_i changes in time by taking into account the flow into and out of the pool due to the various reactions that impinge on it.

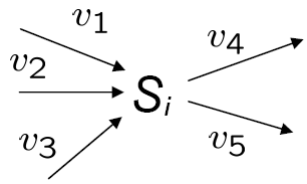


Figure 1.2 Mass-conservation means that changes in the concentration of S_i must be the sum of all input rates minus the sum of all output rates.

The rate of change in the concentration in S_i is given by the mass balance equation:

$$\frac{dS_i}{dt} = (v_1 + v_2 + v_3) - (v_4 + v_5)$$

Where the v terms represent the rates of reaction for the processes that produce and consume S_i. An entire model is described by a series of mass conservation equations, one equation for every internal species in the model. In compact form, such a series of equations is often expressed as a matrix equation:

$$\frac{d\mathbf{s}}{dt} = \mathbf{N}\boldsymbol{v} \tag{1.1}$$

where $\mathbf{N}$ is called the stoichiometry matrix, and $\boldsymbol{v}$ the rate vector. An example will be given in the next few pages.

A Simple Model

The following simple pathway comprise of four reactions, v_1, v_2, v_3 and v_4; three internal species, S_1, S_2, S_3; and two external species, X_o and X_1.

$$X_o \xrightarrow{v_1} S_1 \xrightarrow{v_2} S_2 \xrightarrow{v_3} S_3 \xrightarrow{v_4} X_1 \tag{1.2}$$

By considering mass-conservation, we can write down the three differential equations that govern the dynamics of the pathway as follows:

$$\begin{aligned}
\frac{dS_1}{dt} &= v_1 - v_2 \\
\frac{dS_2}{dt} &= v_2 - v_3 \\
\frac{dS_3}{dt} &= v_3 - v_4
\end{aligned}$$

Note that we do not specify differential equations for the two external species, X_o and X_1 since they are assumed to be fixed boundary species.

The solution of the equations is usually carried out by some standard integration procedure, yielding the concentration of S_1, S_2, and S_3, as a function of time.

As mentioned in a previous section, the differential equations given above are more conveniently described using a matrix notation. Recall that the systems equation was given by:

$$\frac{d\mathbf{s}}{dt} = \mathbf{N}\boldsymbol{v}$$

where **N** is the stoichiometry matrix and v the rate vector. For example, in the case of the simple model, the stoichiometry matrix is given by:

$$\begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{1.3}$$

And the rate vector, v, is given by:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \tag{1.4}$$

The purpose of JDesigner and associated software is to permit users to graphically specify the model, derive the set of differential equations automatically and generate a solution.

2

The Systems Biology Workbench

2.1 Systems Biology Workbench

2.1.1 Installation

Download the Systems Biology Workbench from

`http://www.sys-bio.org/downloads/`

and follow the instructions given by the setup program.

2.1.2 How to Start JDesigner

During installation, an icon will automatically appear in a new program group called Systems Biology Workbench. To start JDesigner, double-click on the JDesigner icon that can be found in the model editor section.

2.1.3 How to Start Jarnac

During installation, an icon representing the Jarnac program will automatically appear in a new program group called Systems Biology Workbench. To start Jarnac, double-click on the Jarnac icon that can be found in the model editor section.

Getting Help

There are a number of ways to get help when using Jarnac. The first is the Help menu. The Help menu has four help options, a traditional Windows help file under 'Contents', a hyperlinked library reference that opens into a browser, the Jarnac Web site and email feedback to the author.

The other kind of help is at the console itself. To get quick help on a function you know the name of, type a question mark followed by the name of the function. For example to get help on the graph function, type:

```
->?graph
graph (matrix) or graph (vector)
->
```

2.1.4 Using JDesigner

On start up the user is presented with the screen shown in Figure 2.1.

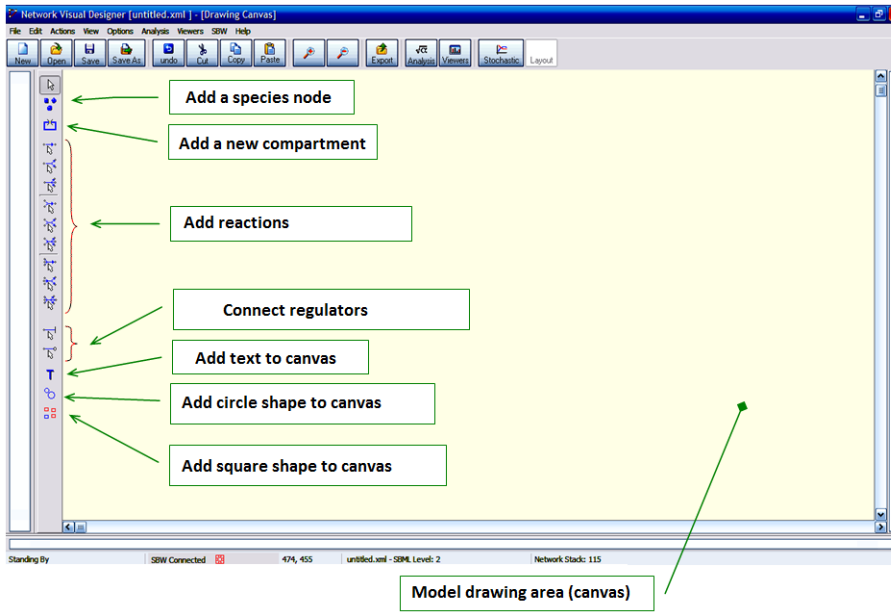


Figure 2.1 Basic JDesigner Controls.

Some useful toolbar buttons:

Add species node: After selecting this button, drag the cursor to any location on the drawing canvas and click to place a node. Each node represents a specific entity in the system, such as a chemical species , a protein or a transcription factor.

Add reaction: JDesigner has the capability to represent reactions involving one, two, or three substrates and one, two, or three products. To add a reaction, select the button with the desired numbers of substrates and products, and then click on each node involved in the reaction. In order to ensure that a reaction placed on the canvas has the desired substrates and products, click on each substrate node first, then each product node. The reaction will appear on the canvas as a set of line

segments emanating from each substrate node terminating in arrows pointing at each product node. Note that even if a reaction is defined as reversible (see below), the arrows will point only at the nodes defined as products in this step.

Add text: Click on the canvas to add descriptive text, i.e. the name of a reaction, compartment, or system.

Moving elements on the canvas

It may be useful to move certain elements around on the canvas, either to highlight an important feature of the system or for aesthetic reasons. To move an element, make sure the arrow button has been depressed. Click on an element on the canvas to select it for movement. The border of a selected element will turn red (in the case of a reaction or text, the entire element will turn red). The way each element may be moved is as follows.

Nodes: Click and drag the node to the desired location. Line segments for all reactions involving the node will curve and lengthen/shorten to accommodate the new location.

Compartment: To move the compartment, click and drag any edge of the compartment. Moving the compartment will also move any node, which has been placed, in the compartment. To change the size of the compartment, hover the cursor over its lower right corner. Click and drag the compartment to the desired size.

Reaction: Clicking on a reaction curve will display line segments tangent to where the curve meets each node involved in the reaction. These segments end in small circles. Click and drag these circles to change the curvature.

Text: Click and drag to move the text to the desired location.

3

JDesigner Tutorials

3.1 Tutorial 1 – Basic Skills

Let us begin by building a simple network. This network will comprise of four molecular species and three reactions. The first thing to do is add the nodes that represent the molecular species. Click on the add node button (see Figure 3.1), move the mouse to the drawing area and click on the left mouse button to 'drop' four nodes on the canvas. Once you have completed this, click on the pointer button, this prevents you from dropping any further nodes onto the canvas. Your screen should look something like Figure 3.1, note that the exact placement of the network nodes may differ.

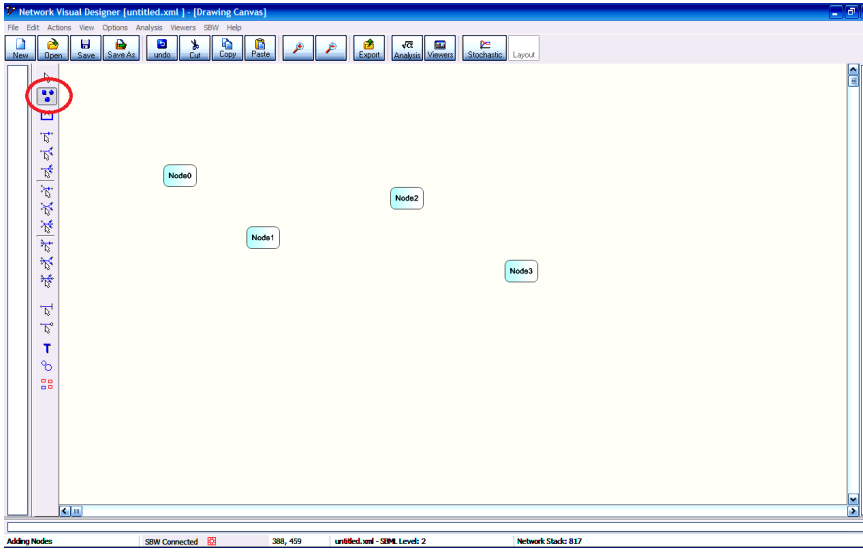


Figure 3.1 Species Nodes Placed onto Canvas.

3.1.1 Change Names, Concentrations and Status of Nodes

To change the properties of a particular node, right-click over the node and select the properties menu item. This will bring up the node properties dialog box, from here you can change a variety of things, including the name of the node, its initial concentration and its status as a boundary condition or not (see Figure 3.2).

Move the mouse over the first node, right-click and select the properties item. You will see the screen shown in Figure 3.2.

Once the property window is visible, you can change the properties of other nodes simply by clicking on the node and the properties window will automatically change to reflect the new node.

In our example, you should make the following change:

Set the concentration of the first node (Node0) to 10 units.

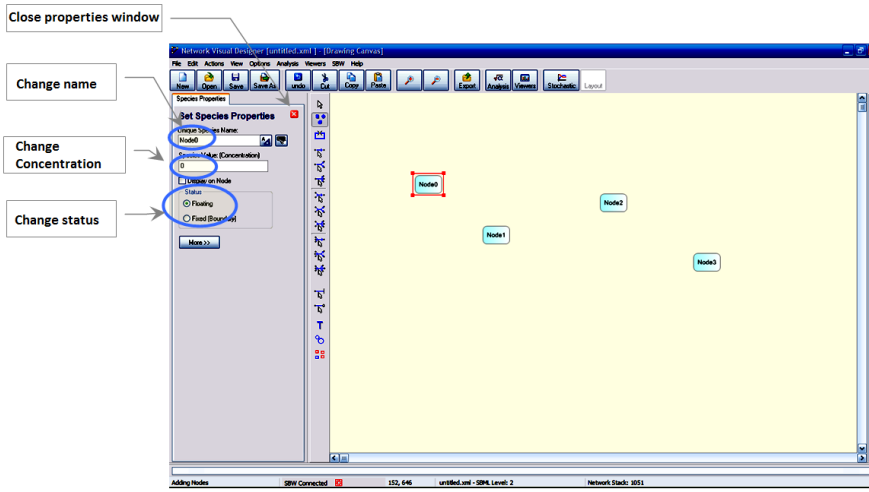


Figure 3.2 Setting node properties.

> Note: Units
> Throughout this discussion you will find that we do not mention the units used in any of the models. It is up to the user to ensure that values used to initialize concentrations, rate constants etc., are dimensionally consistent with each other.

Close the properties window by clicking on the red close button at the top right corner (see Figure 3.2).

Note that a species node can represent any kind of molecular species, for example, small molecules, such as ATP or glucose, large molecules such as proteins, in their phosphorylated or unphosphorylated state, or nucleic acids such as mRNA or tRNA. JDesigner deals with generalized network models and thus any biological chemical network can be represented.

We are now ready to add some reactions.

On the left side tool bar, you will see nine different reaction types. These range from simple uni-uni to tri-tri reactions. We will add

three uni-uni reactions to the model. These reactions will go from the first node to the second node, the second node to the third node and third node to the forth node. To add the first reaction select the uni-uni button, this is the first reaction button (see Figure 3.3). Select the reactant by clicking on Node0 (it should highlight by turning red), then click on the product (Node1). Once you click on the product node, a reaction arc will automatically appear.

Proceed to do this for the other two reactions. Once you have finished, make sure that you **unselect** the reaction button by clicking on the arrow button. If all has gone well you should see the following screen:

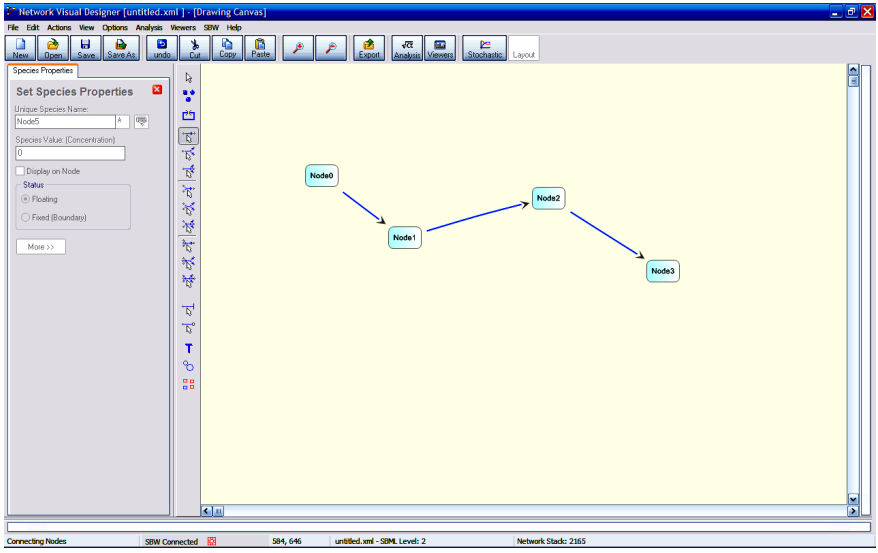


Figure 3.3 Reactions and Species Nodes added to a model.

By default, JDesigner assigns a simple irreversible mass-action rate law to every reaction. You can check this by right-clicking over a reaction arc and selecting the properties item (see Figure 3.4).

From this panel you can change things like the name of the reaction or its stoichiometry and the reaction rate law. You will notice that

the reaction by default has been assigned a simple first-order mass-action rate law. Note also that the rate constant has been assigned a default value of 0.1.

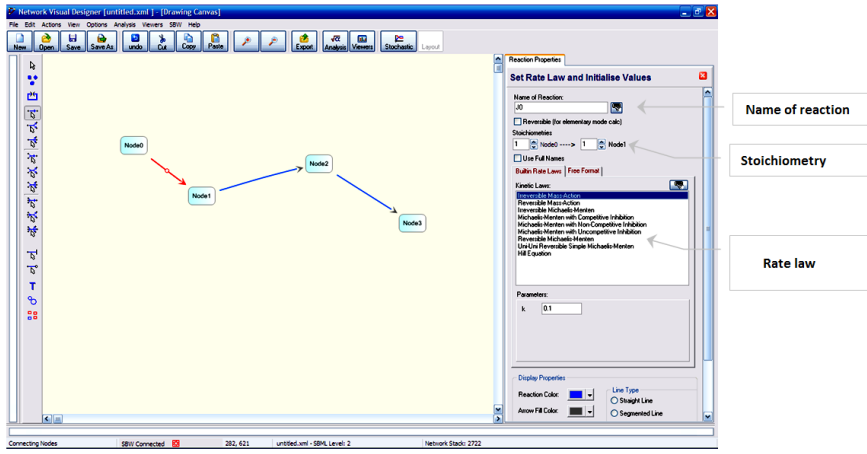


Figure 3.4 Reaction Properties Window.

As an exercise, we will make the following changes to the rate constants:

1. Set the rate constant for the first reaction to 0.6
2. Set the rate constant for the second reaction to 1.0
3. Set the rate constant for the third reaction to 0.15

To change the rate constants just select each reaction in turn using the mouse and the reaction panel will change to reflect the current reaction.

We are now ready to carry out a simple time course simulation. This simulation will show the pathway undergoing a transition as mass flows out from the first node (Node0) to the other nodes in the pathway.

First, close the reaction properties window; this will leave more room on the screen. Click on the red close button in the top right-

hand corner.

> Note: Selecting a Simulator
> JDesigner itself has no simulation capability, but there is choice of simulators available in JDesigner which can be accessed by selecting 'File' and then clicking on 'Preferences' from where you can select a simulator. By default, the roadRunner simulator is selected.

To carry out a simulation you need to bring up two windows, the time course simulation window and the graph plotting window. Go to the horizontal tool bar and select the analysis button, choose Time Course Simulation. Select the viewer button and chose, 'View Graphical Output'. Two new panels should appear. Your screen should look like Figure 3.5:

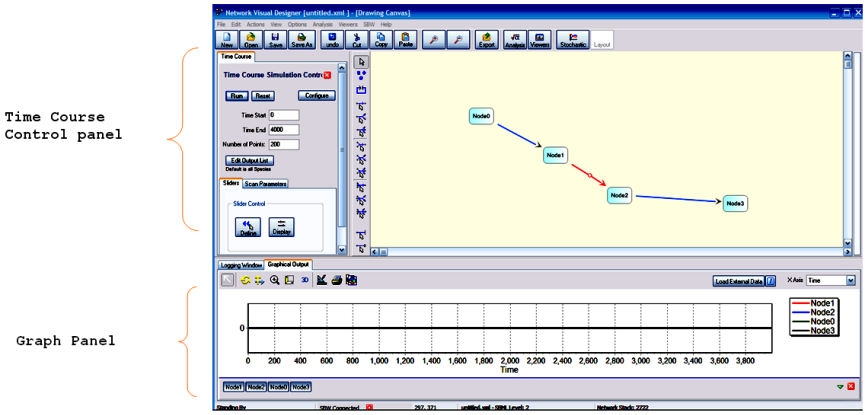


Figure 3.5 Time Course Control Panel and Graph Panel.

You can carry out a simulation immediately by simply clicking on the run button that you can see in the time course control panel. If you do this, you should get the screen shown in Figure 3.6.

The graph in the lower panel shows how all four nodes change in time. As you can see the concentration of Node0 declines as the other nodes fill and empty in turn.

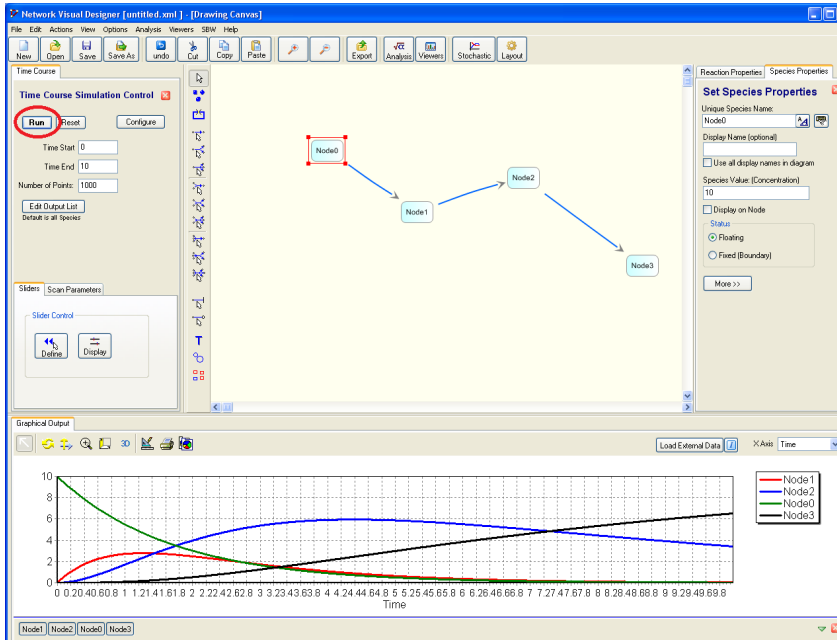


Figure 3.6 A Typical Run Generated from the Time Course Control Panel.

Let us do the simulation again, but this time over a longer period. By default, simulations are run from zero to ten time units. We can change the end time by simply entering a new value into the Time End edit box that you can find below the run button.

Let us set this time to 30 time units. In order to repeat the simulation, we must also **reset** the simulation back to its initial condition. To do this, simply click on the reset button next to the run button (Figure 3.7). If you do not reset the model, the next simulation will simply carry on from the data generated in the last simulation.

The sizes of the various panels in the screen can be changed, for example you can make the graph panel larger by moving the splitter bar (see Figure 3.7). The graph can also be copied to the clipboard or printed.

Now run the simulation again by clicking on the run button, you should see the graph shown in Figure 3.7 and 3.8. Figure 3.8 was obtained by clicking on the Copy to Clipboard button (Figure 3.7) and pasting the image into Word.

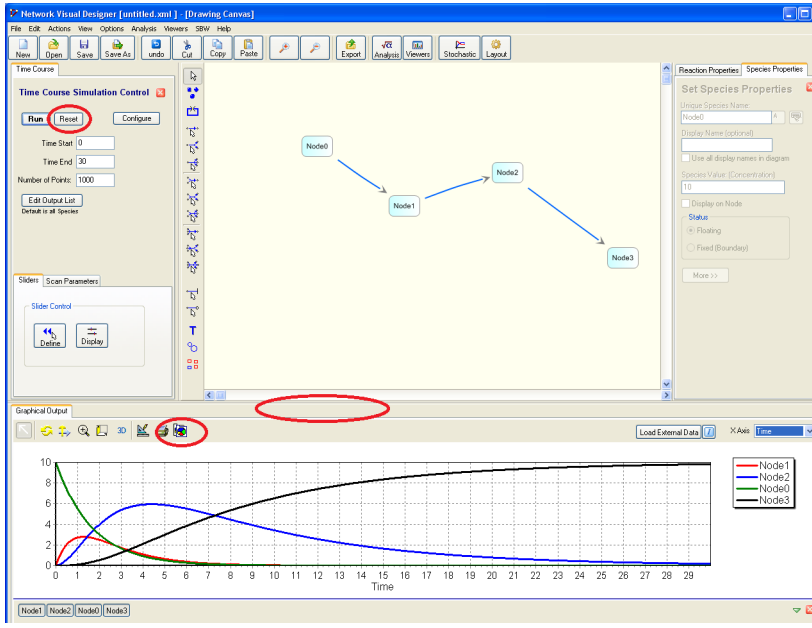


Figure 3.7 Changing Time to Run and Rerunning the Simulation.

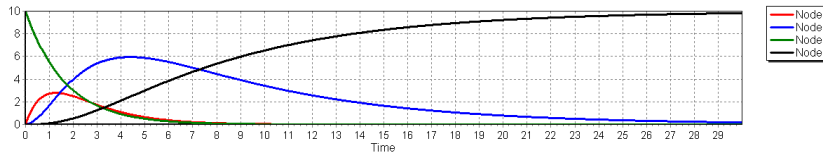


Figure 3.8 Final Output from the Graph Panel.

3.1.2 Saving and Loading Models

All models are saved using native SBML (Systems Biology Markup Language) with additional annotations for storage of graphical elements. These files are compatible with other modeling tools that can import SBML. JDesigner supports both SBML Level 1 and 2. For normal use, it is recommended the models be saved in SBML Level 2 format.

3.1.3 Setting Reaction Rate Laws

We are now going to make a change to the model by setting different rate laws for the three reactions. In the earlier version, all the rate laws were simple irreversible mass-action rate laws. For many models this may not be realistic, instead we will employ in the next version, reversible Michaelis-Menten rate laws.

To change a rate law, move the mouse over the reaction of interest and right-click, this will bring up a popup menu, from this select the reaction properties option. This will bring up the reaction properties panel that we saw in Figure 3.4. Let us have a closer look at the rate law setting section (Figure 3.9). This shows the list of built-in kinetic laws. If you move the mouse over one of the rate laws, and right-click, a popup menu will appear from where you can obtain more details.

In this example, you should select the rate law marked **Reversible**

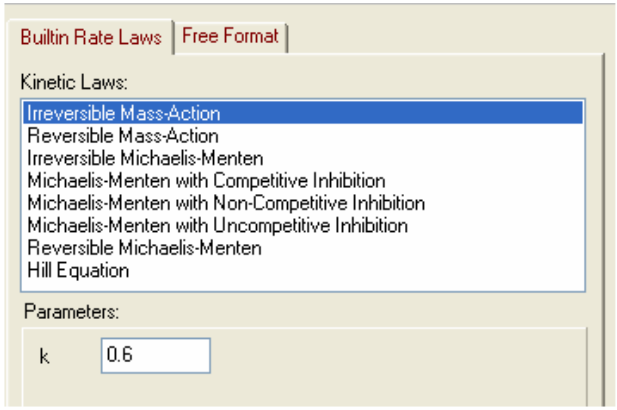


Figure 3.9 Example of a standard list of rate laws.

Michaelis-Menten. Notice that a new list of parameters has been displayed to reflect the selected rate-law. From this panel you can set the values of the different parameters. Change the kinetic parameters to the following values:

Reaction 1:	Vmax = 2.5; Keq = 0.8
Reaction 2:	Vmax = 1.2; Keq = 0.2
Reaction 3:	Vmax = 0.5; Keq = 4.8

Close the reaction properties panel and bring up the time course control panel and graph display panel. Set the time end to 10 time units and press the run button. You should see something like this:

Notice the difference between the new simulation and the earlier one shown in Figure 3.10. Whereas in Figure 3.10, all the mass ended up in the last node (Node3), in the new simulation the mass is distributed. Given the values for the equilibrium constants (0.8, 0.2, 4.8) you should be able to predict the final ratio between Node0 and Node3. Your prediction should correspond to the simulation values.

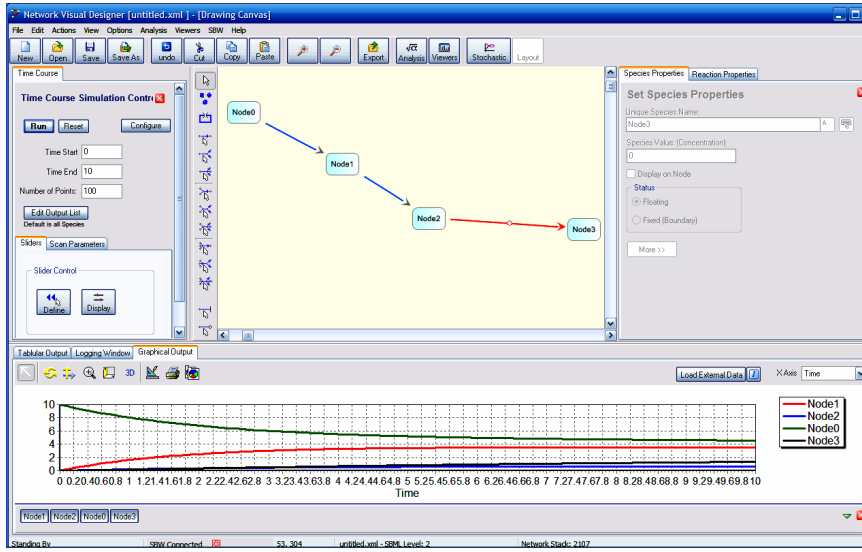


Figure 3.10 Time course using reversible Michaelis-Menten Rate Laws.

NOTE: Adding new built-in rate laws.
The list of rate laws is determined by an external file called ratelaws.xml. When JDesigner starts up, it reads this file, from which it builds the list shown in the rate law panel.

3.1.4 Obtaining Raw Data from a Simulation

Until now, we have plotted results from simulations. There will be times when the raw data is required. This can be easily accomplished by selecting the Table Viewer and rerunning the simulation. Note that you can have the graph and table viewer visible at the same time.

Select the table viewer from the views toolbar button (see Figure 2.1) and select the table viewer. Click on the reset button on the time

course simulation control panel and set the number of points to 10. If you now rerun the simulation and select the table viewer tab you should see the following table of results:

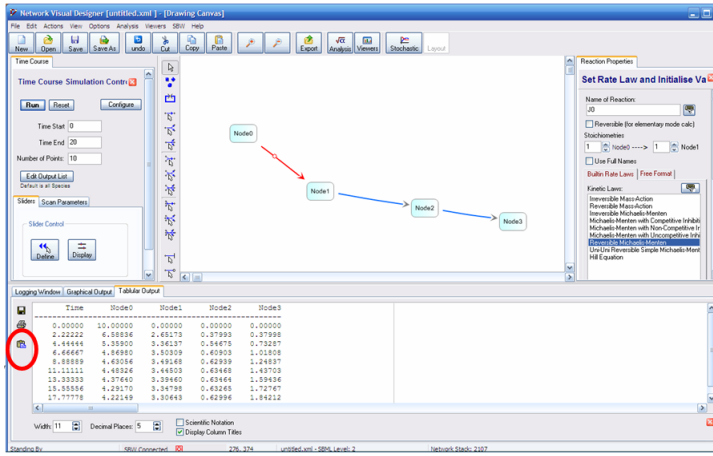


Figure 3.11 Simulation which uses the table viewer.

You can copy the raw numerical data to the clipboard if you wish to process the simulation data in another application, for example Excel. Here we have just copied the raw data into word.

Time	Node1	Node2	Node0	Node3
0.00000	0.00000	0.00000	10.00000	0.00000
2.22222	2.65173	0.37993	6.58836	0.37998
4.44444	3.36137	0.54675	5.35900	0.73287
6.66667	3.50309	0.60903	4.86980	1.01808
8.88889	3.49168	0.62939	4.63056	1.24837
11.11111	3.44503	0.63468	4.48326	1.43703
13.33333	3.39460	0.63464	4.37640	1.59436
15.55556	3.34798	0.63265	4.29170	1.72767
17.77778	3.30643	0.62996	4.22149	1.84212
20.00000	3.26967	0.62704	4.16188	1.94141

Referring to the earlier question about the predicted ratio of Node3 to Node0 we can compute the ratio from the equilibrium constants by computing first the overall equilibrium constant as (0.8 x 0.2 x 4.8 = 0.768). To make sure that the system reaches equilibrium we must simulate for a longer time period so that the concentrations settle to constant values. The following table was derived from a simulation with a time end of 300 time units.

Time	Node0	Node1	Node2	Node3
0.00	10.00	0.00	0.00	0.00
75.00	3.72	2.97	0.59	2.72
150.00	3.67	2.93	0.59	2.81
225.00	3.67	2.93	0.59	2.81
300.00	3.67	2.93	0.59	2.82

The ratio of Node3 to Node0 is 2.82/3.67 is 0.768 as predicted theoretically. Note also that since the system is closed, the total mass remains the same, that is at the end of the simulation, Node0 + Node1 + Node2 + Node3 = 10

3.1.5 Setting Boundary Conditions

In the previous model all the molecular species, Node0 to Node3, were free to change during the simulation, the system was a closed system. Biological systems are however open, that is, there is a net and steady flow of mass from one model boundary to another. In a real system, the system boundaries are usually fixed and as a result the system will reach a steady state rather than equilibrium. Let us now change the previous model by fixing Node0 and Node3, rerun the simulation, and compare it to the original version.

First, close all viewer and time course windows. Changing nodes from float to fixed status is very straightforward. One can either bring up the node properties window and change the setting from

this window, or one can use the shortcut from the popup menu. Either way, change the status of Node0 and Node3 to boundary nodes. Boundary nodes are indicated by shadowed edge compared to floating nodes.

Once you have changed the status of the two edge nodes, bring up the time course simulation and graph panels and run the simulation. The graph below illustrates the evolution of the new system.

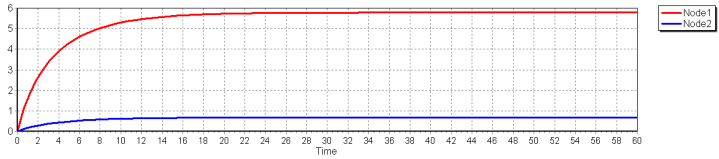


Figure 3.12 Time Course Simulation Showing Evolution to Steady-state.

Notice this time, that the system reaches a steady state.

3.1.6 Steady State Simulations

Before we continue to the next stage, close any time course control panels or viewing panels such as the graphing panel.

What if we wanted to compute the steady-state immediately rather than using a time course simulation? Under the analysis toolbar button there is an additional analysis option, called Steady State Analysis. Selecting the steady-state option will bring up the steady-state panel. This is shown below.

The steady-state panel is very simple, it has a single button marked Run. By clicking run, JDesigner will attempt to find the steady state (assuming there is one) for the system. If a steady-state is located, the values for the steady-state concentrations are indicated in the table below the run button.

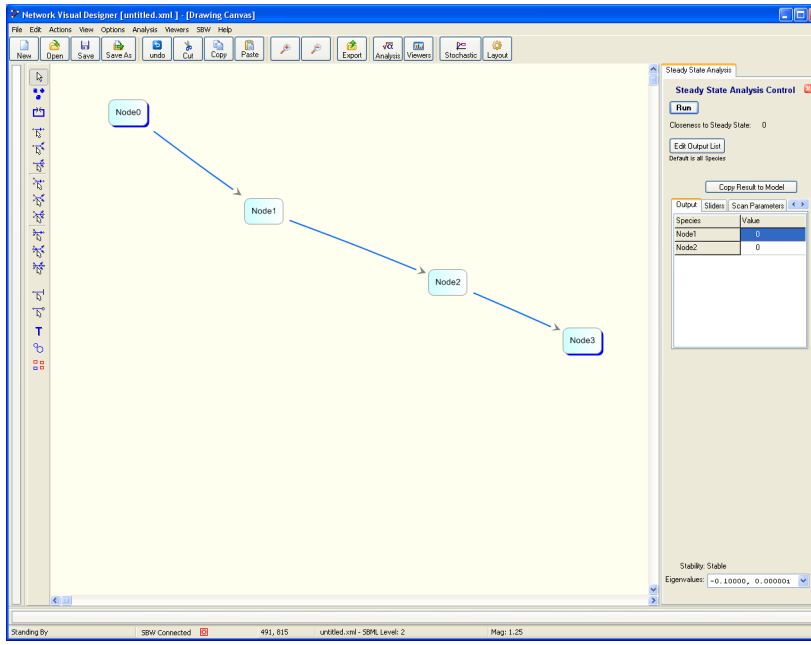


Figure 3.13 Steady-state Control Panel.

3.1.7 Printing Models

Model diagrams may be printed using the print menu item under the File menu. If you have acrobat or CutePDF Writer installed you can also print to pdf files.

3.2 Tutorial 2 – Modeling a Simple Branch

In this tutorial, we will study some of the basic properties of a simple metabolic branch and introduce the notion of a **pathway flux**.

Enter the model shown in Figure 3.14 into JDesigner.

Let X_o, X_1 and X_2 be boundary species. S_1 will be an internal

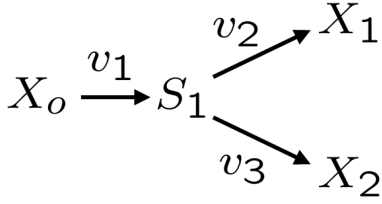


Figure 3.14 Branched Pathway.

species. Assign Reversible Michaelis-Menten rate laws to all three reaction rates. The default values for the parameters can remain unaltered except for the V_m's which should be set to:

Parameter	Value
V_{m_1}	1.5
V_{m_2}	2.0
V_{m_3}	4.0

Set the initial concentration for X_o to be 10 concentration units. All other concentrations leave at zero. Bring up both the steady-state control panel and the table viewer. Compute the steady-state by clicking on the run button. Note that the output only shows a single item, the concentration of S_1 which should be at a value of about 0.031.

We are now going to introduce the notion of a pathway flux.

At any time during the evolution of a pathway, the rates through the different reactions as well as the species concentrations will evolve in time, eventually settling to some steady-state values. The reaction rates that we measure during a simulation are termed fluxes.

We can examine the fluxes of our simple branch very easily. On the steady-state control panel, below the run button you should find the 'Edit Output List' button. If you click on this button you will get the following dialog page

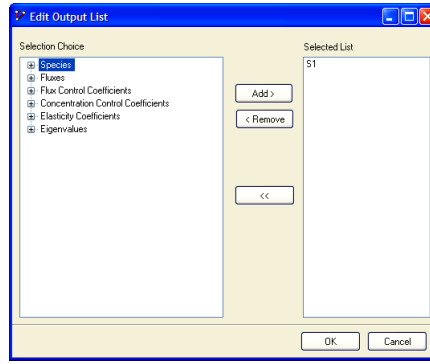


Figure 3.15 Output List Selector Panel.

This panel permits one to select what attributes of a pathway one might like to view. By default, all internal molecular species are displayed. On the left side one can see the kinds of attributes to display, in this case we are interested in the fluxes. Click on the fluxes entry and it should open up to reveal the list of fluxes. In the case of a simple branch, there will be three fluxes, these are indicated by the reaction names, J_0, J_1 and J_2 (see Figure 3.16).

There are two ways to select attributes, one way is to select each one individually, the first flux, the second flux, and so on, each time clicking on the add button. However, in this case we want to view all three fluxes, it is quicker to select the name 'Fluxes' and hit the add button, this will select all three fluxes and add them to the selected list on the right-hand side.

Once you have completed this, click on the OK button and recompute the steady-state. You should now see the following results in the table viewer:

S1	J0	J1	J2
0.03141	1.43404	0.47801	0.95603

What do you observe about the flux values, what pattern is there?

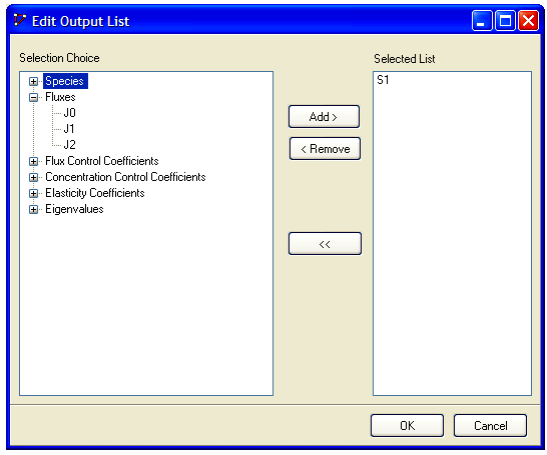


Figure 3.16 Selector Panel showing Flux Symbols.

Can you explain the pattern?

We will now try one final experiment which will investigate the effect of increasing the activity of the enzyme on the second limb, that is we will change the Vmax of J_2

Change the second Vmax from 2.0 to 6.0. Recompute the steady-state of the network.

S1	J0	J1	J2
0.01706	1.45735	0.87441	0.58294

What do you observe? Compare the flux values with the values from the earlier experiment. Explain for example the change in J_2

Comment: J_2 decreases by almost 50%, J_0 increases slightly and J_3 almost doubles. The explanation for this is simple, when we increase the Vmax on J_2 this results in a decrease in the concentration of S_1. This means that the flux through J_2 decreases. The flux through J_2 increases because we have increased the Vmax even though S_1 decreases. The change in J_0 is marginal and increases

because the product inhibition of S_1 has decreased. The net effect is a redistribution of flux from one branch to another.

3.3 Tutorial 3 – Modeling a Simple Cycle

In this tutorial, you will investigate the basic properties of a simple cycle and introduce the idea of parameter scans. Such cycles are very common in signaling networks and thus an understanding of their properties is important.

Enter the model shown in Figure 3.17 into JDesigner.

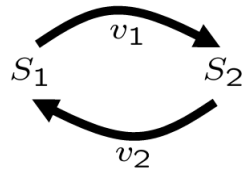


Figure 3.17 Cycle Model.

The differential equations for this model are easily written:

$$\frac{dS_1}{dt} = v_2 - v_1$$
$$\frac{dS_2}{dt} = v_1 - v_2$$

Note that this network has as an important property, namely that

$$\frac{dS_1}{dt} = -\frac{dS_2}{dt}$$

This means that at all times:

$$S_1 + S_2 = \text{ constant}$$

In biological terms, the forward arm, v_1 might be catalyzed by a kinase and the reverse arm, v_2, a phosphatase.

Set both reactions to have irreversible Michaelis-Menten rate laws. We are going to investigate the effect of varying the activity of v_1 on the steady-state concentrations of S2 and see how this response is effected by the values of enzyme Kms. We can change the activity of v_1 by changing the Vmax.

Set the following parameter values for each reaction:

Reaction	Parameter	Value
v_1	V_m	0.1
	K_m	5.0
v_2	V_m	1.0
	K_m	5.0

Initialize the values of S_1 and S_2 to 3 concentration units each.

We are now going to use the parameter scan feature to investigate how the activity of the first v_m affects the steady-state concentration of S_2.

To make sure that you have entered everything correctly, compute the steady-state with the current values. You should get the following steady-state values for S_1 and S_2.

$S_1 = 5.7182$

$S_2 = 0.28179$

Once you can reproduce the above values we can begin to scan the parameter range. To do this, select the scan parameter tab on the steady-state control panel.

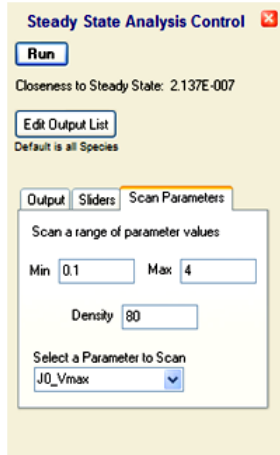


Figure 3.18 Steady-state Control Panel.

First thing to do is select the particular parameter we wish to scan, in this case it will be the first V_m, in this case J_0 V_m. Note, you may have named it something else so make sure the correct one is selected.

Next we must set the Min, Max and Density parameters. Set these to 0.1, 4.0 and 80 respectively. These indicate where to start the scan, when to finish and how many values to compute. Now run the simulation. You should get the following graph:

This graph shows a simple hyperbolic response as the V_m is increased, nothing particularly interesting. We will now see the effect of changing the two reaction Kms on this response.

Change both reaction Kms from 5.0 to 2.0. Run the simulation at these new values. You should get the following graph:

The response has changed from a simple hyperbolic to a sigmoid response. Decrease the Kms further to 0.25.

and further to 0.05. Also reduce the Max value from 4 to 2 in order to rescale the graph. As the Kms fall, the sigmoid character becomes

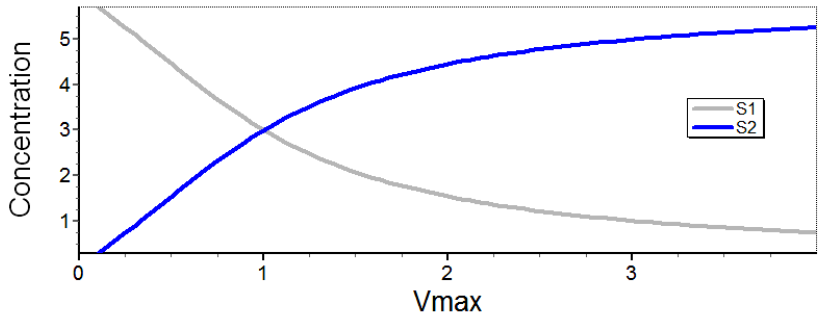


Figure 3.19 Simulation carried out with high Km = 5.0, note change hyperbolic response.

more and more pronounced. In fact, it becomes almost switch like. This property is well known and is called ultrasensitivity. The property enables the network to convert a graded input into an on/off response.

3.4 Tutorial 4 – Modeling True Bistable Systems

Bistable switches are commonly found as network motifs in prokaryotes and in sometimes in eukaryotes. They represent the biochemical equivalent of a light toggle switch. In this tutorial, we will introduce a bistable model and illustrate its unique switching properties as well as introducing the notion of free-format rate laws.

Enter the following model shown in Figure 3.23 into JDesigner (see below for details)

This model illustrates a simple regulatory loop from S_1 to v_1. The regulation is positive in the sense that when the concentration of S_1 increases, the rate through v_1 increases. The problem however

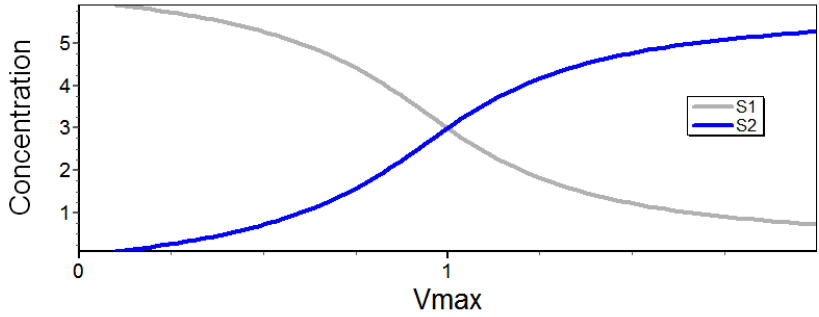


Figure 3.20 Simulation carried out with Km = 2.0, note change to sigmoid response.

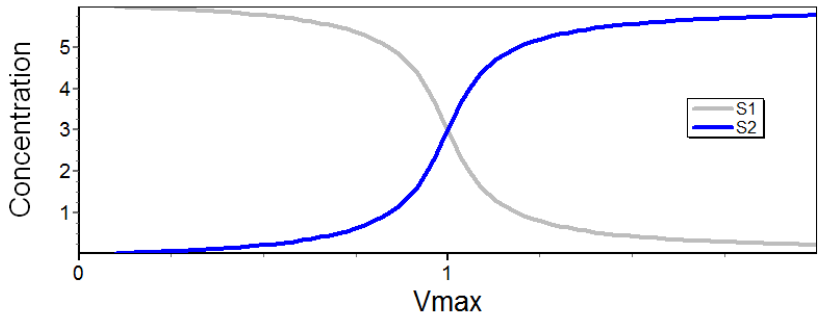


Figure 3.21 Simulation carried out with Km = 0.25.

is that there is not a built-in rate law with this property. Instead, JDesigner permits one to enter a 'free-format' rate law.

To build this model, follow the steps below:

1. Drop down three species nodes and rename them X0, S1, and X1
2. Set X0 and X1 as boundary species.
3. Add uni-uni reactions between

   X0 and S1

   S1 and X1.

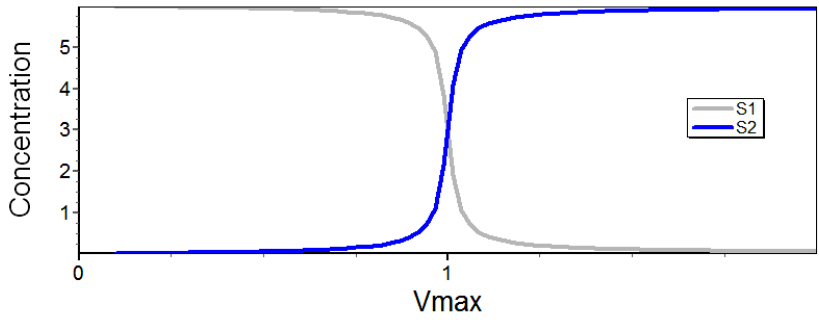


Figure 3.22 Simulation carried out with Km = 0.05.

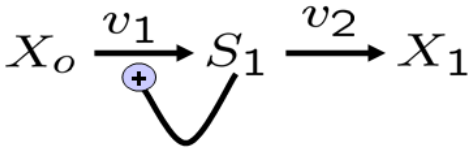


Figure 3.23 Simple network that exhibits bistability.

You should have the following on your screen:

Right-click over the first reaction to bring up the reaction properties panel.

5. Select the tab marked Free-Format

You should see the following:

At the edit box containing `J0_k*X0`, type in the following expression:

```
10+X0*3.2*((S1/0.75)^3.2)/(1+((S1/4.3)^3.2))
```

6. Hit the tab key, this moves the focus and ensures that the equation has been entered. You should see the following on your screen.

7. Next select the second reaction, this time choose an irreversible mass-action rate law with a rate constant of 12.0

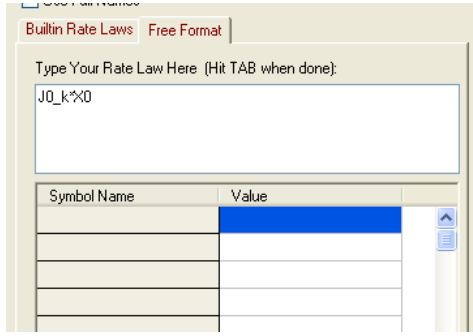


Figure 3.24 Free-Format Rate Law entry.

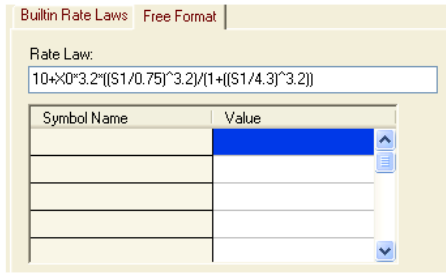


Figure 3.25 Setting the rate law for the first reaction using the free format panel.

8. Set the concentration of X0 to 0.12 units

9. Set the initial concentration of S1 to 0.1

Next close the reaction properties panel and open the time course control panel plus a graph panel .

Set the time end to 1 time unit and run a simulation. If you have correctly entered the model, you should see a graph as shown in Figure 3.26.

Notice that the system approaches a steady-state with a value of roughly 1.0, nothing out of the ordinary there. Let us now change the initial concentration of S1 to 4.0 and carry out the same simula-

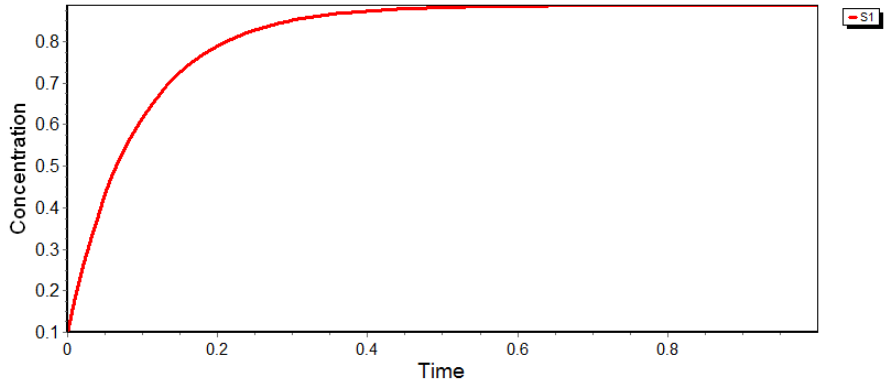


Figure 3.26 Evolution to bistable system towards the lower stable steady-state.

tion.

You will also need to change the axis limits, right click over the Y-axis and set the lower limit to zero and the upper limit to 10, thus:

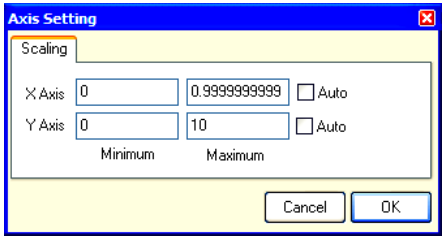


Figure 3.27 Setting the Y-axis limits – obtained by right-clicking on a graph axis.

If you run the simulation (do not click on reset) you should observe the graph shown in Figure 3.28.

What is unusual is that the model has clearly settled to a different steady-state, a high steady-state, compared to the first run. You can

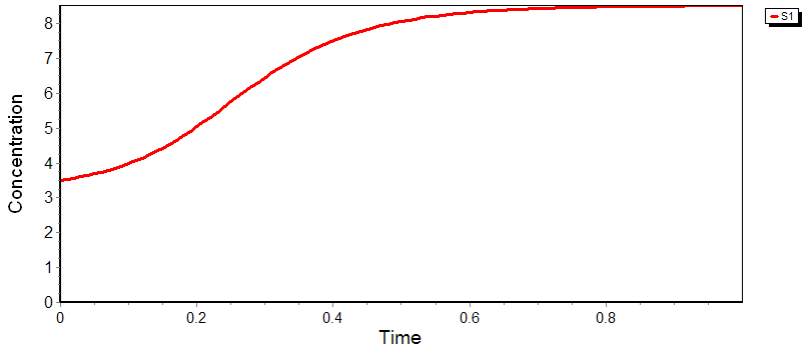


Figure 3.28 Evolution to bistable system towards the high stable steady-state.

return the initial condition of S_1 back to 0.1 and run it again and you will see the system settles to the lower steady-state. This model appears to have two quite different steady-states for the same set of parameter values.

To explain this behavior, the graph shown in Figure 3.29 plots the two reaction rates against the concentration of S_1. Where the lines intersect is the point when the two rates are equal, these correspond to the steady-state points. Because of the particular nature of the positive feedback, the model admits three intersection points, corresponding to three unique steady-state solutions.

The middle intersection point is unstable, but the two outer ones are stable. Note that the lower stable state corresponds to a S_1 value of about 1.0, while the upper steady-state corresponds to a S_1 value of about 8.5. Both these values correspond to the values obtained from the simulations described previously.

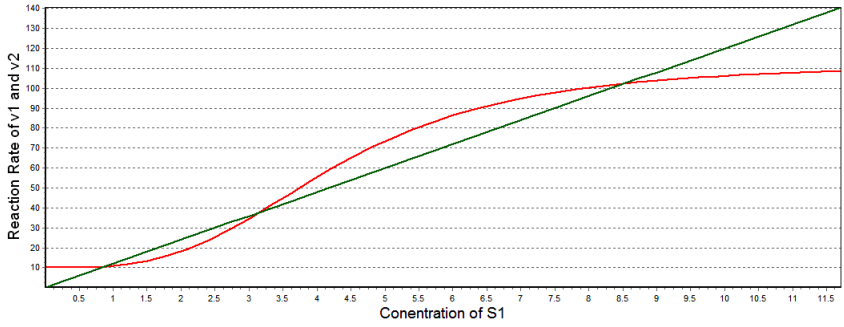


Figure 3.29 Plots of reactions rates, v_1 and v_2 versus S_1. Intersection points show steady-state points.

3.5 Tutorial 5 – Oscillators and Sliders

Circadian rhythms are common occurrences in biological systems, many of these rhythms are controlled by biochemical oscillators. In this tutorial we will investigate a well-known oscillator called the Brusselator. This oscillator is a highly simplified model of the famous Belousov-Zhabotinskii chemical oscillator. The reaction scheme is given below:

$$\begin{aligned} A &\rightarrow X \\ 2X + Y &\rightarrow 3X \\ B + X &\rightarrow Y + C \\ X &\rightarrow D \end{aligned}$$

One of the characteristic features of the scheme is the autocatalytic nature of the second reaction. That is, for every 2 X species consumed, 3 X species are produced. This reaction is therefore quite

unstable. However, as X increases exponentially, it also removes Y exponentially, the result being that at a critical point, the second reaction stops due to lack of Y and the concentration of X collapses. Y now starts to build again due to the third reaction, and in turn will eventually stimulate the autocatalytic reaction. This process of crash and boom continues indefinitely (or until the source material, A and B are used up in a real experiment). In this tutorial we are going to use the *interactive slider* capability to study the effect of the different rate constants on the oscillatory properties of this reaction network.

This model introduces a new concept – non-unity stoichiometry (see reaction two) – which we need to deal with.

The first step is to place the molecular species on the drawing area. These include:

Species Name	Fixed	Concentration
A	Yes	0.5
B	Yes	3.0
C	Yes	0.0
D	Yes	0.0
X	No	3.0
Y	No	3.0

The second step is to add the reactions, if you have done this correctly you should see something like the following. Note that your particular arrangement of reaction arcs might be different, use the mouse to rearrange the nodes and reaction arcs as desired.

Note that to add bimolecular reactions, select the bi-bi reaction type from the left-side toolbar. Use the mouse to select in turn, the two substrates and the two products.

The next step is to change two stoichiometries. Select the second reaction where X is autocatalytically transformed. This should bring

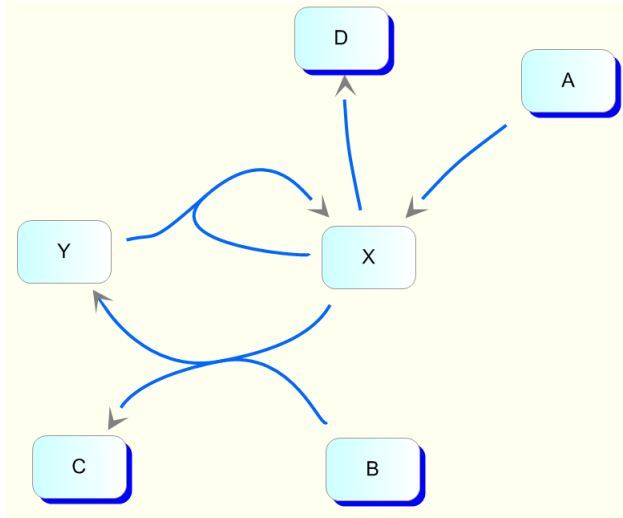


Figure 3.30 Brusselator as Drawn using JDesigner.

up the following reaction properties panel. Locate the stoichiometry settings and adjust the first X stoichiometry to be 2 and the second X stoichiometry to be 3.

The next thing is to change the rate law. The rate law for this reaction is given by:

$$k_2 X X Y$$

Make sure you include the rate constant and set its value to one, use the free-format entry to enter this rate law.

For the other reactions, use the following rate laws:

Reaction	Rate Law	Value of Rate Constant
$A \rightarrow X$	$k_1 A$	1
$2X + Y \rightarrow 3X$	$k_2 X X Y$	1
$B + X \rightarrow Y + C$	$k_3 B X$	1
$X \rightarrow D$	$k_4 X$	1

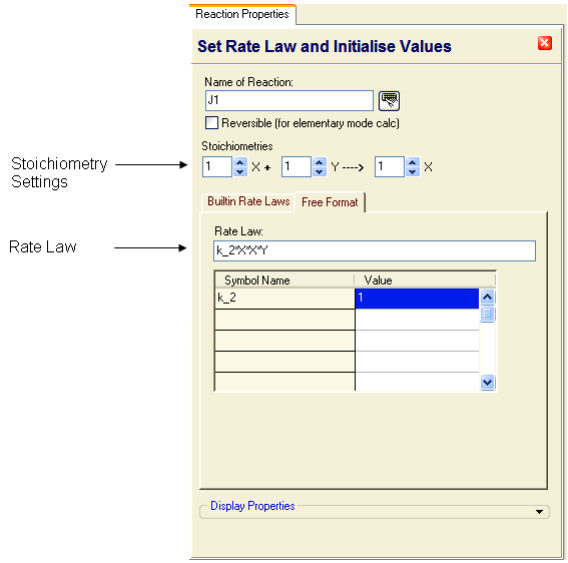


Figure 3.31 Setting Stoichiometries and Rate Laws.

Once the model is entered, bring up the time course simulation panel and the graph panel.

Set the time end to a value of 100 time units, and carry out a simulation. You should observe the following oscillatory output.

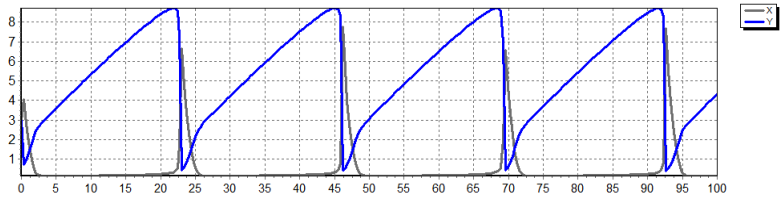


Figure 3.32 Simulation of the Brusselator Reaction Model.

We are now going to investigate how the different rate constants affect the dynamics of this model.

At the lower end of the time-course panel you should see two buttons, one marked Define and the other marked Display . Click on the Define button and you will be presented with a window that permits you to attach slider controls to parameters in the model. We will select all parameters. To select a parameter, select the parameter and click the Add button. Do this for each parameter until the define slider window looks like the display shown in Figure 3.33. If this is the case, click on the close button.

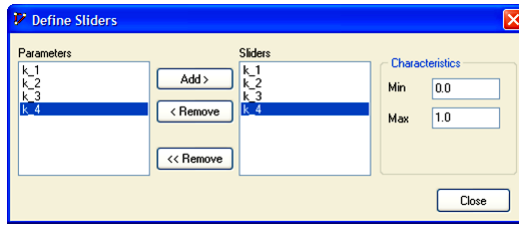


Figure 3.33 Attach Sliders to Parameters.

To display the sliders, click on the button marked Display, this will bring up the slider window (see Figure 3.33). For convenience you may dock this window onto the time-course panel (see Figure 3.34). You will now find that by **gently** moving the sliders the simulation curves will change. If you move the sliders out of range you may find the differential equation solver complaining about numerical difficulties. If this happens try to move the slider back to it original position, it may take some tries to accomplish this.

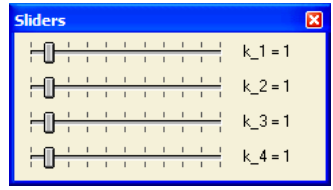


Figure 3.34 Slider Window

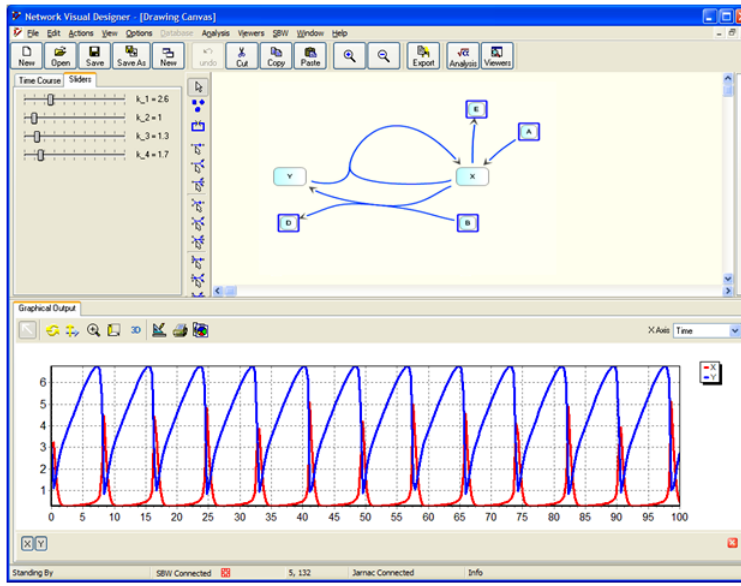


Figure 3.35 Slider Panel Docked with Time Course Panel.

4
Jarnac Tutorials

Jarnac is a script based simulation application. Biochemical models are described by specifying reaction schemes in the form of text. In the following tutorials we repeat the earlier tutorials based on JDesigner but using Jarnac instead.

4.1 Tutorial 1

4.1.1 Basic Pathway Model

Let us begin by building a simple network. This network will comprise of four molecular species and three reactions. First, we will define our network object, species, reactions, and corresponding rate laws. The pathway, p, is the variable that represents the model. Our species can take on any name, such as glucose or ATP but here they

are named Node0, Node1, Node2, and Node3.

```
// Define Pathway
p = defn Pathway

     // Declare species
     var Node0, Node1, Node2, Node3;

     // Define each reaction (J0,J1,J2) and rate law
     // Here, we use irreversible mass-action rate laws
     J0: Node0 -> Node1; k0*Node0;
     J1: Node1 -> Node2; k1*Node1;
     J2: Node2 -> Node3; k2*Node2;
end;
```

Next, we will set the initial concentrations for our species and values for the rate constants. Node1 will start with 10 concentration units and the other species with 0. Note we have named the reactions J_0, J_1, and J_2. The rate constants will be 0.6, 1.0, and 0.15 for reactions J_0, J_1, and J_2, respectively.

```
// Initialize the parameters and variables
p.Node0 = 10; p.Node1 = 0; p.Node2 = 0;
p.Node3 = 0; p.k0 = 0.6; p.k1 = 1.0;
p.k2 = 0.15;
```

To run a time course simulation, we create a matrix using the method sim.eval and pass it to the graph function:

```
// Perform simulation
// First argument Time Start
// Second argument Time End
// Third argument Number Of Points
m = p.sim.eval(0, 10, 100);
graph(m);
```

The arguments in the eval method correspond to the time start, time end and the number of points to generate. Running this script will create the following graph:

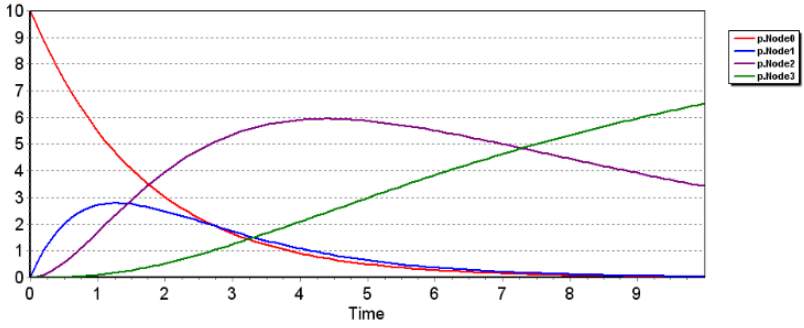


Figure 4.1 Jarnac Generated Graph.

The graph can be copied by clicking the copy to clipboard button and the axes and other graph properties using the Edit Chart button. To obtain the raw data, simply add the line

```
print m;
```

which displays the matrix we created in a tabular format. The eval method can also take an optional 4th argument which lists the entries in the columns of the returned matrix. Without the 4th argument, the default is to return the first column as time and the remaining column as the floating species in the model. the script below shows that we have selected a different set of columns, note we can include simple expressions such as `10*p.Node1`.

```
// Perform simulation
m = p.sim.eval(0, 10, 100,
   [<p.Time>, <10*p.Node1>, <p.Node2+Node3>]);
graph(m);
```

4.1.2 Setting Reaction Rate Laws

We are now going to make a change to the model by setting different rate laws for the three reactions. In the earlier version, all the rate laws were simple irreversible mass-action rate laws. For some models this may not be realistic, instead we will use in the next version, reversible Michaelis-Menten rate laws.

We have the option of typing in our own free-format rate laws or we can take advantage of Jarnac's built-in rate laws:

```
uui (S1, Vm, Km)                                    Uni-Uni Irreversible
uur (S1, S2, Vm, Km1, Km2, Keq)                     Uni-Uni Reversible
bbi (S1, S2, Vm, Km1, Km2, Kia)                     Bi-Bi Irreversible
hill (S, Vm, K, h)                                  Hill Equation
unibi (A, P, Q, Vmf, Vmr, Kma, Kmp, Kmq, Kip, Keq)  Uni-Bi Reversible
biuni (A, B, P, Vmf, Vmr, Kma, Kmb, Kmp, Kia, Keq)  Bi-Uni Reversible
```

The built-in function `uur` evaluates the single substrate reversible Michaelis-Menten model, given by the equation:

$$v = \frac{V_m/K_{m_1}(S - P/K_{eq})}{1 + S/K_{m_1} + P/K_{m_2}}$$

The arguments to uur must be in the order: `uur(S,P,Vm1,Km1,Km2,Keq);`

Rewriting our network from 1.1, we create the following:

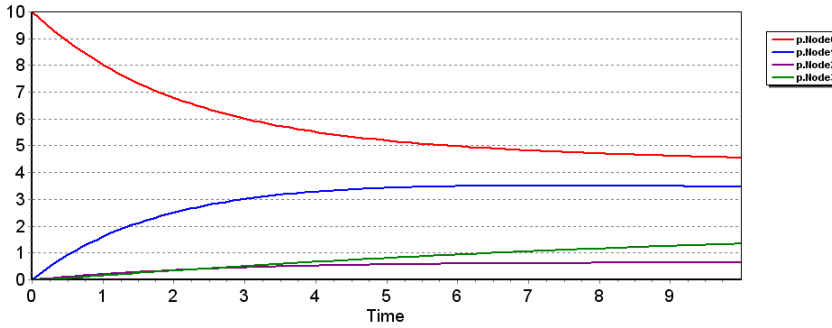
```
// Define Pathway
p = defn Pathway
   // Declare species
   var Node0, Node1, Node2, Node3;

   // Define rxn and rate law (reversible Michaelis-Menten)
   J0: Node0 -> Node1;
       uur(Node0, Node1, J0_Vm, J0_Km1, J0_Km2, J0_Keq);
   J1: Node1 -> Node2;
       uur(Node1, Node2, J1_Vm, J1_Km1, J1_Km2, J1_Keq);
```

```
   J2: Node2 -> Node3;
       uur(Node2, Node3, J2_Vm, J2_Km1, J2_Km2, J2_Keq);
end;
```

Initializing our parameters and variables we have:

```
p.Node0 = 10; p.Node1 = 0; p.Node2 = 0; p.Node3 = 0;
p.J0_Vm = 2.5; p.J0_Km1 = 0.1; p.J0_Km2 = 0.1; p.J0_Keq = 0.8;
p.J1_Vm = 1.2; p.J1_Km1 = 0.1; p.J1_Km2 = 0.1; p.J1_Keq = 0.2;
p.J2\_Vm = 0.5; p.J2_Km1 = 0.1; p.J2_Km2 = 0.1; p.J2_Keq = 4.8;
```

If we run this updated script we should get the following graph:



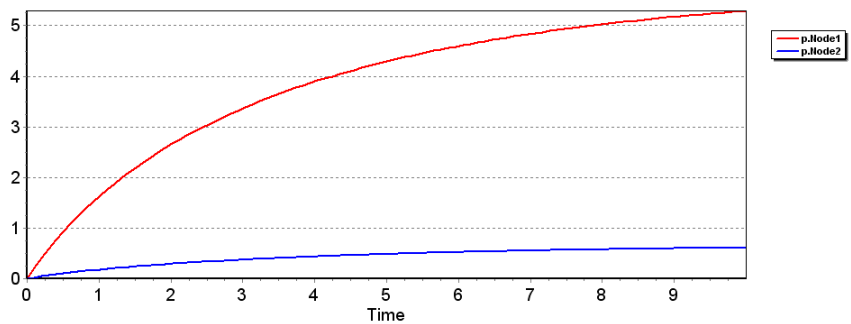
4.1.3 Setting Boundary Conditions

In the previous model all the molecular species, node0 to node3, were free to change during the simulation, the system was a closed system. Biological systems are however open, that is, there is a net and steady flow of mass from one model boundary to another. In a real system, the system boundaries are usually fixed and as a result the system will reach a steady state rather than equilibrium. Let us now change the previous model by fixing Node0 and Node3, rerun the simulation, and compare it to the original version. To set Node0

and Node3 as boundary species, we declare them as `ext` variables and put a $ symbol in front of the species in the reaction declaration.

```
// Define Pathway
p = defn Pathway
   // Declare species
   ext Node0, Node3; // create boundary nodes
   var Node1, Node2; // create floating nodes
   // boundary nodes are denoted by a $ preceding the node
   J0: $Node0 -> Node1;
       uur(Node0, Node1, J0_Vm, J0_Km1, J0_Km2, J0_Keq);
   J1: Node1 -$>$ Node2;
        uur(Node1, Node2, J1_Vm, J1_Km1, J1_Km2, J\_Keq);
   J2: Node2 -$>$ $Node3;
       uur(Node2, Node3, J2_Vm, J2x
       Km1, J2_Km2, J2_Keq);
end
// perform simulation, this time giving data
// for only Node1 and Node2
m = p.sim.eval(0, 10, 100,
     [<p.Time>, <p.Node1>,<p.Node2>]);
graph(m);
```

The graph should look like this:



Notice this time, the system approaches a steady state.

4.1.4 Steady State Simulations

What if we wanted to compute the steady-state immediately rather than using a time course simulation? We can remove the time course simulation code and instead run a steady state Simulation using the p.ss.eval method and print the final concentrations of our floating species:

```
// run Steady State Simulation and print data
p.ss.eval;
println "Node1 =", p.Node1, "Node2 =", p.Node2;
```

The output should look like this:

```
Node1 =  5.78434, Node2 = 0.679977
// Another way to print this data is to use the console
// and enter the method which returns the names of
// the floating species and the method which returns
// concentrations of the species:
->p.fs
Node1, Node2
->p.sv
5.784,     0.68
->
```

4.2 Tutorial 2 - Modeling a Simple Branch

In this tutorial, we will study some of the basic properties of a simple metabolic branch and introduce the notion of a **pathway flux**. Enter the following model into a Jarnac script, where Xo, X_1 and X_2 are boundary species and S_1 is an internal species.

Your script should look like this:

```
// Define Pathway
```

```
p = defn Pathway

     // Declare species
     ext X0, X1, X2;
     var S1;

     // Define each reaction and rate law
     // (here a reversible Michaelis-Menten)
     v1: $X0 -> S1; uur(X0, S1, v1_Vm, v1_Km1, v1_Km2, v1_Keq);
     v2: S1 -> $X1; uur(S1, X1, v2_Vm, v2_Km1, v2_Km2, v2_Keq);
     v3: S1 -> $X2; uur(S1, X2, v3_Vm, v3_Km1, v3_Km2, v3_Keq);
end;
```

Parameter	Value
V_{m_1}	1.5
V_{m_2}	2.0
V_{m_3}	4.0

Set the initial concentration for X_o to be 10 concentration units.
All other concentrations leave at zero:

```
// Initialize the parameters and variables
p.X0 = 10; p.X1 = 0; p.X2 = 0; p.S1 = 0;
p.v1_Vm = 1.5; p.v1_Km1 = 0.1; p.v1_Km2 = 0.1; p.v1_Keq = 0.1;
p.v2_Vm = 2.0; p.v2_Km1 = 0.1; p.v2_Km2 = 0.1; p.v2_Keq = 0.1;
p.v3_Vm = 4.0; p.v3_Km1 = 0.1; p.v3_Km2 = 0.1; p.v3_Keq = 0.1;
```

To make sure your script is running correctly, calculate the steady state and print your results for the steady state concentration of S_1:

```
// run steady state simulation
p.ss.eval;

println "S1 =", p.S1;
```

The output should be:

```
S1 = 0.0314073
```

We are now going to introduce the notion of a pathway flux. At any time during the evolution of a pathway, the rates through the different reactions as well as the species concentrations will evolve in time, eventually settling to some steady-state values. The reaction rates that we measure during a simulation are termed fluxes.

To display this data (calculated by ss.eval) we can add the following lines which will print the flux information:

```
println "Flux v1 =", p.v1;
println "Flux v2 =", p.v2;
println "Flux v3 =", p.v3;
```

The output should be:

```
S1 = 0.0314073
Flux v1 = 1.43404
Flux v2 = 0.478015
Flux v3 = 0.95603
```

Another way to print this data is to use the console and enter the rs method which returns the names of the reactions and the rv method which returns the reaction rates (fluxes):

```
->p.rs
["v1", "v2", "v3"]
->p.rv
{   1.434,    0.478,    0.956 }
->
```

4.3 Tutorial 3 - Modeling a Simple Cycle

In this tutorial, you will investigate the basic properties of a simple cycle and introduce the idea of parameter scans. Such cycles are very common in signaling networks and thus an understanding of their properties is important. The network we will be model is the same as shown below:

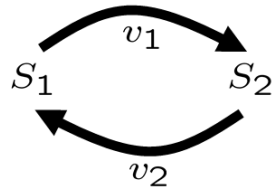


Figure 4.2 Cycle Network

The differential equations for this model are easily written:

$$\frac{dS_1}{dt} = v_2 - v_1$$
$$\frac{dS_2}{dt} = v_1 - v_2$$

Note that this network has as an important property, namely that

$$\frac{dS_1}{dt} = -\frac{dS_2}{dt}$$

This means that at all times:

$$S_1 + S_2 = \text{ constant}$$

In biological terms, the forward arm, v_1 might be catalyzed by a kinase and the reverse arm, v_2, a phosphatase.

Set both reactions to have irreversible Michaelis-Menten rate laws. We are going to investigate the effect of varying the activity of v_1 on the steady-state concentrations of S2 and see how this response is effected by the values of enzyme Kms. We can change the activity of v_1 by changing the Vmax.

Initialize the values of S_1 and S_2 to 3 concentration units each. In biological terms, the forward arm, v_1 might be catalyzed by a kinase and the reverse arm, v_2, a phosphatase.

We will set both reactions to have irreversible Michaelis-Menten rate laws. We are going to investigate the effect of varying the activity of v_1 on the steady-state concentrations of S2 and see how this response is effected by the values of enzyme Kms. We can change the activity of v_1 by changing the Vmax.

Set the following parameter values for each reaction:

Reaction	Parameter	Value
v_1	V_m	0.1
	K_m	5.0
v_2	V_m	1.0
	K_m	5.0

Initialize the values of S_1 and S_2 to 3 concentration units each. Your script up to this point should look like this:

```
// Define Pathway
p = defn Pathway

     // Declare species
     var S1, S2;
     // Define each reaction and rate law
     // (here, reversible Michaelis-Menten)
```

```
    v1: S1 -$>$ S2; uui(S1, v1_Vm, v1_Km);
    v2: S2 -$>$ S1; uui(S2, v2_Vm, v2_Km);
end;
// Initialize the parameters and variables
p.S1 = 3; p.S2 = 3;
p.v1_Vm = 0.1; p.v1_Km = 5.0;
p.v2_Vm = 1.0; p.v2_Km = 5.0;
```

We are now going to use the parameter scan feature to investigate how the activity of the first Vmax affects the steady-state concentration of S_2.

To make sure that you have entered everything correctly, compute the steady-state with the current values. You should get the following steady-state values for S_1 and S_2.

$S_1 = 5.7182$

$S_2 = 0.28179$

Once you can reproduce the above values we can begin to scan the parameter range. The first thing to do is select the particular parameter we wish to scan, in this case it will be the first Vmax, `p.v1_Vm`.

Next we must set the Min, Max and Density parameters. Set these to `p.v1_Vm`, 4.0 and 80 respectively. These indicate where to start the scan, when to finish, and how many values to compute:

```
// Initialize Scan Parameters
ScanMin = p.v1_Vm;
ScanMax = 4.0;
ScanDensity = 80;
```

Next, we will create a matrix which will hold the data we will will generate as we change the values of Vmax. The function `m = matrix(i,j)` creates a matrix with i rows and j columns. We can access individual cells with the syntax `m[i,j]`. The function SetColumnName is useful for labeling our matrix and uses the syntax `SetColumnName(m, i, "ColName")`, where m is the matrix, i

is the column number, and "ColName" is the string name we wish to assign to the column.

```
// Create matrix with ScanDensity
// of rows and 3 columns

m = matrix(ScanDensity, 3);
SetColumnName (m, 1, "Vmax");
SetColumnName (m, 2, "S1");
SetColumnName (m, 3, "S2");
```

After creating the matrix, we will write a `for` loop which calculates the steady state at each different Vmax value, populating the matrix. Then we graph the matrix:

```
for i=1 to ScanDensity do
    begin
    // short simulation to avoid errors in steady-state
    p.sim.eval(0, 10, 10, []);
    p.ss.eval;
    // add new data to row i in vector form
    m[i] = {p.v1_Vm, p.S1, p.S2};
    // increment Vmax
    p.v1_Vm = p.v1_Vm + (ScanMax-ScanMin)/ScanDensity;
    end;
graph(m);
```

Now run the simulation. You should get the graph shown in Figure 4.3.

This graph shows a simple hyperbolic response as the Vmax is increased, nothing particularly interesting. We will now see the effect of changing the two reaction Kms on this response.

Change both reaction Kms from 5.0 to 0.25. Run the simulation at these new values. You should get the graph show in Figure **??**.

Reduce the Max value from 4 to 2 in order to rescale the graph and then decrease the Kms further to 0.25. As the Km falls, the

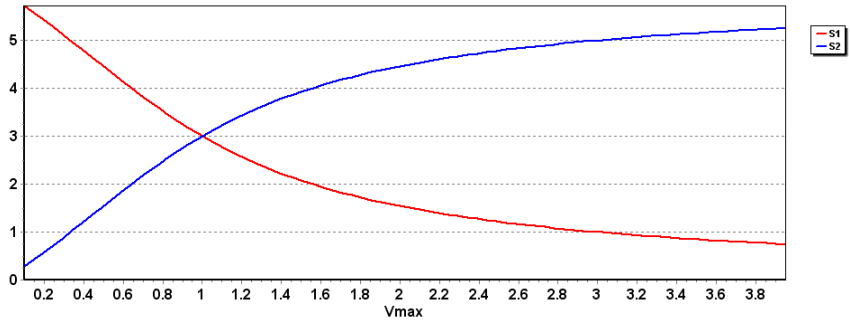


Figure 4.3 Cycle Network

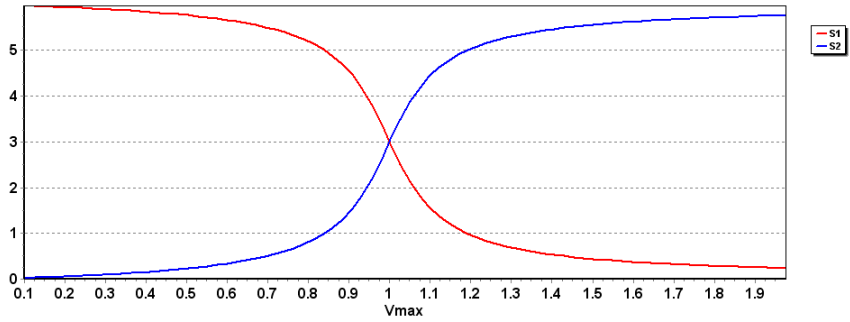


Figure 4.4 Cycle Network

hyperbolic character becomes sigmoidal. In fact, it becomes almost switch like is the K_m is low enough. This property is well known and is called ultrasensitivity. The property enables the network to convert a graded input into an on/off response:

4.4 Tutorial 4 - Modeling a Bistable System

Bistable switches are commonly found as network motifs in prokaryotes and in sometimes in eukaryotes. They represent the biochemical equivalent of a light toggle switch. In this tutorial, we will introduce a bistable model and illustrate its unique switching properties as well as introducing the notion of free-format rate laws.

We will be entering the model shown in Figure 4.5 into Jarnac (see below for details).

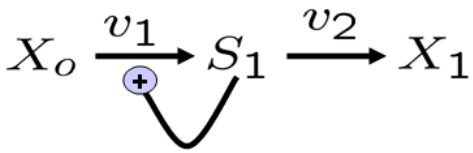


Figure 4.5 Simple network that exhibits bistability.

This model illustrates a simple regulatory loop from S1 to v_1. The regulation is positive in the sense that when the concentration of S1 increases, the rate through v_1. Since there is not a built-in rate law with this property, we will instead enter a 'free-format' rate law:

```
v1: $X0 -> S1;
     10+X0*3.2*((S1/0.75)^3.2)/(1+((S1/4.3)^3.2));
```

Our second reaction will be an irreversible mass action rate law:

```
v2:  S1 -> $X1; k0*S1;
```

Our variables will be initialized as follows:

```
p.X0 = 0.12; p.S1 = 0.1; p.X1 = 0;
p.k0 = 12;
```

We will then run a time course simulation. Our script should look like this:

```
// Define Pathway
p = defn Pathway

   // Declare species
   ext X0, X1;
   var S1;

   // Define each reaction and free-format rate law
   v1: $X0 -> S1; 10+X0*3.2*((S1/0.75)^3.2)/(1+((S1/4.3)^3.2));
   v2: S1 -> $X1; k0*S1;
end;

// Initialize the parameters and variables
p.X0 = 0.12; p.S1 = 0.1; p.X1 = 0;
p.k0 = 12;

// Define simulation variables
TimeStart = 0;
TimeEnd = 1;
NumDataPoints = 100;

// perform simulation
m = p.sim.eval(TimeStart, TimeEnd, NumDataPoints,
       [<p.Time>, <p.S1>]);
```

Figure 3.26 shows the low steady state and Figure 3.28 the high steady state.

We will now do something different and write a script to repeat a simulation multiple times. In this case we will carry out a time course simulation ten times, each time using a different initial value for S1. If the system is bistable we should see different trajectories depending on where the initial condition is.

```
p = defn Pathway
```

```
   // Declare species
   ext X0, X1;
   var S1;

   // Define each reaction and free-format rate law
   v1: $X0 -> S1; 10+X0*3.2*((S1/0.75)^3.2)/(1+((S1/4.3)^3.2));
   v2: S1 -> $X1; k0*S1;
end;

// Initialize the parameters and variables
p.X0 = 0.12; p.S1 = 0.1; p.X1 = 0;
p.k0 = 12;

// Define simulation variables
TimeStart = 0;
TimeEnd = 1;
NumDataPoints = 100;

// perform simulation
m = p.sim.eval(TimeStart, TimeEnd, NumDataPoints,
       [<p.Time>, <p.S1>]);

initialS1 = 0.2;
for i = 1 to 10 do
    begin
    p.S1 = initialS1;
    m1 = p.sim.eval(TimeStart, TimeEnd, NumDataPoints, [<p.S1>]);
    m = augc (m, m1);
    initialS1 = initialS1 + 0.5;
    end;
graph (m);
```

4.5 Tutorial 5 - Reaction Stoichiometry and Oscillators

Circadian rhythms are common occurrences in biological systems, many of these rhythms are controlled by biochemical oscillators.

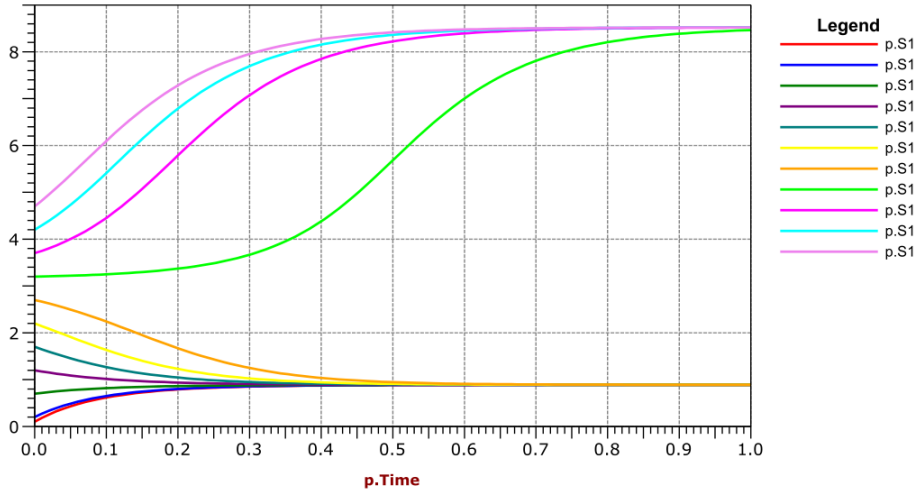


Figure 4.6 Simple network that exhibits bistability.

In this tutorial we will investigate a well-known oscillator called the Brusselator. This oscillator is a highly simplified model of the famous Belousov-Zhabotinskii chemical oscillator. The reaction scheme is given below:

One of the characteristic features of the scheme is the autocatalytic nature of the second reaction. That is, for every 2 X species consumed, 3 X species are produced. This reaction is therefore quite unstable. However, as X increases exponentially, it also removes Y exponentially, the result being that at a critical point, the second reaction stops due to lack of Y and the concentration of X collapses. Y now starts to build again due to the third reaction, and in turn will eventually stimulate the autocatalytic reaction. This process of crash and boom continues indefinitely (or until the source material, A and B are used up in a real experiment). In this tutorial we are going to study the effect of the different rate constants on the oscillatory properties of this reaction network.

This model introduces a new concept – non-unity stoichiometry (see

reaction two) – which we need to deal with.

The first step is to identify the species we will be dealing with. These include:

Species Name	Fixed	Concentration
A	Yes	0.5
B	Yes	3.0
C	Yes	0.0
D	Yes	0.0
X	No	3.0
Y	No	3.0

The second step is to identify the reactions. In this case, we have reactions with coefficients larger than one. We include this in the reaction definition by simply placing integer coefficients in front of the different species.

The reactions and rate laws we will be using in this example are listed below:

Reaction	Rate Law	Value of Rate Constant
$A \rightarrow X$	$k_1 A$	1
$2X + Y \rightarrow 3X$	$k_2 XXY$	1
$B + X \rightarrow Y + C$	$k_3 BX$	1
$X \rightarrow D$	$k_4 X$	1

When we add a time course simulation to 100 time units, our complete script should look like the following:

```
// Define Pathway
p = defn Pathway

     // Declare species
     ext A, B, C, D;
```

```
    var X, Y;

    // Define each reaction and rate law
    J1: A -> X; k_1*A;
    J2: 2X + Y -> 3X; k_2*X*X*Y;
    J3: B + X -> Y + C; k_3*B*X;
    J4: X -> D; k_4*X;
end;

// Initialize the parameters and variables
p.A = 0.5; p.B = 3.0; p.C = 0.0;
p.D = 0.0; p.X = 3.0; p.Y = 3.0;
p.k_1 = 1.0; p.k_2 = 1.0; p.k_3 = 1.0;
p.k_4 = 1.0;

// Define simulation variables
TimeStart = 0;
TimeEnd = 100;
NumDataPoints = 1000;

// Perform simulation
m = p.sim.eval(TimeStart, TimeEnd, NumDataPoints,
       [<p.Time>, <p.X>, <p.Y>]);
graph(m);
```

Figure 4.7 shows the graph produced by the script above

Changing the rate constants will give us different oscillating patterns. For example, setting $k_1 = 2.6, k_2 = 1, k_3 = 1.3,$ and $k_4 = 1.7$ produces the graph shown in Figure 4.8.

4.6 Tutorial 6 - Stochastic Simulation

Stochastic simulation is an important modeling approach given what we now know about the low number of molecules involved in some

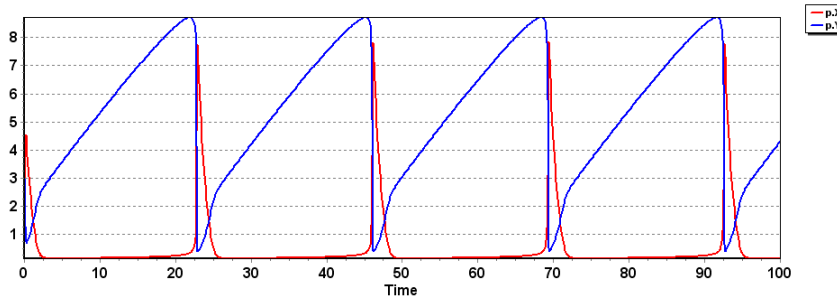


Figure 4.7 Brusselator Simulation Using Jarnac

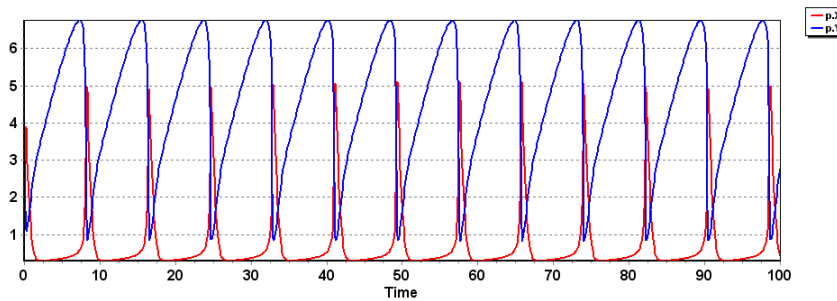


Figure 4.8 Brusselator Simulation Using Jarnac

biochemical processes. Jarnac offers a command `gillespie` that can be used to invoke a Gillespie based simulator. By way of an example the script below shows how a simple stochastic simulation can be carried out. Figure 4.10 shows the output from this simulation.

```
p = defn newModel
    $Xo -> S1; k1*Xo;
    S1 -> S2; k2*S1;
    S2 -> $X1; k3*S2;
end;
```

```
p.k1 = 0.2; p.k2 = 0.4; p.k3 = 2;
p.Xo = 50; p.S1 = 0; p.S2 = 0;

m = gillespie (p, 0, 30, [<p.time>, <p.S1>, <p.S2>]);
graph (m);
```

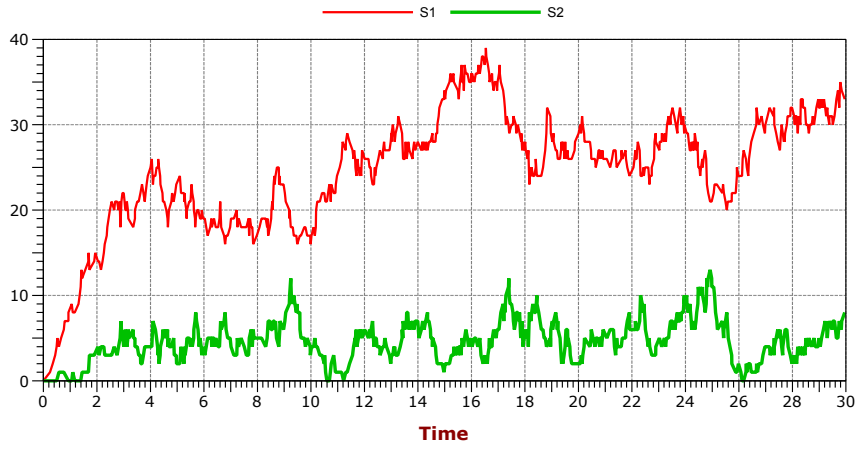


Figure 4.9 Stochastic Simulation

As in the previous simulations we can also invoke events during simulations. Figure 4.10 shows a simulation where at a time 30, the k_1 rate constant is reduced and the simulation continued. Note the use of `augr` to merge the two simulations.

```
p = defn newModel
    $Xo -> S1; k1*Xo;
    S1 -> S2; k2*S1;
    S2 -> $X1; k3*S2;
end;

p.k1 = 0.2; p.k2 = 0.4; p.k3 = 2;
```

```
p.Xo = 50; p.S1 = 0; p.S2 = 0;

// Simulate the first part up to 20 time units
m1 = gillespie (p, 0, 30, [<p.time>, <p.S1>, <p.S2>]);
p.k1 = p.k1 / 6;
m2 = gillespie (p, 30, 60, [<p.time>, <p.S1>, <p.S2>]);

graph (augr(m1,m2));
```

The `gillespie` call takes four arguments, the model variable, the time start, time end and a list of values to return to the caller. Note that the number of points is not given.

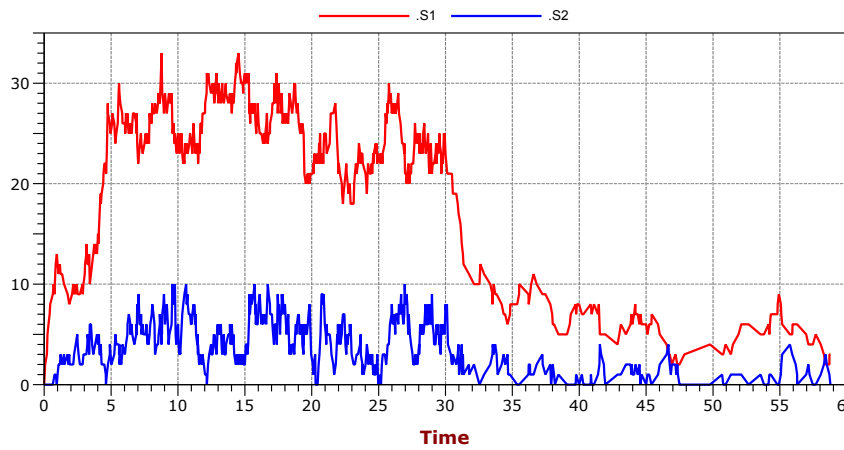


Figure 4.10 Using events in a stochastic simulation