Essentials for Biochemical Modeling

Herbert M. Sauro
University of Washington
Seattle, WA

PASI 2013 Edition

Ambrosius Publishing

First Edition, version 0.9
Published by Ambrosius Publishing and Future Skill Software
www.analogmachine.org

Typeset using $\LaTeX\ 2_\varepsilon$, TikZ, PGFPlots, WinEdt
and 11pt Math Time Professional 2 Fonts

ISBN 10: x-xxxx-xxxx-x (ebook)
ISBN 13: xxx-x-xxxxxxx-x-x (ebook)
ISBN 10: x-xxxxxxx-x-x (paperback)
ISBN 13: xxx-x-xxxxxxx-x-x (paperback)

Printed in the United States of America.

Mosaic image modified from Daniel Steger's Tikz image (http://www.texample.net/tikz/examples/mosaic-from-pompeii/

Front-Cover: Metabolic pathway image from JWS online (Jacky Snoep) with permission. The pathway depicts the glycolytic pathway from Lactococcus lactis using the Systems Biology Graphical Notation (SBGN). Ref: Hoefnagel, Hugenholtz and Snoep, 2002, Time dependent responses of glycolytic intermediates in a detailed glycolytic model of Lactococcus lactis during glucose run-out experiments. Mol. Biol. Reports 29, 157-161

Contents

Other topics:

1. More on bifurcation
2. Where to get data?
3. What rate laws to choose?
4. Preface, mention that we’re not going to talk about Boolean, petri nets etc.

Preface

This book is an introduction to modeling biochemical pathways. The book should be suitable for undergraduates in their early (Junior, USA, second year UK) to mid years at college. The book can also serve as a reference guide for researchers and teachers.

The latest edition together with free software and other material can be found at `www.analogmachine.org`, `www.sbw-app.org` and the research site, `www.sys-bio.org`.

Given the breadth of modeling biochemical systems I've had to be strict on what to include and what not to include. The text itself reflects somewhat by own bias in the field. I concentrate on two poplar and successful approaches to building, namely using differential equation and stochastic dynamics. I leave it to others to write but Boolean networks, Petri nets and the many other modeling approaches that exist. The other thing I do not talk about it the selection of rate laws and the limitations and assumptions of the various rate laws that one can use in building a biochemical models. Many of these details can be found in the companion book Enzyme Kinetics for Systems Biology. Lastly, I major omission in the book is spatial modeling, an area that is becoming increasingly important as researches turn to modeling larger systems such as tissues and organs.

As with my earlier text book on Enzyme Kinetics for Systems Biology I have decided to publish this book myself via a service called Createspace that is part of Amazon. Over the years I've had many offers from publishers to publish text books but have found the contracts they offer to be far too restrictive. Two restrictions in particular stand out, the loss of copyright on the text as well as any figures but more problematic in today's environment, the inability to rapidly update the text when either errors are found or new material needs to be added. With today's print on demand technology there is no reason for these restrictions.

There are many people and organizations who I should thank but foremost must be my infinitely patient wife, Holly, who has put up with the many hours I have spent working alone in our basement or long hours at the department and who contributed significantly to editing this book. I am also most grateful to the National Science Foundation and the National Institutes of Health who paid my summer salary so that I could allocate the time to write, edit and research. I would also like to thank the many undergraduates, graduates and colleagues who have directly or indirectly contributed to this work. In particular I want to thank my two teachers, David Fell and the late Henrik Kacser who I had the privilege to work with as a graduate student and postdoctoral fellow. I had many hours of fruitful and stimulating conversations with Jannie Hofmeyr, Athel Cornish-Bowden, Vijay Chickarmane, Jim Burns and last but not least Luis Azerenza. More recently I should thank my graduate students, in particular Frank Bergmann (author of SBW) who was (and is) a brilliant programmer, and Deepak Chandran (author of TinkerCell) who developed a very deep understanding of how networks operate. I thank them for they dedication and steadfast enthusiasm while they worked in my lab.

Thanks to the authors of the TEX system, MikTeX (2.9), TikZ (2.1), PGF-Plots (1.8.39) and WinEdt (6.0) for making available such amazing tools to technical authors.

Finally, I should thank Michael Corral (http://www.mecmath.net/) and Mike Hucka (sbml.org) whose LATEXwork inspired some of the visual styles I used in the text.

August 2013 HERBERT M. SAURO
Seattle, WA

1 Stoichiometric Networks

1.1 Prologue

Science fiction writers have for many years written stories about engineering life. Three in particular stuck a cord in my own mind, Frank Herbert's early work (1966), The Eyes of Heisenberg, describes a future where micromanipulation of cells at the molecular level is common place. Another author, Harry Harrison in his Eden trilogy (1984), describes an earth populated by intelligent dinosaurs who have complete mastery over genetic engineering and can manipulate living organisms at will. Finally Greg Bear writes a story called Blood Music (1985) that describes a renegade bioengineer who reengineers his own lymphocytes with intelligence which then begin to alter and "improve" his own genetic constitution with dire effects. These and many other stories, going all the way back to Shelley's Frankenstein, have predicted that one day the ability to control and change living systems at an unprecedented level would become a reality. Obviously such control could be used for good or bad and science fiction writers have recognized both aspects in their writings. There are considerable ethical issues at stake and many fear that such power could do great harm.

But what of the good? The ability to regenerate a severed spinal column, the replacement of disfiguring skin burns, or a final cure for one of the most feared diseases, cancer, could all eventually be resolved if we had a better understanding of living systems. One way is to be able to predict before hand what our interventions might lead to. A real understanding of the dynamic response of a cell in response to arbitrary perturbations would be of considerable help when looking for suitable targets for therapeutic intervention or the design or new abilities via synthetic biology.

If we were to administer a drug or cocktail of drugs, or change the expression of one or more genes, can we predict the outcome? At the moment not very well, and in fact the easiest way to find out what will happen is to actually do the experiment, potentially a long a expensive process. When the designers of space probes wish to land a robot on Mars, they don't first launch a rocket to see if it will work. If this were the approach it would probably involve many trials to get the trajectories right, an impossibly expensive approach. Instead, engineers use computer models to predict before hand how much fuel to burn and what planets to use for a sling shot. It is a remarkable feat of engineering and physics that space probes can be flown with considerable precision to their destination.

This leads to the question whether as biologists, we too could predict the outcome of interventions before they are even attempted? In recent decades there has been growing interest and progress in the possibility of building computer models of living organisms that can be used to make such predictions. For example we might build a computer simulation of a signaling pathway and use it to investigate possible targets for disrupting and/or control, or a synthetic biologist who might need to design a genetic network that can control when a cell will replicate or begin to produce a useful commodity. All these applications could be helped if a computer model were available.

This book will be concerned with introducing some of the main concepts and techniques in building and running computer models. What we won't cover is how biochemical systems operate or how they can be analyzed. Such topics will be reserved for a separate volume.

We will start by considering stoichiometric networks.

1.2 Stoichiometric Networks

Almost all cellular processes involve some kind of chemical process such as binding, unbinding or transformation, often in specific stoichiometric amounts. In addition, such processes have direction and show conservation of mass. We will classify cellular reactions as either elementary or non-elementary. Before continuing it is recommended that the reader reviewer Appendix A which describes basic kinetics and a variety of important terminology.

Elementary Reactions

Chemical reactions that involve no reaction intermediates, other than a single transition state, are called **elementary reactions**. Elementary reactions

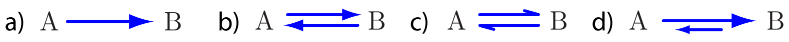


Figure 1.1 Simple Transformations. a) A single arrow, indicates positive rate direction. b) Two arrows showing explicit reversibility. c) Common barb style used to indicate reversibility. d) Reversibility with dominant arrow indicating positive direction.

have been depicted in a number of ways in the literature. For example, the transformation of one species into another can be represented by a simple line with an arrow at the tip. The direction of the arrow indicates the direction of the positive reaction rate (Figure 1.1). For example, if a reaction rate is -0.75 mol l^{-1}, this means that the reaction proceeds in the opposite direction indicated by the arrow at a rate of 0.75 mol l^{-1}. As this example implies, a single arrow head does not necessarily indicate that the reaction is irreversible. Such a determination should be indicated by author of the diagram. Often however reversibility is explicitly indicted by using multiple arrows. These come in various forms. One approach is to use two lines and add arrow heads to both the reactant and product line as shown in Figure 1.1b. Other authors add a smaller reverse arrow as shown in Figure 1.1d or more commonly use a barbed style as shown in Figure 1.1c.

In example (c) and (d) it is not possible however to obtain the positive direction of the rate.

For a bimolecular reaction that depicts dissociation or association the notation is shown in Figure 1.2 (a) and (b). This style makes it clear that

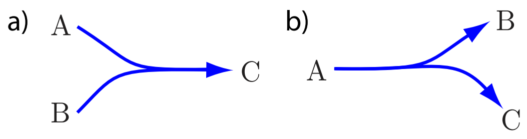


Figure 1.2 Dissociation and Association Reactions. (a) Equal stoichiometric proportions of compounds A and B combine to form a complex C. (b) Likewise, complex A dissociates into equal proportions of B and C.

there is a stoichiometric constraint between A and B and B and C. For example, that one molecule of A reacts with one molecule of B to form one molecule of C.

The simple association and dissociation reactions can be naturally extended to depict the situation where both association and dissociation occur in the same reaction as show in Figure 1.3.

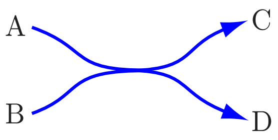
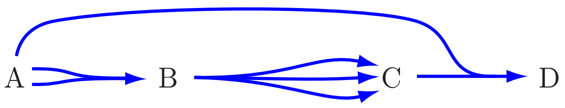


Figure 1.3 A bimolecular interaction, coupling one process, A to C to another B to D. Equal proportions of A and B combine to form equal proportions of C and D.

One area that is frequently problematic is visually depicting reactions with non-unity stoichiometry. The previous examples assume that each molecular species has a stoichiometry of one. However, what if species A in Figure 1.2 has a stoichiometry of 2 and B a stoichiometry of 3, how should these be represented. Figure 1.3 shows three depictions that have been in used by authors in the past. Sometimes simple arc extensions may be used to indicate the stoichiometry, Figure 1.3a. A variation of (a) is to use small barbs at the tips of the reaction arcs [21] where the number of barbs indicates the stoichiometry, Figure 1.3b. Finally stoichiometric numbers may be placed near the tips of the arcs Figure 1.3c. The use of numbers to indicate stoichiometry permits the display of fractional stoichiometries.

Example 1.1

In the following network made from elementary reactions, write out the individual reactions, taking care to indicate the correct stoichiometries.



Answer:

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

Non-Elementary Reactions

Non-elementary reactions include all reactions that have hidden reaction intermediates. The most common is the enzymatic reaction where the enzyme-substrate complex and free enzyme are rarely shown in network diagrams. The effect of hiding intermediates is that it is now possible to include regulatory links. For example, an enzyme may be regulated by an allosteric effector where the mechanism might be quite complex. Very often, this mechanism will be hidden and instead the action of the effector will be represented by a simple regulatory line. For example, if an enzyme that catalyzes the conversion of species S_1 to S_2 is inhibited by an effector molecule, M, then we will often depict this circumstance as shown in Figure 1.4.

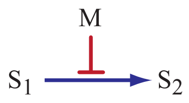


Figure 1.4 Modifier, M, inhibiting a non-elementary reaction such as a reaction catalyzed by an allosteric enzyme.

In hiding detailed mechanisms we also, when converting the diagrams to a mathematical model, invoke certain assumptions. In the case of a simple enzyme mechanism, we will often assume rapid-equilibrium or steady state of the enzyme substrate complex. Sometimes these assumptions are reasonable other times they are not. A discussion of this will be left to another volume.

The key thing to remember is that whenever one sees a regulatory link in a reaction step it always means that the reaction is non-elementary and hides other mechanistic details. The use of non-elementary reactions is a high level representation because exploding every non-elementary reaction into the full set of elementary reactions would make the network much more complex to view. Figure 1.5 illustrates an example of a relatively simple pathway drawn using non-elementary reactions together with a feedback inhibition step and the equivalent exploded view of the same system. The exploded view is clearly more complex. The mechanism chosen for the inhibition is the simplest possible and therefore the exploded view could be much more complex.

Text Representation

Simple text representations of reaction networks have been used for many years. Text representations are particulary easy to implement for computer consumption of network information. For example, a linear chain of four reactions can represented as follows:

Sometimes if a species is converted to a waste product, such as degradation fragments, them often the symbol, Ø will be used to represent the empty species set, for example:

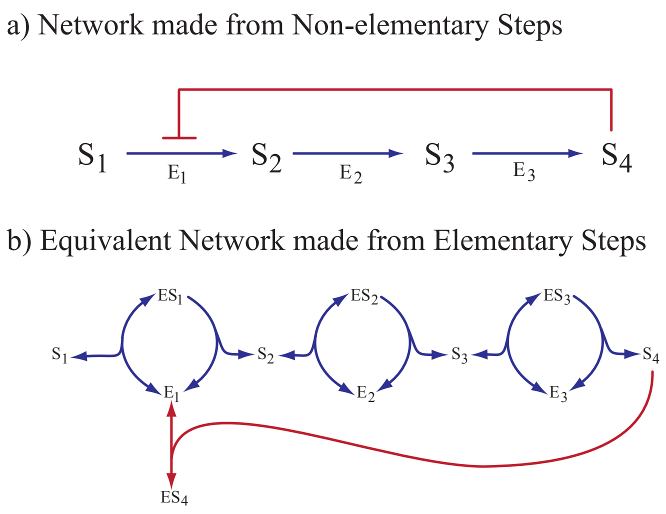

a) Network made from Non-elementary Steps
S1
E1
S2
E2
S3
E3
S4
b) Equivalent Network made from Elementary Steps
ES1
ES2
ES3
S1
S2
S3
S4
E1
E2
E3
ES4

Figure 1.5 Equivalent networks made from non-elementary and elementary components.

$$A + B \text{ -> } 2\,C$$
$$B \text{ -> } \emptyset$$

1.3 Standard Visualization Notation

Cellular networks have been depicted on wall charts for many decades using an informal notation. With the increased interest in recent years in protein and gene regulatory networks the variety of notations has also proliferated. As a result there have been some efforts, must notably the Systems Biology Graphical Notation (SBGN), to define a standard set of node and edge symbols to represent stoichiometric networks. In the next few section we will briefly look at the visual representation of metabolic networks using SBGN. For gene regulatory networks a notation similar to

$$
\begin{aligned}
A &\text{ -> } B\\
B &\text{ -> } C\\
2\,C + D &\text{ -> } 4\;E\\
E &\text{ -> } 2\;F
\end{aligned}
$$

Figure 1.6 Simple textual representation of a linear chain of four reactions and five molecular species.

that used by biotapestry [63] will be used because it is concise and easy to read.

Enzyme Catalyzed Reactions

SBGN can represent enzyme catalyzed reaction using the SBGN process description notation. For example, Figure 1.7 illustrates the SBGN approach to representing an enzyme catalyzed reaction. Round nodes represent small molecules such as DHAP, ATP and F6P. Rounded rectangles are used to represent macromolecules, in this case enzymes TPase (Triose phosphate Isomerase) and PFK (phosphofructokinase). In the second reaction, ATP also negatively regulates the reaction.

1.4 Mass-Balance Equations

Consider a simple network comprising two reactions, v_1 and v_2, with a common species, S. We will assume that the first reaction, v_1 produces S and the second reaction, v_2, consumes S (Figure 1.8).

According to the law of conservation of mass, any observed change in the amount of species, S must be due to the difference between the inward rate, v_1 and outward rate, v_2. That is, the change in S will be given by the

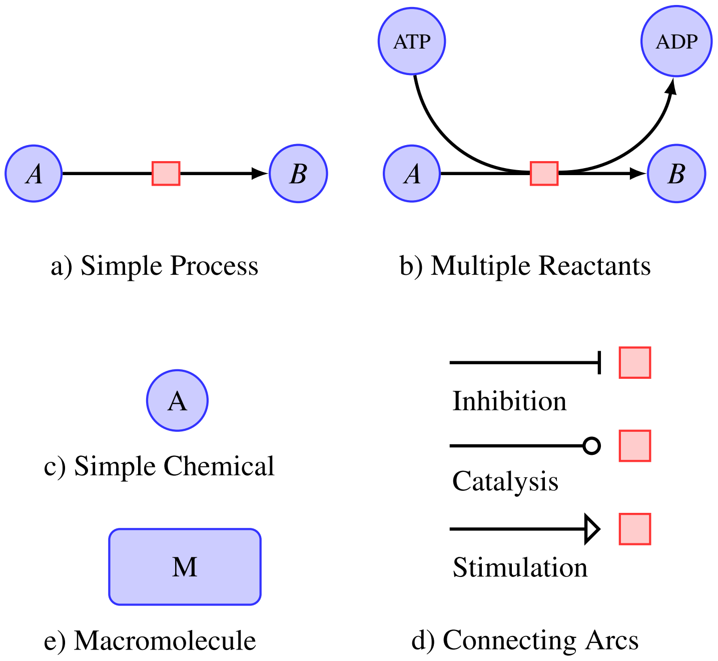


Table 1.1 Basic Symbols used in SBGN

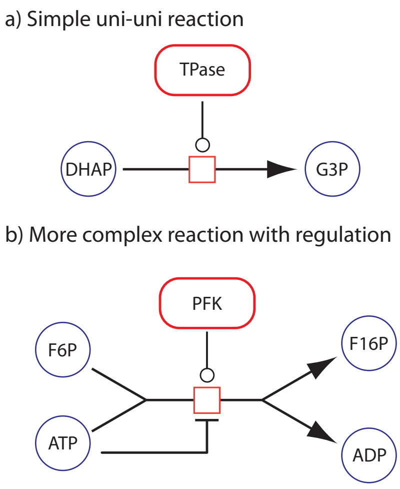

a) Simple uni-uni reaction
TPase
DHAP
G3P
b) More complex reaction with regulation
PFK
F6P
ATP
F16P
ADP

Figure 1.7 SBGN notation for enzyme catalyzed reactions.

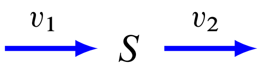


Figure 1.8 Simple Two Step Pathway.

difference in the two rates, leading to the differential equation:

$$\frac{dS}{dt} = v_1 - v_2. \tag{1.1}$$

The above equation is called a **mass-balance equation**. Often S will be expressed in concentration (mol l^{-1}) but it is mass that is conserved not concentration. We can reexpress equation 1.1 as:

$$\frac{dS_a}{dt}\frac{1}{V} = v_1 - v_2$$

where S_a is the amount of S in moles and V is the volume. Alternatively we can write:

$$\frac{dS_a}{dt} = V(v_1 - v_2)$$

This assumes that the reaction rates are expressed in mol l^{-1} t^{-1}. Often models assume a constant unit volume so that numerically:

$$\frac{dS}{dt} = \frac{dS_a}{dt}$$

and this will be the case is all the examples in this chapter. Although we will write the rate of change in terms of concentration, it is implied that we are dealing with a constant unit volume so that the change in concentration is the same as the change in amount. If movement is from one compartment to a compartment with a different volume then it is necessary to factor in the volume difference and express the rate of change in amounts.

For more complex systems such as the one shown in Figure 1.9 where there are multiple inflows and outflows, the mass-balance equation is given by:

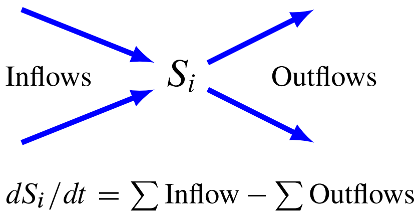


Figure 1.9 Mass Balance: The rate of change in species S_i is equal to the difference between the sum of the inflows and the sum of the outflows

$$\boxed{\frac{dS_i}{dt} = \sum \mathit{Inflows} - \sum \mathit{Outflows} \qquad (1.2)}$$

For an even more general representation, we can write the mass-balance equations by taking into account the stoichiometric coefficients. The rate at which a given reaction, v_j contributes to change in a species, S_i is given by the stoichiometric coefficient of the species, S_i with respect to the reaction, c_{ij}, **multiplied** by the reaction rate, v_j (See equation A.1). That is, a reaction j contributes, $c_{ij} v_j$ rate of change in species S_i. For a species, S_i with multiple reactions producing and consuming S_i, the mass-balance equation (assuming constant unit volume) is given by:

$$\frac{dS_i}{dt} = \sum_j c_{ij} v_j \tag{1.3}$$

where c_{ij} is the stoichiometric coefficient for species i with respect to reaction, j. For reactions that consume a species, the stoichiometric coefficient is often **negative** otherwise the stoichiometric coefficient is **positive** (See Appendix A). In considering the simple example in Figure 1.8, the stoichiometric coefficient for S with respect to v_1 is $+1$ and for v_2 is -1. That is

$$\frac{dS}{dt} = c_{s1} v_1 + c_{s2} v_2$$

or

$$\frac{dS}{dt} = (+1) v_1 + (-1) v_2 = v_1 - v_2$$

The way in which the construction of the mass-balance equation is described may seem overly formal, however the formality allows software to be written that can automatically convert network diagrams into mass-balance differential equations.

Example 1.2

Consider a linear chain of reactants from S_1 to S_5 shown in Figure 1.10. Write out the mass-balance equations for this simple system.

$$\begin{tikzcd} S_1 \arrow[r, "v_1"] & S_2 \arrow[r, "v_2"] & S_3 \arrow[r, "v_3"] & S_4 \arrow[r, "v_4"] & S_5 \end{tikzcd}$$

Figure 1.10 Simple Straight Chain Pathway.

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

$$\frac{dS_3}{dt} = v_2 - v_3 \qquad \frac{dS_4}{dt} = v_3 - v_4$$

$$\frac{dS_5}{dt} = v_4 \tag{1.4}$$

Each species in the network is assigned a mass-balance equation which accounts for the flows into and out of the species pool.

Example 1.3

Write out the mass-balance equation for the following branched system:

$$\begin{tikzcd} & & {} & \\ {} \arrow[r, "v_1"] & S_1 \arrow[ur, "v_2"] \arrow[dr, "v_3"'] & & {} \\ & & S_2 \arrow[ur, "v_4"] \arrow[dr, "v_5"'] & \\ & & & {} \end{tikzcd}$$

Figure 1.11 Multi-Branched Pathway.

The mass-balance equations are given by:

$$\frac{dS_1}{dt} = v_1 - v_2 - v_3$$

$$\frac{dS_2}{dt} = v_3 - v_4 - v_5$$

Example 1.4

Write out the mass-balance equation for the more complex pathway:

$$\begin{aligned} A + X &\xrightarrow{v_1} 2X \\ X + Y &\xrightarrow{v_2} Z \\ Z &\xrightarrow{v_3} Y + B \end{aligned}$$

This example is more subtle because we must be careful to take into account the stoichiometry change between the reactant and product side in the first reaction (v_1). In reaction v_1, the stoichiometric coefficient for X is $+1$ because two X molecules are made for every one consumed. Taking this into account the rate of change of species X can be written as:

$$\frac{dX}{dt} = -v_1 + 2v_1 - v_2$$

or more simply as $v_1 - v_2$. The full set of mass-balance equations can therefore be written as:

$$\frac{dA}{dt} = -v_1 \qquad \frac{dX}{dt} = v_1 - v_2$$

$$\frac{dY}{dt} = v_3 - v_2 \qquad \frac{dZ}{dt} = v_2 - v_3$$

$$\frac{dB}{dt} = v_3$$

The last example (1.4) illustrates a very important aspect of converting a network diagram into a set of differential equations. The process is potentially **lossy**. That is, it is not always possible to fully recover the original network diagram from the set of derived differential equations. This is because in one or more of the reactions the stoichiometries may cancel out. In the example (1.4) the reaction, $A + X \longrightarrow 2X$ is not recoverable from the final set of differential equations. Instead if we reverse engineered the differential equations the first reaction would be:

$$A \rightarrow X$$

which is not like the original. This is not perhaps a common occurrence although in protein signaling pathways it might be more common than other kinds of networks. What it means however is that sharing models by exchanging differential equations is **not** recommended. This is one reason why standard exchange formats such as SBML [48] store models explicitly as a set of reactions not as a set of differential equations. Many models are exchanged using Matlab which means that much of the biological information, particularly information on the underlining network, is lost. Exchanging models via computer languages such as Matlab is therefore not recommended.

Example 1.5

Write out the mass-balance equation for pathway:

$$\begin{aligned} S_1 + S_3 &\xrightarrow{v_1} S_2 \\ 2S_2 &\xrightarrow{v_2} S_3 \\ S_3 &\xrightarrow{v_3} 3S_4 \end{aligned}$$

In this example we have non-unity stoichiometries in the second and third reaction steps. The mass-balance equations are given by:

$$\frac{dS_1}{dt} = -v_1 \qquad \frac{dS_2}{dt} = v_1 - 2v_2$$

$$\frac{dS_3}{dt} = v_2 - v_3 \qquad \frac{dS_4}{dt} = 3v_3$$

From the previous examples we can see that it is fairly straight forward to derive the balance equations from a visual inspection of the network. Many software tools exist that will assist in this effort by converting network diagrams, either represented visually on a computer screen (for example, JDesigner) or by processing a text file that lists the reactions in the network (for example via Jarnac) into a set of differential equations (See Appendix E).

1.5 Stoichiometry Matrix

When describing multiple reactions in a network, it is convenient to represent the stoichiometries in a compact form called the **stoichiometry matrix**, traditionally denoted by $\mathbf{N}$, where the symbol $\mathbf{N}$ refers to number[1]. The stoichiometry matrix is a m row by n column matrix where m is the number of species and n the number of reactions:

$$\mathbf{N} = m \times n \text{ matrix}$$

The columns of the stoichiometry matrix correspond to the individual chemical reactions in the network, the rows to the molecular species, one row per species. Thus the intersection of a row and column in the matrix indicates whether a certain species takes part in a particular reaction or not, and, according to the sign of the element, whether there is a net loss or gain of substance, and by the magnitude, the relative quantity of substance that takes part in that reaction. That is the elements of the stoichiometry matrix **do not** concern themselves with the rate of reaction. This latter point is particular important when we will consider in a later chapter the various stoichiometric analyses that can be carried out purely on the stoichiometry without **any** reference to reaction rate laws.

[1]Some recent flux balance literature uses the symbol $\mathbf{S}$

The stoichiometric matrix is not concerned with describing the reaction rates. Reaction rates are given by the rate laws which is a separate vector (See section 1.8).

In general the stoichiometry matrix has the form:

$$\begin{array}{cc} & \longleftarrow \quad v_j \quad \longrightarrow \\ \mathbf{N} = \begin{array}{c}\uparrow\\ S_i\\ \downarrow\end{array} & \begin{bmatrix} c_{ij} & \cdots & \cdots \\ \vdots & & \\ \vdots & & \end{bmatrix} \end{array}$$

where c_{ij} is the stoichiometry coefficient for the i^{th} species and j^{th} reaction. As was mentioned before the stoichiometry matrix is in general a lossy representation. That is, it is not always possible to revert back to the original biochemical network from which the matrix was derived. For example consider the simple stoichiometry matrix:

$$\mathbf{N} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}$$

The most obvious network that this matrix could have been derived from is:

$$\begin{aligned} A &\longrightarrow B \\ B &\longrightarrow C \end{aligned}$$

But equally plausible is this network:

$$\begin{aligned} 2A &\longrightarrow A + B \\ B &\longrightarrow C \end{aligned}$$

If is not possible from the stoichiometry matrix alone to determine which was the original network.

Example 1.6

Write out the stoichiometry matrix for the simple chain of reactions which has five molecular species and four reactions as shown below. The four reactions are labeled, v_1 to v_4.

$$S_1 \xrightarrow{v_1} S_2 \xrightarrow{v_2} S_3 \xrightarrow{v_3} S_4 \xrightarrow{v_4} S_5$$

The stoichiometry matrix for this simple system is given by:

$$\mathbf{N} = \begin{array}{cccccc} & v_1 & v_2 & v_3 & v_4 & \\ \left[\begin{array}{c} \\ \\ \\ \\ \\ \end{array}\right. & \begin{array}{c} -1 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} & \begin{array}{c} 0 \\ -1 \\ 1 \\ 0 \\ 0 \end{array} & \begin{array}{c} 0 \\ 0 \\ -1 \\ 1 \\ 0 \end{array} & \begin{array}{c} 0 \\ 0 \\ 0 \\ -1 \\ 1 \end{array} & \left.\begin{array}{c} \\ \\ \\ \\ \\ \end{array}\right] \begin{array}{c} S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{array} \end{array}$$

The rows and columns of the matrix have been labeled for convenience. Normally the labels are absent.

Example 1.7

Write out the stoichiometry matrix for the multibranched pathway shown in Figure 1.11

$$\mathbf{N} = \begin{array}{cccccc} & v_1 & v_2 & v_3 & v_4 & v_5 \\ \left[\begin{array}{c} \\ \\ \end{array}\right. & \begin{array}{c} 1 \\ 0 \end{array} & \begin{array}{c} -1 \\ 0 \end{array} & \begin{array}{c} -1 \\ 1 \end{array} & \begin{array}{c} 0 \\ -1 \end{array} & \begin{array}{c} 0 \\ -1 \end{array} \left.\begin{array}{c} \\ \\ \end{array}\right] \begin{array}{c} S_1 \\ S_2 \end{array} \end{array}$$

1.6 Reversiblity

Up to this point nothing has been said about whether a given reaction is reversible or not. When dealing with kinetic models, reversibility often manifests itself as a **negative reaction rate** in the rate law. For example the rate law for the simple mass-action reversible reaction, $A \rightleftharpoons B$ is often given by:

$$v = k_1 A - k_2 B$$

When this reaction goes in the reverse (right to left) direction, the reaction rate, v, will be negative. This may not be apparent from the stoichiometry matrix, which in this case will be:

$$\mathbf{N} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Information on reversibility is therefore traditionally found in the rate law. Depending on the modeling problem, reversibility can be made more explicit in the stoichiometry matrix by specifying a **separate** reaction path for the reverse reaction. For example, in the previous example we might instead represent the system by two separate rate laws:

$$\begin{aligned} A \rightarrow B \quad v_f &= k_1 A \\ B \rightarrow A \quad v_r &= k_2 B \end{aligned}$$

in which case the stoichiometry matrix now becomes:

$$\mathbf{N} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix}$$

Splitting a reaction into separate forward and reverse steps might not always be possible however. For example an enzyme catalyzed reversible reaction such as $A \rightleftharpoons B$ **cannot** be represented using:

$$\frac{dB}{dt} = v_f - v_r$$

where the forward (v_f) and reverse (v_r) rates might be represented by irreversible Michaelis-Menten rate laws because the individual reactions are not independent but are connected by the shared enzyme pool. In such cases, the full enzyme mechanism in terms of elementary steps should be used.

To illustrate that we can apply the stoichiometry matrix to other kinds of networks, let us look at a simple signaling network and two simple gene regulatory networks.

1.7 Network Types

Signaling Networks

Figure 1.12 illustrates a simple protein signaling network, comprising two double phosphorylation cycles coupled by inhibition by protein C on the lower double cycle (D, E and F). In this model, all species are proteins and we assume that protein A and D are unphosphorylated, B and E singly phosphorylated and C and F doubly phosphorylated. C acts as a kinase and phosphorylates D and E. The reverse reactions, v_2, v_4, v_7 and v_8 are assumed to be catalyzed by phosphatases.

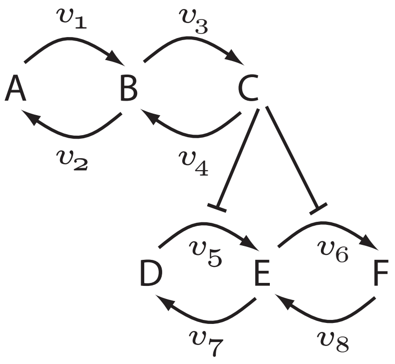


Figure 1.12 Simple Signaling Network. Protein C inhibits the activity of reactions v_5 and v_6.

There is no specified stoichiometric mechanism for the inhibition on v_5 and v_6. Therefore the stoichiometric matrix will contain no information on this. The stoichiometric matrix for this system will look like:

$$\mathbf{N} = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 & v_8 \end{array} \\ \left[\begin{array}{cccccccc} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{array}\right] \end{array} \tag{1.5}$$

The stoichiometric matrix can be seen to be composed of two separate blocks corresponding to the two cycle layers. It is important to note that whenever there are **regulatory** interactions in a pathway diagram, these **do not** appear in the stoichiometry matrix. Instead, such information will reside in the rate law that describes the regulation. If however the mechanism for the regulation is made explicit then details of the regulation will appear in the stoichiometry matrix. Figure 1.13 will shows a simple example of an inhibitor I regulating a reaction, S to P. On the left is displayed the implicit regulatory interaction. All we see is a blunt ended arrow indicating inhibition. In this case, details of the regulation will be found in the rate law governing the conversion of S to P. On the right is displayed an explicit mechanism, a simple competitive inhibition. In this case details of the inhibition mechanism will find its way into the stoichiometry matrix, although from an inspection of the stoichiometry matrix it is not obvious what kind of regulation it is.

Figure 1.14 shows a comparison of the implicit and explicit models in terms of the stoichiometry matrix. In each case the rate laws also change. In the implicit form, the rate law will be a Michaelis-Menten competitive inhibition model whereas in the explicit model, the rates laws (now multiplied in number) will be simple mass-action rate laws. The choice of what to use, an implicit or explicit model, will depend entirely on the type of question that the model is being used to answer. **There is no right or wrong way to do this**, the details of a model will depend on the type of question being asked.

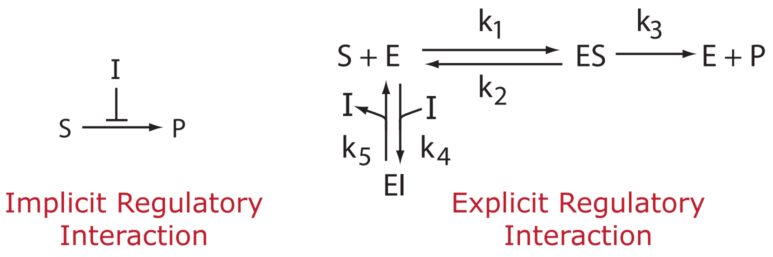

I
S
P
Implicit Regulatory Interaction
S + E
k1
k2
ES
k3
E + P
I
I
k5
k4
EI
Explicit Regulatory Interaction

Figure 1.13 Example of implicit and explicit depiction of a regulatory interaction. The left-hand mechanism involving inhibitor I will not appear in the stoichiometry matrix whereas the explicit mechanism, right-hand figure, it will.

Gene Regulatory Networks

Consider a transcription factor P_1 that represses a gene with expression rate v_3 shown in Figure 1.15, left panel. In this model we have production of P_1 from reaction v_1 and degradation of P_1 via v_2. The construction of the stoichiometry matrix will depend on how we represent the regulated step, v_3. If regulation is implied, i.e. there is no explicit kinetic mechanism, then the regulation will not appear in the stoichiometry matrix. For the network on the left in Figure 1.15, the stoichiometry matrix will be given by:

$$\mathbf{N} = \begin{array}{c} \\ P_1 \end{array} \begin{array}{c} \begin{array}{cc} v_1 & v_2 \end{array} \\ \left[\begin{array}{cc} 1 & -1 \end{array} \right] \end{array} \tag{1.6}$$

The stoichiometry matrix has only one row indicating that there is only one species in the model, P_1 and there is no hint in the stoichiometry matrix that there is regulation. In this model, P_1 is not explicitly sequestered by the operator site that is upstream of the gene. We make the significant assumption that when P_1 regulates, it is itself is not affected in any way.

Consider now that the interaction between P_1 and v_3 is made mechanistically explicit. The right hand network in Figure 1.15 shows one possible way in which to represent the interaction of the transcription factor,

$$\mathbf{N} = \begin{array}{c} \\ S \\ P \\ I \end{array} \begin{array}{c} v_1 \\ \left[\begin{array}{c} -1 \\ 1 \\ 0 \end{array}\right] \end{array} \qquad \mathbf{N} = \begin{array}{c|ccccc|} & v_1 & v_2 & v_3 & v_4 & v_5 \\ S & -1 & 1 & 0 & -1 & 1 \\ P & 0 & 0 & 1 & 0 & 0 \\ I & 0 & 0 & 0 & -1 & 1 \\ ES & 1 & -1 & 1 & 0 & 0 \\ EI & 0 & 0 & 0 & 1 & -1 \end{array}$$

Implicit Explicit

Figure 1.14 Stoichiometry matrices corresponding to the two models in Figure 1.13

P_1 with gene v_3. In the explicit model, the transcription factor, P_1 is assumed to bind to a repressor site preventing gene expression. In the ex-

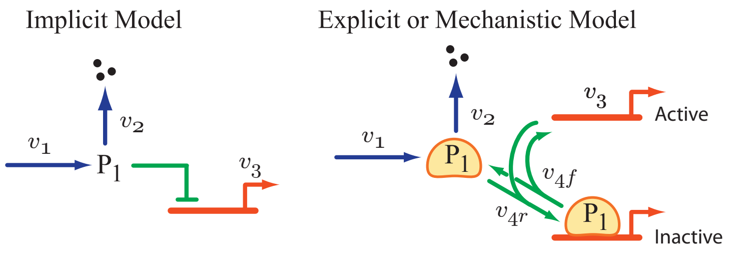


Figure 1.15 Two simple gene regulatory networks involving gene repression. On the left side is the implicit model where P_1 represses v_3, on the right side is the explicit model showing a more detailed mechanism for the regulation.

plicit model there are two new species, designated active gene and inactive gene. The stoichiometry matrix will therefore include two additional rows corresponding to these two new species. The stoichiometry matrix for the explicit model is shown below:

$$\mathbf{N} = \begin{array}{r} \\ P_1 \\ P_1(\text{Active}) \\ P_1(\text{InActive}) \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_{4r} & v_{4f} \end{array} \\ \left[\begin{array}{rrrr} 1 & -1 & -1 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{array}\right] \end{array} \tag{1.7}$$

In this case, P_1 is actively sequestered on to the operator site and therefore appears in the stoichiometry matrix. Processes such as consumption, production or sequestration by some binding mechanism will appear as columns in the stoichiometry matrix. Regulation that is often depicted by arrow or blunt ends are modeled in the rate law itself and therefore do not appear in the stoichiometry matrix.

In conclusion, regulation does not appear explicitly in a stoichiometry matrix unless the regulation is represented in an explicit mechanistic scheme. The choice of implicit or explicit representations depends on the question being asked and the availability of suitable data.

1.8 The System Equation

Equation 1.3, which describes the mass balance equation, can be reexpressed in terms of the stoichiometry matrix to form the **system equation**.

$$\frac{d\mathbf{s}}{dt} = \mathbf{N}\boldsymbol{v} \tag{1.8}$$

where $\mathbf{N}$ is the $m \times n$ stoichiometry matrix and $\boldsymbol{v}$ is the n dimensional rate vector, whose ith component gives the rate of reaction i as a function of the species concentrations. $\mathbf{s}$ is the m vector of species.

Looking again at the simple chain of reactions in Figure 1.10, the system equation can be written down as:

$$\frac{d\mathbf{s}}{dt} = \mathbf{N}\boldsymbol{v} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \tag{1.9}$$

If the stoichiometry matrix is multiplied into the rate vector, the mass-balance equations show earlier (1.4) are recovered. To illustrate that the system equation might look like for a simple system, consider the model in Jarnac format:

```
p = defn cell
     A -> B; k1*A - k2*B;
     B -> C; k3*B - k4*C;
end;

p.k1 = 0.1; p.k2 = 0.02;
p.k3 = 0.3; p.k4 = 0.04;
p.A  = 10;  p.B  = 0; p.C = 0;
```

The system equation for this model will be given by:

$$\frac{d\mathbf{s}}{dt} = \mathbf{N}\boldsymbol{v} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} k_1 A - k_2 B \\ k_3 B - k_4 C \end{bmatrix} \tag{1.10}$$

All stoichiometric interactions are placed in the stoichiometry matrix.

The example shown in Figure 1.12 and Figure 1.15 illustrated non-stoichiometric interactions, namely two inhibition interactions from C to reactions v_5 and v_6 and repression on v_3 by P_1. As was noted, these interactions do no occur in the stoichiometry matrix. Instead they will be found in the rate vector, $\boldsymbol{v}$ in the form of a particular rate law.

The stoichiometry matrix represents the mass transfer connectivity of the network and contains information on the network's structural mass-transfer characteristics. These characteristics fall into two groups, relationships among the species and relationships among the reaction rates. These relationships will be considered in detail in another volume.

1.9 Jarnac

The modeling platform Jarnac [91] provides facilities to extract the stoichiometry matrix from a model. The command for generating the stoichiometry matrix is `p.sm` assuming the model is stored in the variable `p`. The script and results of a run are given below:

```
p = defn cell
     J1: A -> B; k1*A - k2*B;
     J2: B -> C; k3*B - k4*C;
end;

p.k1 = 0.1; p.k2 = 0.02;
p.k3 = 0.3; p.k4 = 0.04;
p.A  = 10;  p.B  = 0; p.C = 0;

// Print out the stoichiometry matrix
println p.sm;
```

If this script is run, the output is as shown below:

```
       J1     J2
A{{    -1      0 }
B {     1     -1 }
C {     0      1 }}
```

Note that in Jarnac, matrices are labeled, this is useful for identifying the corresponding species and reactions in the stoichiometry matrix.

Further Reading

1. Sauro HM (2012) Enzyme Kinetics for Systems Biology. 2nd Edition, Ambrosius Publishing ISBN: 978-0982477335

2. Stephanopoulos G, Aristidou A, and Nielsen J (1998) Metabolic engineering: principles and methodologies. Academic Press, ISBN: 978-0126662603

3. Palsson BO (2006) Systems Biology Systems Biology: Properties of Reconstructed Networks. Cambridge University Press, ISBN: 978-0521859035

Exercises

1. Explain the difference between the terms: Stoichiometric amount, Stoichiometric coefficient, rate of change (dX/dt) and reaction rate (v_i). Refer to Appendix A to answer this question.

2. Determine the stoichiometric amount and stoichiometric coefficient for each species in the following reactions:

$$\begin{aligned}
A &\longrightarrow B\\
A + B &\longrightarrow C\\
A &\longrightarrow B + C\\
2A &\longrightarrow B\\
3A + 4B &\longrightarrow 2C + D\\
A + B &\longrightarrow A + C\\
A + 2B &\longrightarrow 3B + C
\end{aligned}$$

3. Derive the set of differential equations for the following model in terms of the rate of reaction, v_1, v_2 and v_3:

$$\begin{aligned} A &\xrightarrow{v_1} 2B \\ B &\xrightarrow{v_2} 2C \\ C &\xrightarrow{v_3} \end{aligned}$$

4. Derive the set of differential equations for the following model in terms of the rate of reaction, v_1, v_2 and v_3:

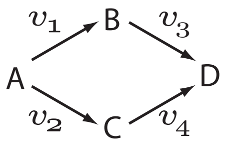
$$\begin{aligned} A &\xrightarrow{v_1} B \\ 2B + C &\xrightarrow{v_2} B + D \\ D &\xrightarrow{v_3} C + A \end{aligned}$$

5. Write out the stoichiometry matrix for the networks in question 3 and 4

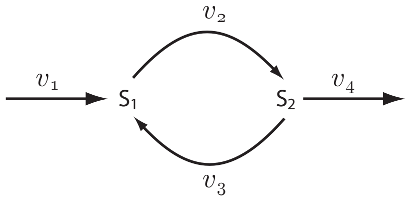
6. Enter the previous models, 3 and 4, into Jarnac and confirm that the Jarnac stoichiometry matrices are the same as those derived manually in the exercises.

7. Derive the stoichiometry matrix for each of the following networks. In addition write out the mass-balance equations in each case.

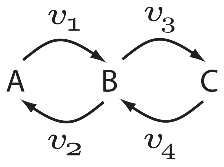
(a)



(b)



(c)



(d)

$$A + X \xrightarrow{v_1} B + Y \qquad B + X \xrightarrow{v_2} Y$$

$$B \xrightarrow{v_3} C \qquad C + X \xrightarrow{v_4} D + Y$$

$$D + Y \xrightarrow{v_5} X \qquad X \xrightarrow{v_6} Y$$

$$X + W \xrightarrow{v_7} 2Y \qquad 2Y \xrightarrow{v_8} X + W$$

8. For the irreversible enzyme catalyzed reaction, $A \rightarrow B$:

(a) Write out the stoichiometry matrix.

(b) Write out the stoichiometry matrix in terms of the elementary reactions that make up the enzyme mechanism.

9. A gene G_1 expresses a protein p_1 at a rate v_1. p_1 forms a tetramer (4 subunits), called p_1^4 at a rate v_2. The tetramer negatively regulates a gene G_2. p_1 degrades at a rate v_3. G_2 expresses a protein, p_2 at a rate v_9. p_2 is cleaved by an enzyme at a rate v_4 to form two protein domains, p_2^1 and p_2^2. p_2^1 degrades at a rate v_5. Gene G_3 expresses a protein, p_3 at a rate v_6. p_3 binds to p_2^2 forming an active complex, p_4 at a rate v_{10}, which can bind to gene G_1 and activate G_1. p_4 degrades at a rate v_7. Finally, p_2^1 can form a dead-end complex, p_5, with p_4 at a rate v_8.

10. (a) Draw the network represented in the description given above.

(b) Write out the differential equation for each protein species in the network in terms of $v_1, v_2, \ldots$.

(c) Write out the stoichiometric matrix for the network.

11. Write out the differential equations for the system depicted in equation 1.9.

12. Given the following stoichiometry matrix, write out the corresponding network diagram. Why might this process not fully recover the original network from which the stoichiometry matrix was derived?

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \end{array}
\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\
\left[\begin{array}{ccccc}
-1 & 0 & -1 & 0 & 0 \\
1 & -1 & 0 & 0 & 3 \\
0 & 2 & -1 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & -1 & 0
\end{array}\right] \end{array} \tag{1.11}$$

13. Why is it better to store a model as a list of reactions rather than a set of differential equations?

2 Introduction to Modeling

2.1 Introduction

The universe is a very large place and to study it in its entirely would not be practical. Instead we always study a small portion of the universe, often under very controlled conditions which we call a **system**. Everything else other than the system is called the **surroundings**. Between the system and the surroundings we try to enforce strict rules on how the system interacts with the surroundings. These interactions occur at the system **boundary**.

The **system** is a defined region of the universe that we wish to study.

The **surroundings** is everything else other than the system.

The **boundary** is the interface between the system and the surroundings.

The word system derives from the Greek term to mean "place together", that is a system is generally one or more parts working together. There are however certain characteristics about systems that we impose that make them very suitable to study.

In order to make the study of a particular system possible we will often have strict conditions on how the system interacts with the rest of the universe. If the system were allowed to freely interact with the surroundings without any restrictions then we're effectively back to studying the entire universe again. When we study a system we will usually make sure that we know exactly how the system interacts with the surroundings and in ways that we can control.

To make this more concrete, consider an animal cell that consumes glucose, produces waste products and generates a small amount of heat. The animal cell will be our system. In an experiment we may arrange things so that the concentration of glucose and waste products outside the cell are kept relatively constant during the experiment. In addition we can keep the animal cell in a thermostatically control temperature bath. By keeping the environment constant we are insulating the animal cell from any extraneous changes that might inadvertently affect our system in unknown ways. We therefore have control over the system. If we didn't, we would find it very difficult to study the behavior of the animal cell.

The actual boundary of the system is however entirely at the discretion of the experimenter and depends on practical as well and scientific considerations. The important point is that the boundary is under **our strict control**, at least in principle. The nature of this control also determines whether our system is open, closed or isolated.

In general, the experimenter decides the location of the boundary that exists between the system and the surroundings. Often this is accomplished by having strict control over the surroundings.

2.2 Open, Closed, and Isolated Systems

When considering systems it is usual to distinguish between three types of boundary between the system and the surroundings (i.e. rest of the universe). These types are called **isolated, closed and open** systems. Each of these systems represent an idealized state. In practice we can often approximate them quite well. An isolated system, as the name suggests, is completely cut off from the rest of the universe, that is neither energy or matter can be transferred across the isolated systems's boundary. A closed system is one that only transfers energy, for example heat, work or light.An open system is one that can exchange both energy and mass with the surroundings.

The distinction between a closed and open system in biology is very important. Open systems are characteristic of biological systems. For example, glycolysis is a pathway for converting an external nutrient source, such as glucose, into available energy, such as ATP or heat and waste products such as lactate or ethanol. That is it exchanges mass and energy with the surroundings. Without mass and energy exchange biological systems would eventually run to thermodynamic equilibrium and cease to function. All models of living biological model are therefore open.

The previous example of an experiment done on an animal cell illustrates an open system. In this case the cell consumed glucose and produced waste products in addition to generating heat. Even though there was the free exchange of matter and energy we nevertheless made sure that we continually replenished the glucose so that it appeared to the cell that the level of glucose was constant. Likewise we ensured that waste products didn't build up and that the generated heat didn't cause the temperature to rise because we immersed the experiment in a heat bath. From the perspective of the animal cell, the surroundings appeared constant even though there was a continue exchange between the surroundings and the animal cell.

Example 2.1

For each of the following systems, decide whether the system is isolated, closed or open. Comment on the nature of the surroundings.

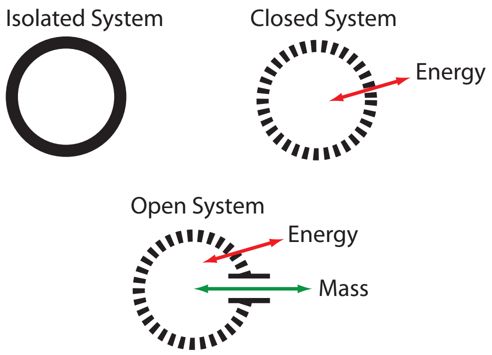


Figure 2.1 Open and Closed Systems.

i) A system represented by a mechanical clock slowly winds down in a room controlled by a thermostat.

The clock starts with an amount of potential energy in the wound spring which slowly dissipates, ultimately as heat which is transferred to the surroundings. No mass is exchanged with the room. The clock is therefore a closed system. Because the clock is in a temperature controlled room, the temperature of the room appears constant to the clock even though the clock dissipates heat.

ii) A car engine running idle in the open air.

The car engine is burning fuel that generates both waste gases and heat. The heat and waste gases are lost to the surroundings. At the same time the car engine takes in oxygen. The car system is therefore open since it exchanges both matter and energy with the surroundings. In addition, given that the exchange takes place in the open, the surrounding temperature, oxygen and carbon dioxide levels appear constant because the large volume of the atmosphere acts as a buffer. We assume that the fuel tank is part of the system and therefore we do not consider the loss of fuel to be an exchange of mass with the surroundings.

iii) A bacterial culture is grown in batch and kept in a sealed and insulated chamber.

The batch vessel is isolated and therefore the culture itself is an isolated system. There is no exchange of mass or energy with the surroundings. However, if we focus our attention on a single bacterium we would have to conclude that a single cell is an open system which consumes nutrients, produces waste and generates heat. However, the bacterial surroundings are not kept constant and the temperature as well as waste products rise with the loss of nutrients. Eventually the nutrients are used up, the culture dies and the system tends to thermodynamics equilibrium.

2.3 Models

There are many ways to describe systems, these range from pictures or cartoons, to verbal and mathematical representations. Collectively, these descriptions are called **models**. A model is our way of describing a particular system. The Oxford English dictionary defines a model in the following way:

> "A simplified or idealized description or conception of a particular system, situation, or process, often in mathematical terms, that is put forward as a basis for theoretical or empirical understanding, or for calculations, predictions, etc."

This definition embodies a number of critical features that defines a model, the most important is that a model represents an **idealized description**, a simplification, of a real world process. This may at first appear to be a weakness but simplification is usually done on purpose. Simplification allows us to comprehend the essential features of a complex process without being burdened and overwhelmed by unnecessary detail.

A more interesting way to describe models is to use mathematics, a language designed for logical reasoning. Mathematical models are useful in biology for a number of reasons, but the three most important are increased precision, prediction, and the capacity for analysis. Analysis can be carried out either by simulation or by mathematical analysis. Although visual models can be used to make predictions, the kinds of predictions that can be made are limited. The use of mathematical models opens up whole new vistas of study which visual models simply can not match.

A **model** is a simplified description of a system. A model can be used to represent known facts about the system and hypotheses concerning the system's operation. Models can be described using pictures, plain text, mathematics or computer software.

2.4 Build a Simulation Model

Water Tank Model

Figure 2.2 shows two water tanks. The first tank is fed with water at a rate Q_1 (m^3 s^{-1}). This tank drains into a second tank at a rate Q_2 which in turn drains to waste at a rate Q_3. The second tank has an additional feed of water flowing in at a rate Q_4. The height of the water level in each tank is given by h_1 and h_2 respectively. Each tank has a cross sectional area A.

The rate of change in the volume of water in a given tank is the rate at which the water enters minus the rate at which it leaves. For example for the first tank we have:

$$\frac{dV_1}{dt} = Q_1 - Q_2$$

If we want the equation in terms of the rate of change of height then we needs to recall that $V = Ah$, that is:

$$\frac{dV}{dt} = A\frac{dh}{dt}$$

so that:

$$\frac{dh_1}{dt} = \frac{Q_1 - Q_2}{A}$$

Using Torrielli's Law we know that the rate of water flowing out of a given tank, i, is equal to:

$$Q_i = K_i\sqrt{h}$$

Where K_i is a constant related to the resistance of the output pipe. Therefore for the first tank we have:

$$\frac{dh_1}{dt} = \frac{Q_1 - K\sqrt{h}}{A}$$

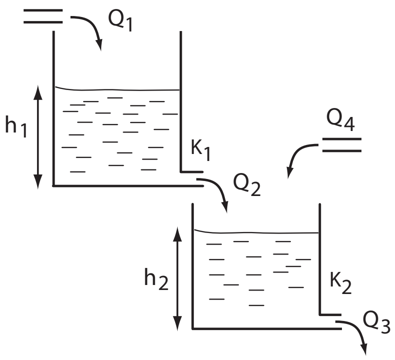


Figure 2.2 Water Tank Model

Given this information, answer the following questions:

a) Plot the rate of outflow, Q_2, as a function of the height of water, h_1 at a given resistance, K_1.

b) Assuming that Q_1 and Q_4 are fixed, and we start with both tanks empty, what do you expect to happen over time as water flows in?

c) Write out the differential equations (ODEs) that describe the rate of change in the tank water levels, h_1 and h_2.

d) Build a computer model of the tank system, assign suitable values to the parameters in the model and run the simulation to plot the height of water in the tanks over time. Assume both tanks are empty at time zero.

e) Investigate the effect of increasing and decreasing the resistance parameters, K_1 and K_2 on the model.

2.5 Types of Model

Models come in various forms including verbal, written text, visual, mathematical and others. Molecular biology, has a long tradition of using visual models to represent cellular structure and function; one need only look through a modern textbook to see instances of visual models on every page. Visual models have been immensely useful at describing complicated biological processes but are limited in their scope.

Like visual models, mathematical models can serve at least two important purposes in systems biology:

- Conceptual Models

  Conceptual models serve as test-beds for investigating basic principles, for example the effects of feedback or sequestration. Conceptual models are not intended to describe actual real biological systems but are instead employed to aid reasoning about particular aspects of biological networks. Conceptual models are frequently to illustrate properties of biological networks.

- Concrete Models

  Concrete models are constructed to model a specific real system, such as glycolysis, apoptosis or the sporulation circuit in Bacillus subtilis. They represent a **working hypothesis** for a particular biological system and allows us to generate predictions about the real system and falsifiable statements about the model.

Concrete Models

What makes a good concrete model? There are a range of properties that a good model should have, but probably the most important are **accuracy**, **predictability** and **falsifiablity**.

▷ A model is considered **accurate** if the model is able to describe current experimental observations, that is a model should be able to reproduce the current state of knowledge.

▷ A **predictive** model should be able to generate insight and/or predictions that are beyond current knowledge. Without this ability a model is considerably less useful, some would even suggest useless.

▷ Finally, a model should be **falsifiable**. By this we mean that a model cannot be proved be true only disproved. The only discipline where statements can be actually be proved to be true or false is mathematics. Starting with a set of axioms, mathematicians derive theorems that can be shown beyond any doubt to be true or false. In contrast scientific models based on observations **cannot be proved correct**. This is because it is simply not possible to test every possible circumstance in which the model may apply. Instead we are left with two options:

Model Falsification

We can falsify a model by finding one observation that the model fails to predict. In this case the model must be changed or abandoned. For example, the statement, RNA is never transcribed into DNA can be falsified simply by finding one instance where it happens (e.g. the life cycle of the HIV virus). Although the idea of falsifying a model is appealing, in practice it is not often used. This leads to the second option, model validation which is probably the most commonly used approach.

Model 'Validation'

Model validation is based on the idea that predictions made by the model are verified by experiment. The word validate may imply that once a model is 'validated', the model can now be considered a true representation of the real system but this is not correct. A validated model is simply one where our **confidence** in a model's ability to predict and provide insight has **increased**. As already suggested, no model is correct. The utility of a model is based on how well it can make useful predictions and how well it fits existing knowledge. Models will have a certain scope within which they are useful. For example, Newtonian mechanics is useful for describing objects traveling at sub-light speeds, higher velocities are beyond the scope of the Newtonian model. Michaelis-Menten kinetics is useful for describing steady state kinetics but is less useful for describing transient behavior if the enzyme concentration is higher or comparable to the substrate concentration. One role of validation is to delineate the scope of a model.

Validation therefore serves two purposes, to describe the scope of a model and to increase confidence in the ability of the model to make useful predictions. It is important to understand that validation does no 'prove' that a model is correct since no such statement can easily be made.

Other Attributes

There are other attributes of a model that are desirable but not essential, these include **parsimonious** and **selective**. A parsimonious model is a model that is as simple as possible, but no simpler. Occam's infamous razor which states that "Entities should not be multiplied beyond necessity" and argues that given competing and equally good models, the simplest is preferred. Finally, since no model can represent everything in a given problem, a model must be **selective** and represent those things most relevant to the task at hand.

2.6 Model Variables

One of the first decisions to make when developing a model is to identify the external (the causes) and internal variables (effects). In systems biology, these variables often include the concentrations of molecular species or voltages across membranes.

The internal variables of the system are often termed the **state variables** or **dynamic variables** and will change in time as the system evolves. The values of the state variables are determined by the model or system. In biochemical models the state variables are the various molecular species such as metabolites, proteins, mRNA etc. In this book such variables will also be referred to as **floating species** to reflect the fact that these species can change during a simulation or experiment.

The state of a system at time t is described by a set of **state variables**:

$$\mathbf{x}(t)$$

They are the smallest set of variables that define the state of the system.

In contrast to the floating species, the external variables are in principle under the strict control of the experimenter. Often the external variables are clamped to some fixed values (*cf.* voltage clamp) but could also be varied in some controlled way by the experimenter. The external variables are also called **boundary variables** because they are considered to be at the boundary of the system and the rest of the universe.

Physically, external variables are usually clamped by some kind of buffering mechanism. The buffering mechanism can simply be a large external reservoir so that any exchange of mass between the system and the external reservoir has a negligible effect on the external concentration. Alternatively there may be active mechanisms maintaining an external concentration. A classic example of active maintenance of an external variable is the voltage clamp used in electrophysiology.

External concentrations may also simply be slow moving compared to the timescale of the model so that over the study period, the external concentrations change very little. A typical example of the latter is the study of a metabolic response over a timescale that is shorter than gene expression. This permits a modeler to study a metabolic pathway without considering the effect of changes in gene expression.

The separation between the internal and external variables depends ultimately on practical considerations and the particular questions that the experimenter and modeler wish to answer. However, once the choice is made, the separation is strictly adhered to during the course of a study. This means for example that the environment surrounding the physical system will, **by definition**, be **unaffected** by the behavior of the system. If for some reason parts of the environment do change as a result of the system and can in turn affect the system in some way, then these parts should be considered part of the system.

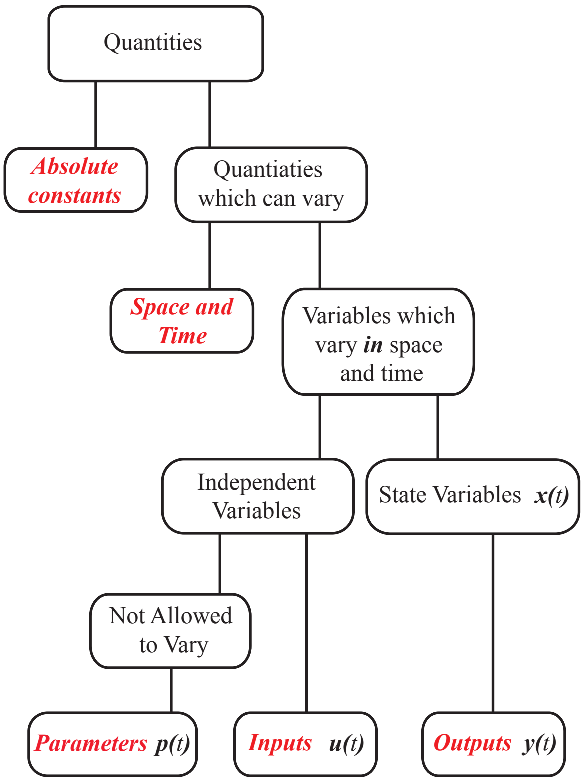


Figure 2.3 Classification of quantitative terms.

Mathematical Descriptions of Models

Table 2.3 lists the many different ways in which a model can be represented and simulated. However one thing they all have in common is that the models are expressed first in mathematical form. The form of this expression determines how the model will be constructed and how it will be solved or simulated. In particular decisions must be made about how the variables and parameters in the system are best described. For example the model variables can be described either using **discrete** or **continuous** variables. For example the change in the level of water in a tank is more reasonably described using a continuous variable such as the height. On the other hand it might be more realistic to describe the dynamics of lion predation on the Serengeti using a discrete model where individual lions can be represented. It might not make much sense to refer to 34.67 lions in a model. The choice of whether to use a discrete or continuous description depends entirely on the system being studied and the questions posed.

Another important categorization is whether the model should be represented in a **deterministic** or **stochastic** form. A deterministic model is one where if we repeated the simulation using the same starting conditions we would get exactly the same result again. That is the future state of the model is completely determined by its initial starting point. The model of the water tanks filling up is an example of a deterministic model. In contrast, each time we run a computer simulation of a stochastic model we would get a slightly different outcome even though the starting conditions are the same.

A **discrete variable** is one that cannot take on all values within a given numeric range. For example, the number of aeroplanes in the sky at any one time is a discrete number. In statistics this is generalized further to a finite set of states, such as true/false or combinations in a die throw.

Continuous variables can assume all values within a given numeric range. For convenience we will often represent a measurement as a continuous variable. For example we will often use a continues variable to represent the concentration of a solute as it is unwieldy to refer to the concentration of a solute as 5,724,871,927,315,193,634,656 molecules per liter.

The reason for this is that each step in the simulation is determined by one or more random processes. To give an example, modeling lion predation on the Serengeti could be modeled as a stochastic process. It is not guaranteed that a Lion will catch its prey every time, instead there is a probability it will succeed. To model this process the computer simulation would throw a die to determine whether the Lion had succeeded or not. Repeatedly running such a simulation again would naturally give a slightly different outcome because the die throws would be different in each run.

In systems biology stochastic models have been shown to be very important in reproducing certain behaviors. A deterministic model based on ordinary differential equations assumes a continuum of values for concentration. This ignores the fact that cellular process operate at the molecular level and concentrations could be described using discrete values representing the number of molecules. However, because we often deal with systems containing tens of thousands of particles we assume that we can describe concentration as a continuous variable and therefore differential equations are quite suitable for these situations. For systems where the particulate number is very low, of the order of tens of particles, the use a continuum measure might seem unreasonable. However, an additional and more important problem arises when dealing with low particulate numbers. At low concentrations, Brownian motion becomes a significant factor in determining reaction rates. The time at which a molecule binds or is transformed, becomes a probabilistic property. As a result of these factors, models of systems containing low particulate numbers are better modeled using a stochastic approach [123, 93].

A **deterministic model** is one where a given input will always produce the same output. For example, in the equation, $y = x^2$, setting x to 2 will always yield the output 4.

A **stochastic model** is one where the processes described by the model include a random element. This means that repeated runs of a model will yield slight different outcomes.

We can now therefore classify a model as a combination of the above attributes. The water tank model uses a deterministic, continuous approach. The lion model might use a discrete and stochastic approach. Table 2.1 shows the four combinations and example where each combination might be appropriately used.

Type	Example
Continuous/Deterministic	Projectile motion
Continuous/Stochastic	Brownian motion
Discrete/Deterministic	Large population dynamics
Discrete/Stochastic	Small population dynamics

Table 2.1 Examples of different kinds of model

Forcing Functions

As described earlier, it is common to make sure that the surroundings do not change during the duration of the study. For example we might make sure that the pH remains constant by using a buffer solution. The key point is that the experimenter has complete control over the experiment. In some cases it is useful for an experimenter to change the surrounding conditions in a controlled fashion. For example, he/she might slowly increase the concentration of an administered drug or make a step change in a variable such as the concentration of an enzyme. In systems theory such controlled changes are often called **forcing functions**.

Intensive and Extensive Properties.

In science a distinction is made between physical quantities termed intensive and extensive. An intensive property is a physical quantity whose value does not depend on the size of the system. Examples include pressure, density, concentration and temperature. An extensive property is a physical quantity whose value does depend on the size of the system, examples include mass, volume, energy and entropy.

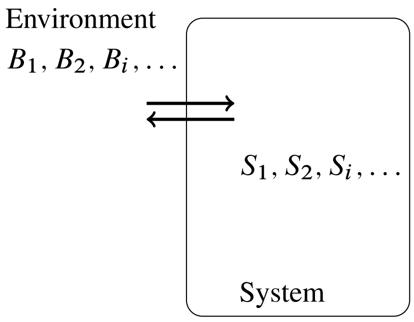


Figure 2.4 System and Environment: $S_1, S_2, S_i, \ldots$ are state variables that may change during the evolution of the system; $B_1, B_2, B_i, \ldots$ are boundary variables that are clamped to certain values by the observer. The exchange arrows represent the exchange of mass between the environment and the system.

2.7 Model Parameters

So far, only floating and boundary variables have been introduced as measurable quantities in the system. The floating variables are quantities that evolve in time and the boundary variables are concentrations or voltages that are clamped by the observer or under the control of a forcing function. There is a third set of quantities which are often called the **system parameters**. These include the various kinetic constants and enzyme activity factors whose values are determined by a combination of thermodynamic, physical or genotypic properties. Some system parameters will appear in the reaction rate laws as kinetic constants, others will be related to physical characteristics such as the volume of the system (assuming the volume is constant during the study period). As with boundary variables, the system parameters are in principle under the control of the experimenter and are not a function of the model itself. One can imagine for example changing kinetic constants via site directed mutagenesis or changing enzyme activities by altering promoter efficiencies. The actual choice of system parameters in any particular model will depend on physical limitations and the question being considered by the researcher.

Example Figure 2.5 illustrates a simplified model of glycolysis. The corresponding Table 2.2 lists the various variables and parameters that have been identified in the model. The concentration of glucose and ethanol are assumed to be boundary variables, that is controlled by the observer. This can be arranged by supplying glucose from a large volume compartment so that during its consumption there is only a negligible change in its concentration. Likewise we assume that ethanol is discharged into a large volume.

In the model another set of concentrations that are assumed to be constant are the NAD and NADH cofactors. This may be an unreasonable assumption to make because we know that the redox potential can change. We must assume here that the model builder has good reason for making this assumption and will make this explicit when the model if formally published. It is important that the model builder be specific about these decisions and why they were made. Such choices are necessary when building a model and great care should be made when making them. One simple way to justify this assumption is that if the model adequately predicts experiments that are of interest to the experimenter then it seems reasonable that a floating redox potential is not important. However as demands on the model to make further predictions expands there may come a time when the model fails and assumptions such as the fixed redox potential may need to be revisited.

The modeler also makes an additional assumption about ATP. Since glycolysis is ostensibly the pathway for generating ATP, some way to simulate ATP consumption is necessary, this is achieved by including a single step that hydrolyzes ATP to ADP even though we know that ATP consumption is a complex process involving many separate reactions. The response of the pathway to changing ATP demand can be simulated by perturbing the ATP demand step. The assumptions made in building this model may appears to be completely unreasonable but one sure test is to determine how well the model reproduces current knowledge about the system and whether the model can make useful predictions that can be later shown to be correct. If either of these tests fail then we know that the assumptions

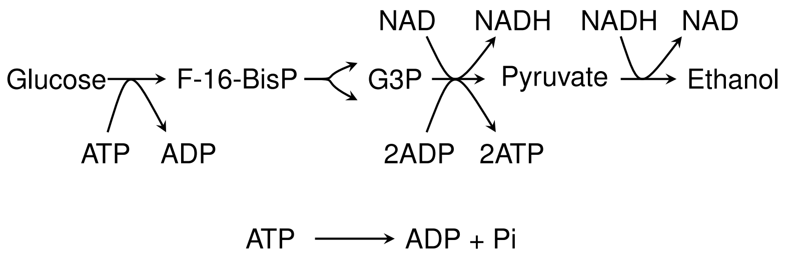


Figure 2.5 A simplified glycolytic pathway. Many reactions have been condensed and ATP consumption has been simplified to a single process $ATP \rightarrow ADP + Pi$

State Variables	System Parameters	Boundary Variables
F-16-BisP	Kinetic Constants	Glucose
G3P	Enzyme Activities	Ethanol
Pyruvate	Volume	NAD
ATP	Temperature	NADH
ADP		Pi

Table 2.2 Variables and parameters for the simplified glycolytic model 2.5. We assume that glucose and ethanol are clamped by the observer using large volume sinks. We assume that during the period of study that the concentrations of NAD and NADH remain essentially unchanged. F-16-BisP = Fructose-1,6-bisphosphate; G3P = Glyceraldehyde-3-Phosphate; Pi = Phosphate

about the model need to be amended. One very important point to make. It is easy to look on a model and state that it is unrealistic because it misses out certain features. However, is **unrealistic** if it fails to be useful, i.e. explain current knowledge and make useful predictions.

The realism of a model can only be judged with respect to its purpose.

Steps in Building a Model

To summarize we can break down the approach to building a model into at least four stages.

- Define the system boundaries.
- Define the simplifying assumptions.
- Invoke physical laws to describe the system processes.
- Test (validate) the model against experimental data.

Different Ways to Represent Physical Models

The tank model described in section 2.4 was built using a set of ordinary differential equations (ODEs) and solutions to these equations were obtained using a digital computer. There are however many other ways to build and find solutions to models. Table 2.3 lists some of the more common and interesting approaches that people have used in the past to construct and simulate models.

Table 2.3: Different ways to construct and solve physical models

Electrical Circuits	General purpose analog computer [109]
	WWII V2 guidance system [121]
	Neuromorhpic electronics [13, 106]

Mechanical and Fluid	Slide rule [119]
	Curta [112]
	Tide predicting machine [120]
	Computing projectile trajectories [118]
	Differential analyzer (solves ODEs) [113]
	Antikythera mechanism (planetary motion) [110]
	Water tanks - MONIAC economic model [116]
Purely Mathematical	Algebraic Equations
	Linear differential equations
	Linear difference equations
	Partial differential equations
	Probabilistic models
	Statistical models
Digital Computer	Solving ODEs and PDEs
	Agent based models (multicellular systems)
	Cellular automata [111]
	Emergent systems (Ant models) [114]
	Fractal models [115]
	Neural networks [117]

2.8 Dimensions and Units

Variables and parameters that go into a model will be expressed in some standard of measurement. In science the recognized standard for units are the SI units. These include units such as the *meter* for length, *kilogram* for mass, *second* for time, *Joules* for energy, *kelvin* for temperature and the *mole* for amount. The mole is of particular importance because it is a means to measure the number of particles of substance irrespective of the mass of substance itself. Thus 1 mole of glucose is the same amount as 1 mole of the enzyme glucose-6-phosphate isomerase even though the mass of each type of molecule is quite different. The actual number of particles in 1 mole is defined as the number of atoms in 12 grams of carbon-12 which has been determined empirically to be 6.0221415×10^{23}. This definition means that 1 mole of substance will have a mass equal to the molecular weight of the substance, this makes is easy to calculate the number of moles using the following relation

$$\text{moles} = \frac{\text{mass}}{\text{molecular weight}}$$

The concentration of a substance is expressed in moles per unit volume and is usually termed the molarity. Thus a 1 molar solution means 1 mole of substance in 1 litre of volume.

Dimensional Analysis

Dimensional analysis is a simple but effective method for uncovering mistakes when formulating kinetic models. This is particularly true for concrete models where one is dealing with actual quantities and kinetic constants. Conceptual models are more forgiving and don't usually require the same level of attention because they tend to be simpler.

Amounts of substance is usually expressed in moles and concentrations in moles per unit volume (mol l^{-1}). Reaction rates can be expressed either in concentrations or amounts per unit time depending on the context (mol t^{-1}, mol l^{-1} t^{-1}).

Rate constants are expressed in differing units depending on the form of the rate law, the rate constants in simple first order kinetics are expressed in per unit time (t^{-1}), while in second order reactions the rate constant is expressed per concentration per unit time (mol^{-1} t^{-1}).

In dimensional analysis, units on the left and right-hand sides of expressions must have the same units (or dimensions). There are certain rules for combining units when checking consistency in units. Only like units can be added or subtracted, thus the expression $S + k_1$ cannot be summed because the units of S are likely to be mol l^{-1} and the units for k_1, t^{-1}. Even something as innocent looking as $1 + S$ can be troublesome because S has units of concentration but the constant value '1' is unit-less. Quantities with different units can be multiplied or divided with the units for the overall expression computed using the laws of exponents and treating the unit symbols as variables.

Example 2.2

Determine the overall units for the expression, $k_1\ S/K_m$ where the units for each variable are $k_1(t^{-1}\ l)$, $S(\text{mol}\ l^{-1})$ and $K_m(\text{mol}\ l^{-1})$.

We first write out the expression in terms of the individual units:

$$t^{-1} l\ \text{mol}/(\text{mol}\ l^{-1})$$

by treating the symbols are algebraic variables we can see that the symbol mol will cancel and using exponents rules we can bring the l^{-1} term up to the the denominator to yield:

$$t^{-1} l^2$$

In exponentials such as e^x, the exponent term must be dimensionless, or at least the expression should resolve to dimensionless, thus e^{kt} is permissible but e^k is not if for example k is a first-order rate constant. Trigonometric functions will always resolve to dimensionless quantities because the argument will be an angle which can always be expressed as a ratio of lengths which will by necessity have the same dimension.

2.9 Classification of Models

In addition to classifying models as discrete/continuous and detemrinsitic/stochastic there are additional properties of models that can be used to categorize them (Table 2.4).

Dynamic and Static Models

A static model is one where the variables of the system do not change in time. For example, a circuit made up of only resistors can be modeled as a static system because there are no elements in the circuit that can store or dissipate charge thus currents and voltages are considered instantaneous without any time evolution. Most interesting model are dynamic.

1. Linear or Non-Linear
2. Dynamic or Static
3. Time invariant or time dependent
4. Lumped or distributed parameter models

Table 2.4 Additional categories of models

Time Invariant Systems

A time invariant model is one where the model does not explicitly depend on time. This means that given a time invariant model, running the model at $t = 0$ or $t = 10$ makes no different to the time evolution of the model. If a parameter of the system depends on time then the model is called time dependent. An example of a time dependent model is where we apply a drug in the form of a pulse and the duration of the pulse depends on when the drug is administered. An example of a time dependent non-biological model is a parking lot where the price of a ticket depends on the time of day. We will have more to say about time invariant systems in a later chapter when we will talk about linear time invariant systems (LTI).

Lumped and Distributed Parameter Models

Many complex models can be approximated with a single number. For example we often describe a resistor using a single value, its resistance. In reality, the resistor has a length, a diameter and a chemical composition. The resistance is a function of all these properties that make up the resistor. We could model the resistor by slicing up the resistor in to many small compartments and compute the resistance as a systemic property. In the former case we have what is called a lumped parameter model, in the second case a distributed parameter model.

2.10 Linear and Non-Linear Models

When we use mathematics to describe physical systems there is a great divide that separates on one side **linear** and on the other **nonlinear models**. This separation is fundamental and places hard limits on what we can and cannot do with mathematical analysis.

One one side we have linear systems, where inputs to a system result in their weighted sum appearing in the outputs. The output is a superposition of the inputs. The simplest linear system is given by the relation $y = ax$, where x is the input and y the output. We know this is linear for the following reason. Let us apply two separate inputs, x_1 and x_2 to this system. This gives us outputs, ax_1 and ax_2 respectively. If we now apply the sum of the inputs, $x_1 + x_2$ we get $a(x_1 + x_2)$ as the output, which as we can see is simply the sum of the separate inputs.

$$ax_1 + ax_2 = a(x_1 + x_2)$$

This property is called the property of **additivity** and can be generalized as follows. A mathematical model, $f(x)$, shows additivity if the following is true:

$$f(x_1 + x_2 + \ldots) = f(x_1) + f(x_2) + \ldots$$

This states that the sum of multiple inputs applied simultaneously is equivalent to applying the inputs separately. Nonlinear systems do not follow this rule. Strictly speaking a linear system also needs to satisfy **homogeneity** (or scaling), that is, $f(ax) = af(x)$. Combining additivity and homogeneity gives us the general rule of linearity called **superposition**:

$$\boxed{f(ax_1 + bx_2 + \ldots) = f(ax_1) + f(bx_2) + \ldots}$$

Any system that satisfies superposition is a linear system. Any system that does not, is a nonlinear system. Table 2.5 illustrates some functions that are nonlinear.

x^n	$\sqrt[n]{x}$
xy	$\sin(x)$
e^x	$\log(x)$
$(dy/dy)^n$	$V_m S/(S + K_m)$

Table 2.5 Examples of nonlinear functions

Example 2.3

Show that the function e^x is nonlinear.

We first apply separate inputs, x_1 and x_2, to the function and compute the sum of the output, that is:

$$e^{x_1} + e^{x_2}$$

We next take the sum of the inputs, $x_1 + x_2$ and apply the sum to the function, that is:

$$e^{x_1+x_2}$$

To obey additivity, the two expressions much be equal. However, $e^{x_1+x_2} = e^{x_1}e^{x_2}$ which is not the same as $e^{x_1} + e^{x_2}$. Therefore e^x is a nonlinear function.

Similarly we can also easily show that homogeneity ($f(ax) = af(x)$) is not true because if should be evident that:

$$ae^x \neq e^{ax}$$

To appreciate the difference between linear and nonlinear functions, consider the system $y = x^2$. Let us apply two separate inputs, x_1 and x_2 to give outputs x_1^2 and x_2^2. If we now apply the inputs simultaneously, that is $y = (x_1 + x_2)^2$, we obtain $x_1^2 + x_2^2 + 2x_1x_2$. We see that the output is not simply $x_1^2 + x_2^2$ but includes an additional term, $2x_1x_2$. This term is the nonlinear contribution. Imagine that this difference now enters further nonlinear processes, leading to further changes. Eventually the output looks nothing like the input. This small change makes most nonlinear systems difficult to understand and analyse.

Unless the system has an infinite number of solutions (degenerate) or has the trivial solution (that is the solution is zero), linear systems will admit only one solution. In contrast it is possible for nonlinear systems to admit multiple solutions, that is given a single input, a nonlinear system can regurgitate one of a number of possible distinct outputs. To makes matters worse, in the majority of mathematical models we find in biochemical networks, it is not even impossible to find the solutions mathematically. That is we cannot actually describe mathematically how an output depends on an input other than by doing brute-force computer simulation. Understanding nonlinear systems whether we find them in biology or elsewhere is a huge unresolved problem.

Whereas there is a complete theory of linear systems, no such thing exists for nonlinear systems. When dealing with nonlinear systems we are often forced to use computer simulation.

There is one approach we can take that can help use deal with nonlinear models. If we were to draw a nonlinear curve on a graph and zoom in closer the curve would eventually look like a straight line; that is we can turn a nonlinear system into a linear one but only in small regions of the system's behavior where linearity dominates. This process, is called linearization and is a powerful technique for studying nonlinear systems.

2.11 Linearization

When modeling nonlinear systems, we have two options, to simulate or to linearize. Simulation will be considered later, here we will briefly look at a technique called linearization. To linearize a model means replacing the nonlinear version with a linear approximation which is easier to understand. It should be emphasized that in the process we loose valuable information but enough information is preserved to make linearization an extremely useful and popular tool.

One of the most useful results in mathematics is the **Taylor series** (See Appendix B for review). This is a way of approximating a mathematical function by using an infinite polynomial series such as the following:

$$f(x) = c_o + c_1 x + c_2 x^2 + c_3 x^3 + \ldots \tag{2.1}$$

We can represent any continuous function using such a polynomial. For example, we can represent sin(x) using the formula:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \tag{2.2}$$

Without going into the details (See Appendix B) the Taylor series is a means for defining the c_i terms in the polynomial series (2.1) given any continuous function. The Taylor series is always defined around some operating point, x_o and the distance from that operating point, x. The Taylor series is given by:

$$f(x) = f(x_o) + \frac{\partial f}{\partial x}(x - x_o) + \frac{1}{2!}\frac{\partial^2 f}{\partial x^2}(x - x_o)^2 + \ldots + \frac{1}{n!}\frac{\partial^n f}{\partial x^n}(x - x_o)^n + \ldots \tag{2.3}$$

The various derivatives in the Taylor series must be evaluated at x_o. For this to work, we must be able to derive, at least in principle, all the derivatives. This means the function $f(x)$ must be continuous, that is there should be no holes or sudden breaks (discontinuity) in the curve that the function describes. The number of terms in the Taylor series determines the degree of approximation, the fewer terms the more approximate the series is. For example, the most approximate expression is given by using only the first term, $f(x_o)$. However, $f(x_o)$ is a constant so this represents a very poor approximation. To make the approximation more useful we include the first two terms of the Taylor series:

$$f(x) \approx f(x_o) + \frac{\partial f}{\partial x}(x - x_o) \tag{2.4}$$

Provided x is close to x_o, the approximation is good. For example, let us form the Taylor series for the function, $y = \sin(x)$ around $x_o = 0$. We

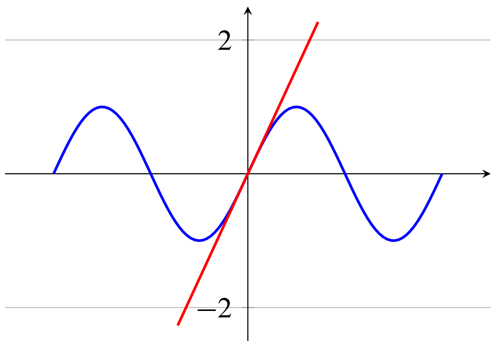


Figure 2.6 Linearized $\sin(x)$ function represented by the straight line through zero.

should recall that $\sin(0) = 0$ and $\cos(0) = 1$, then writing out the Taylor series:

$$y = \sin(0) + \frac{\partial \sin(x_o)}{\partial x}(x-0) + \frac{1}{2!}\frac{\partial^2 \sin(x)}{\partial x^2}(x-0) + \ldots$$

$$y = 0 + 1x + 0 - \frac{1}{3!}x^3 + 0 + \frac{1}{5!}x^5 + \ldots$$

That is:

$$y = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$$

Note this is the same as equation 2.2. The linear approximation is given by the first two terms:

$$y = \sin(0) + \frac{\partial \sin(x_o)}{\partial x}(x-0)$$

Since $\sin(0) = 0$ and $\partial \sin(x_0)/\partial x = \cos(0) = 1$, the linear approximation is therefore $y = x$, a straight line running through the origin (Figure 2.6). We have linearized the sin function.

To illustrate linearization with another example consider the simple nonlinear function, $y = x^2$. To linearize we must first choose an operating point around which we will linearize, for example, $x_o = 2$. According to the second term in the Taylor series we need to find the derivative, df/dx so that the first two terms of the Taylor series (Equation 2.4) become:

$$f(x) = f(2) + 2x(x-2) = 4 - 4x + 2x^2$$

This gives a second order polynomial as the approximation. To obtain the linear approximation we evaluate the derivative at the operating point ($x_o = 2$), that is $df/dx = 2x = 4$ so that the final linear approximation is then given by:

$$f(x) = 4 - 4(x-2) = 4x - 4$$

Figure 2.7 shows the original nonlinear function together with the linear approximation.

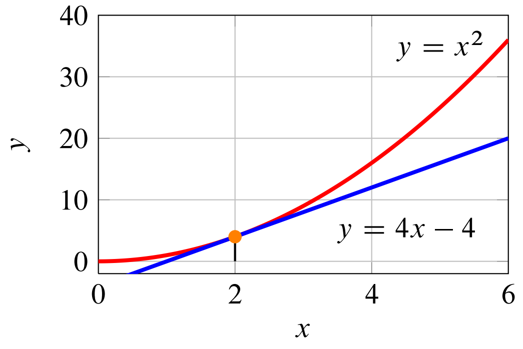


Figure 2.7 Taylor series approximation of $y = x^2$ at the operating point, $x_o = 2$.

Equation 2.4 can also be written in the form

$$f(x) \simeq f(a) + \frac{df}{dx}\delta x$$

where $\delta x = (x - x_o)$. If the equation f is a function of more than one variable then additional terms appear. For example the linearization of $f(x, y)$ will give:

$$f(x, y) = f(x_o, y_o) + \frac{\partial f}{\partial x}\delta x + \frac{\partial f}{\partial y}\delta y \tag{2.5}$$

As before, the derivatives must be evaluated at the operating point.

Example 2.4

Linearize the following equation at $x_o = 2$:

$$y = \frac{x^3}{x + 1}$$

To linearize we must apply equation 2.4. We first compute, $f(x_o)$. Since $x_o = 2$, then:

$$f(x_o) = 8/3$$

Next we form the derivative $\partial f/\partial x$:

$$\frac{\partial f}{\partial x} = \frac{2x^3 + 3x^2}{(x + 1)^2}$$

At $x_o = 2$ the derivative is given by:

$$\frac{\partial f(2)}{\partial x} = \frac{28}{9}$$

Inserting $f(x_o)$ and the derivative into:

$$f(x) \approx f(x_o) + \frac{\partial f}{\partial x}(x - x_o)$$

yields:

$$f(x) \approx \frac{8}{3} + \frac{28}{9}(x - x_o) = \frac{8}{3} + x\frac{28}{9} - \frac{56}{9} = \frac{28x - 32}{9}$$

Example 2.5

Linearize the following equation at $x_o = 1$ and $y_o = 0$:

$$f(x, y) = x^2 - 2xy - \sin(y)$$

To linearize a two dimensional system we must apply equation 2.5. We first compute, $f(x_o, y_o)$. Since $x_o = 1$ and $y_o = 0$, then:

$$f(x_o, y_o) = 1$$

Next we form the two derivatives $\partial f/\partial x$ and $\partial f/\partial y$:

$$\frac{\partial f}{\partial x} = 2x - y \qquad \frac{\partial f}{\partial y} = -2x - \cos(y)$$

At $x_o = 1$ and $y_o = 0$ the derivatives are given by:

$$\frac{\partial f(1,0)}{\partial x} = 2 \qquad \frac{\partial f(1,0)}{\partial y} = -3$$

Inserting $f(x_o, y_o)$ and the derivatives into:

$$f(x, y) = f(x_o, y_o) + \frac{\partial f}{\partial x}(x - x_o) + \frac{\partial f}{\partial y}(y - y_o)$$

yields:

$$f(x, y) \approx 2x - 3y - 1$$

2.12 Approximations

By their very nature, models involve making assumptions and approximations. The best modelers are those who can make the most shrewd and reasonable approximations without compromising a model's usefulness. There are however some kinds of approximations which are useful in most problems, these include:

- Neglecting small effects.

- Assuming that the system environment is unchanged by the system itself.
- Replacing complex subsystems with lumped or aggregate laws.
- Assuming simple linear cause-effect relationships where possible.
- Assuming that the physical characteristics of the system do not change with time.
- Neglecting noise and uncertainty.

Neglecting small effects. This is the most common approximation to make. In many studies there will always be parts of the system that have a negligible effect on the properties of the system, at least during the period of study. For example, the rotation of the earth, the cycle of the moon or the rising and setting of the sun will most likely have a negligible influence when studying the action of an enzyme. Assuming of course we are not studying circadian rhythms.

Assuming that the system environment is unchanged by the system itself. This is a basic assumption in any study. The minute a system starts to affect the environment in an uncontrolled way we have effectively extended the system boundaries to include more of the environment. It will often be the case that the interface between the environment and the system will not be perfect so that there will be some effect that the system has on the environment. So long as this effect is small we can assume that the environment is not affected by the system.

Replacing complex subsystems with lumped or aggregate laws. Lumping subsystems is a commonly used technique in simplifying cellular models. The most important of these is the use of aggregate rate laws, such as Michaelis-Menten or Hill like equations to model cooperativity. Sometimes entire sequences of reactions can be replaced with a single rate law.

Assuming simple linear cause-effect relationships. In some cases it is possible to assume a linear cause-effect between an enzyme reaction rate and the substrate concentration, this is especially true when the substrate concentration is below the K_m of the enzyme. Linear approximations make it much easier to understand a model.

Physical characteristics do not change with time. A modeler will often assume that the physical characteristics of a system do not change, for example the volume of a cell, the values of the rate constants or the temperature of the system. In many cases such approximations are perfectly reasonable.

Neglecting noise and uncertainty. Most models make two important approximations. The first is that noise in the system is either negligible or unimportant. In many non-biological systems such an approximation might be quite reasonable. However cellular phenomena often operate at the molecular level. Biological systems are susceptible to noise generated from thermal effects as a result of molecular collisions. For many systems the large number of particles ensures that the noise generated in this way is insignificant and in many cases can be safely ignored. For some systems such as prokaryotic organisms, the number of particles can be very small. In such cases the effect of noise can be significant and therefore must be included as part of the model.

2.13 Example Model

Before we leave this chapter let us look at building a model of a simple chain of four enzyme catalyzed reactions (Figure 2.8). Let us construct a mathematical model of this system.

$$\begin{tikzcd} S_1 \arrow[r, "v_1"] & S_2 \arrow[r, "v_2"] & S_3 \arrow[r, "v_3"] & S_4 \arrow[r, "v_4"] & S_5 \end{tikzcd}$$

Figure 2.8 Simple Straight Chain Pathway.

The first thing we can do is decide where the boundary of the pathway is, assuming there will be one. A convenient place to have a boundary is the start and end metabolite of the pathway, that is S_1 and S_5. We will assume that these two metabolites are **fixed** and are unaffected by the system (Figure 2.9).

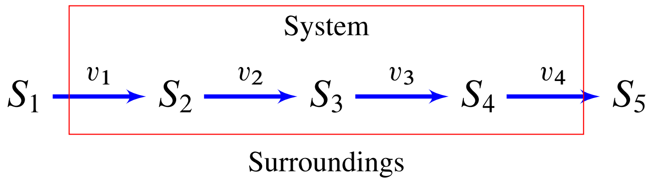


Figure 2.9 Simple Straight Chain Pathway with system shown in a box and S_1 and S_5 outside the system. We assume S_1 and S_5 are fixed.

The metabolites that can change in time include S_2, S_3 and S_4. Recall that such metabolites are called the **state variables** or **floating species**. We can now write the differential equations that represent the rates of change of S_2, S_3 and S_4. Note that there will be no differential equations assigned to S_1 and S_5 because these are fixed and unchanging. According to mass-balance (Cover in more detail in the next chapter) the following differential equations must be true:

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

$$\frac{dS_3}{dt} = v_2 - v_3$$

$$\frac{dS_4}{dt} = v_3 - v_4$$

Next we must decide on the rate laws, v_1, v_2, v_3 and v_4. This is possibly the most difficult part to building a model and a detailed examination of the literature will need to be done to decide what are the most appropriate rate laws to use. The companion text book, "Enzyme Kinetics for Systems Biology" [92] gives much more detail on rate laws in general. Here a mixture of rate laws will be used to illustrate the kinds of rate laws that one might employ. For example a simple reversible mass-action rate law may be used for the first reaction v_1, that is:

$$v_1 = k_1 S_1 - k_2 S_2$$

This rate law introduces two new parameters, the rate constants, k_1 and k_2. These are fixed and unaffected by the model. For the second reaction, let us use a simple allosteric regulated rate law. That is we will assume that the reaction v_2 is allosterically inhibited by S_4. For this we can use the simplest exclusive Monod, Wyman, Changeax model [75]:

$$v_2 = V_m \frac{S_2 \left(1 + S_2/K_m\right)^4}{\left(1 + S_2/K_m\right)^4 + L \left(1 + S_4/K_I\right)^4}$$

where the Hill coefficient is equal to four, L is the allosteric constant, K_I is the inhibition constant, K_m is the substrate concentration at half-maximal activity and V_m the maximal velocity.

The third rate law will be a simple irreversible but product inhibited Michaelis-Menten rate law, that is:

$$v_2 = V_m \frac{S_3}{S_3 + K_m \left(1 + S_3/K_p\right)}$$

where V_m is the maximal velocity of the reaction, K_m is the Michaelis constant and K_p is the product inhibition constant. The last reaction, v_4 will be assigned a simple irreversible mass-action rate law:

$$v_4 = k_3 S_4$$

where k_3 is the rate constant. In total the model has ten parameters, two boundary species and three state (or floating) species. The model can be completed by assigning values to all the parameters, boundary species and initial conditions to the state variables. Once the model is described it can be entered into a simulation tool such as Jarnac [91], See Appendix E or JDesigner [96, 6] and the evolution of the system in time studied.

Further Reading

1. Sauro HM (2011) Enzyme Kinetics for Systems Biology. ISBN: 978-0982477311

Exercises

1. Choose from the following options. A model is:
   (a) an attempt to form an exact replica of reality.
   (b) something that bears no resemblance to the real system.
   (c) a simplification of the real world.

2. List the three most desirable attributes of a model.

3. When we "validate" a model which of the following do we most likely mean:
   (a) We show that the model represents the truth about the real system.
   (b) We increase our confidence in the model's predictive power.
   (c) We prove that the model is correct.

4. Explain the difference between accuracy and predictability in relation to the quality of a model.

5. The authors of a published biochemical model claim that their model has been validated, what do they mean by this?

6. *E. coli* can be considered a cylindrical volume with length 2 μm and diameter 1 μm. A reaction is known to occur in E. coli with an intensive rate of 0.5 mmol $s^{-1}\ l^{-1}$, what is the rate of reaction per volume of *E. coli*? If Avogadro's number is 6.022×10^{23}, express the rate in terms of molecules converted per second per *E. coli*.

7. Show that the following functions are nonlinear with respect to x:
   (a) $\sin(x)$
   (b) e^x
   (c) $V_m x/(x + K_m)$

8. Linearize the following functions:

(a) $4x^2 + 6x - 10$ at $x = 1$

(b) $V_m x/(x + K_m)$ at $x = 0$ and $x = K_m$

3 Cellular Networks

The study of cellular networks is one of the defining characteristics of systems and synthetic biology. Such networks involve the coordinated interaction of thousands of molecules that include nucleic acids, proteins, metabolites and other small molecules. Descriptions of these elaborate networks can be found in text books, on wall charts, and more recently in databases such as EcoCyc, RegulonDB, KEGG or as shown in Table 3.1.

3.1 Overall Organization

Biological networks can be organized into three broad categories (Fig. 3.4), gene regulatory, protein and metabolic networks. In the metabolic category, small molecules are chemically transformed by enzymes. These molecules – or metabolites – serve either as energy sources or as building blocks for more complex molecules, particularly polymers such as polysaccharides, nucleic acids and proteins.

The protein networks constitute a major part of the decision making and nano-machine apparatus of a cell. We can divide the decision making protein networks into two subgroups. One subgroup involves transcription factor proteins that regulate gene expression, forming what are called gene

Table 3.1 Online *E. coli* resources

Online Resource	URL
EcoCyc	`http://ecocyc.org/`
RegulonDB	`http://regulondb.ccg.unam.mx/`
KEGG	`http://www.genome.jp/kegg/`
STRING	`http://string.embl.de/`

regulatory networks (GRNs). The second subgroup constitutes the signalling pathways that integrate information about the internal and external environments and modulate both the metabolic and gene regulatory networks.

The metabolic, protein, and gene regulatory networks each have a characteristic mode of operation and differ by the molecular mechanisms employed and their respective operating time scale. In general, metabolic networks operate on the smallest time scale, followed by protein networks and gene regulatory networks.

This picture is of course a simplified view. For example it omits the extensive RNA network that may be present particularly in eukaryotic cells. Protein signalling networks are also involved in a variety of other related functions such as cytoskeleton control and cell cycle regulation. In addition, there is considerable overlap between the different systems with gene regulatory networks and protein control networks interlinked [11].

3.2 Network Representation

There are different ways to represent cellular networks depending on how the information will be used and what kinds of questions are asked. Traditionally cellular networks have been described using a **stoichiometric** formalism. Such networks are mechanistic in nature, consistent with the laws of mass conservation and will often include kinetic laws describing transformations of species from one form to another through binding/unbinding or molecular reorganization. In recent years an alternative representation, which might be termed **non-stoichiometric**, has gained significant popularity with the advent of high-throughput data collection. Non-stoichiometric networks, of which there are a great variety, include interaction networks which describe the relationship, usually via some physical interaction but sometimes also functional, between molecular species or functional entities such as genes or proteins. Non-stoichiometric networks tend to be more course grained compared to stoichiometric networks but their study has proved to be very popular due in large part to the availability of vast new data sources. That, coupled with the unprecedented interest in networks in general has made the study of non-stoichiometric networks an intellectually fruitful area of study.

In this book we will be primarily concerned with modeling stoichiometric networks.

3.3 Metabolic Networks

The first cellular networks to be discovered were the metabolic pathways such as Glycolysis in the 1930s and the Calvin cycle in the 1940s. The first metabolic pathways were elucidated by a combination of enzymatic inhibitors and the use of radioisotopes such as Carbon-14. The Calvin cycle for example was discovered by following the fate of carbon when algae were exposed to ^{14}C-labeled CO_2. With the development of microbial genetics, significant progress was made in uncovering other pathways by studying mutants and complementing different mutants of a given pathway to determine the order of steps. The reaction steps in a metabolic pathway are catalysed by enzymes and we now know there are thousands of enzymes in a given organism catalyzing a great variety of pathways. The collective sum of all reaction pathways in a cell is referred to as metabolism and the small molecules that are interconverted are called metabolites.

Traditionally, metabolism is classified into two groups, anabolic (synthesis) and catabolic (breakdown) metabolism. Coupling between the two metabolic groups is achieved through cofactors of which a great variety exist. Two widely distributed cofactors include the pyridine nucleotides in the form of NAD^+ and $NADP^+$, and the adenine nucleotides in the form of ATP, ADP and AMP. These cofactors couple redox and phosphate respectively by forming reactive intermediates that enables catabolism to drive anabolism. A primary catabolic process is cellular respiration where starting molecules such as glucose are oxidized in a stepwise fashion. The energy released is captured in the form of ATP and the oxidized products of water and carbon dioxide are released as waste. ATP can be used in turn to drive anabolic processes such as amino acid or lipid biosynthesis. In general, metabolic pathways tend to be regulated via allosteric regulation. This is where a metabolite can regulate the reaction rate of an enzyme by binding to a site on the enzyme other than the catalytic site. Such interactions form a network of feedback and feedforward regulation. Figure 3.2 shows a metabolic pathway of glycolysis from Lactococcus lactis. On the left, glucose enters the cell which is then converted in a series of reactions to ethanol and a variety of other small molecules.

Metabolic networks are by far the fastest (excluding ion transfer mechanisms) in terms of their response to perturbations and can operate in a time scale from microseconds to seconds. This reflects the need to rapidly adjust the supply of molecular building blocks and energy as supply and demand fluctuate. Physically the rapid response of metabolic networks is achieved by allosteric control where the fast diffusion of small molecules can bind and rapidly alter the activity of selected enzymes.

Figure 3.3 shows a section of the glycolytic pathway which converts glucose to pyruvate with the production of ATP and NADH. The diagram also shows the many negative and positive feedback and feedforward regulatory loops in glycolysis. Not all of these are present in all organisms, however many are. Note the six regulatory signals that converge on 6-Phosphofructose-1-kinase (also known as phosphofructokinase) and Fructose Bisphosphatase (Labelled 2 and 3). One of the aims of metabolic control analysis is to quantify regulation and to understand the operational principles of such control.

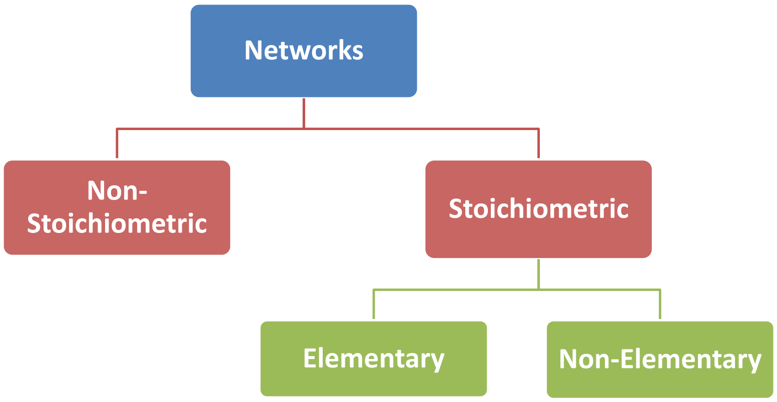


Figure 3.1 Cellular networks are often represented using two common approaches, non-stoichiometric and stoichiometric. Non-stoichiometric networks are characterized by a lack of stoichiometric information and mass conservation. Stoichiometric networks are classified according whether they are elementary or not. Elementary networks are those where the reactions cannot be broken into simpler forms. Non-elementary networks may have one or more reaction steps which represent an aggregate of two or more elementary reactions, the aggregation being dependent on some particular assumptions such as quasi-steady state or equilibrium.

3.4 Protein Networks

Protein networks are by far the most varied networks found in biological cells. They range from proteins involved in controlling gene expression, the cell cycle, coordinating and processing signals from the internal and external environments, to highly sophisticated nano-machines such as parts of the ribosome or the bacterial flagella motor.

Protein networks can be studied on different levels, broadly classified as either stoichiometric or non-stoichiometric networks. The non-stoichiometric networks can be as simple as considering the physical associations between different proteins (often through the formation of protein complexes). Such networks, also termed interaction networks, have been elucidated

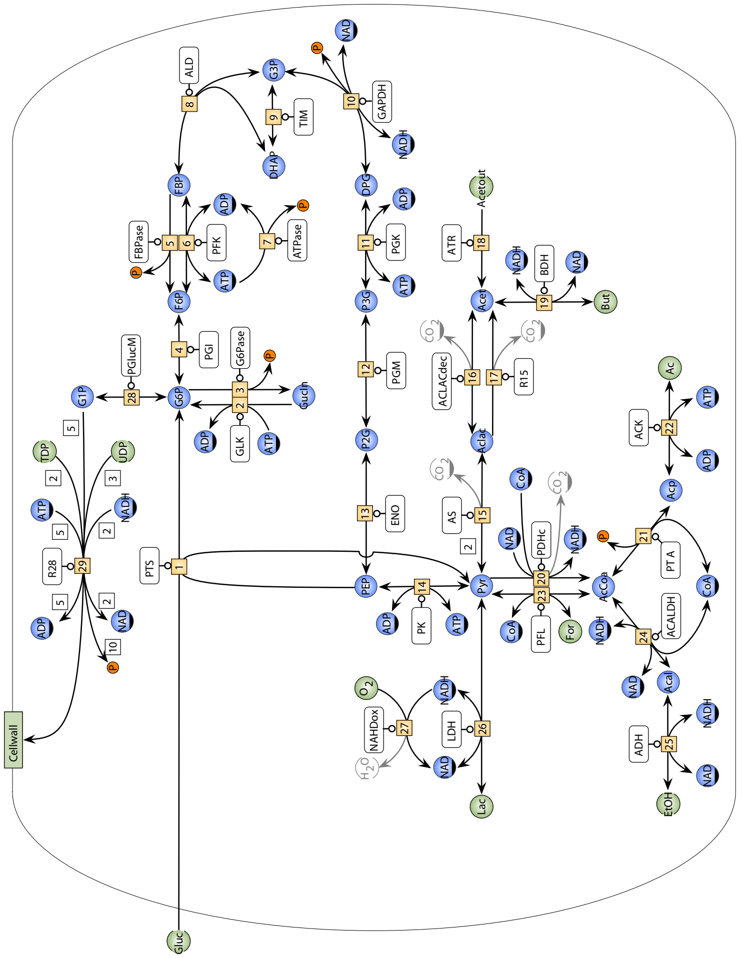


Figure 3.2 Metabolic Pathway: Metabolic pathway image from JWS online (Jacky Snoep) with permission. The pathway depicts the glycolytic pathway from Lactococcus lactis using the Systems Biology Graphical Notation (SBGN) [60, 44].

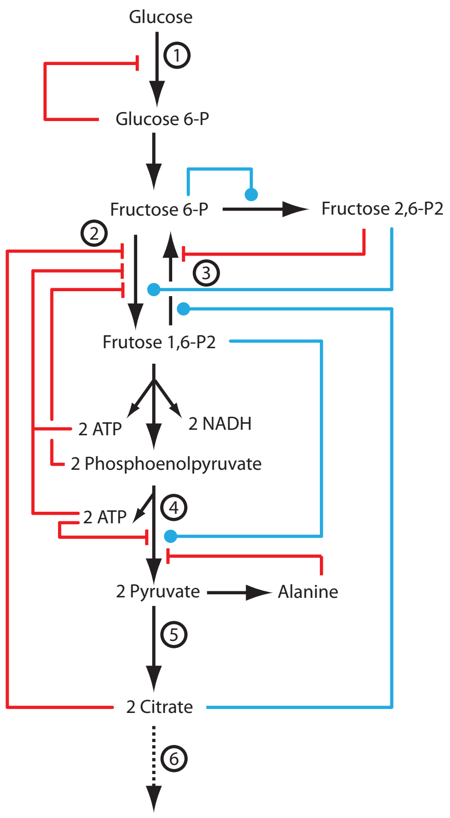


Figure 3.3 A section of glycolysis with negative and positive regulation shown. 1. Hexokinase; 2. 6-Phosphofructose-1-kinase; 3. Fructose bisphosphatase; 4. Pyruvate kinase; 5. Entry to Citric acid cycle; 6. To oxidative respiration.

largely with the help of high-throughput methods. An interaction is formed if two proteins, A and B are known to associate.

Another descriptive level involves functional and stoichiometric networks formed from a consideration of specific stoichiometric binding events, covalent modification (most notably phosphorylation) and degradation. Here two protein A and B might form a complex with a stoichiometric relation-

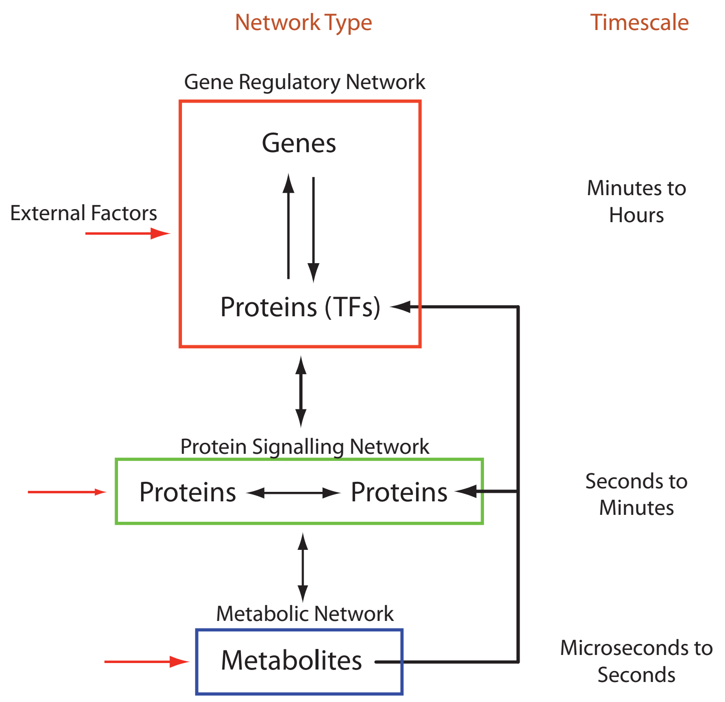


Figure 3.4 Network Overview. The figure illustrates the three main network layers, metabolic, protein and gene. TF – Transcription Factors.

ship and given association constant.

Protein-Protein Networks

Work on uncovering protein networks has be ongoing since the 1950s and considerable detail has accumulated on many different pathways across different organisms. Traditional methods, though laborious [26, 24] have been used extensively to gain detailed knowledge on phosphorylation sites, protein structure, the nature of membrane receptors and the constitution and function of protein complexes. More recently high-throughput methods, though more course grained, have been used to elucidate large swaths of protein-protein interaction networks. For example, in yeast, large scale studies have identified approximately 500 different protein complexes [36, 57] and their relationships to each other.

A popular high-throughput technique that has been used to uncover protein-protein interaction networks is the Yeast two-hybrid method [30, 84]. Other methods such as phage display [101, 39] and particularly affinity purification and mass spectrometry have also been successfully employed [36, 57]. The Yeast two-hybrid method (Figure 3.5) is based on the idea that eukaryotic transcriptional activators consist of two domains, a DNA binding domain (DB) and an activation domain (AD). The activation domain is responsible for recruiting the RNA polymerase to begin transcription. What is remarkable is that the two domains do not have to be covalently linked in order to function correctly, they need to be in close proximity. The Yeast two-hybrid method is based specifically on this property.

Assume we want to know whether two proteins, X and Y interact with each other. In the two-hybrid method, protein X is fused with the DB domain (known as the bait protein) and the second protein, Y, is fused with the AD domain (known as the prey protein). These two fused proteins are now expressed in Yeast. If the two proteins X and Y interact in some way, they will bring the DB and AD domains close to each other resulting in an active transcriptional activator. If the gene downstream of the DNA binding sequence is a reporter gene, then the interaction of X and Y can be detected.

A common reporter gene is the lacZ gene which codes for β-galactosidase and which produces a blue coloring in Yeast colonies through the metabolism of exogenously supplied X-gal (5-bromo-4-chloro-3-indolyl-β-D-galactoside).

There are some caveats with the yeast two-hybrid method however. Although two proteins may be observed to interact, the protein in their natural setting may not be expressed at the same time or may be expressed but in different compartments. In addition, using the method to identify interactions between non-yeast proteins may be invalid because of the alien environment of yeast cells. As with many high-throughput methods caution is advised when interpreting the data.

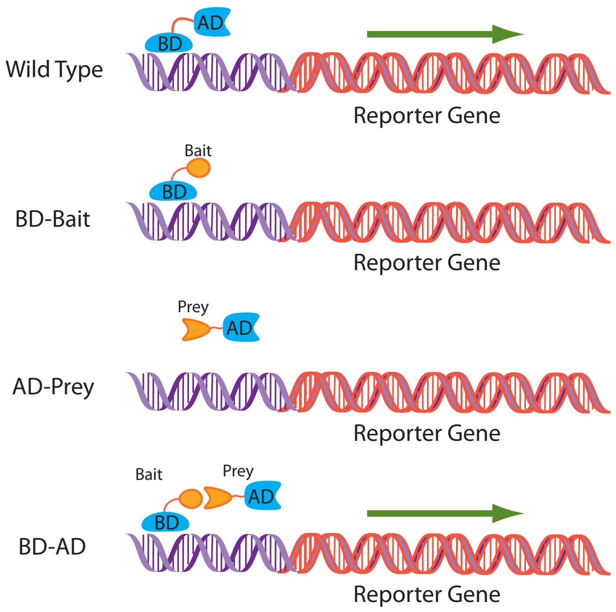


Figure 3.5 Yeast two-hybrid. The wild-type transcription factor is composed of two domains, BD and AD. Both are essential for transcription. Two fusion proteins are made, BD-Bait and AD-Prey. Bait and Prey are the two proteins under investigation. If the two protein Bait and Prey interact, BD and AD are brought together resulting in a viable transcription factor that can be used to express a reporter gene.

Using techniques such as yeast two-hybrid, one of the first interaction graphs to be published was the protein interaction graph of Saccharomyces cerevisiae [107, 50]. Subsequent analysis of this map was conducted by Jeong et al. [51] and included 1870 proteins nodes and 2240 interaction edges. Such graphs give a birds-eye view of protein interactions (Fig. 3.6).

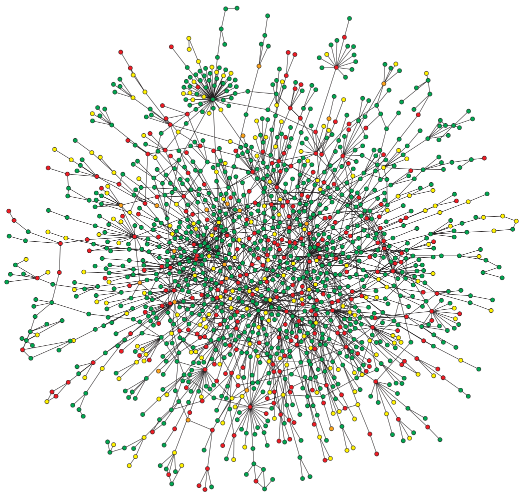


Figure 3.6 The poster child of interaction networks, one of the earliest yeast protein interaction networks generated from yeast two-hybrid measurements. Each node represents a protein and each edge an interaction. In addition, the graph nodes have been annotated such that red indicate lethal phenotypic effect if removed, green non-lethal, orange slow growth, and yellow unknown. Adapted from Barabási and Oltvai [3] but originally published in arXiv and Nature [51].

Signalling and Control Networks

Many protein-protein networks operate as signal processing networks and are responsible for sensing external signals such as nutritional (for example by changes in glucose levels) or cell to cell signals such as insulin. Other signalling networks include control networks that are concerned with monitoring and coordinating internal changes, the most well known of these includes the cell cycle control network. Many external signals act by binding to cell-surface receptor proteins such as the large family of receptor tyrosine kinases and G-protein coupled receptors [56]. Once a signal is internalized through the cell-surface receptors, other proteins, including protein kinases and phosphatases continue to process the signal often in coordination with other signaling networks. Eventually the signalling pathway terminates on target proteins that leads to a change in the cell's behavior. Such targets can include a wide variety of processes such as metabolic pathways, ion channels, cytoskeleton, motor proteins and gene regulatory proteins.

The molecular mechanisms employed by signalling and control pathways include covalent modification, degradation and complex formation. Covalent modification, in particular, is a common mechanism used in signaling networks and includes a variety of different modifications such as phosphorylation, acetylation, methylation, ubiquitylation, and possibly others [14]. As a result the structure and computational abilities [97] of such networks is likely to be extremely elaborate. It has been estimated from experimental studies that in *E. coli*, 79 proteins can be phosphorylated [65] on serine, threonine and tyrosine side groups whereas in Yeast, 4000 phosphorylation events involving 1,325 different proteins have been recorded [86].

The cell cycle control network is an excellent example of a sophisticated protein control network that coordinates the replication of a biological cell. The cell cycle includes a number of common molecular mechanisms that are found in many other protein networks. These can be grouped into three broad types: phosphorylation, degradation and complex formation. Phosphorylation is a common mechanism for changing the state of a protein and involves phosphorylation on a number of sites on the protein surface including serine/threonine and tyrosine. In prokaryotes, histidine, arginine, or lysine can also be phosphorylated. Phosphorylation is mediated by kinases. The Human genome may have over 500 kinase encoding genes [68]. The effect of phosphorylation is varied but most often causes the altered protein to change catalytic activity, to change the protein's 'visibility' to other proteins, or to mark the protein for degradation. For example, src is a tyrosine kinase protein involved in cell growth. It has two states, active and inactive. When active, it has the capacity to phosphorylate other proteins. Deactivation of src is achieved by phosphorylation of a tyrosine group on the C-terminal end of the protein. Dephoshorylation of the tyrosine group by tyrosine phosphatases results in the activation of the protein.

Phosphorylation can also be used to inactivate enzymes such as glycogen synthase by the glycogen synthase kinase 3 protein. In the Yeast cell cycle, the protein Wee1 is phosphorylated and inactivated by the complex Cdc2-Cdc13. Active Wee1 in turn (i.e. the unphosphorylated form) can inactivate Cdc2-Cdc13 by phosphorylating the Cdc2 subunit.

In addition to changing the activity of proteins, phosphorylation can also be used to mark proteins for degradation. For example, the protein Rum1 that is part of the Yeast cell cycle control network can be phosphorylated by Cdc2-Cdc13. Once phosphorylated, Rum1 is degraded. Degradation itself is an important mechanism used in protein networks and allows proteins to be rapidly removed from a network according to the cell state. Degradation is usually mediated by ubiquitylation. For example, Cdc2-Cdc13, via Ste9 and APC is marked for degradation by ubiquitylation (Rum1 is similarly processed once phosphorylated). Once marked this way, such proteins can bind to the proteasome where they are degraded. Finally, binding of one protein to another can change the target protein's activity or visibility. An example of this is the inactivation of Cdc2-Cdc13 by Rum1. When unphosphorylated, Rum1 binds to Cdc2-Cdc13, rendering the resulting complex inactive.

Different combinations of these basic mechanisms are also employed. For example, phosphorylation of complexes can lead to the dissociation of the complex, or the full activity of a protein may require multiple phosphorylation events. Although signalling networks can appear highly complex and varied, most of them can be reduced to the three fundamental mechanisms of covalent modification, selective degradation and complex formation (Fig 3.7).

These examples highlight fundamental mechanisms by which protein control networks can be assembled into sophisticated decision making systems.

In higher eukaryotic cells, particulary human, around 2% of the protein-coding part of the genome is devoted to encoding protein kinases, with perhaps 10% of the coding region dedicated to proteins involved in signalling networks. It has also been suggested that possibly as much as 30% of all cellular proteins in yeast and human can be phosphorylated [22].

The actual size of the networks themselves is however even larger that

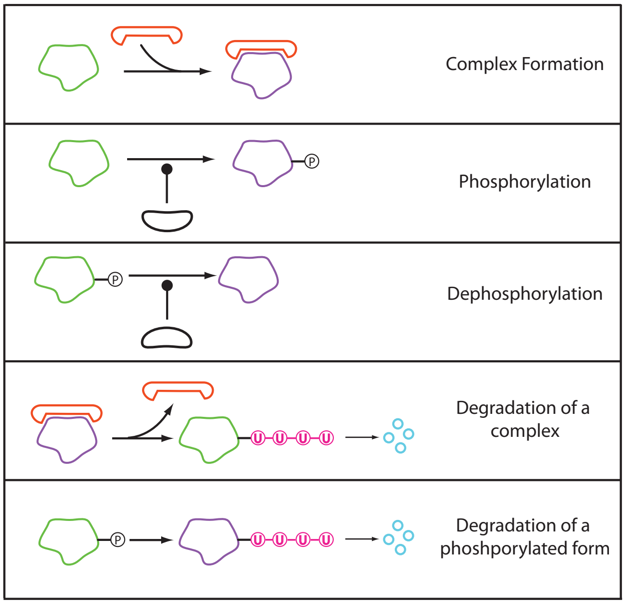


Figure 3.7 Fundamental Protein Mechanisms.

these numbers suggest because of the significant number of covalent variants and binding permutations. For example, p53, the tumor suppressor protein, has between 17 and 20 phosphorylation sites alone [105]. If every combination were phenotypically significant, (as unlikely as that might be), this amounts to at least 131,072 different states.

Ptacek and Snyder [87] have published a review on elucidating phosphorylation networks where more detailed information is provided.

3.5 Gene Regulatory Networks

In prokaryotes, the control of gene expression is relatively well understood. Transcription factors control gene expression by binding to special upstream DNA sequences called operator sites. Such binding results in the

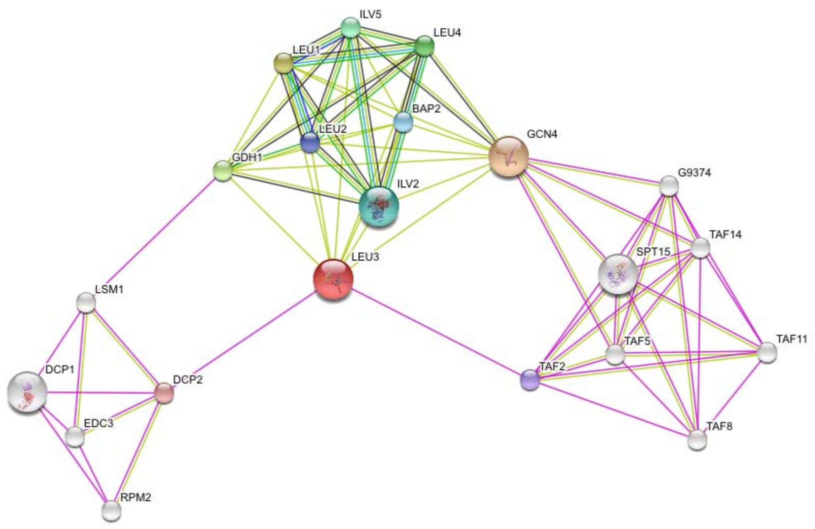


Figure 3.8 A Small Protein-Protein Interaction Map. This image was taken from the STRING web site (Search Tool for the Retrieval of Interacting Genes/Proteins, `http://string.embl.de/`). The image displays a small segment of the protein interaction map centered around LEU3, the transcription factor that regulates genes involved in leucine and other branched chain amino acid biosynthesis.

activation or inhibition of gene transcription. Multiple transcription factors can also interact to control the expression of a single gene. Such interactions can emulate simple logical functions (such as AND, OR etc.) or more elaborate computations. Gene regulatory networks can range from a single controlled gene to hundreds of genes interlinked with transcription factors forming a complex decision making circuit. Different classes of transcription factors exist, for example the binding of some transcription factors to operator sites is modulated by small molecules, the classical example being the binding of allolactose (a disaccharide very similar to lactose) to the lac repressor or cAMP to the catabolite activator protein (CAP). Alternatively a transcription factor may be expressed by one gene and either directly modulate a second gene (which could be its self), or via other transcription factors integrate multiple signals onto another gene. Additionally, some transcription factors only become active when phosphorylated or unphosphorylated by protein kinases and phosphatases. Like protein signaling and control networks, gene regulatory networks can be elaborate, structurally and computationally.

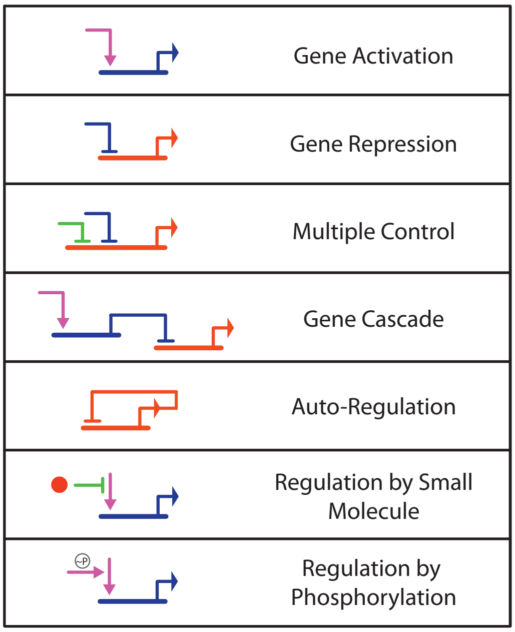


Figure 3.9 Simple gene regulatory patterns.

Significant advances have been made in developing high-throughput methods that can be used to determine protein-gene networks. Of particular interest are ChIP-chip [90, 2] and the more recently developed ChIP-seq [69] screening method – Chromatin immunoprecipitation microarray/Sequencing. ChIP works by treating cells with formaldehyde which crosslinks the DNA to the transcription binding protein if it is bound to the DNA. The cells are then lysed and the DNA fragmented into small 1 kB or less fragments. A specific antibody is then used to bind to the DNA-binding protein of interest and precipitate the protein and associated DNA fragment. The precipitated DNA pieces are released by reversing the crosslinking. In ChIP-chip, the released DNA pieces are hybridized to a microarray that enables the bound protein to be located on the Genome. A more recent version that is gaining popularity is ChIP-seq. In this procedure the microarray stage is abandoned and instead the released DNA pieces are sequenced. Once sequenced, the location on the Genome can be determined. These methods have been successfully used to determine the gene-protein network of a number of organisms, with yeast being the first [61]. Alternatively other approaches have focused on determining gene-protein networks from literature mining and careful curation or even prediction of putative binding sites.

In general, gene regulatory networks are the slowest responding networks in a cell and work from minutes to hours depending on the organism, with bacterial gene regulatory networks tending to operate more quickly.

The most extensive gene regulatory network database is RegulonDB [49, 32] which represents the gene regulatory network of *E. coli*. In depth reviews covering the structure of regulatory networks can be found in the works of Alon [99] and Seshasayee [98].

Although the description of the three main network types may give the impression that they act independently of each other ,this is most definitely not the case. In general, the different networks will often act together. For example, Figure 3.11 shows a small example taken from Caulobacter [16] showing a mixed gene regulatory and protein network.

3.6 Genome Sizes

How big are cellular networks? To answer this question we can look at whole genomes. The sizes of genomes vary considerable from the minuscule 159,662 bases of the symbiotic bacterium called Carsonella ruddii, which lives off sap-feeding insects, to the Whisk fern comprised of 2.5×10^{11} bases. Some of this size difference is related to the complexity of the organism, simpler organisms requiring fewer genes. However the correlation, although positive, is not entirely linear. For example *E. Coli* has roughly 4,300 genes on a genome of size 4.6 Mb, while humans

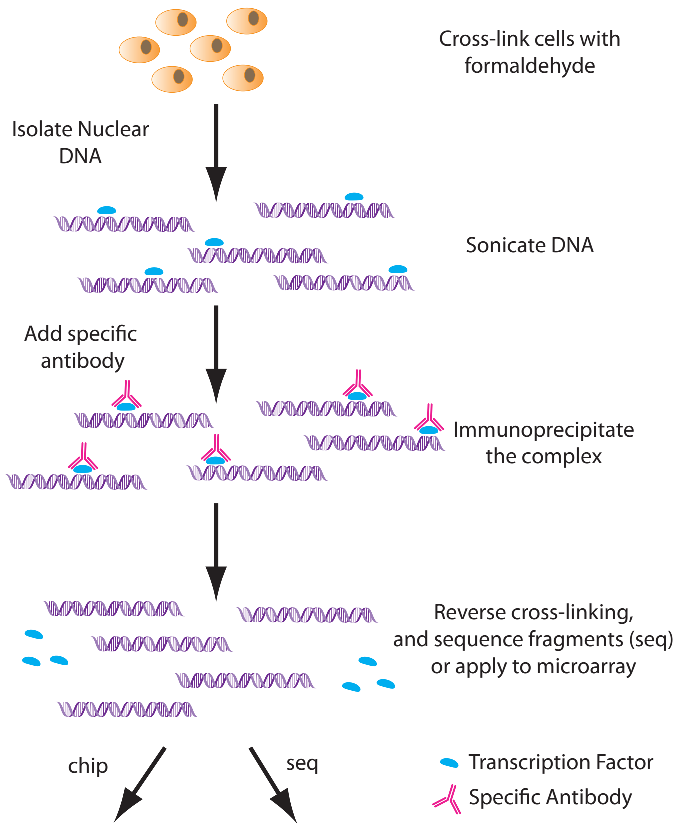

Cross-link cells with formaldehyde
Isolate Nuclear DNA
Sonicate DNA
Add specific antibody
Immunoprecipitate the complex
Reverse cross-linking, and sequence fragments (seq) or apply to microarray
chip
seq
Transcription Factor
Specific Antibody

Figure 3.10 ChIP-chip and ChIP-seq methods for determining transcriptional binding sites. Adapted from [69].

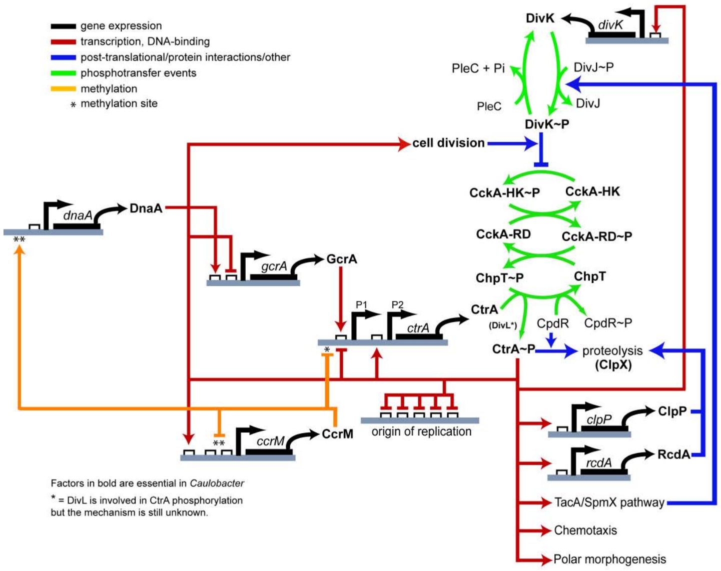


Figure 3.11 Example of a mixed network involving gene regulatory and protein phosphorylation networks in Caulobacter. Blunt ends to regulatory arcs indicate inhibition while arrow ends indicate activation. Image from BioMed Central [16].

have roughly 25,000 genes on a genome of about 3000 Mb. The *E. Coli* genome is quite dense with roughly 88% of the genome coding for proteins [104] with the remainder being made up of RNA coding, promoter sequences and so on. The human genome on the other hand is very sparse with only about 2% of the genome actually coding for protein. There is ongoing speculation as to why the human genome is so sparse and what the role of the other 98% might play. Some evidence suggests an extensive RNA based regulatory network [71] that is coded in at least some of the non-coding sequences (the so-called junk DNA).

Figure 3.12 shows an example of a small genome from Mycoplasma genitalium. This organism is a small parasitic bacteria that lives in primate genital and respiratory tracts and is the smallest known free-living bacteria. The genome of this organism has 521 genes in total, 482 or these code for protein with the remaining 39 reading frames coding for tRNA and rRNA.

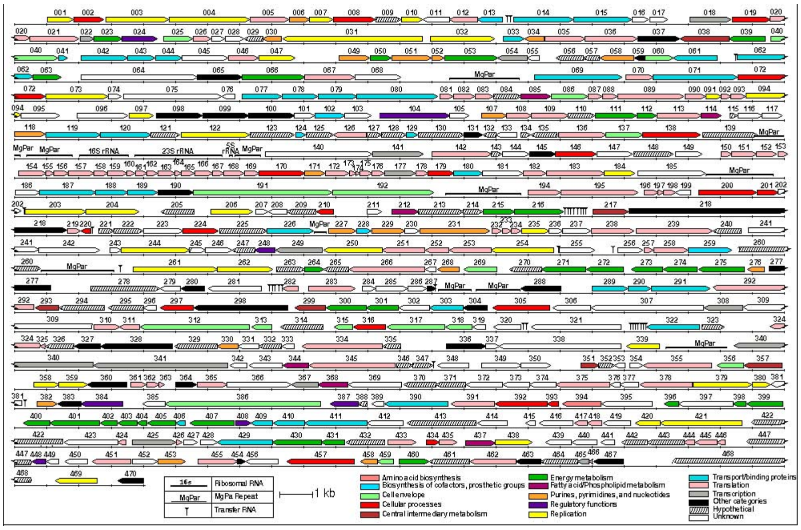


Figure 3.12 Small genome with 521 genes from Mycoplasma genitalium. Image taken from BioCyc.

The 482 genes that encode proteins in Mycoplasma genitalium include a wide variety of functions (Figure 3.13) that cover areas such as energy metabolism, replication and the cell envelope. Even for such a small organism there are still eight genes of unknown function.

Eukaryotic genes, especially Human, are also fragmented into segments called exons (coding) and introns (non-coding). This segmentation allows different forms of protein to be derived from the same gene by splicing together different exons. Although the apparent number of genes is of the order of 25,000, probably increases this number significantly [15, 108]. Finally, many proteins, particularly those involved in signalling pathways also have alternative forms due to covalent modification such as phosphorylation or methylation. This again increases the actual number of states. In others words the number of genes in a genome gives a lower limit to the size of a cellular network, particularly in eukaryotic organisms. The size

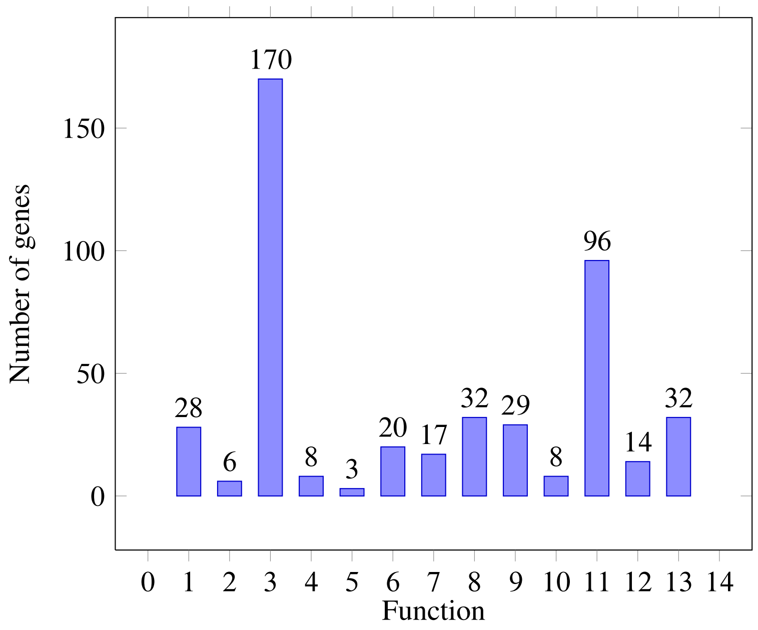


Figure 3.13 1: cell Envelope; 2: Regulatory; 3: Unknown; 4: Central Metabolism; 5: Cofactor Biosynthesis; 6: Purine/Pyrumdine metabolism; 7: Transcription; 8: Transport; 9: Replication/Repair; 10: Lipid Metabolism; 11: Translation: 12: Cellular Processes; 13: Energy Production.

of a given genome is therefore a poor indicator organism complexity. To give a better idea of the size and complexity of a small genome, let's look more closely at a specific one, *E. coli*.

3.7 *E. coli*

The bacterium *E. coli* is probably one the best understood organisms so is worth considering some of its features in detail. Much of the information provided here comes from the EcoCyc and RegulonDB online databases and their respective publications [52, 32].

Table 3.2 A comparison of genome sizes (base pairs) and estimated number of genes. Data from Taft and Mattick [104].

Organism	Genome Size	Est. Number of Genes
E. coli	4,639,221	4,316
Bacillus subtilis	4,214,810	4,100
Saccharomyces cerevisiae	12,100,000	6,000
Caenorhabditis elegans	97,000,000	19,049
Arabidopsis thaliana	115,409,949	25,000
Drosophila melanogaster	120,000,000	13,600
Mus musculus	2,500,000,000	37,000
Homo sapiens	3,000,000,000	30,000

E. coli is approximately a cylindrical body, with a length of about $2\mu m$ and diameter of about $0.8\mu m$. These dimensions offer a convenient translation between concentration and number of molecules in *E. coli*. Thus 1 nM concentration roughly translates to one molecule per *E. coli* cell (See exercises at end of chapter). For example, ATP is present at a concentration of approximately 2 mM, this means there are roughly 2,000,000 molecules of ATP in a single *E. coli* cell.

The *E. coli* genome is composed of 4,639,221 base pairs ($490\mu m$ in diameter) encoding at least 4,472 genes. Of this number, 4,316 code for proteins, with the remainder coding for various RNA products such as tRNAs and rRNAs. The genes in *E. coli*, like other prokaryotes, do not have segmented genes (genes made of introns and exons); that is, a gene in *E. coli* is a contiguous sequence of DNA translated into the final protein without editing. In addition there is very little non-coding DNA in *E. coli* with almost 88% of the genome coding for proteins.

Almost one quarter of all proteins produced by gene expression in *E. coli* form multimers, proteins composed of multiple subunits. Many of these multimers are homomultimers, meaning they are made up of the same subunits. Some of these proteins can also be covalently modified by phosphorylation, methylation or other means. There are estimated to be at least 171 transcription factors, that directly control gene expression. This number provides insight into the size of the gene regulatory network. The EcoCyc database reports at least 48 small molecules and ions that regulate these transcription factors.

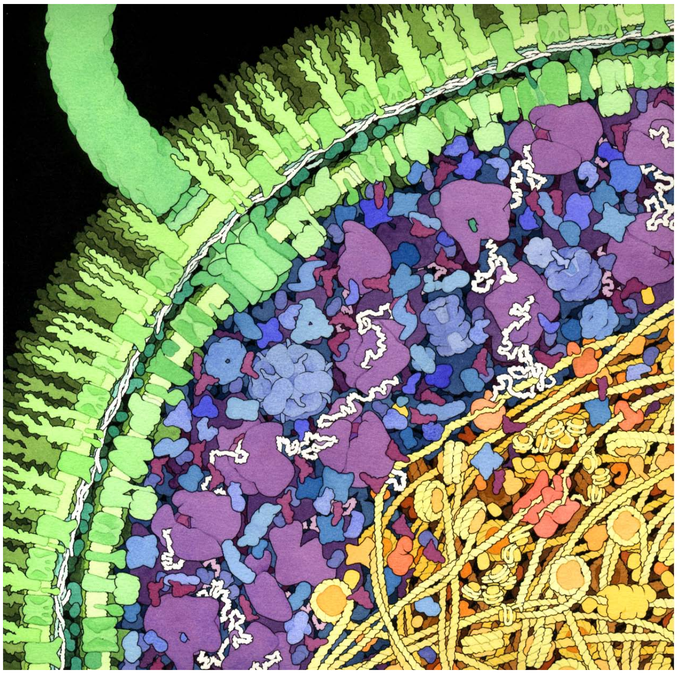


Figure 3.14 Artists impression (With permission Goodsell) of a cross-section through *E. Coli* illustrating the high density of proteins and other molecules in the the cytoplasm.

Of the 4,316 genes in *E. coli*, 3,384 (76%) have been assigned a biochemical function. There are at least 991 genes involved directly in metabolism with a further 355 genes involved in transport. Other gene functions include DNA replication, recombination and repair, protein folding, transcription, translation and regulatory proteins. An inventory of small molecules has not been thoroughly made but EcoCyc records at least 1352 unique small organic molecules but it is likely incomplete.

These statistics suggest large numbers of interactions among many thousands of cellular components forming extensive networks.

Given the size of a single *E. Coli* cell, the concentration of protein in the cytoplasm and the average diameter of a protein (5 nm), it is estimated that the average spacing (center to center) between proteins is about 7 nm. This suggests that the cytoplasm is quite dense. David Goodsell (`http://mgl.scripps.edu/people/goodsell`) is well known for his evocative illustrations of subcellular spaces. Figure 3.14 illustrates his rendition of a cross-section through *E. Coli*.

Property	Dimensions
Length	2 to 3 μm
Diameter	$\simeq 1 \mu$m
Volume	1×10^{-15} L
Optimal generation time	20 to 30 mins
Translation rate	40 amino acids per sec
Transcription rate	70 nucleotides per sec
Number of ribosomes per cell	18,000
Average protein diameter	5 nm
Average concentration of protein	5-8 mM
Average number of proteins	3,600,000

Table 3.3 Basic Information on *E. coli*

There are two useful sites for obtaining basic operating information on *E. Coli*. The first site is the *E. Coli*. statistics site at Project (`http://gchelpdesk.ualberta.ca/CCDB/cgi-bin/STAT_NEW.cgi`). The other is a more generic and community based web site called BIONUMBRS (The Database of Useful Biological Numbers). Publications from the project also supply many useful statistics on *E. Coli* [52, 53].

The number of molecules in a typical *E. coli* varies with the molecule type. For example there are approximately 2,000,000 Na ions while only 300,000 tryptophan molecules. The larger the molecule the fewer their number, Table 3.4. For example transcription factors are only present in numbers ranging from 10s to 100s, whereas ions are present in the millions.

Molecule	Estimated Number
Ions	Millions
Small Molecules	10,000 - 100,000
Metabolic Enzymes	1000 - 10,000s
Signaling Molecules	100 - 1000s
Transcription factors	10s to 100s
DNA	1 - 10s

Table 3.4 Orders of magnitude for various molecule types.

A significant study by Bennett et al. [5] measured over 100 metabolites levels in the main metabolic pathways of glucose-fed, exponentially growing *E. coli.* The average concentration was found to be 0.22 mM. We can compare this with the average K_m (concentration of substrate that gives half maximal activity) of approximately 0.1 mM as reported by the database. This suggests that on average enzymes operate above their half maximal activity. However, a more detailed analysis revealed considerable variability among different metabolite types. For example, cofactors such as ATP and NAD$^+$ were at concentrations significantly above their K_{ms}. In contrast, substrate-enzyme pairs where the concentration was below the K_m were dominated by enzymes catalyzing nucleotide, nucleoside, nucleobase and amino acid degradation reactions. On the other hand, the glycolytic pathway, tricarboxylic acid cycle, and the pentose-phosphate pathways all showed substrate concentration that were similar to their K_m values.

We can also consider how fast processes occur in *E. coli*. As suggested earlier in the chapter, metabolic responses are the fastest followed by protein signaling networks and gene regulatory networks. Table 3.7 lists some estimated for various biological processes.

The number of molecules and the rate of various processes gives some idea of the magnitude of systems we are dealing with. However, the economy of a typical cell, how ATP is distributed to different processes, and how supply and demand are maintained is largely not understood since many of these processes are difficult to measure. Moreover, there is no economic

Ions	Estimated Numbers
Na	3,000,000
Ca	2,300,000
Fe	7,000,000
Small Molecules	**Estimated Numbers**
Alanine	350,000
Pyruvate	370,000
ATP	2,000,000
ADP	70,000
NADP	240,000

Table 3.5 Small molecules estimates in *E. coli*.

theory that describes the life of a cell.

3.8 Network Motifs

At first sight the complex biochemical maps we see appear to have little order. However upon closer examination, patterns emerge. One way to discern these patterns is to compare real biochemical network with random networks and to look for a given pattern in each. For example, let's say we identify a pattern of regulation which we label p_1. We look for the occurrence of p_1 in both the real biochemical network and the randomly generated network. If we find that the pattern is statistically enriched in the real biochemical network compared to the randomly generated one, then we say we have found a network **motif**.

A motif is a subgraph within a network that occurs more often than one would expect by random chance alone. Such subgraphs can be simple triangles, squares etc. It is assumed that such motifs occur more frequently because they confer some functional advantage; their identification is therefore considered of some importance. Locating motifs in a large network entails a three stage process:

Signaling Proteins	Estimated Numbers
LacI	10 to 50
CheA kinase	4,500
CheB	240
CheY	8,200
Chemoreceptors	15,000
Metabolic Enzymes	**Estimated Numbers**
Phosphofructokinase	1,550
Pyruvate Kinase	11,000
Enolase	55,800
Phosphoglycerate kinase	124,000
Malate Dehydrogenase	3,390
Citrate Synthase	1,360
Aconitase	1630

Table 3.6 Estimated numbers for larger molecules in *E. coli*.

- Estimating the frequency of each isomorphic subgraph in the target network.
- Generating a suitable random graph to test the significance of the frequency data.
- Compare the target network with the random graph.

The critical stage is generating a suitable random model for comparison. The approach is to generate a random network which has a degree distribution that is the same as the degree distribution (See end of chapter for glossary) of the real target network [74, 73, 72]. One way to accomplished this is by starting with the target network itself and randomizing edges in such as way that the original degree distribution is preserved. This is carried out multiple times in order to generate a population of random networks. Once the random and target networks are ready, additional algorithms are invoked to count the number of given motifs. The frequency

Process	Rate
Cell Division Time	50 minutes
Rate of Replication	2000 bp/s
Protein Synthesis	1000 proteins/s
Lipid Synthesis	20,000 lipids/s
Ribosome Rates	25 amino acids per sec per ribosome
Number of ATP to make one cell	55 billion ATPs

Table 3.7 *E. coli* grown on minimal media plus Glucose. Data from Phillips et al (2010) and *E. coli* stats reference: http://ccdb.wishartlab.com.

distribution of the motif in the random networks is then compared to the frequency distribution in the target network. A simple significance test can be carried out using the z-score. The z-score is computed by subtracting the number of a given motif in the target network from the mean number of the same motif in the randomized networks. This difference is then normalized by dividing by the standard deviation of the motif count in the random population. If the z-score is greater than zero then it means that the observed number of motifs is greater than the mean, while a negative z-score indicates that the observed number of motifs is below the mean. A z-score of two indicates that the observed value is two standard deviations above the mean which can also be roughly interpreted as the 95% confidence level, That is if a z-score is two or above then we can say that the number of motifs is significantly different from a random network.

$$z = \frac{n - n_r}{\sigma_r}$$

The definition of a motif, while useful, has important restrictions. For example, consider a large electronic circuit containing transistors, resistors and capacitors. A motif search in such a circuit my find an overabundance of amplifier like motifs compared to a completely random circuit. However, such an analysis will not find specialist circuits such as a resonance filter, which may only occur once in the circuit. The motifs that are located using this approach therefore need to be fairly common in the network.

The operational definition of motifs therefore excludes motifs which may only appear once in a network but whose role is critical to the function of the network [100].

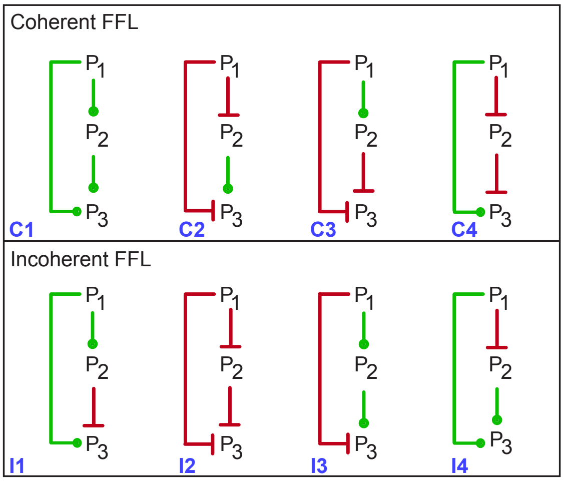


Figure 3.15 Full complement of feedforward motifs, classified into coherent and incoherent types.

Figure 3.16 shows the relative abundance of the different kinds of feedforward motifs found in *E. coli* and Yeast. This data indicates that two types predominate in both organisms, Coherent Type 1 (C1) and Incoherent Type 1 (I1). Within each group four combinations of regulation can be discerned. Figure 3.16 shows the relative abundance of the different kinds of feedforward motifs found in *E. coli* and Yeast. This data indicates that two types predominate in both organisms, Coherent Type 1 (C1) and Incoherent Type 1 (I1). What is particularly interesting is that these two types have distinct behavioral properties.

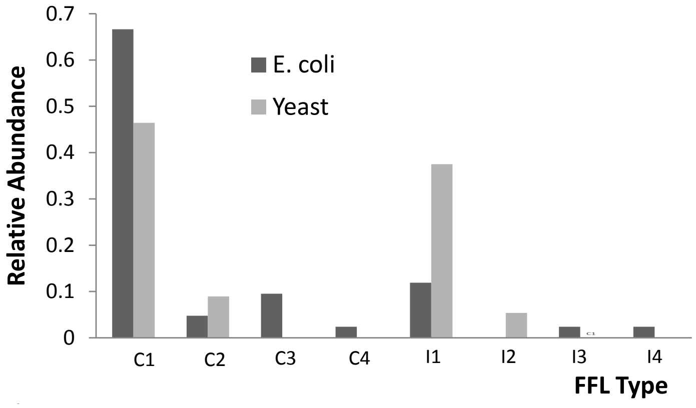


Figure 3.16 Relative abundance of different FFL types in Yeast and *E. coli*. Labels on the x-axis refer to the particular FFL motifs seen in Figure 3.15. Data taken from [66]

Properties of the Coherent Type I Motif

The best way to understand the properties of the coherent type I feed-forward network is to build a computer model and run a simulation. Figure 3.17 illustrates a possible genetic feedforward coherent type I network.

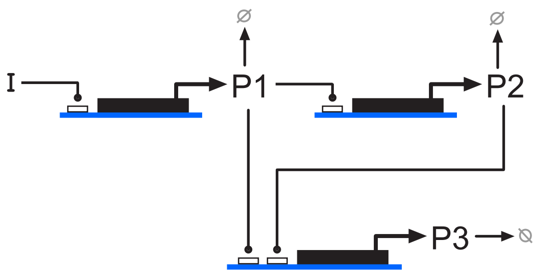


Figure 3.17 Example of a feedforward coherent type I genetic network.

Figure 3.18 illustrates one of the key properties of the coherent type I feed-forward network.

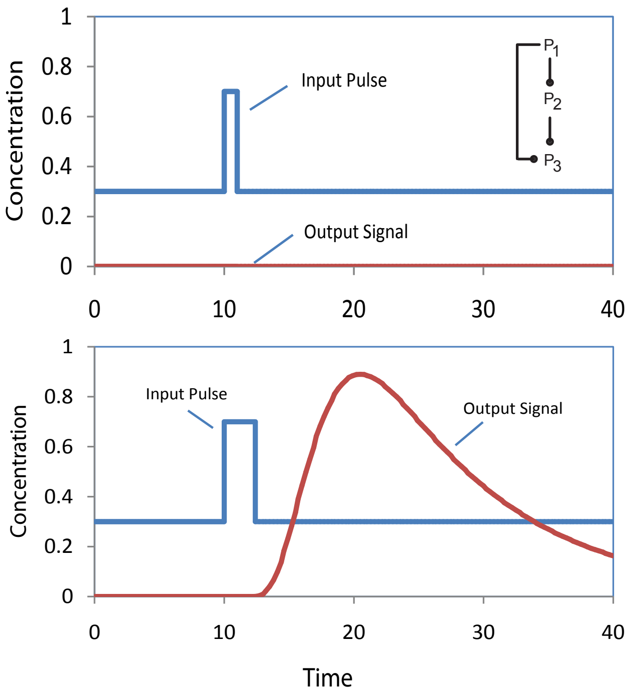


Figure 3.18 Simulation of a coherent Type I feedforward network. The top panel shows the effect of a narrow pulse in the concentration of P_1. The output signal, P_3, shows no response. In the lower panel, the input signal, P_1, is wider, this gives time for the feedforward loop to activate as shown by the large change in output signal, P_3. The coherent type I FFL will only 'fire' if the input pulse is of sufficient width. This means that the network can act as a noise filter. See Jarnac script 3.1

The output signal, P_3 is controlled by from P_2 and P_1. If a pulse signal is applied to P_1, the signal travels two routes to get to P_3. If the pulse is too short, P_2 does not have sufficient time to increase in order to reach the threshold level at P_3, as a result, short pulses do not result in P_3 turning on.

This effect is shown in the upper panel of Figure 3.18. In sharp contrast, if the P_1 pulse width is wide enough, P_2 has sufficient time to increase so that is reaches the threshold level and activates P_3. The network therefore acts as a transient signal filter. Only signals of sufficient duration will result in activation.

Properties of the Incoherent Type I Motif

The incoherent type I feedforward network (Figure 3.19) has a completely different response compared to the coherent type I network.

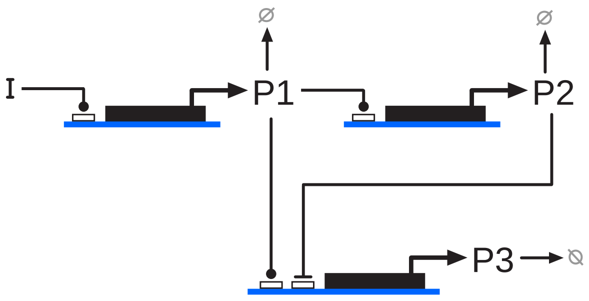


Figure 3.19 Example of a feedforward incoherent type I genetic network. Note that the regulation on P_3 is antagonistic compared to the coherent type I network.

The incoherent type I network has a number of interesting behaviors, including pulse generator, concentration band detector and frequency band pass filter. The pulse generator simulation is shown in Figure 3.20 where an input step function is applied to the input, P_1 and the effect on P_3 observed. In this case, P_3 rises rapidly then falls off in a pulse like manner even though the input signal remains on. This is due to the delay in the inhibition route, initially the inhibition is weak and the output raises, but eventually P_2 increases and begins to repress the production of P_3. The second type of behavior that the network can display is to act as a concentration band detector. The simulation shown in Figure 3.21 shows the network turning on in a specific range of input signal. The position of the

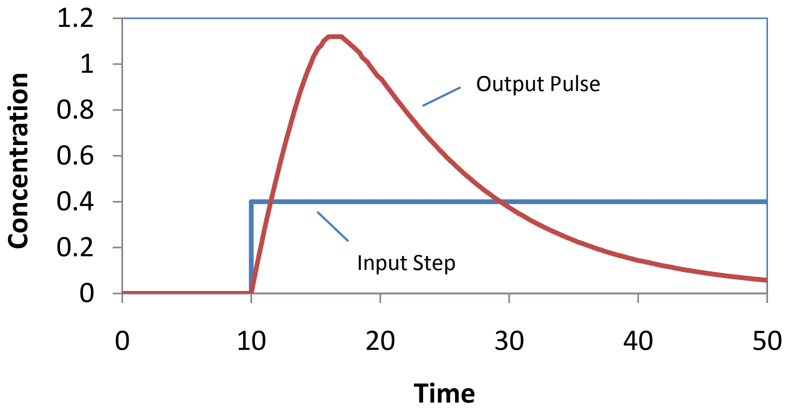


Figure 3.20 Simulation of the pulse generation characteristics of a incoherent Type I feedforward network. The x-axis shows the change in the input signal over time. At 10 time units, a step signal is applied, this causes the output to rise then fall in a pulse like manner. The width of the pulse can be adjusted by changing the degree of cooperativity on P_3. See Jarnac script 3.2

peak in the response can be adjusted by changing the strength of inhibition and the input threshold from P_1 to P_3.

Finally, an incoherent type I network can act as a band pass filter. That is the network will respond more strongly if the input signals varying at a specific range of frequencies. That is at high and low frequencies the network will not respond but at mid range frequencies it will.

A series of synthetic incoherent type I networks have been built in *E. coli* that illustrate the band pass behavior [29]. Further details on the properties of feedforward networks can be can also be found in the book by [1].

Menagerie of Motifs

The feedforward network described in the previous section is one of many different kinds of motifs that have been identified. It would take an entire book to describe them all. Instead we will summarize them here, together

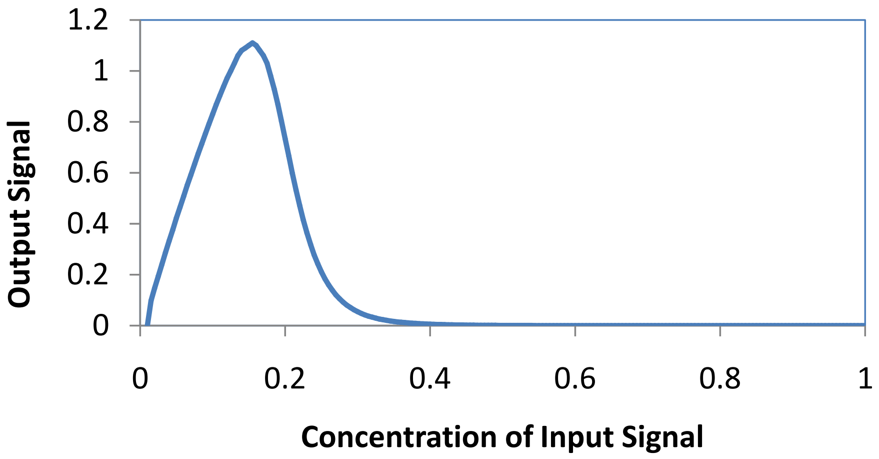


Figure 3.21 Simulation of the concentration band detector characteristics of a incoherent Type I feedforward network. The X-axis shows the change in the input signal while the Y-axis shows the steady state concentration in the output signal. The network only responds in a particular range of input concentration. See Jarnac script 3.3.

with their basic dynamic properties.

Figures 3.22, 3.23 and 3.24 show a variety of motifs. No doubt many more natural patterns remain to be discovered in addition to new motifs designed by the synthetic biology community [31].

Further Reading

General

1. Bray D (2011) Wetware: A Computer in Every Living Cell. Yale University Press. ISBN: 978-0300167849

2. Goodsell D S (2009) The machinery of life. Springer, 2nd edition. ISBN 978-0387849249

3. Phillips R, Kondev J and Theriot J (2010) Physical Biology of the Cell. Garland Science. ISBN 978-0-8153-4163-5

Specific

1. Alberts et al, (2002) General Principles of Cell Communication http://www.ncbi.nlm.nih.gov/books/NBK26813/
2. Brown TA (2006) Genomes 3, Garland Science, 3rd edition. ISBN: 978-0815341383
3. Gerhard M and Schomburg D (2012) Biochemical Pathways: An Atlas of Biochemistry and Molecular Biology, Wiley, 2nd edition. ISBN: 978-0470146842
4. Hancock J (2010) Cell Signalling, Oxford University Press, 3rd edition. ISBN: 978-0199232109
5. Hartl DL (2008) Genetics: Analysis Of Genes And Genomes. Jones & Bartlett Learning, 7th edition. ISBN: 978-0763772154
6. Nelson DL and Cox MM (2008) Wetware: Lehninger Principles of Biochemistry. W. H. Freeman, 5th edition. ISBN: 978-0716771081
7. Salway JG (2004) Metabolism at a Glance, Wiley-Blackwell, 3rd edition. ISBN: 978-1405107167

Exercises

In the following exercises, use the data given in the main text, and Tables 3.3, 3.4, 3.5, and 3.6.

1. How many *E. coli* cells laid end to end would fit across the full stop at the end of this sentence? Assume a diameter of the full stop to be 0.5 mm.
2. Estimate the volume of an *E. coli* cell.

3. Calculate the surface area of an *E. coli* cell. If a typical membrane protein is 5 nm in diameter, estimate the number of membrane proteins that can be laid out on the membrane if the center-center distance between each protein is 6 nm.

4. Show that a 1 nM concentration is roughly equivalent to 1 molecule in a volume of one *E. coli* cell.

5. Estimate the number of protein molecules a typical *E. coli* cell can make per second assuming that the average protein is 360 amino acids long. Assume that the number of proteins in a cell is 3,000,000. How long would it take to make 3,000,000 proteins?

6. If it takes 1,500 ATP molecules to make an average protein, how long would it take before all the ATP is used up? Assume the ATP is not being replaced.

7. What are the visual symbols often used to represent activation and repression in biochemical networks?

8. Draw a similar diagram to the glycolysis regulatory diagram (Figure 3.3) but for the lysine, threonine and methionine biosynthesis pathway from *E. coli*.

Appendix

See http://sbw-app.org/jarnac/ for more details of Jarnac.

```
// Coherent Type I Genetic Network, noise filter
p = defn cell
       $G2 -> P2; Vmax2*P1^4/(Km1 + P1^4);
        P2 -> $w; k1*P2;

       $G3 -> P3; Vmax3*P1^4*P2^4/(Km1 + P1^4*P2^4);
        P3 -> $w; k1*P3;
end;

p.Vmax2 = 1;
```

```
p.Vmax3 = 1;

p.Km1 = 0.5;
p.k1 = 0.1;

p.P1 = 0;
p.P2 = 0;
p.P3 = 0;

p.ss.eval;
println p.sv;

// Pulse width
// Set to 1 for no effect
// Set to 4 for full effect
h = 1;

p.P1 = 0.3;
m1 = p.sim.eval (0, 10, 100, [<p.Time>, <p.P1>, <p.P3>]);
p.P1 = 0.7; // Input stimulus
m2 = p.sim.eval (10, 10 + h, 100, [<p.Time>, <p.P1>, <p.P3>]);
p.P1 = 0.3;
m3 = p.sim.eval (10 + h, 40, 100, [<p.Time>, <p.P1>, <p.P3>]);

m = augr (m1, m2);
m = augr (m, m3);
graph (m);
```

Listing 3.1 Script for Figure 3.17

```
// Incoherent Type I Genetic Network, Pulse generator
p = defn cell
      $G1 -> P2; t1*a1*P1/(1 + A1*P1);
       P2 -> $w; gamma_1*P2;

       $G3 -> P3; t2*b1*P1/(1 + b1*P1 + b2*P2 + b3*P1*P2^8);
       P3 -> $w; gamma_2*P3;
end;

p.P2 = 0;
```

```
p.P3 = 0;
p.P1 = 0.01;
p.G3 = 0;
p.G1 = 0;

p.t1 = 5;
p.a1 = 0.1;
p.t2 = 1;
p.b1 = 1;
p.b2 = 0.1;
p.b3 = 10;
p.gamma_1 = 0.1;
p.gamma_2 = 0.1;

// Time course response for a step pulse

p.P1 = 0.0;
m1 = p.sim.eval (0, 10, 100, [<p.Time>, <p.P1>, <p.P3/1>]);
p.P1 = 0.4; // Input stimulus
m2 = p.sim.eval (10, 50, 200, [<p.Time>, <p.P1>, <p.P3/1>]);

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

Listing 3.2 Script for Figure 3.20

```
// Incoherent Steady State Response
p = defn cell
      $G1 -> P2; t1*a1*P1/(1 + A1*P1);
       P2 -> $w; gamma_1*P2;

       $G3 -> P3; t2*b1*P1/(1 + b1*P1 + b2*P2 + b3*P1*P2^8);
       P3 -> $w; gamma_2*P3;
end;

p.P2 = 0;
p.P3 = 0;
p.P1 = 0.01;
p.G3 = 0;
p.G1 = 0;
```

```
p.t1 = 5;
p.a1 = 0.1;
p.t2 = 1;
p.b1 = 1;
p.b2 = 0.1;
p.b3 = 10;
p.gamma_1 = 0.1;
p.gamma_2 = 0.1;

// Steady state response
n = 200;
m = matrix (n, 2);
for i = 1 to n do
    begin
    m[i,1] = p.P1;
    m[i,2] = p.P3;
    p.ss.eval;
    p.P1 = p.P1 + 0.005;
    end;
graph (m);
```

Listing 3.3 Script for Figure 3.21

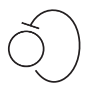
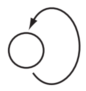
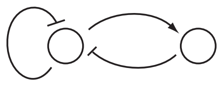
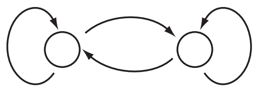
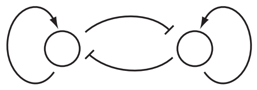
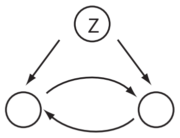
Motif Structure	Name	Dynamic Properties
a) 	Negative Autoregulaton	1. Noise suppression 2. Accelerated Response 3. HIgh Fedelity Amplifier 4. Feedback Oscillator
b) 	Positive Autoregulaton	1. Bistability 2. Memory Unit 3. Op Amp Component
c) 	Relaxation Oscillator	Adaptable Oscillatory
d) 	Double Positive Feedback	Memory Unit where both units are **either** on or off.
e) 	Double Negative Feedback	Memory Unit where one unit is off the **other** is on
f) 	Regulated Double Positive Feedback	Memory unit that records an event in Z

Figure 3.22 Motifs

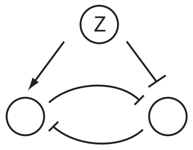
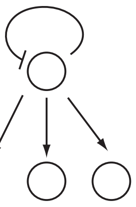
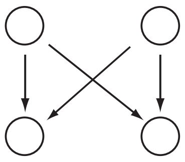
Motif Structure	Name	Dynamic Properties
a) 	Regulated Double Negative Feedback	Mempory unit where nodes switch in opposite directions due to an event Z.
b) 	Single Input Module (SIM)	1. Master/Slave Regulator 2. Temporal Sequencer - last gene activated is the first gene deactivated.
c) 	Bi-Fan	1. Neural network type computations 2. Synchronizers 3. Filters

Figure 3.23 Motifs

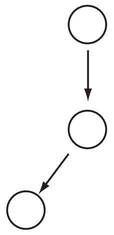
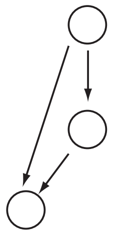
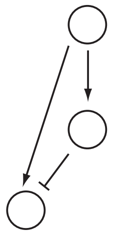
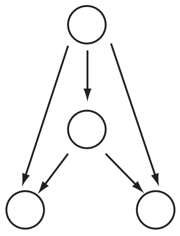
Motif Structure	Name	Dynamic Properties
a) 	Cascade Unit	1. Noise filter 2. NonLinear amplifier
b) 	Coherent Feedforward	1. Noise rejection 2. Pulse shifter
c) 	Incoherent Feedforward	1. Pulse generator 2. Concentration detector 3.Response time accerlerator
d) 	Multi-Output FFL	Pulse train generator

Figure 3.24 Feedforwrd Network Motifs

4 How Systems Behave

4.1 System Behavior

Ultimately we are interested in what kinds of behavior systems can display, how that behavior is generated and with that understanding how systems can be manipulated and controlled. As we proceed through the book we will encounter many different kinds of behavior. At this stage however it is worth describing the states that are fundamental to all systems. These states fall into three groups: **(Thermodynamic) equilibrium**, **steady state**, and **transients**. In the literature the terms equilibrium and steady state are often used to mean the same thing but here they will be used to describe two very different states.

The simplest and arguably the least interesting is equilibrium, or more precisely thermodynamic equilibrium.

4.2 Equilibrium

Thermodynamic equilibrium, or simply equilibrium, refers to the state of a system when all forces are balanced. In chemistry, thermodynamic equilibrium is when all forward and reverse rates are equal. This also means that the concentration of chemical species are also unchanging and all net flows are zero. Equilibrium is easily achieved in a closed system. For example, consider the simple chemical isomerization:

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B \tag{4.1}$$

Let the net forward rate of the reaction, v, be equal to $v = k_1 A - k_2 B$. The rates of change of A and B are then given by:

$$\frac{dA}{dt} = -v \qquad \frac{dB}{dt} = v$$

The equilibrium constant for this system is given by:

$$K_{eq} = \frac{B_{eq}}{A_{eq}} = \frac{k_1}{k_2}$$

At equilibrium dA/dt and dB/dt equal zero, that is $Ak_1 = Bk_2$, or $v = 0$. The analytical solution to the chemical isomerization can be derived as follows. Given that the system is closed we know that the total mass in the system, $A + B$ is constant. This constant is given by the sum of the initial concentrations of A and B which we will define as $A_o + B_o$. Note that $A_o + B_o = A(t) + B(t)$ is always true. We assume that the volume is constant and set to unit volume, this allows us to state that the sum of the concentrations is conserved. The differential equation for A is given by:

$$\frac{dA}{dt} = k_2 B - k_1 A$$

Before solving this equation, let us replace B by the term $A_o + B_o - A$. This yields:

$$\frac{dA}{dt} = k_2 A_o + k_2 B_o - k_2 A - k_1 A = k_2(A_o + B_o) - A(k_1 + k_2)$$

The easiest way to solve this equation is to use Mathematica or Maxima. The Mathematica command is `DSolve[A'[t] == k2 (Ao + Bo) - A[t] (k1 + k2), A[0] == Ao, A[t], t]`, where `A[0] == Ao` sets the initial condition for the concentration of A to be A_o. By implication, the initial condition for B_o is $(A_o + B_o) - A_o = B_o$. The result of applying the Mathematica command yields the following solution:

$$A(t) = \frac{(A_o + B_o)k_2}{k_1 + k_2} + \frac{e^{-(k_1+k_2)t} v_{\text{initial}}}{k_1 + k_2}$$

The first term in the equation is independent of time and equals the equilibrium concentration of A. The first term is a function of the total mass in the system $(A_o + B_o)$ which means that the equilibrium solution is independent of the starting concentrations so long as the total remains the same. That is starting conditions of $A_o = 1; B_o = 9$, $A_o = 6; B_o = 4$ will also read to the same equilibrium concentrations.

The second term is time dependent and describes the evolution of the system when the initial concentrations of A and B are not set to the equilibrium concentrations. The initial concentrations are set in the term v_{initial} which is the reaction rate, v, at $t = 0$. The second term also has an exponential component which approaches zero as time goes to infinity so that at infinite time we are left with the first term which equals the concentration of A when $dA/dt = dB/dt = 0$.

At equilibrium the reaction rate can be computed by substituting the equilibrium concentration of A and B into the reaction rate, $v = k_2 B - k_1 A$. We note that the equilibrium concentration of A is given by:

$$A_{eq} = \frac{(A_o + B_o)k_2}{k_1 + k_2}$$

and for B by subtracting A_{eq} from $A_o + B_o$. When the A_{eq} and B_{eq} relations are substituted into v, the result is:

$$v = 0$$

From this somewhat long winded analysis, it has been determined for the closed reversible system, at infinite time, the concentrations of A and B

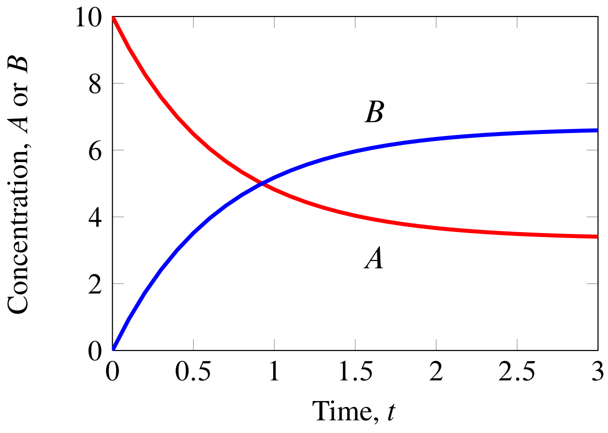


Figure 4.1 Time course for equilibration of the reversible reaction in model 4.1 where $k_1 = 1, k_2 = 0.5, A_o = 10, B_o = 0$. The ratio of the equilibrium concentration is given by k_1/k_2. Jarnac model: 6.2

reach some constant values and that the net rate, v is zero. Note also that the ratio of the final concentration for A and B equals the equilibrium constant. The system is therefore at thermodynamic equilibrium.

In biochemical models it is often assumed that when the forward and reverse rates for a particular reaction are very fast compared to the surrounding reactions that the reaction is said to be in **quasi-equilibrium**. That is, although the entire system may be out of equilibrium there may be parts of the system that can be approximated as though they were in equilibrium. This is often done to simplify the modeling process.

Living organisms are not themselves at thermodynamic equilibrium, if they were then they would technically be dead. Living systems are open so that there is a continual flow of mass and energy across the system's boundaries.

4.3 Steady State

The **steady state**, also called the stationary state, is where the rates of change of all species, dS/dt are zero **but** at the same time the net rates are non-zero, that is $v_i \neq 0$. This situation can only occur in an open system, that is a system that can exchange matter with the surroundings. To convert the simple reversible model described in the last section into an open system we need only add a source reaction and a sink reaction as shown in the following scheme:

$$X_o \overset{v_o}{\rightarrow} A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B \overset{k_3}{\rightarrow} \tag{4.2}$$

In this case simple mass-action kinetics is assumed for all reactions. It is also assumed that the source reaction, with rate v_o, is irreversible and originates from a boundary species, X_o, that is X_o is fixed. In addition it is assumed that the sink reaction, with rate constant, k_3 is also irreversible. For the purpose of making it easier to derive the time course solution, the reverse rate constant, k_2 will be assumed to equal zero and we will set the initial conditions for A and B to both equal zero. The mathematical solution for the system can again be obtained using Mathematica:

$$\begin{aligned} A(t) &= v_o \frac{1 - e^{-k_1 t}}{k_1} \\ B(t) &= v_o \frac{k_1 \left(1 - e^{-k_3 t}\right) + k_3 \left(e^{-k_1 t} - 1\right)}{k_3 \left(k_1 - k_3\right)} \end{aligned} \tag{4.3}$$

As t tends to infinity $A(t)$ tends to v_o/k_1 and $B(t)$ tends to v_o/k_3. In addition, the reaction rate through each of the three reaction steps is v_o. This can be confirmed by substituting the solutions for A and B into the reaction rate laws. Given that v_o is greater than zero and that A and B reach constant values given sufficient time, we conclude that this system eventually settles to a steady state rather than thermodynamic equilibrium.

Thermodynamic Equilibrium and Steady State

Thermodynamic equilibrium (or equilibrium for short) and the steady state are distinct states of a chemical system. If we consider a system where every part is in equilibrium then we can be sure of two things. That the species concentrations are unchanging and most importantly there are no net flows of mass or energy within the system or between the system and the environment. A system that is in equilibrium must have the following properties:

$$\frac{d\mathbf{s}}{dt} = 0$$

$$\text{for all } i: v_i = 0$$

where v_i is the net reaction rate for the i^{th} reaction step. When a biological system is at equilibrium, we say it is **dead**. Thermodynamically we can also say that entropy production is at zero and has reached its **maximum value**.

The steady state has some similarities with the equilibrium state, species concentrations are still unchanging, however **there are net flows** of energy and mass within the system and with the environment. Systems at steady state must therefore be open and will necessarily continuously dissipate any gradients between the system and the external environment. This means that one or more v_is must be non-zero

The steady state is defined when all dS_i/dt are equal to zero while one or more reaction rates are non-zero:

$$\frac{d\mathbf{s}}{dt} = 0$$

$$v_i \neq 0$$

Thermodynamically, we can also say that the entropy production of the system at steady state is lower than the entropy production in the environment. In some of the literature the terms equilibrium and steady state are used interchangeably resulting in possible confusion. In this book, the word equilibrium will be used to refer to a system at thermodynamic equilibrium not at steady state.

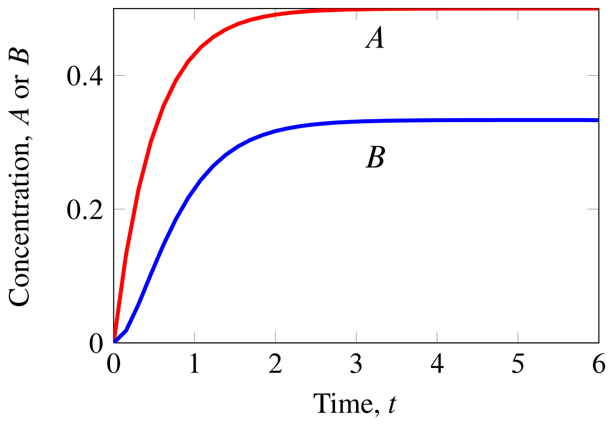


Figure 4.2 Time course for an open system reaching steady state in model 4.4 where $v_o = 1, k_1 = 2, k_2 = 0, k_3 = 3, A_o = 0, B_o = 0$. X_o is assumed to be fixed. The Jarnac model: 6.3

The system displays a continuous flow of mass from the sink to the source. This can only continue undisturbed so long as the source material, X_o never runs out and that the sink is continuously emptied. Figure 4.2 shows a simulation of this system.

At steady state, the rate of mass transfer across a reaction is often called the flux, or J.

At steady state the net reaction rate is also called the **pathway flux**, often symbolized with the letter J.

We can sometimes also calculate the steady state in a different way. In the last example we used the simplified model:

$$X_o \xrightarrow{v_o} A \xrightarrow{k_1} B \xrightarrow{k_3} \tag{4.4}$$

The differential equations for this system were:

$$\frac{dA}{dt} = v_o - k_1 A$$

$$\frac{dB}{dt} = k_1 A - k_3 B$$

If we set the rates of change to zero:

$$0 = v_o - k_1 A$$
$$0 = k_1 A - k_3 B$$

We have equations in two unknown, A and B. We can solve for A and B to obtain:

$$A = v_o / k_1$$

$$B = v_o / k_3$$

Usually however we cannot solve the equations and so must revert to computer simulation or using specialist software (such as Jarnac) to compute the steady state. The script below show a Jarnac model where we ask Jarnac to solve for the steady state using the command `p.ss.eval`.

```
p = defn model
     $Xo -> A;  vo;
      A -> B;   k1*A;
      B -> $X1; k3*B;
end;

// Set up the model initial conditions
p.Xo = 1;   p.X1 = 0;
p.k1 = 0.2; p.k3 = 0.3;
p.vo = 0.5;

// Evaluation the steady state
p.ss.eval;

println "Steady State values:", p.A, p.B;
```

```
// Output follows:
Steady State values:  2.5  1.66667
```

We will talk a lot more about the steady state in Chapter 6.

4.4 Transients

The final simple behavior that a system can show is a transient. A transient is usually the change that occurs in the species concentrations as the system moves from one state, often a steady state, to another. Equation 4.3 shows the solution to a simple system that describes the transient behavior of species A and B. Figure 4.2 illustrates the transient from an initial condition, in this case from a non-steady state condition to a steady state. A periodic (such as an oscillation) or a chaotic system may be considered a transient, one that is unable to settle to a fixed steady state. In the case of a system showing periodic behavior, the transient repeats itself indefinitely at regular intervals called the period. In a chaotic system, the transient never repeats the exact same trajectory but will continue indefinitely.

4.5 Setting up a Model in Software

There are many software tools both commercial and free (including open source) that one can use to build models of cellular networks. In this book we will use Jarnac [91], a software tool written by the author. Jarnac is a script based tool where one enters a model as a text file, the model is then compiled, run and the results displayed. Jarnac is currently a windows based application. It is quite easy to set up models using Jarnac but it also has a fairly complete programming language built-in that allows advanced users to do some sophisticated analysis. A brief introduction on how to use Jarnac is given in Appendix E. For those who wish to use other tools, such as COPASI (`http://www.copasi.org`), CellDesigner (`http://celldesigner.org/` or even Matlab (`http://www.mathworks.com`, it is easy to convert Jarnac files into standard Systems Biology Markup Language (SBML) or Matlab scripts (See Appendix E) and then load the models into the simulation tool of choice.

4.6 Effect of Different Kinds of Perturbations

When we talk about model dynamics we mean how species levels and reaction rates change in time as the model evolves. There are a number of ways to elicit a dynamic response in a model, the two we will consider here are perturbations to species and perturbations to model parameters.

Effect of Perturbing Floating Species

Let us consider a two step pathway of the following form:

$$\begin{tikzcd} X_o \arrow[r, "v_1 = k_1 Xo"] & S_1 \arrow[r, "v_2 = k_2 S_1"] & X_1 \end{tikzcd}$$

We assume that X_o and X_1 are fixed. If the initial concentration of S_1 is zero then we can run a simulation and allow the system to come to steady state. This is illustrated in Figure 6.4

Once at steady state we can consider applying perturbations to see what happens. For example, Figure 6.7 illustrates the effect of injecting 0.35 units of S_1 at $t = 20$ and watching the system evolve. The Jarnac script used to generate this graph is shown in the chapter Appendix. In practice this could be accomplished this by injecting 0.35 units of S_1 into the volume where the pathway operates. What we observe is that the concentration of S_1 initially jumps by the amount 0.35, then relaxes back to its steady state concentration before the perturbation was made (Figure 6.7). When we apply perturbations to species concentrations and the change relaxes back to the original state, we call the system **stable**. We will return to the topic of stability in the next section.

Figure 6.7 illustrates perturbing one of the floating molecular species by physically adding a specific amount of the substance to the pathway. In many cases we will find that the system will recover from such perturbations as we see in Figure 6.7. We are not limited to single perturbations,

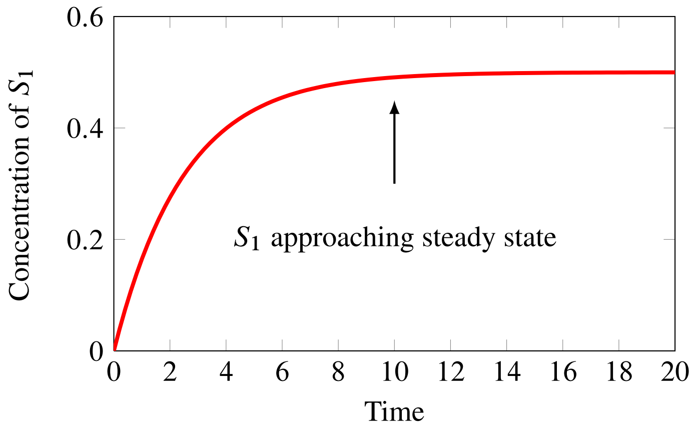


Figure 4.3 S_1 approaching steady state. Jarnac model: 6.4

Figure 6.5 shows multiple perturbations, both positive and negative. Not all systems show recovery like this but those that do not are called **unstable**. That is when we perturb a species concentration, instead of the perturbation relaxing back, it begins to diverge.

Effect of Perturbing Model Parameters

In addition to perturbing floating species we can also perturb the model parameters. Such parameters include kinetic constants and boundary species. When changing a parameter we can do it in two ways, we can make a permanent change or we can make a change and at some time later return the parameter to its original value. Assuming that the steady state is stable, a temporary change will result in the steady state changing then recovering to the original state when the parameter is changed back. Figure 6.6 shows the effect of perturbing the rate constant, k_1 and then restoring the parameter to its original value at some time later.

In some applications other types of perturbations are made. For example in studying the infusion of a drug where the concentration of the drug is a model parameter, one might use a slow linear increase in the drug concen-

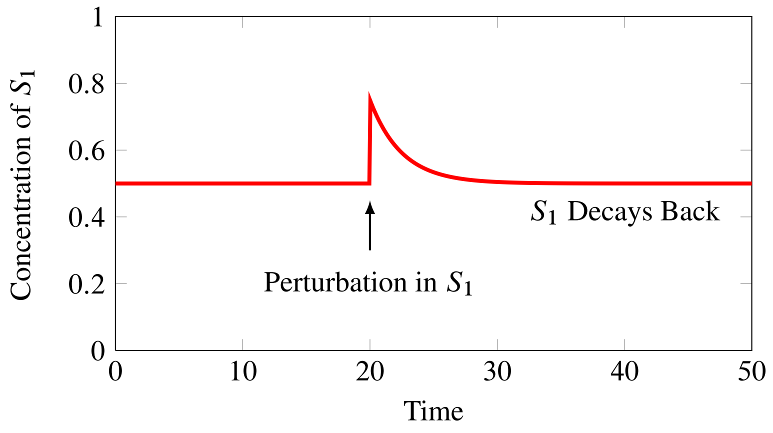


Figure 4.4 Stability of a simple biochemical pathway at steady state. The steady state concentration of the species S_1 is 0.5. A perturbation is made to S_1 by adding an additional 0.35 units of S_1 at time $= 20$. The system is considered stable because the perturbation relaxes back to the original steady state. Jarnac model: 6.5

.

tration. Such a perturbation is called ramp. More sophisticated analyzes might require a sinusoidal change in a parameter, an impulse, a pulse or an exponential change. The main point to remember is that parameter changes will usually result in changes to the steady state concentrations and fluxes.

For completeness, Figure 4.7 shows what happens when we perturb both a parameter and a species concentration. As expected the species concentration does not recover to the original steady state.

4.7 Sensitivity Analysis

Sensitivity analysis at steady state looks at how particular model variables are influenced by model parameters. There are at least two main reasons why it is interesting to examine sensitivities. The first is a practical one. Many kinetic parameters we use in building biochemical models can

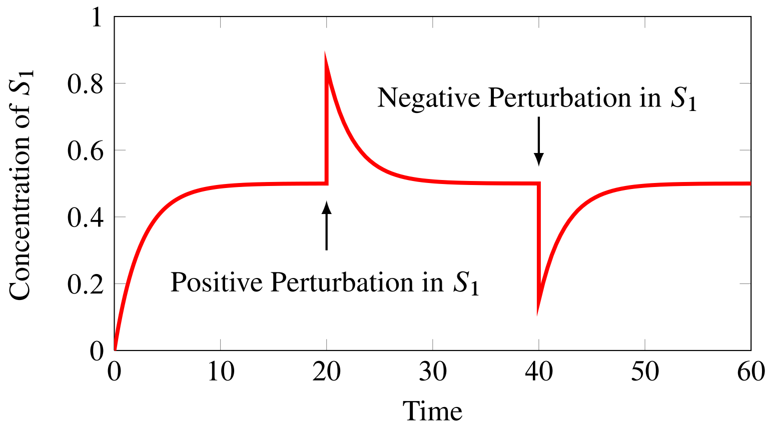


Figure 4.5 Multiple Perturbations. The steady state concentration of the species S_1 is 0.5 and a perturbation is made to S_1 by adding an additional 0.35 units of S_1 at time $=$ 20 and removing 0.35 units at time $=$ 40. In both cases the system relaxes back. Jarnac script: 6.6

have a significant degree of uncertainty about them. By determining how much each parameter has an influence on the model's state we can decide whether we should improve our confidence in the particular parameter. A parameter that has considerable influence but at the same time has significant uncertainty is a parameter that should be determined more carefully by additional experimentation. On the other hand a parameter that has little influence but has significant uncertainly associated with it, is relatively unimportant. A sensitivity analysis can therefore be used to highlight parameters that need better precision.

The second reason for measuring sensitivities is to provide insight. The degree to which a parameter can influence a variable tells us something about how the network is responding to perturbations. Such a study can be used to answer questions about robustness and adaptation. We will delay further discussion of this important topic to the second half of the book where we will describe it much more detail.

How are sensitivities represented? Traditionally there are two way, one based on absolute sensitivities and the second based on relative sensitiv-

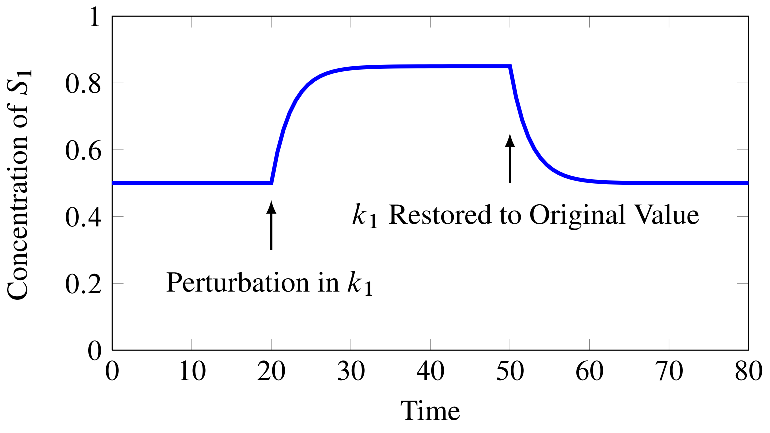


Figure 4.6 Effect of Perturbing Model Parameters. Jarnac script: 6.7

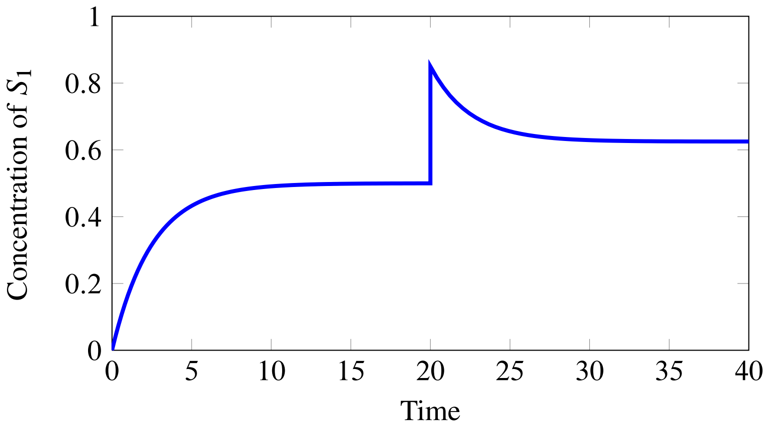


Figure 4.7 Effect of Perturbing Model Parameters and Species Concentration.

ities. Absolute sensitivities are simply given by the ratio of the absolute change in the variable to the absolute change in the parameter. That is:

$$S = \frac{\Delta V}{\Delta p}$$

where V is the variable and p the parameter. This equation shows finite changes to the parameter and variable. Unfortunately because most systems are nonlinear and therefore the value for the sensitivity will be a function of the size of the finite change. To make the sensitivity independent of the size of the change, the sensitivity is usually defined in terms of infinitesimal changes:

$$S = \frac{dV}{dp}$$

Although absolute sensitivities are simple they have one drawback, the value can be influenced by the units used to measure the variable and parameter. Often in making experimental measurements we won't be able to measure the quantity using the most natural units, instead we may have measurements in terms of fluorescence, colony counts, staining on a gel and so on. Is is most likely that the variable and parameter units will be quite different and each laboratory may have its own way particular units is uses. Absolute sensitivities are therefore quite difficult to compare.

To get round the problem of units, many people will use relative sensitivities, These are simple scaled absolute sensitivities:

$$S = \frac{dV}{dp}\frac{p}{V}$$

The sensitivity is defined in terms of infinitesimal changes for the same reason cited before. The reader may also recall that elasticities are measured in this way. Relative sensitivities are immune from the units we use to measure quantities but also relative sensitivities correspond more closely to how many measurements are made, often in terms of relative or fold changes. In practice steady state relative sensitivities should be measured by taking a measurement at the operating steady state, making a perturbation (preferable a small one), waiting for the system to reach a new steady state then measuring the system again. It is important to be aware that the steady state sensitivities measure how a perturbation in a parameter moves the system from one steady state to another.

4.8 Robustness and Homeostasis

Biological organisms are continually subjected to perturbations. These perturbations can originate from external influences such as changes in temperature, light or the availability of nutrients. Perturbations can also arise internally due to the stochastic nature of molecular events or by natural genetic variation. One of the most remarkable and characteristic properties of living systems is their ability to resist such perturbations and maintain very steady internal conditions. For example the human body can maintain a constant core temperature of 36.8°C ± 0.7 even though external temperatures may vary widely. The ability of a biological system to maintain a steady internal environment is called **homeostasis**, a phrase introduced by Claude Bernard almost 150 years ago. Modern authors may also refer to this behavior as **robustness**, although this word is used in many other contexts.

There are a number of mechanisms that are used in biology to maintain homeostasis. Perhaps the most common is negative feedback. This is where the difference between the desired output and the actual output is used to modulate the process that determines the output. For example, if the output is lower than the desired output then the process will increase the output. Such systems are found at multiple levels in a living organism, including subcellular processes such as metabolism and multicellular processes that control for example the level of glucose in the blood stream. One way to measure the degree of robustness or homoeostasis in a system is to use sensitivity analysis. We investigate the use of negative feedback to maintain concentrations within a narrow range in a later chapter.

Further Reading

1. Kipp E, Herwig R, Kowald A, Wierling C and Lehrach H (2005) Systems Biology in Practice, Wiley-VCH Verlag
2. Steuer R, Junker BH (2008) Computational Models of Metabolism: Stability and Regulation in Metabolic Networks, Advances in Chemical Physics, Volume 142, (ed S. A. Rice), John Wiley & Sons, Inc.
3. Stucki JW (1978) Stability analysis of biochemical systems–a practical guide. Prog Biophys Mol Biol. 33(2):99-187.

Exercises

1. Describe the difference between thermodynamic equilibrium and a steady state.
2. Write out the differential equations for the system $A \leftarrow B \leftarrow C$ where the reactions rates are given by:

$$v_1 = k_1 A - k_2 B$$
$$v_2 = k_3 B - k_4 C$$

   Find the concentrations of A, B and C when the rates of change are zero $dA/dt = 0, dB/dt = 0, dC/dt = 0$. Show that this system is at thermodynamic equilibrium when the rates of change are zero.
3. What do we mean by the phrase quasi-equilibrium?
4. Find the mathematical expression for species A and B that describes the steady state for the network:

$$X_o \underset{k_2}{\overset{k_1}{\rightleftharpoons}} A \overset{k_3}{\rightarrow} B \overset{k_4}{\rightarrow} \tag{4.5}$$

   Assume that X_o is fixed and that all reactions are governed by simple mass-action kinetics.

5. Explain what is meant by a stable and unstable steady state.
6. The steady state of a given pathway is stable. Explain the effect in general terms on the steady state if:

   a) A bolus of floating species is injected into the pathway

   b) A permanent change to a kinetic constant.
7. Determine whether the steady state for the system 4.5 is stable or not.
8. Why are scaled sensitivity sometime of more advantage that unscaled sensitivities?
9. Use a software tool of your choice to visualize the phase plot for the following system:

$$\frac{dx}{dt} = 2.55x - 4.4y$$
$$\frac{dy}{dt} = 5x + 2.15y$$

Appendix

See Appendix E for more details of Jarnac.

```
p = defn cell
      A -> B;  k1*A;
      B -> A;  k2*B;
end;

p.A = 10; p.k1 = 1;
p.B = 0;  p.k2 = 0.5;

m = p.sim.eval (0, 6, 100);
graph (m);
```

Listing 4.1 Script for Figure 4.1

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

p.vo = 1;
p.k1 = 2; p. k2 = 0;
p.k3 = 3;

m = p.sim.eval (0, 6, 100);
graph (m);
```

Listing 4.2 Script for Figure 4.2

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

p.k1 = 0.2;
p.k2 = 0.4;
p.Xo = 1;
p.S1 = 0.0;

m = p.sim.eval (0, 20, 100, [<p.time>, <p.S1>]);
graph (m);
```

Listing 4.3 Script for Figure 6.4

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

p.k1 = 0.2;
p.k2 = 0.4;
p.Xo = 1;
```

```
p.S1 = 0.5;

// Simulate the first part up to 20 time units
m1 = p.sim.eval (0, 20, 100, [<p.time>, <p.S1>]);

// Perturb the concentration of S1 by 0.35 units
p.S1 = p.S1 + 0.35;

// Continue simulating from last end point
m2 = p.sim.eval (20, 50, 100, [<p.time>, <p.S1>]);

// Merge and plot the two halves of the simulation
graph (augr(m1, m2));
```

Listing 4.4 Script for Figure 6.7

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

p.k1 = 0.2;
p.k2 = 0.4;
p.Xo = 1;
p.S1 = 0.0;

// Simulate the first part up to 20 time units
m1 = p.sim.eval (0, 20, 100, [<p.time>, <p.S1>]);

// Perturb the concentration of S1 by 0.35 units
p.S1 = p.S1 + 0.35;

// Continue simulating from last end point
m2 = p.sim.eval (20, 40, 50, [<p.time>, <p.S1>]);
// Merge the data sets
m3 = augr(m1, m2);
// Do a negative perturbation in S1
p.S1 = p.S1 - 0.35;

// Continue simulating from last end point
```

```
m4 = p.sim.eval (40, 60, 50, [<p.time>, <p.S1>]);

// Merge and plot the final two halves of the simulation
graph (augr(m3, m4));
```

Listing 4.5 Script for Figure 6.5

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

p.k1 = 0.2;
p.k2 = 0.4;
p.Xo = 1;
p.S1 = 0.5;

// Simulate the first part up to 20 time units

m1 = p.sim.eval (0, 20, 5, [<p.time>, <p.S1>]);

// Perturb the parameter k1
p.k1 = p.k1*1.7;

// Simulate from the last point
m2 = p.sim.eval (20, 50, 40, [<p.time>, <p.S1>]);

// Restore the parameter back to ordinal value
p.k1 = 0.2;

// Carry out final run of the simulation
m3 = p.sim.eval (50, 80, 40, [<p.time>, <p.S1>]);

// Merge all data sets and plot
m4 = augr(augr(m1, m2), m3);
graph (m4);
```

Listing 4.6 Script for Figure 6.6

5 Running Simulations

5.1 Introduction

In this chapter we will look at ways to solve the differential equations that we generate when we build a model of a system. This may seem unnecessary today given that we have software applications that hide all the details and solutions can be obtained literately at the click of a button. However it is still useful to know that basic approach and limitations of black box solver applications so that if problems do arise one is in a better position to make an informed judgement on how to proceed. To begin, consider the simplest possible model, the first-order irreversible degradation of a molecular species, S into product P:

$$S \rightarrow P$$

The differential equation for this simple reaction is given by the familiar form:

$$\frac{dS}{dt} = -k_1 S \tag{5.1}$$

Our aim is to solve this equation so that we can describe how S changes in time. There are at least two ways to do this, we can either solve the equation mathematically or use a computer to obtain a numerical solution. Mathematically we can either solve the system using traditional methods from calculus or we can use more sophisticated methods such as the Laplace transform, often employed by engineers.

To use the traditional method we move the dependent variables onto one side and the independent variables onto the other, this is called the separation of variables. In the above equation one can easily do this by dividing both sides by S to give:

$$\frac{dS}{dt}\frac{1}{S} = -k_1 \tag{5.2}$$

In differential calculus, the derivative of $\ln y$ with respect to t is

$$\frac{d \ln y}{dt} = \frac{dy}{dt}\frac{1}{y}$$

This means we can rewrite equation 5.2 in the following form:

$$\frac{d \ln S}{dt} = -k_1$$

We now integrate both sides with respect to dt, that is:

$$\int \ln S dt = -k_1 \int dt$$

$$\ln S = -k_1 t + C$$

where C is the constant of integration. If we assume that at $t = 0$, $S = S_o$, then $\ln S_o = C$. Substituting this result into the solution gives:

$$\ln S = -k_1 t + \ln S_o$$

$$\ln\left(\frac{S}{S_o}\right) = -k_1 t$$

Taking anti-natural logarithms on both sides and multiplying both sides by S_o gives:

$$S = S_o e^{-k_1 t} \tag{5.3}$$

For simple systems such as this, it is possible to obtain analytical solutions but very quickly we are confronted with the fact that for 99.99% of real problems, no mathematical solution exists. In such cases, we must carry out numerical simulations.

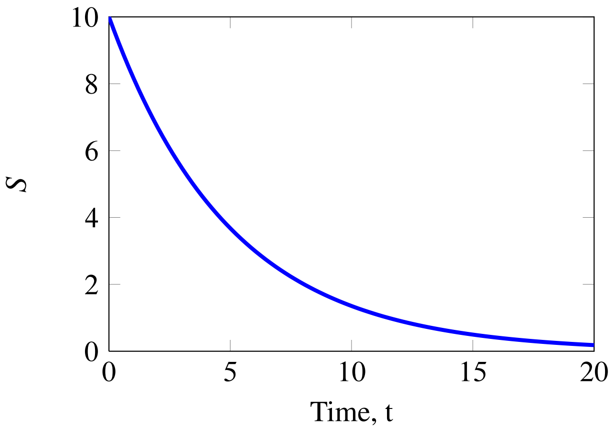


Figure 5.1 Expponential decay from the equation: $S = S_o e^{k_1 t}$ where $S_o = 10, k_1 = 0.2$.

5.2 Numerical Solutions

In the last section we saw how it was possible to solve a differential equation mathematically. If the system of differential equation is linear there are systematic methods for deriving a solution. Most of the problems we encounter in biology however are non-linear and for such cases mathematical solutions rarely exist. Because of this, computer simulation is often used instead. Since the 1960s, almost all simulations have been carried out using digital computers. Before the advent of digital computers, analog computer were frequently used where an analog of the system was built using either mechanical or more commonly, electrical analogs of concentrations. Here we will focus on methods used on digital computers.

The general approach to obtaining a solution by computer is as follows:

1. Construct the set of ordinary differential equations, with one differential equation for every molecular species in the model.
2. Assign values to all the various kinetic constants and boundary species.
3. Initialize all floating molecular species to their starting concentrations.
4. Apply an integration algorithm to the set of differential equations.
5. If required, compute the fluxes from the computed species concentrations
6. Plot the results of the simulation.

Step four is obviously the key to the procedure and there exist a great variety of integration algorithms. We will describe three common approaches to give a flavor of how they work. Other than for educational purposes (which is significant), it is rare now for a modeler to write their own integration computer code because many libraries and applications now exist that incorporate excellent integration methods. Here we will focus on some of the the approaches themselves and some of the software tools that can be used.

An integration algorithm approximates the behavior of what is, strictly speaking, a continuous system, on a digital computer. Since digital computers can only operate in discrete time, the algorithms convert the continuous system into a discrete time system. This is the reason why digital computers can only generate approximations. In practice a particular discrete step size, h, is chosen, and solution points are generated at the discrete points up to some upper time limit. As we will discover, the approximation generated by the simplest method is dependent on the size of the step size and in general the smaller the step size the more accurate the solution. However since computers can only represent numbers to a given precision (usually 15 digits on modern computers), it is not possible to continually reduce the step step in the hope of increasing the accuracy. For one thing, the algorithm will soon reach the limits of the precision of the computer and secondly, the smaller the step size the longer it will take to compute the solution. There is therefore often a tradeoff made between accuracy and computation time.

Let us first consider the simplest method, the Euler method, where the tradeoff between accuracy and computer time can be clearly seen.

Euler Method

The Euler method is the simplest possible way solve a set of ordinary differential equations. The idea is very simple. Consider the following differential equation that describes the degradation rate of a species, S:

$$\frac{dS}{dt} = -k_1 S$$

The Euler method uses the rate of change of S to predict the concentration at some future point in time. Figure 5.2 describes the method in detail. At time t_1, the rate of change in S is computed from the differential equation using the known concentration of S at t_1. The rate of change is used to compute the change in S over a time interval h, using the relation, $h\ dS/dt$. The current time, t_1 is incremented by the time step, h and the procedure repeated again, this time starting at t_2. The method can be summarized by the following two equations which represent one step in an iteration that repeats until the final time point is reached:

$$\begin{aligned} y(t+h) &= y(t) + h\,\frac{dy(t)}{dt} \\ t_{n+1} &= t_n + h \end{aligned} \tag{5.4}$$

Figure 5.2 also highlights a problem with the Euler method. At every iteration, there will be an error between the change in S we predict and what the change in S should have been. This error is called the **truncation error** and will accumulate at each iteration. If the step size is too large, this error can make the method numerically unstable resulting in wild swings in the solution.

Figure 5.2 also suggests that the larger the step size the larger the truncation error. This would seem to suggest that the smaller the step size the more accurate the solution will be. This is indeed the case, up to a point. If the step size becomes too small there is the risk that roundoff error which will propagate at each step into the solution. In addition, if the step size is too small it will require a large number of iterations to simulate even a small time period. The final choice for the step size is therefore a compromise between accuracy and effort. A theoretical analysis of error propagation in the Euler method indicates that the error accumulated over the entire integration period (called the **global error**) is proportional to the step size. Therefore halving the step size will reduce the global error by half. This means that to achieve even modest accuracy, small step sizes are necessary. As a result, the method is rarely used in practice. The advantage of the Euler method is that it is very easy to implement in computer code or even on a spreadsheet.

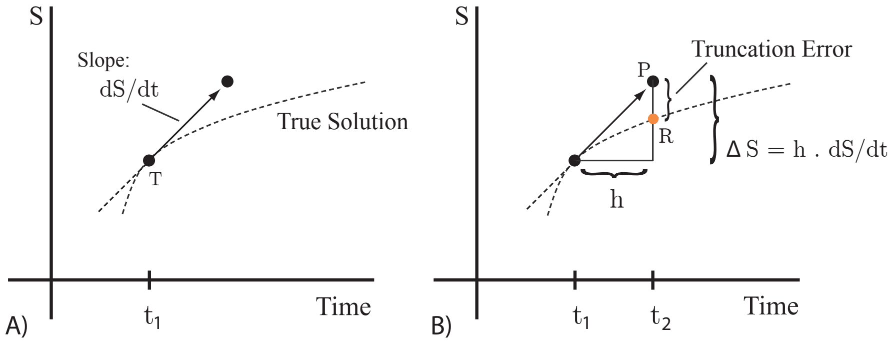


Figure 5.2 Euler Method. Starting at t_1, the slope dS/dt at t_1 is computed (Panel A). The slope is used to project forward to the next solution in time step, h, to t_2 (Panel B). The new solution at t_2 is indicated by P. However the solution is given by point R, located on the solution curve at t_2. Reducing the step size h will reduce the error between the exact and the projected solution, but will simultaneously increase the number of slope projections necessary to compute the solution over a given time period.

The Euler method can also be used to solve systems of differential equations. In this case all the rates of change are computed first followed by the application of the Euler equation 5.4. As in all numerical integration methods, the computation must start with an initial condition for the state variables at time zero. The algorithm is described using pseudo-code in Algorithm 1.

```
$n$ = Number of state variables
$y_i$ = $i^{\text{th}}$ variable
Set timeEnd
currentTime = 0
$h$ = stepSize
Initialize all $y_i$ at currentTime

while currentTime < timeEnd do
    for $i = 1$ to $n$ do
        $dy_i = f_i(y)$

    for $i = 1$ to $n$ do
        $y_i(t + h) = y_i(t) + h\, dy_i$

    currentTime = currentTime + $h$
end while
```

Algorithm 1: Euler Integration Method, $f_i(y)$ represents the i^{th} differential equation from the system of ordinary differential equations.

Example 5.1

Solve the decay differential equation 5.1 using the Euler method. Assume $k_1 = 0.2$ and the concentration of S_o and P are time $= 0$ is 10 and 0 respectively. Assume a step size, h, of 0.4. Form a table of four columns, write out the solution to two three decimal places. The 4^{th} column should include the exact solution (Equation 5.3) for comparison.

Time	Numerical Solution (S)	dS/dt	Exact Solution
0	10	2	10
0.4	9.2	1.84	9.23
0.8	8.464	1.6928	8.52
1.2	7.787	0.01	7.87
...			

Table 5.1 Solution Table using a step size of $h = 0.4$.

Figure 5.3 shows the effect of different steps sizes on the Euler method. Four cases are shown, in the worse case the solution is unbounded and the computer will eventually crash with an overflow error. The second case is where the result is bounded but the solution bears no resemblance at all to the actual solution. The third case shows that the solution is beginning to resemble the actual solution but irregularities appear near the beginning of the integration. The final case shows the actual solution generated from a specialized integrator.

Modified Euler or Heun Method

As indicated in the last section, the Euler method, though simple to implement, tends not to be used in practice because it requires small step sizes to achieve reasonable accuracy. In addition the small step size makes the Euler method computationally slow. A simple modification however can be made to the Euler method to significantly improve its performance. This approach can be found under a number of headings, including the modified Euler method, the Heun or the improved Euler method.

The modification involves improving the estimate of the slope by averaging two derivatives, one at the initial point and another at the end point. In order to calculate the derivative at the end point, the first derivative must be used to predict the end point which is then corrected by using the averaged slope (Figure 5.4). This method is a very simple example of a predictor-corrector method. The method can be summarized by the

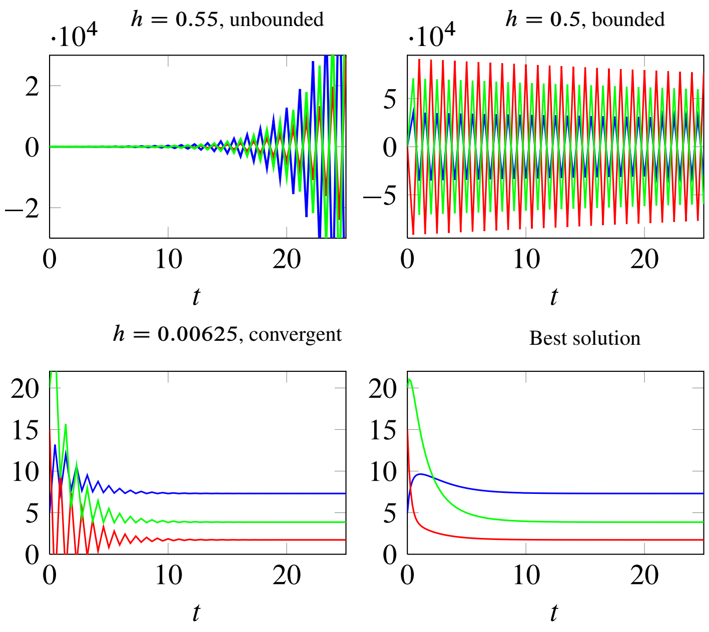


Figure 5.3 Effect of different step sizes on the Euler method using a simple linear chain of reactions where each reaction follows reversible mass-action kinetics: $X_o \underset{k_2}{\overset{k_1}{\rightleftharpoons}} S_1$, $S_1 \underset{k_4}{\overset{k_3}{\rightleftharpoons}} S_2$, $S_2 \underset{k_6}{\overset{k_5}{\rightleftharpoons}} S_3$, $S_3 \overset{k_7}{\rightleftharpoons} X_1$ where $k_1 = 0.45, k_2 = 0.23, k_3 = 0.67, k_4 = 1.2, k_5 = 2.3, k_6 = 0.3, k_7 = 0.73$, $X_o = 10$, $X_1 = 0$, $S_1 = 5$, $S_2 = 15$, $S_3 = 20$.

following equations:

$$\text{Predictor: } y(t+h) = y(t) + h\,\frac{dy(t)}{dt} \tag{5.5}$$

$$\text{Corrector: } y(t+h) = y(t) + \frac{h}{2}\left(\frac{dy(t)}{dt} + \frac{dy(t+h)}{dt}\right) \tag{5.6}$$

$$t_{n+1} = t_n + h \tag{5.7}$$

Figure 5.4 describes the Heun method graphically. A theoretical analysis of error propagation in the Heun method shows that it is a second order method, that is if the step size is reduced by a factor of 2, the global error reduced by a factor of 4. However, to achieve this improvement, two evaluations of the derivatives is required per iteration, compared to only one for the Euler method. Like the Euler method the Heun method is also quite easy to implement.

Runge-Kutta

The Heun method described in the previous section is sometimes called the RK2 method where RK2 stands for 2nd order Runge-Kutta method. The Runge-Kutta methods are a family of methods developed around the 1900s by the German mathematicians, Runge and Kutta. In addition to the 2nd order Heun method there are also 3rd, 4th and even 5th order Runge-Kutta methods. For hand coded numerical methods, the 4th order Runge-Kutta algorithm (often called RK4) is probably the most popular among modelers. The algorithm is a little more complicated in that it involves the evaluation and weighted averaging of four slopes.

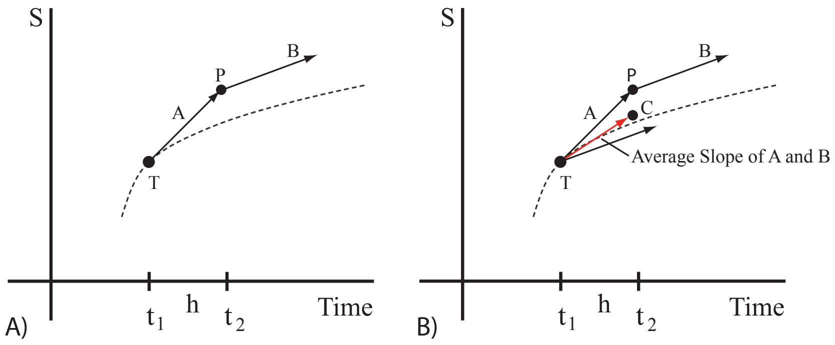


Figure 5.4 Heun Method. Starting at t_1, the slope A at T is computed. The slope is used to predict the solution at point P using the Euler method. From point P the new slope, B is computed (Panel A). Slopes A and B are now averaged to form a new slope C (Panel B). The averaged slope is used to compute the final prediction.

```
$n$ = Number of state variables
$y_i = i^{\text{th}}$ variable
Set timeEnd
currentTime = 0
$h$ = stepSize
Initialize all $y_i$ at currentTime

while currentTime < timeEnd do
   for $i = 1$ to $n$ do
      $a_i = f_i(y)$

   for $i = 1$ to $n$ do
      $b_i = f_i(y + h\,a)$

   for $i = 1$ to $n$ do
      $y_i(t+h) = y_i(t) + \frac{h}{2}(a_i + b_i)$

   currentTime = currentTime + $h$
end while
```

Algorithm 2: Heun Integration Method. $f_i(y)$ is the i^{th} ordinary differential equation

In terms of global error, however, RK4 is considerably better than Euler or the Heun method and has a global error of the order of four. This means that halving the step size will reduce the global error by a factor or 1/16. Another way of looking at this is that the step size can be increased 16 fold over the Euler method and still have the same global error. The method can be summarized by the following equations which have been simplified by removing the dependence on time:

$$
\begin{aligned}
k_1 &= h\ f\left(y_n\right) \\
k_2 &= h\ f\left(y_n + \frac{k_1}{2}\right) \\
k_3 &= h\ f\left(y_n + \frac{k_2}{2}\right) \\
k_4 &= h\ f\left(y_n + k_3\right) \\
y(t+h) &= y(t) + \frac{1}{6}\left(k_1 + 2\,k_2 + 2\,k_3 + k_4\right) \\
t_{n+1} &= t_n + h
\end{aligned}
$$

Figure 5.5 shows a comparison of the three methods, Euler, Heun and RK4 in solving the Van der Pol equations. The Van der Pol equations is a classic problem set that is often used when comparing numerical methods. The equations model an oscillating system, inspired originally from modeling vacuum tubes but also later formed the basis for developments in modeling action potentials in neurons. The Figure shows that the Heun and RK4 methods are very similar, at least for the Van der Pol equations, though this is not always be the case. For this particular model the solution generated by the RK4 method is very similar to the best possible solution that can be obtained by numerical solution. Notice how bad the Euler method is.

```
$n$ = Number of state variables
$y_i = i^{\text{th}}$ variable
timeEnd = 10
currentTime = 0
$h$ = stepSize
Initialize all $y_i$ at currentTime

**while** currentTime < timeEnd **do**
    **for** $i = 1$ to $n$ **do**
        $k_{1i} = hf(y_i)$

    **for** $i = 1$ to $n$ **do**
        $k_{2i} = hf(y_i + k_{1i}/2)$

    **for** $i = 1$ to $n$ **do**
        $k_{3i} = hf(y_i + k_{2i}/2)$

    **for** $i = 1$ to $n$ **do**
        $k_{4i} = hf(y_i + k_{3i})$

    **for** $i = 1$ to $n$ **do**
        $y_i(t + h) = y_i(t) + \frac{h}{6}\,(k_{1i} + 2\,k_{2i} + 2\,k_{3i} + k_{4i})$

    currentTime = currentTime + $h$
**end while**
```

Algorithm 3: 4th Order Runge-Kutta Integration Method

Variable Step Size Methods

In the previous discussion of numerical methods for solving differential equations, the step size, h, was assumed to be fixed. This makes implementation quite straight forward but also make the methods inefficient. For example, if the solution is at a point where it changes very little then the method could probably increase the step size without loosing accuracy while at the same time achieve a considerable speedup in the time it takes

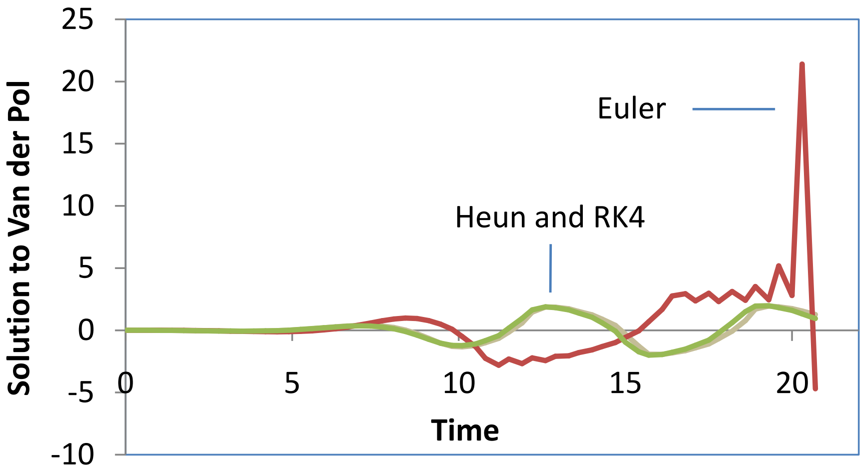


Figure 5.5 Comparison of Euler, Heun and RK4 numerical methods at integrating the Van der Pol dynamic system: $dy_1/dt = y_2$ and $dy_2/dt = -y_1 + (1 - y_1 y_1) y_2$. The plots show the evolution of y_1 in time. The RK4 solution is almost indistinguishable from solutions generated by much more sophisticated integrators. Step size was set to 0.35.

to generate the solution. Likewise if at a certain point in the integration the solution starts to change rapidly, it would be prudent to lower the step size to increase accuracy. Such strategies are implemented in the **variable step size methods**.

The approach used to automatically adjust the steps size can vary from quite simple approaches to very sophisticated methods. The simplest approach is to carry out two integration trials, one at a step size of h and another trial using two steps size but at $h/2$. The software now compares the solution generated by the two trials. If the solutions are significantly different then the step size must be reduced. If the solutions are about the same then it might be possible to increase the step size. These tests are repeatedly carried out, adjusting the step size as necessary as the integration proceeds. This simple variable step size approach can be easily incorporated into some of the simpler algorithms particularly the fourth order Runge-

Kutta where it is called the variable step-size Runge-Kutta. Another approach to adjusting the step size is called the Dormand-Prince method [27]. This method carries out two trials based on the fourth and fifth order Runge-Kutta. Any difference between the two trials is used to adjust the step size. Matlab's ode45 implements the Dormand-Prince method. Similar methods to Dormand-Prince include the Fehlberg `http://en.wikipedia.org/wiki/Runge-Kutta-Fehlberg_method` and more recently the Cash-Karp method [19].

Many of these simple adjustable step size solvers are quite effective. Sometimes they can be slow however especially for the kinds of problem we find in biochemical models. In particular there is a class of problem called stiff problems which generally plagues the biochemical modeling community. Stiff models require highly specialized solvers which have been developed in the last four decades.

Stiff Models

Many differential equations we encounter in biochemical models are so-called **stiff** systems. The word stiff apparently comes from earlier studies on spring and mass systems where the springs had large spring constants and therefore difficult to stretch. A stiff system is often associated with widely different time scales in a system, for example when the rate constants are widely different in a biochemical model. Such systems may have molecular species whose decay rates are very fast compared to other components. This means that the step size has to be very small to accommodate the fast processes even though the rest of the system could be accurately solved using a much larger step size. The overall result is the need for very small steps sizes and therefore significant computational cost and the possibility of roundoff error which will tend to be amplified by the large time constants in the fast system. The net result are solutions which bear no resemblance to the true solution.

Most modern simulators will employ specific stiff algorithms for solving stiff differential equations. Of particular importance is the sundials suite [23] and odepack [43]. Sundials includes a number of very useful, well written and documented solvers. In particular the CVODE solver is very well suited to finding solutions to stiff differential equations. As a result sundials is widely used in the biochemical modeling community (for example by Jarnac and roadRunner). Before the advent of sundials, the main workhorse for solving stiff systems was the suite of routines in odepack. Of particular importance was LSODA which in the 1990s was very popular and is still a valuable set of software (currently used in COPASI). The original stiff differential equation solver was developed by Gear in the 1970s and is still used in Matlab in the form of ode15s.

5.3 Matlab Solvers

Although this isn't a book about Matlab, it is worth mentioning how Matlab can be used to solve differential equation. Matlab offers a range of solvers to solve ordinary differential equations. Possibly the two most common solvers use are the `ode45` and `ode15s` solvers.

The `ode45` solver implements a variable step size Runge-Kutta method. The variable step size is achieved by comparing solutions using a forth and fifth order Runge-Kutta. If the error between the two methods is too big, then the step size is reduced, otherwise if the step size is below a given threshold then the program will attempt to increase the step size. This method is also called the Dormand-Prince method. The basic syntax for `ode45` is:

```
[t,y] = ode45(@myModel, [t0, tend], yo, [], p);
```

where

`myModel` is the function containing the differential equations

`t0`, tend is the initial and final values for the independent variable, t.

`yo` is a vector of initial conditions

`p` is set of parameters for the model, this can be any size.

The empty vector in the call is where additional options particular to ode45 can be placed.

For example, to solve the set of ODEs:

$$\frac{dy_1}{dt} = v_o - k_1 y_1$$

$$\frac{dy_2}{dt} = k_1 y_1 - k_2 y_2$$

We would write the following .m file and load it into Matlab

```
function dy = myModel(t, y, p)
dy = zeros (2,1);
vo = p(1);
k1 = p(2);
k2 = p(3);
dy(1) = vo - k1 y(1);
dy(2) = k1 y(1) - k2 y(2);
```

We would then call the solver as follows:

```
p = [10, 0.5, 0.35]
y0 = [0, 0]
[t, y] = ode45 (@myModel, [0, 20], y0, [], p)
```

Although many problems can be solved using `ode45`, for some models that are stiff, `ode45` is insufficient and will fail to give the correct solution. In these cases `ode15s` is recommended. `ode15s` is a variable order solver and amongst other things it uses the well know method called Gear's method. Like `ode45`, `ode15s` is also a variable step size method. `ode45` might be faster than `ode15s` on simple problems but with today's fast computers the difference is hardly noticeable. Therefore one might as well use `ode15s` for all problems. `ode15s` is called in the same way as `ode45`.

5.4 Other Software

Although sometimes it may seem to be the case, Matlab isn't the only software that can be used to solved differential equations. For example

Mathematica is an example of another commercial tool that can be used to solve differential equations.

For those who require more control or who are unable to purchase a commercial tool there are many free applications and professionally developed open source software libraries that can be used very effectively. Octave (http://www.gnu.org/software/octave/) is an open source tool that is very similar to Matlab, in fact even the syntax is similar if not identical. SciLab (http://www.scilab.org/) is another free Matlab like application. If you like programming in Python then Sage (http://www.sagemath.org/index.html) is an excellent option. There are therefore many alternatives and often free options to using Matlab.

For those who require much more control and higher performance than can be achieved by the tools mentioned above then there is the Sundials C/C++ library developed by the Department of Energy. In particular within Sundials there is the CVODE library that is used by many of the commercial tools. CVODE implements an advanced Gear like algorithm using a variable order and variable step size approach. It is highly suited for stiff systems and is the preferred method for those who need to write their own code. One final library worth mentioning is the GPL (GNU General Public License) licensed GSL library (http://www.gnu.org/software/gsl/. Although very comprehensive, the GPL license restricts its use to other GPL licensed source code and therefore its ODEs solvers cannot be used as easily as the CVODE library.

5.5 Specialized Software

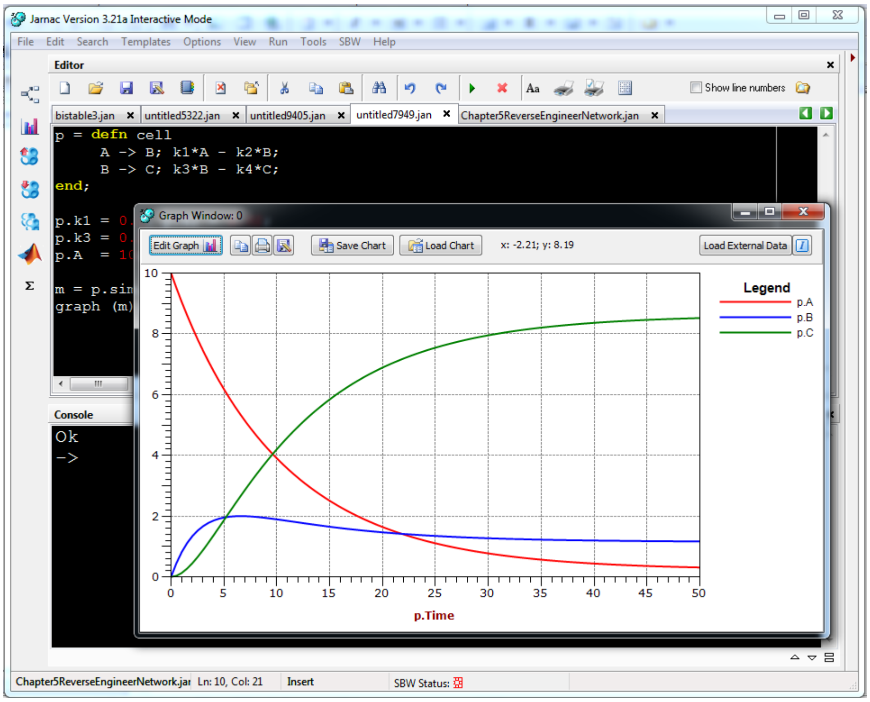
Simulating biochemical networks has a long history dating back to at least the 1940s [20]. The earliest simulations relied on building either mechanical or electrical analogs of biochemical networks. It was only in the late 1950s, with the advent of digital computers and the development of specialized software tools [33] that the ability to simulate biochemical networks became more widely available. In the intervening years up to 1980, a handful of other software applications were developed [17, 18, 82] to help the small community of modelers. In more recent years, particularly since the early 1990s, there has been a significant increase in interest in modeling biochemical processes and a wide range of tools is now available to the budding systems biologist. Many tools have been developed by practicing scientists and are therefore available for free and often as open source.

In this book we will be using the authors' modeling tool Jarnac [91]. The main reason for this is that Jarnac is a script based modeling application which makes it easy to show the model script in the text of the book. Most tools do not allow this because either they use a visual approach to modeling, such as JDesigner [8] or CellDesigner [55] or use graphical user interfaces such as COPASI [47] or iBiosim [78]. All these tools however export and import the standard modeling language SBML (See section 5.7). However SBML is written in XML is it is not a suitable format for displaying in a text book.

Jarnac

Jarnac [91] is a rapid prototyping script based tool that was developed as a successor to SCAMP [95]. It is distributed as part of the Systems Biology Workbench which makes installation a one-click affair. Jarnac was developed in the late 1990s before the advent of portable GUI toolkits which explains why it only runs under Windows although it runs well under Wine (Windows emulator) thus permitting it to run under Linux or the Apple Mac. Visually, Jarnac has two main windows 5.6, a console where commands can be issued and results returned and an editor where control scripts and models can be developed. The application also has a plotting window which is used when graphing commands are issued.

Jarnac implements two languages, a biochemical descriptive language which allows users to enter models as reaction schemes (similar to a SCAMP script [95]) and a second language, the model control language which is a full featured scripting language that can be used to manipulate and analyze a model. The main advantage of Jarnac over other tools is that models can be very rapidly built and modeled. From the authors own experience with using simulation tools over the years, Jarnac probably offers the fastest development time for model building of any tool. Models can also be imported or exported as SBML. Jarnac offers many analysis capabilities including support for metabolic control analysis, structural analysis of networks and stochastic simulation. It has no explicit support for parameter fitting but this is easily remedied by transferring a model directly to a tool such as COPASI via SBW. A more detailed description of Jarnac is given in Appendix E.


Jarnac Version 3.21a Interactive Mode
File Edit Search Templates Options View Run Tools SBW Help
Editor
Show line numbers
bistable3.jan
untitled5322.jan
untitled9405.jan
untitled7949.jan
Chapter5ReverseEngineerNetwork.jan
Console
Graph Window: 0
Edit Graph
Save Chart
Load Chart
x: -2.21; y: 8.19
Load External Data
Legend
p.A
p.B
p.C
p.Time
Chapter5ReverseEngineerNetwork.jan Ln: 10, Col: 21 Insert SBW Status:

```
p = defn cell
     A -> B; k1*A - k2*B;
     B -> C; k3*B - k4*C;
end;

p.k1 = 0.
p.k3 = 0.
p.A  = 10

m = p.sim
graph (m)
```

```
Ok
->
```

Figure 5.6 Screen shot of Jarnac with simulation results

5.6 Stochastic Kinetics

Up to now we've considered how to run simulations of continuous models that can be described using ordinary differential equations. However, as mentioned in Chapter 2 there is a strong case to model reaction systems using a discrete stochastic approach. The reason for this is that at low numbers of molecules, individual reaction events can play a dominant role in determining the evolution of the system. Since the exact time at which a reaction will occur is only known as a probability, simulating reaction events becomes a stochastic process. In addition, since we're dealing with individual reaction events, we must keep track of exact molecule numbers. The companion book, 'Enzyme Kinetics for Systems Biology' [93], describes in detail the theory and algorithms behind stochastic simulations and here we will only describe the basic algorithm and software that can be used to simulate a stochastic system. The most common approach to stochastic simulation is to use the Gillespie method for simulation. Consider a system described by the reaction network:

$$X_o \overset{v_1}{\rightarrow} S_1 \overset{v_2}{\rightarrow} S_2 \overset{v_3}{\rightarrow} X_1$$

At each iteration in the simulation, two random choices are considered, the first is which reaction among the three will 'fire', that is react? The second random choice is to decide when in the future the selected reaction will actually 'fire'. In addition we must also decrement the number of reactants by one and increase the number of products by one for the chosen reaction. For example if the second reaction is chosen to react then when it reacts one molecule of S_1 will disappear and one molecule of S_2 will appear. The method is applied repeatedly until a maximum time for simulation is reached.

There are many variants on the method, some are faster and some are approximate. The review by Pahle [81] covers many of these variants and is well worth consulting. There are a number of software tools that support stochastic simulation, examples include COPASI [47], Dizzy [88] and Jarnac [91]. Manninen et al [67] has a review of some of the software tools. Here we will show how to carry out a stochastic simulation using Jarnac. Figure 5.1 shows a Jarnac listing that generated the plots shown in Figure 5.7. The key line in the script is:

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

This takes four arguments. The first argument is the model variable, in this case, p. The second and third arguments set the time start and time end for the simulation. The forth argument sets the columns in the matrix that will be returned by the Gillespie method, in this case time and the levels of the two variables, S_1 and S_2. Note that the amounts for the species have been set to integer values reflecting the fact that we are now dealing with discrete molecules.

```
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);
```

Listing 5.1 Script for Figure 6.6

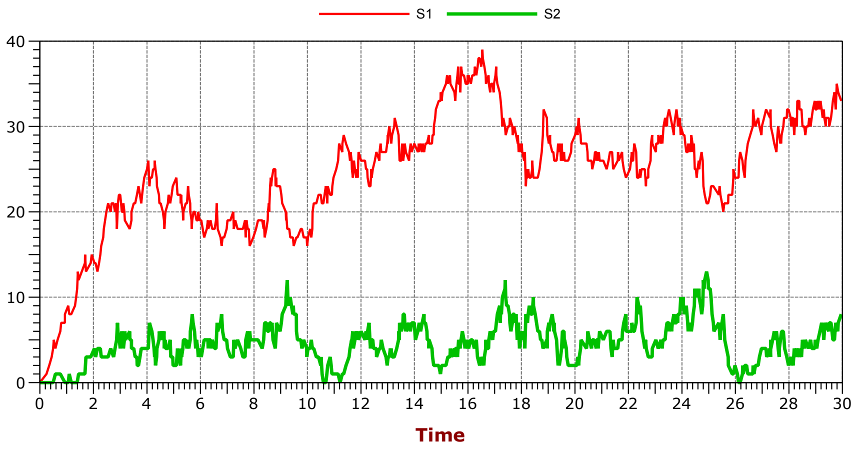


Figure 5.7 Stochastic Simulation using Jarnac.

As with continuous simulations, it is possible to carry out a number separate runs where other events can be imposed in between the runs. For example, we might want to decrease one of the rate constants by a factor of six at a certain time point in the simulation and then carry the simulation on. Listing 5.2 shows one simulation being carried out from 0 to time 30. At this point one of the rate constants is decreased six fold, then the simulation is started up again, but this time setting the time start to the end time of the previous simulation. Finally, both matrices from the two runs are merged and the entire simulation plotted.

```
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));
```

Listing 5.2 Script for Figure 6.6

5.7 Modeling Standards and Databases

The last 10 years has seen a significant increase in the number of simulation tools, all of which use different formats to store models. It was soon realized that some form of standardization for model exchange was necessary. As result two proposed standards emerged: CellML [40] and SBML [48] (Systems Biol9ogy Markup Language). CellML is primarily a notation for representing biochemical models in a strict mathemat-

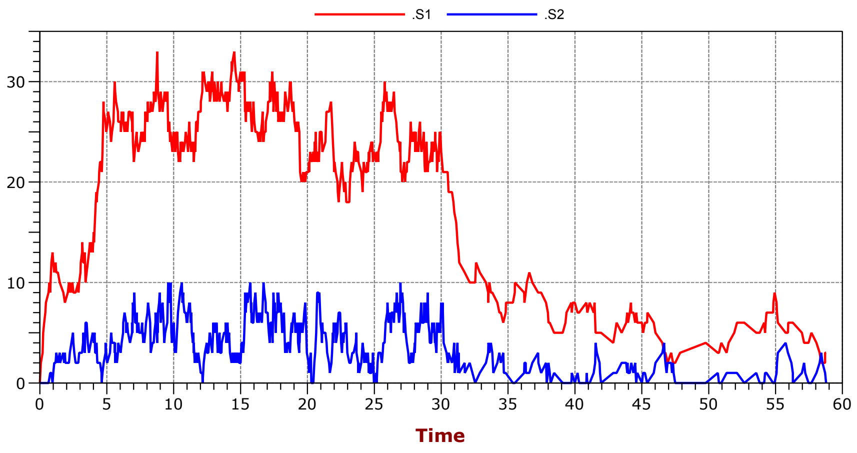


Figure 5.8 Stochastic Simulation using Jarnac showing how any event can be superimposed between two consecutive simulations.

ical form, as a result it is in principle completely general. SBML on the other hand uses a biologically inspired notation to represent networks from which a mathematical model can be generated. Each has its strengths and weaknesses, SBML has a simpler structure compared to CellML, as a result there is more software support for SBML. Most software tools at the present time support import and export of SBML. Both standards have very active communities with intra cellular models being primarily the domain of SBML and physiological models for CellML. Here we will focus on SBML.

SBML

SBML is based on XML and closely follows the way existing modeling packages represent models. For example SBML represent biochemical networks as a list of chemical transformations. SBML employs specific elements to represent spatial compartments, molecular species and parameters. In addition to these, SBML also has provision for rules which can be used to represent constraints, derived values and general math which for one reason or another cannot be transformed into a chemical scheme.

SBML, like any standard, evolves with time sbml.org. Major revisions of the standard are captured in levels, while minor modifications and clarifications are captured in versions. An example of a major change within the standard would be the use of MathML in level two of SBML, whereas level one encoded infix strings to denote reaction rates and rules. A minor change on the other hand would for example be the introduction of semantic annotations that can be added to SBML level two version three, whereas this was not possible in a supported fashion in earlier versions (see section 5.7). The latest level of SBML is level three where new functionality can be supported through extension packages.

Graphical Layout

Graphical modeling applications [8] routinely enhance computational models by layout annotations. Recently the SBML community has decided on a common standard on how to embed the layout information within SBML. The layout extension [35] allows a model to store the size and dimension of all model elements, along with textual annotations and reactions. Originally the intention was to embed the layout extension in a model annotation for level 2 versions of SBML but with the upcoming level 3 the layout extension will be added to the SBML as a first-class construct. LibSBML has been modified to provide access to all elements of the Layout Extension. Also several reference implementations exist [8, 25].

Whereas the layout extension is concerned with representing simple elements, the Systems Biology Graphical Notation (SBGN) (http://sbgn.org) aims to standardize the visual language of computational models unambiguously. While this standard is still in development and strictly speaking independent of the SBML effort, experience in other fields such as electrical engineering has demonstrated the essential need for standardizing the visual notation for representing models in diagrammatic form.

MIRIAM

Model Definition Languages such as SBML and CellML target the exchange of models. They aim to pass on the quantitative computational models from one software tool to another. However these description formats do not concern themselves with semantic annotations. Both SBML and CellML have launched efforts to remedy this problem. Both communities agreed on the Minimum Information Requested In the Annotation of biochemical Models (MIRIAM, [59]). These annotations aim to further the confidence in quantitative biochemical models, making it easier and more precise to search for particular biochemical models, enabling researchers to identify biological phenomena captured by a biochemical model and perhaps most importantly to facilitate model reuse and model composition.

In order to call a model MIRIAM compliant, the model has to be encoded in a standard format, such as SBML. Furthermore it needs to be tied to a reference description, describing the properties and results that can be obtained from the model. Parameters of the computational model have to be provided so that the model can be loaded into a simulation environment where the results can be reproduced. Other information that has to be provided is a name for the model, the creator of the model the date and time of the last modification as well as a statement about the terms of distribution.

SBO – Systems Biology Ontology

In order to assign meaning to model constituents an ontology specific to Systems Biology has been developed: The Systems Biology Ontology (SBO, `http://www.ebi.ac.uk/sbo/`). The ontology consists of five controlled vocabularies and two relationships: is-part-of and is-a. Qualifying model participants, say as enzyme, macromolecule, metabolite or small species such as an ion will make it easier to generate meaning from the model. It will make the generation of standard visual notations such as SBGN possible. Moreover it presents a solution on how to interpret the model computationally, as the SBO allows tagging a model as continuous, discrete or logical model. One could even go a step further, making kinetic interactions in a model obsolete, by just referencing that the rate law is one specified by an ontology identifier (e.g.: tagging a reaction as following Henri-Michaelis Menten enzyme kinetics and specifying the parameters). The SBO is community driven and new terms or modifications to the existing ontology can be requested by the community.

Other Ontologies

The most recent developments in CellML and particularly the SBML communities revolve around the creation of ontologies and refining the exchange semantics. Apart from classifying model constituents with an appropriate ontology, one of the current areas of interest is describing the dynamical behavior of a model. The “Terminology for the Description of Dynamics“ (TEDDY, [1]) provides a rich ontology to describe and quantify what kind of behavior a computational model is able to exhibit (e.g.: the characteristics of a model could describe bifurcation behavior where the functionality of a model could be described as featuring oscillations or switch behavior). However knowing that a model exhibits interesting behavior is not enough: more information is needed in order to recreate that behavior. The “Minimum Information About a Simulation Experiment” (MIASE, [2]) project focuses in this problem. MIASE will help to describe the simulation algorithms and the simulation tool used along with all needed parameter settings. In order to do so it will use the Kinetic Algorithm Ontology (KiSAO) that relates simulation algorithms and methods to each other. As these ontologies are still currently under development, it will be interesting to see how they progress and are taken up by the community.

Lastly we should mention BioPAX [64], Biological Pathway Exchange. BioPAX is an XML based format that will act as a bridge between different pathway databases and data. In relation to modeling software, BioPAX may offer a means to embed rich annotation data into an SBML or CellML model. Some of this capability is being addressed to a limited extend by

[1] `http://www.ebi.ac.uk/compneur-srv/teddy/`
[2] `http://www.ebi.ac.uk/compneur-srv/miase/`

the new ontologies being developed at EBI in Cambridge, UK. However, BioPAX, given it role to allow a common exchange of biological data between pathway database, may offer a useful complementary way to bind data to computational models.

Human Readable Formats

SBML and CellML are examples of formats that use XML to represent information. One advantage to using XML is that there is much software available to assist in reading and manipulating XML based data. However, XML is not suited for human consumption but is designed strictly to be read by computer software. In order for humans to build and read models, human readable formats are required, often these are text based but sometimes they are graphical. In relation to text based formats, there has been a long tradition to using human readable formats for representing biochemical models, starting with BIOSSIM [34]. Other examples of early human readable formats include work by Park [82] and Burns [18] to cite but a few. In more recent years simulators such as SCAMP [95] and METAMOD [46] also introduced human readable formats to define models. Both software tools were developed in later years into Jarnac and PySCeS respectively.

Other formats of interest include composable languages developed by James McCollum at the University of Miami, Sauro and Bergmann [7] at the University of Washington and Michael Pederson at the University of Edinburgh [83]. Blinov, Faeder, Goldstein and Hlavacek developed BioNetGen [12] which is a rule based format for representing systems with multiple states, Cyto-Sim, which we have mentioned previously, incorporates an interesting human readable language for representing biochemical systems. The SBML community [122] has also developed a human readable script called SBML-shorthand. This notation maps directly on to SBML but is much easier to hand write compared to SBML (as are all these human readable formats). The shorthand is also much less verbose and uses infix to represent expressions rather than MathML. Finally we should mention a lisp based language called little b (http://www.littleb.org/) being developed at Harvard University. The aim of little b is to allow biologists to build models quickly and easily from shared parts.

Databases

Along with the standardization of model representation there has been an obvious desire to create model repositories where models published in journals can be stored and retrieved. There are at the present time, five repositories with varying degrees of quality and usability. Probably the most promising is the UK based searchable , BioModels Database, which at the current time (July 2008) holds over one hundred and seventy fully curated and working models that can be downloaded in standard SBML as well as other formats. BioModels also has the great benefit of providing programmatic access to its database via web services which allows any software program to access the database seamlessly across the internet. Models stored in the BioModels Database are curated, meaning that models will reproduce the original's authors intention. In addition, the models are liberally annotated so that model components can be referenced from other database sources.

Another large database has been assembled by the CellML community [62] which has over three hundred models stored in CellML format. From their site it is possible in principle to convert the CellML into standard C code for compilation in to a working model.

The JSim group at the University of Washington has a large database of physiological models http://nsr.bioeng.washington.edu/Models/ stored in the mathematical language used by the JSim simulation application. These models can only be read by JSim and currently there is no simply way to translate these to any of the common exchange formats though this is likely to change in the future.

Another small but very useful database is the JWS online database developed by Brett Olivier and Jacky Snoep [80] which has almost eighty fully working models. JWS allows export in both SBML and the script format PySCeS which can be easily translated to other formats such as Jarnac script. JWS is arguably one of the first databases of models although physiological models such as those supported by JSim have been available for longer. Many if not all the models on JWS have also been ported to the

BioModels Database and vice versa.

Another database, DOQCS `http://doqcs.ncbs.res.in/`focuses on signaling networks which contains over two hundred models. Models in DOQCS can only be downloaded however in Genesis format [10] which limits the portability to other frameworks. Recently the DOQCS database has been merged with the BioModels Database.

Further Reading

Unfortunately there are very few reasonably priced books on numerical analysis. The two most popular expensive books are by Press and Burden and are included here for reference. Both books can be bought second-hand at reasonable prices and the content has not changed significantly between editions. One could even argue that the code examples in the latest edition of Press are actually worse than the previous editions. However the main problem with the Press book is that the source code itself has very strict licensing rules making the code almost worthless. However, the book is excellent at explaining how the various algorithms work and for this reason alone the book is worth it. The 2nd edition can now be purchased at very low prices.

1. Press, Teulolsky, Vetterling and Flannery (2007) Numerical Recipes. Cambridge University Press, 3rd Edition ISBN-10: 0521880688

2. Burden and Faires (2010) Numerical Analysis. Brooks Cole, 9th Edition. ISBN-10: 0538733519

3. Pahle J Biochemical simulations: stochastic, approximate stochastic and hybrid approaches. Briefings in Bioinformatics 10(1), 53-64, 2009

4. Sauro HM (2011) Enzyme Kinetics for Systems Biology, 2nd Edition. ISBN: 978-0982477335

5. Sauro, HM, and Bergmann FT (2009) Software Tools for Systems Biology in Systems Biomedicine: Concepts and Perspectives, edited

by Edison T. Liu, Douglas A. Lauffenburger. ISBN 978-0-12-372550-9

For the budget conscious buyer I can highly recommend the Dover edition:

1. Dahlquist and Björck (2003) Numerical Methods. Dover Publications ISBN-10: 0486428079

Exercises

1. Implement the Euler method in your favorite computer language and use the code to solve the following two problems. Set initial conditions, $S_1 = 10, S_2 = 0$. Set the rate constants to $k_1 = 0.1; k_2 = 0.25$. Investigate the effect of different steps sizes, h, on the simulation results.

   a) $dS_1/dt = -k_1 S_1$

   b) $dS_1/dt = -k_1 S_1;\ dS_2/dt = k_1 S_1 - k_2 S_2$

2. The following model shows oscillations in S_1 and S_2 at a step size of $h = 0.044$ when using the Euler method. By using simulation, show that these oscillations are in fact an artifact.

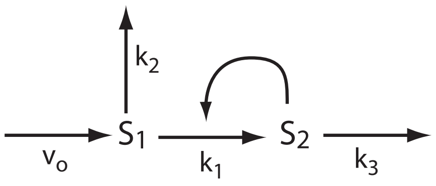
```
p = defn cell
      $Xo -> S1; k1*Xo;
       S1 -> S2; k2*S1;
       S2 -> $X1; k3*S2;
end;

p.k1 = 23.4; p.k2 = 45.6; p.k3 = 12.3
p.Xo = 10; p.S1 = 0; p.S2 = 0;
```

3. Find out what differential equation solver the Python SciPy Packages supports.

4. Construct of a model of the following system using Jarnac. Let the reaction associated with the positive feedback (k_1) be governed by the following rate law:

$$k_1 S_1 (1 + c S_2^q)$$

All other reactions are governed by first-order kinetics except the first reaction which has a constant rate of v_o. Set the constants to the following values: $v_o = 8; c = 1.0; k_1 = 1; k_2 = 5; k_3 = 1$ and $q = 3$. Study the effect of changing v_o on the dynamics of the system.



5. Download the model `BIOMD0000000010` from Biomodels and load it into Jarnac. Run a simulation of the model.

6 The Steady State

6.1 Steady State

In chapter 4 we briefly introduced the idea of the steady state. In this chapter we will investigate the steady state in more details.

The **steady state** is one of the most important states to consider in a dynamical model. In the literature it is also sometimes referred to as the stationary solution or state, singular points, fixed points, or even equilibrium. We will avoid the use of the term equilibrium because of possible confusion with thermodynamic equilibrium.

The steady state is the primary reference point from which to consider a model's behavior. At steady state, the concentrations of all molecular species are constant and there is a net flow of mass through the network. This is in contrast to systems at thermodynamic equilibrium, where, although concentrations are constant there is no net flow of mass across the system's boundaries.

The steady state is where the rates of change of all species, dS/dt are zero but at the same time the net rates are non-zero, that is $v_i \neq 0$. This situation can only occur in an open system, that is a system that can exchange matter with the surroundings.

Equation 4.3 from a previous chapter described the time evolution for the system

$$X_o \overset{v_1}{\rightarrow} A \overset{v_2}{\rightarrow} B \overset{v_3}{\rightarrow} X_1$$

we repeat the equations here for convenience:

$$\begin{aligned} A(t) &= v_o \frac{1 - e^{-k_1 t}}{k_1} \\ B(t) &= v_o \frac{k_1 \left(1 - e^{-k_3 t}\right) + k_3 \left(e^{-k_1 t} - 1\right)}{k_3 \left(k_1 - k_3\right)} \end{aligned} \tag{6.1}$$

As t tends to infinity, $A(t)$ and $B(t)$ tends to:

$$A(\infty) = \frac{v_o}{k_1} \qquad B(\infty) = \frac{v_o}{k_3}$$

The reaction rate through each of the three reaction steps is v_o. This can be confirmed by substituting the solutions for A and B into the reaction rate laws. Given that v_o is greater than zero and that A and B reach constant values given sufficient time, we conclude that this system eventually settles to a steady state rather than thermodynamic equilibrium. The system displays a continuous flow of mass from the source to the sink. This can only continue undisturbed so long as the source material, X_o never runs out and that reaction k_3 is irreversible. Figure 6.1 shows a simulation of this system.

Graphical Procedure

We can also illustrate the steady state using a graphical procedure. Consider the simple model below:

$$\begin{tikzcd} X_o \arrow[r, "v_1"] & S_1 \arrow[r, "v_2"] & X_1 \end{tikzcd}$$

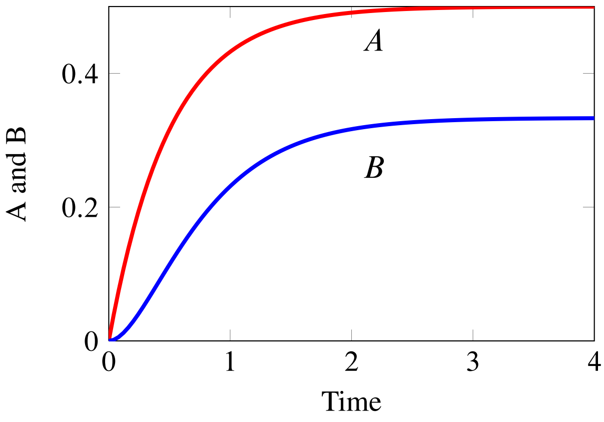


Figure 6.1 Time course for an open system reaching steady state. $X_o \overset{k_o}{\rightarrow} A \overset{k_1}{\rightarrow} B \overset{k_3}{\rightarrow}$ where $v_o = 1, k_1 = 2, k_3 = 3, A_o = 0, B_o = 0$. X_o is assumed to be fixed. Jarnac model: 6.3

where X_o and X_1 are fixed boundary species and S_1 is a species that can change (also called floating species). For illustration purposes we will assume some very simple kinetics for each reaction, v_1 and v_2. Let us assume that each reaction is governed by simple first order mass-action kinetics,

$$\begin{aligned} v_1 &= k_1 X_o \\ v_2 &= k_2 S_1 \end{aligned}$$

where k_1 and k_2 are both first-order reaction rate constants. In Figure 6.2 both reaction rates have been plotted as a function of the floating species concentration, S_1.

Note that the reaction rate for v_1 is a horizontal line because it is unaffected by changes in S_1 (no product inhibition). The second reaction, v_2 is shown as a straight line with slope, k_2. Notice that the lines intersect. The intersection marks the point when both rates, v_1 and v_2 are equal. This point marks the steady state concentration of S_1. By varying the value of k_2 we can observe the effect it has on the steady state. For example, Fig-

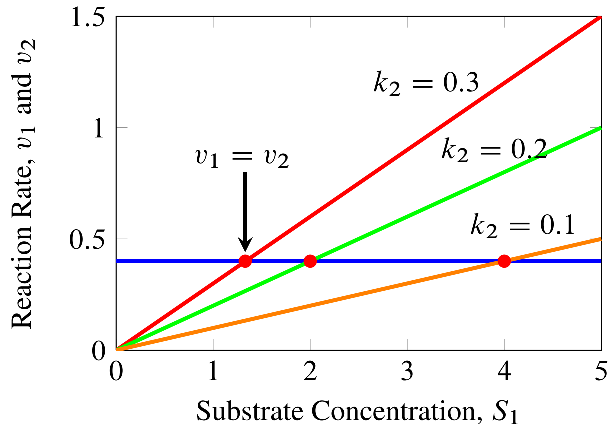


Figure 6.2 Plot of reaction rates versus concentration of S_1 and different values for k_2 for the system $X_o \rightarrow S_1 \rightarrow X_1$. The intersection of the two lines marks the steady state point where $v_1 = v_2$. $X_o = 1, k_1 = 0.4$. Note that as k_2 is decreased the steady state level of S_1 increases.

ure 6.2 shows that as we **decrease** k_2 the concentration of S_1 increases. This should not be difficult to understand, as k_2 decreases, the activity of reaction v_2 also decreases. This causes S_1 to build up in response.

In this simple model it is also straightforward to determine the steady state of S_1 mathematically which amounts to finding a mathematical equation that represents the intersection point of the two lines. We recall that the model for this system comprises a single differential equation:

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1$$

At steady state, we set $dS_1/dt = 0$, from which we can solve for the steady state concentration of S_1 as:

$$S_1 = \frac{k_1 X_o}{k_2} \tag{6.2}$$

This solution tells us that the steady state concentration of S_1 is a function of **all** the parameters in the system. We can also determine the steady state rate, usually called the pathway flux and denoted by J, by inserting the steady state value of S_1 into one of the rate laws, for example into v_2:

$$J = k_2 \frac{k_1 X_o}{k_2} = k_1 X_o$$

This answer is identical to v_1 which is not surprising since in this model the pathway flux is completely determined by the first step and the second step has no influence whatsoever on the flux. This simple example illustrates a rate limiting step in the pathway, that is one step, and one step only, that has complete control over the pathway flux.

More Complex Model

A slightly more realistic model is the following:

$$\begin{tikzcd} X_o \arrow[r, "v_1"] & S_1 \arrow[r, "v_2"] & S_2 \arrow[r, "v_3"] & X_1 \end{tikzcd}$$

where the rate law for the first step is now reversible and is given by:

$$v_1 = k_1 X_o - k_2 S_1$$

The remaining steps are governed by simple irreversible mass-action rate laws, $v_2 = k_3 S_1$ and $v_3 = k_4 S_2$. The differential equations for this system are:

$$\begin{aligned} \frac{dS_1}{dt} &= (k_1 X_o - k_2 S_1) - k_3 S_1 \\ \frac{dS_2}{dt} &= k_3 S_1 - k_4 S_2 \end{aligned}$$

The steady state solution for S_1 and S_2 can be obtained by setting both differential equations to zero to yield:

$$\begin{aligned} S_1 &= \frac{k_1 X_o}{k_2 + k_3} \\ S_2 &= \frac{k_3 k_1 X_o}{(k_2 + k_3) k_4} \end{aligned}$$

The steady state flux, J, can be determined by inserting one of the solutions into the appropriate rate law, the easiest is to insert the steady state level of S_2 into v_3 to yield:

$$J = \frac{k_3 k_1 X_o}{k_2 + k_3}$$

Once the first step is reversible we see that the steady state flux is a function of all the parameters except k_4 indicating that the first step is no longer the rate limiting step. The equation shows us that the ability to influence the flux is shared between the first and second steps. There is no rate limiting step in this pathway. Note that if we set $k_2 = 0$ then the solution reverts to the earlier simpler model.

We can also make all three steps reversible $(k_f S_i - k_r S_{i+1})$, so that the solution is given by:

$$\begin{aligned} S_1 &= \frac{X_o k_1 (k_4 + k_5) + X_1 k_4 k_6}{k_3 k_5 + k_2 (k_4 + k_5)} \\ S_2 &= \frac{X_1 k_6 (k_2 + k_3) + X_o k_1 k_3}{k_3 k_5 + k_2 (k_4 + k_5)} \end{aligned}$$

The last example illustrates the increase in complexity of deriving a mathematical solution after only a modest increase in model size. In addition, once more complex rate laws as used, such as Hill equations or Michaelis-Menten type rate laws, the solutions become exceedingly difficult to derive. As a result steady states tend to be computed numerically rather than analytically.

6.2 Computing the Steady State

In those (many) cases were we cannot derive an analytical solution for the steady state we must revert to numerical methods. There are at least two methods that can be used here. The simplest approach is to run a time course simulation for a sufficiently long time so that the time course trajectories eventually reach the steady state. This method works so long as the steady state is stable, it cannot be used to locate unstable steady states because such trajectories diverge. In addition, the method can sometimes be very slow to converge depending on the kinetics of the model. As a result, many simulation packages will provide an alternative method for computing the steady state where the model differential equations are set to zero and the resulting equations solved for the concentrations. This type of problem is quite common in many fields and is often represented mathematically as the quest to find solutions to equations of the following form:

$$f(x, p) = 0 \tag{6.3}$$

where x is the unknown and p one or more parameters in the equations. All numerical methods for computing solutions to equation 6.3 start with an initial estimate for the solution, say x_o. The methods are then applied iteratively until the estimate converges to the solution. One of the most well known methods for solving equation 6.3 is called the **Newton-Raphson method**. It can be easily explained using a geometric argument, Figure 6.3. Suppose x_1 is the initial guess for the solution to equation 6.3. The method begins by estimating the slope of equation 6.3 at the value x_1, that is df/dx. A line is then drawn from the point $(x_1, f(x_1))$, with slope df/dx until it intersects the x axis. The intersection, x_2, becomes the next guess for the method. This procedure is repeated until x_i is sufficiently close to the solution. For brevity the parameter, p, is omitted from the following equations. From the geometry shown in Figure 6.3 one can write down the slope of the line, $\partial f/\partial x_1$ as:

$$\frac{\partial f}{\partial x_1} = \frac{f(x_1)}{x_1 - x_1}$$

This can be generalized to:

$$\frac{\partial f}{\partial x_k} = \frac{f(x_k)}{x_k - x_{x+1}}$$

or by rearrangement:

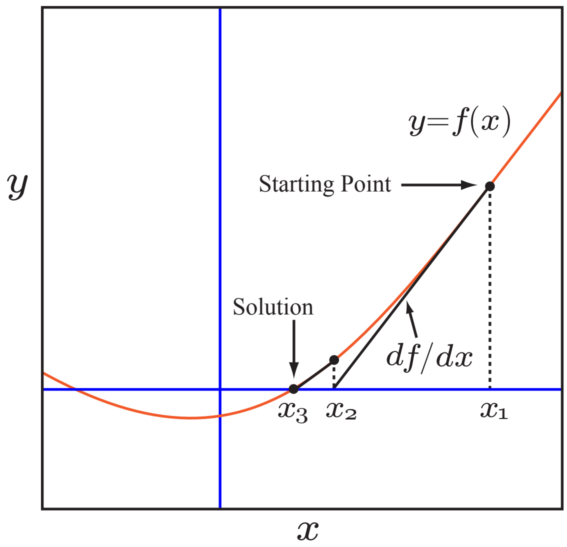


Figure 6.3 The geometry of Newton-Raphson's method

$$x_{k+1} = x_k - \frac{f(x_k)}{\partial f/\partial x_k} \tag{6.4}$$

In this form (6.4) we see the iterative nature of the algorithm.

Before the advent of electronic calculators that had a specific square root button, calculator users would exploit the Newton method to estimate square roots. For example, if the square root of a number, a is equal to x, that is $\sqrt{a} = x$, then it is true that:

$$x^2 - a = 0$$

This equation looks like an equation of the form 6.3. We can therefore apply the Newton formula (equation 6.4) to this equation to obtain

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right) \tag{6.5}$$

Table 6.1 shows a sample calculation using this equation to compute the square root of 25. Note that only a few iterations are required to reach convergence.

Iteration	Estimate
0	15
1	8.33333
2	5.666
3	5.0392
4	5.0001525
5	5.0

Table 6.1 Newton method used to compute the square root of 25, using equation 6.5 with a starting value of 15.

One importance point to bear in mind, the Newton-Raphson method is not guaranteed to converge to the solution, this depends heavily on the start point and the nature of the system being solved. In order to prevent the method from continuing without end in the case when convergence fails if is often useful to halt the method after a maximum of iterations, say one hundred iterations. In a case like this, a new initial start is given and the method tried again. In biochemical models we an always run a time course simulation for a short while and use the end point of that as the starting point for the Newton method. This approach is much more reliable. If the method does converge to a solution there are various ways to decide whether convergence has been achieved. Two such tests include:

1. Difference between Successive Solutions Estimates. We can test for the difference between solution x_i and the next estimate, x_{i+1}, if the absolute difference, $|x_i - x_{i+1}|$ is below some threshold then we

assume convergence has been achieved. Alternatively we can check whether the relative error is less than a certain threshold (say, 1%). The relative error is given by

$$\epsilon = \frac{x_{i+1} - x_i}{x_{i+1}} \times 100\%$$

The procedure can be made to stop at the i-th step if $|f(x_i)| < \epsilon_f$ for a given ϵ_f.

2. Difference between Successive dS_i/dt Estimates. Here we estimate the rates of change as the iteration proceeds and assume convergence has been archived when the difference between two successive rates of change are below some threshold. If we are dealing with a model will more than one state variable then we can construct the sums of squares of the rates of change:

$$\sum \left(\frac{dS_i}{dt} \right)^2$$

The Newton method can be easily extended to systems of equations so that we can write the Newton method in matrix form:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} \right]^{-1} \boldsymbol{f}(\boldsymbol{x}_k) \tag{6.6}$$

If m is the number of state variables or floating species in the model, then $\boldsymbol{x}_k$ is an m dimensional vector of species concentrations, $\boldsymbol{f}(\boldsymbol{x})$ is a vector containing the m rates of change and $\partial \boldsymbol{f}(\boldsymbol{x})/\partial \boldsymbol{x}$ the $m \times m$ Jacobian matrix.

Newton Algorithm

1. Initialize the values of the concentrations, **s**, to some initial guess, obtained perhaps from a short time course simulation.

2. Compute the values for $\boldsymbol{f}(\mathbf{s})$, that is the left-hand side of the differential equations ($\boldsymbol{ds}/\mathbf{dt}$).
3. Calculate the matrix of derivatives, $\partial \boldsymbol{f}/\partial \mathbf{s}$ that is $d(\boldsymbol{ds}/\mathbf{dt})/\boldsymbol{ds}$, at the current estimate for $\mathbf{s}$.
4. Compute the inverse of the matrix $\partial \boldsymbol{f}/\partial \mathbf{s}$
5. Using the information calculated so far, compute the next guess $\mathbf{s}_{k+1}$
6. Compute the sums of squares of the new value of $\boldsymbol{f}(\mathbf{s})$ at $\mathbf{s}_{k+1}$. If the value is less than some error tolerance then assume the solution has been reached, else return to step 3, using $\mathbf{s}_{k+1}$ as the new starting point.

Although the Newton method is seductively simple, it requires the initial guess to be sufficiently close to the solution in order for it to converge. In addition convergence can be slow or not occur at all. A common problem is that the method can overshoot the solution and will then being to rapidly diverge.

A further strategy that is frequently used to compute the steady state is to first use a short time course simulation to bring the initial estimate closer to the steady state. The assumption here is that the steady state is stable. The final point computed in the time course is used to seed a Newton like method, if the Newton method fails to converge then a second time course simulation is carried out. This can be repeated as many times as desired. If there is a suspicion that the steady state is unstable, one can also run a time course simulation backwards in time. In general there is no sure way of computing the steady state automatically and sometimes human intervention is required to supply good initial estimates.

As a result of these issues the unmodified Newton method is rarely used in practice for computing the steady state of biochemical models. One common variant, called the Damped Newton method is sometimes used. Both Gepasi and SCAMP use the Damped Newton method for computing the steady state. This method controls the derivative, df/dx by multiplying the derivative by a factor α, $0 < \alpha < 1$ and can be used to prevent overshoot. There are many variants on the basic Newton method and good simulation software will usually have reasonable methods for estimating the steady state.

In the last ten years more refined Newton like methods have been devised and one that is highly recommended is NLEQ2 (http://www.zib.de/en/numerik/software/ant/nleq2.html. This is used by both Jarnac and PySCeS for computing the steady state. The stiff solver suite sundials (https://computation.llnl.gov/casc/sundials/main.html also incorporates an equation solver, however experience has shown that is it not quite as good as NLEQ2.

Solving the Steady State for a Simple Pathway

We are going to illustrate the use of the Newton-Raphson method to solve the steady state for the following simple pathway. We will assume that all three reactions are governed by simple mass-action reversible rate laws. Species X_o and X_1 are assumed to be fixed and only S_1 and S_2 and floating species or state variables.

$$\begin{tikzcd} X_o \arrow[r, "v_1"] & S_1 \arrow[r, "v_2"] & S_2 \arrow[r, "v_3"] & X_1 \end{tikzcd}$$

The differential equations for the model are as follows:

$$\begin{aligned} \frac{dS_1}{dt} &= (k_1 X_o - k_2 S_1) - (k_3 S_1 - k_4 S_2) \\ \frac{dS_2}{dt} &= (k_3 S_1 - k_4 S_2) - (k_5 S_2 - k_6 X_1) \end{aligned} \tag{6.7}$$

The values for the rate constants and the boundary conditions are given in Table 6.2.

This is a problem with more than one variable (S_1 and S_2) which means we must use the Newton-Raphson matrix form (6.6) to estimate the steady state. To use this we require two vectors, $\boldsymbol{x}_k$ and $\boldsymbol{f}(\boldsymbol{x}_k)$ and one matrix,

Parameter	Value
k_1	3.4
k_2	0.2
k_3	2.3
k_4	0.56
k_5	5.6
k_6	0.12
X_o	10
X_1	0

Table 6.2 Values for example (6.7).

$\partial \boldsymbol{f}(\boldsymbol{x})/\partial \boldsymbol{x}$. The $\boldsymbol{x}_k$ vector is simply:

$$\boldsymbol{x}_k = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix}$$

The $\boldsymbol{f}(\boldsymbol{x}_k)$ vector is given by the values of the differential equations:

$$\boldsymbol{f}(\boldsymbol{x}_k) = \begin{bmatrix} (k_1 X_o - k_2 S_1) - (k_3 S_1 - k_4 S_2) \\ (k_3 S_1 - k_4 S_2) - (k_5 S_2 - k_6 X_1) \end{bmatrix}$$

The $\partial \boldsymbol{f}(\boldsymbol{x})/\partial \boldsymbol{x}$ matrix is the 2 by 2 Jacobian matrix. To compute this we need to form the derivatives which in this case is straight forward given that the differential equations are simple. In cases involving more complex rates laws, software will usually estimate the derivatives by numerical means. In this case however it is easy to differentiate the equations to obtain the following Jacobian matrix:

$$\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \dfrac{d(dS_1/dt)}{dS_1} & \dfrac{d(dS_1/dt)}{dS_2} \\ \dfrac{d(dS_2/dt)}{dS_1} & \dfrac{d(dS_2/dt)}{dS_2} \end{bmatrix} = \begin{bmatrix} -k_2 - k_3 & -k_4 \\ k_3 & -k_4 - k_5 \end{bmatrix}$$

Notice that the elements of the Jacobian contain only rate constants. This is because the model we are using is linear. This also means we need only evaluate the Jacobian and its inverse once. If we used nonlinear rate laws such as the Michaelis-Menten rate law, the Jacobian matrix would also contain terms involving the species concentrations and in this case the Jacobian would need to be reevaluated at each iteration because the value for the species concentration will change at each iteration. For the current problem the Jacobian and its inverse is given by:

$$\text{Jacobian} = \begin{bmatrix} -2.86 & 5.6 \\ -0.56 & -11.2 \end{bmatrix}$$

$$\text{Jacobian}^{-1} = \begin{bmatrix} -0.3876 & -0.1938 \\ -0.01938 & -0.09898 \end{bmatrix}$$

Table 6.3 shows the progress of the iteration as we apply equation 6.6. What is interesting is that convergence only takes one iteration. This is because the model is linear. Nonlinear models may require more iterations. We can also see that after the first iteration the rates of change have very small values, this is usually due to very small numerical errors in the computer arithmetic but anything as small as 10^{-14} can be considered zero.

Iteration	S_1	S_1	dS_1/dt	dS_2/dt
0	1	1	36.74	-10.64
1	13.18	0.6589	2.8×10^{-14}	-1.16×10^{-13}

Table 6.3 Newton-Raphson applied to a Three Step Pathway with Linear Kinetics. Starting values for S_1 and S_2 are both set at one. Convergence occurs within one iteration. Note that the values for the rates of change are extremely small at the end of the first iteration, indicating we have converged.

Computing the Steady State Using Software

The previous section showed how to compute the steady state using the Newton method. In practice we would not write our own solver but use existing software to accomplish the same thing. To illustrate this, the following Jarnac script will define and compute the steady state all at once:

```
// Define model
p = defn cell
    $Xo -> S1;  k1*Xo - k2*S1;
     S1 -> S2;  k3*S1 - k4*S2;
     S2 -> $X1; k4*S2 - k6*X1;
end;

// Initialize value
p.Xo = 10;  p.X1 = 0;
p.k1 = 3.4; p.k2 = 0.2;
p.k2 = 2.3; p.k3 = 0.56;
p.k4 = 5.6; p.k6 = 0.12;

// Initial starting point
p.S1 = 1; p.S2 = 1;

// Compute steady state
p.ss.eval;
println p.S1, p.S2;
```

Running the above script yields steady state concentrations of 13.1783 and 0.658915 for S_1 and S_2 respectively, which is the same if we compare these values to those in Table 6.3. Other tools will have other ways to compute the steady state, for example graphical interfaces will generally have a button marked steady state that when selected will compute the steady state for the currently loaded model.

When using Matlab the function `fsolve` can be use to solve systems of nonlinear equation and in Mathematica one would use `FindRoot`.

6.3 Effect of Different Perturbations

When we talk about model dynamics we mean how species levels and reaction rates change in time as the model evolves. There are a number of ways to elicit a dynamic response in a model, the two we will consider here are perturbations to species and perturbations to model parameters.

If the initial concentration of S_1 is zero then we can run a simulation and allow the system to come to steady state. This is illustrated in Figure 6.4

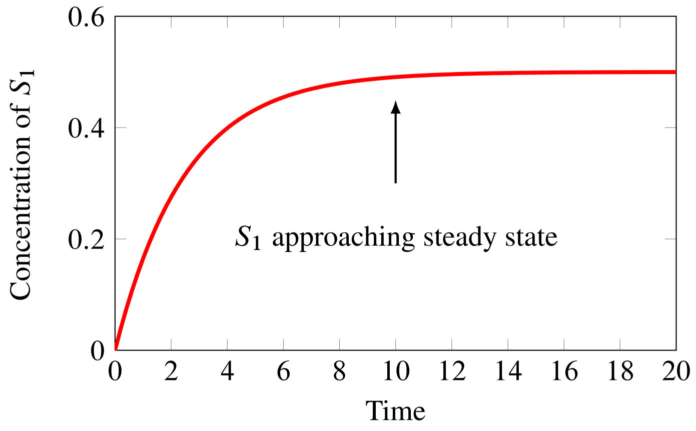


Figure 6.4 S_1 approaching steady state. Jarnac model: 6.4

Effect of Perturbing Floating Species

Once at steady state we can consider applying perturbations to see what happens. For example, Figure 6.7 illustrates the effect of injecting 0.35 units of S_1 at $t = 20$ and watching the system evolve. The Jarnac script used to generate this graph is shown in the chapter Appendix. In practice we would be accomplished this by injecting 0.35 units of S_1 into the volume where the pathway operates. What we observe is that the concentration of S_1 relaxes back to its steady state concentration before the perturbation was made (Figure 6.7). When we apply perturbations to species concentrations and the change relaxes back to the original state, we call the system **stable**.

Figure 6.7 illustrates one kind of perturbation that can be made to a biochemical pathway, in this case perturbing one of the floating molecular species by physically adding a specific amount of the substance to the pathway. In many cases we will find that the system will recover from such perturbations as we see in Figure 6.7. We are not limited to single perturbations, Figure 6.5 shows multiple perturbations, both positive and negative. Not all systems show this behavior, and those that do not are called **unstable**. That is when we perturb a species concentration, instead of the perturbation relaxing back, it begins to diverge.

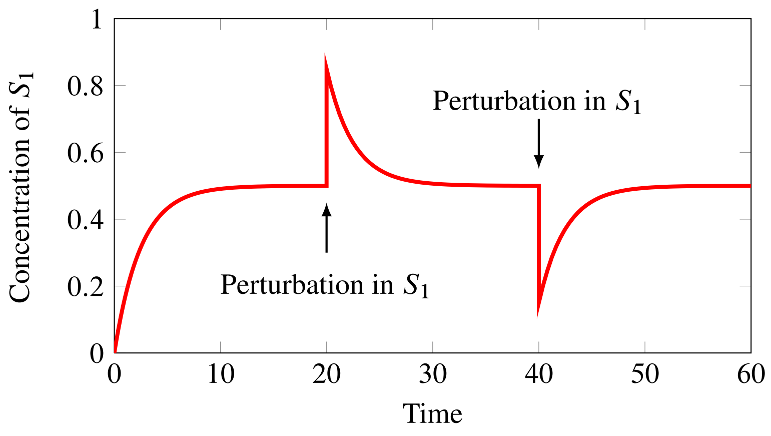


Figure 6.5 Multiple Perturbations. The steady state concentration of the species S_1 is 0.5 and a perturbation is made to S_1 by adding an additional 0.35 units of S_1 at time $=$ 20 and removing 0.35 units at time $=$ 40. In both cases the system relaxes back. Jarnac script: 6.6

Effect of Perturbing Model Parameters

In addition to perturbing floating species we can also perturb the model parameters. Such parameters include kinetic constants and boundary species. Equation 7.2 shows how the concentration of species S_1 depends on all the parameters in the model. Moreover, changing any of the parameters results in a change to the steady state concentration of S_1 and in turn the steady state flux. When changing a parameter we can do it in two ways, we can make a permanent change or we can make a change, then some time later can return the parameter value to its original value. Assuming that the steady state is stable, a temporary change will result in the steady state changing then recovering to the original state when the parameter is changed back. Figure 6.6 shows the effect of perturbing the rate constant, k_1 and then restoring the parameter back to its original value at some time later. Listing 6.7 shows the Jarnac script that was used to generate this plot. In some applications other types of perturbations are made. For example in studying the infusion of a drug where the concentration of the drug is a model parameter, one might use a slow linear increase in the drug concentration. Such a perturbation is called ramp. More sophisticated analyses might require a sinusoidal change in a parameter, an impulse or an exponential change. These inputs are described more fully in Chapter **??**. The main point to remember is that parameter changes will usually result in changes to the steady state concentrations and fluxes.

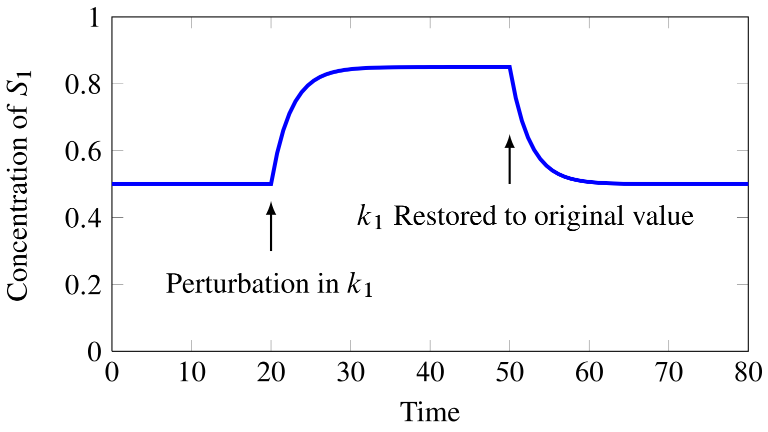


Figure 6.6 Effect of Perturbing Model Parameters using the Jarnac script **??**.

6.4 Stability and Robustness

Biological organisms are continually subjected to perturbations. These perturbations can originate from external influences such as changes in temperature, light or the availability of nutrients. Perturbations can also arise internally due to the stochastic nature of molecular events or by genetic variation. One of the most remarkable and characteristic properties of living systems is their ability to resist such perturbations and maintain very steady internal conditions. For example the human body can maintain a constant core temperature of 36.8°C ±0.7 even though external temperatures may vary widely. The ability of a biological system to maintain a steady internal environment is called **homeostasis**, a phrase introduced by Claude Bernard almost 150 years ago. Modern authors may also refer to this behavior as **robustness**.

6.5 Introduction to Stability

A closely related concept to robustness is stability. We can define the stability of a pathway in the following way:

A biochemical pathway is dynamically stable at steady state if small perturbations in the floating species concentrations relax back to the original state.

We can illustrate a stable system using a simple two step model.

Let us assume that the two step pathway has the following form:

$$\begin{tikzcd} X_o \arrow[r, "v_1 = k_1 Xo"] & S_1 \arrow[r, "v_2 = k_2 S_1"] & X_1 \end{tikzcd}$$

Figure 6.7 illustrates the results from a simulation of a simple two step biochemical pathway with one floating species, S_1. The Jarnac script used to generate this graph is given in Table 6.8. The initial concentrations of the model are set so that it is at steady state, that is no transients are seen between $t = 0$ and $t = 20$. A $t = 20$ a perturbation is made to the concentration of S_1 by adding 0.25 units of S_1. This could be accomplished by injecting 0.25 units of S_1 into the volume where the pathway resides. The system is now allowed to evolve further. If the system is stable, the perturbation will relax back to the original steady state, as it does in the simulation shown in Figure 6.7. This system therefore appears to be stable.

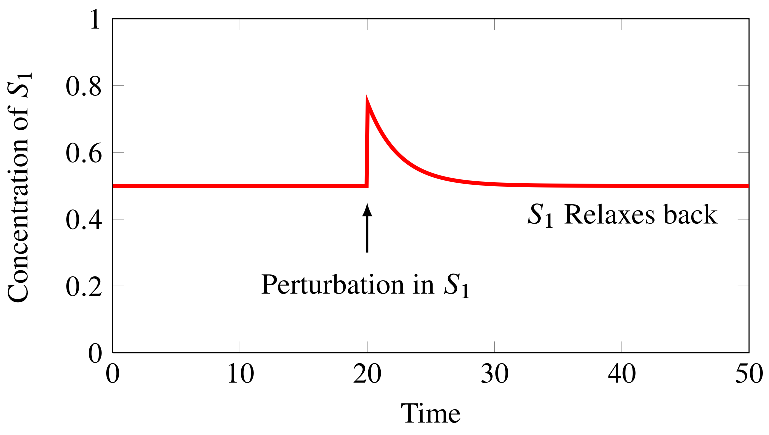


Figure 6.7 Stability of a simple biochemical pathway at steady state. The steady state concentration of the species S_1 is 0.5. A perturbation is made to S_1 by adding an additional 0.25 units of S_1 at time = 20. The system is considered stable because the perturbation relaxes back to the original steady state. See Table 6.8 for Jarnac listing.

The differential equation for the single floating species, S_1, is given by

$$\frac{dS_1}{dt} = k_1 Xo - k_2 S_1 \tag{6.8}$$

with a steady state solution $S_1 = k_1 Xo/k_2$. The question we wish to ask here is whether the steady state is stable or not? We can show that the two step model is stable be using the following mathematical argument. If the system is at steady state, let us make a small perturbation to the steady state concentration of S_1, δS_1 and ask what is the rate of change of $S_1 + \delta S_1$ as a result of this perturbation, that is what is $d(S_1 + \delta S_1)/dt$? The new rate of change equation is rewritten as follows:

$$\frac{d(S_1 + \delta S_1)}{dt} = k_1 X_o - k_2(S_1 + \delta S_1)$$

If we insert the solution for S_1 (equation 7.2) into the above equation we are left with:

$$\frac{d\delta S_1}{dt} = -k_2 \delta S_1 \tag{6.9}$$

In other words the rate of change of the disturbance, δS_1 is negative, that is, the system attempts to reduce the disturbance so that the system returns back to the original steady state. Systems with this kind of behavior are called **stable**. If the rate of change in S_1 had been positive instead of negative however, the perturbation would have continued to diverge away from the original steady state and the system would them be considered **unstable**.

Dividing both sides of equation 7.3 by δS_1 and taking the limit $\delta S \to 0$, we find that $\partial(dS_1/dt)/\partial S_1$ is equal to $-k_2$. The stability of this simple system can therefore be determined by inspecting the sign of $\partial(dS_1/dt)/\partial S_1$ which can be easily determined by taking the derivative of the differential equations with respect to the species concentrations. For larger systems the stability of a system can be determined by looking at all the terms $\partial(dS_i/dt)/\partial S_i$ which are given collectively by the expression:

$$\frac{d(d\mathbf{s}/dt)}{d\mathbf{s}} = \boldsymbol{J} \tag{6.10}$$

where $\boldsymbol{J}$ is called the Jacobian matrix containing elements of the form $\partial(dS_i/dt)/\partial S_i$. Equation 7.3 can also be written as

$$\frac{d(\delta\mathbf{s})}{dt} = \boldsymbol{J}\delta\mathbf{s} \tag{6.11}$$

This is a set of linear differential equations that describes the rate of change in the perturbation δS. $\boldsymbol{J}$ is given by

$$\boldsymbol{J} = \begin{bmatrix} \dfrac{\partial S_1/dt}{\partial S_1} & \cdots & \dfrac{\partial S_1/dt}{\partial S_m} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial S_m/dt}{\partial S_1} & \cdots & \dfrac{\partial S_m/dt}{\partial S_m} \end{bmatrix} \tag{6.12}$$

Equation 7.5 is an example of an homogeneous linear differential equation and has the general form:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{A}\boldsymbol{x}$$

As we will see in a later chapter, solutions to such equations are well known and take the form:

$$x_j(t) = \sum_{k=1}^{n} \beta_{jk} e^{\lambda_k t}$$

That solution involves the sum of exponentials, $e^{\lambda_k t}$. The exponents of the exponentials are given by the eigenvalues of the matrix, $\boldsymbol{A}$, namely the Jacobian matrix 6.12. If the eigenvalues are negative then the exponents decay (stable) whereas if they are positive the exponents grow (unstable). We can therefore determine the stability properties of a given model by computing the eigenvalues of the Jacobian matrix and looking for any positive eigenvalues. Note that the elements of the Jacobian matrix will often be a function of the species levels, it is therefore important that the Jacobian be evaluated at the steady state of interest.

There are many software packages that will compute the eigenvalues of a matrix and there are a small number packages that can compute the Jacobian directly from the biochemical model. For example, the script below is taken from Jarnac, it defines a simple model, initializes the model values, computes the steady state and then prints out the eigenvalues of the Jacobian matrix. For a simple one variable model, the Jacobian matrix only has a single entry and the eigenvalue corresponds to that entry. The output from running the script is given below showing that the eigenvalue is -0.3. Sine we have a negative eigenvalue, the pathway must be stable to perturbations in S_1.

```
p = defn model
     $Xo -> S1;  k1*Xo;
      S1 -> $X1; k2*S1;
end;

// Set up the model initial conditions
```

```
p.Xo = 1;   p.X1 = 0;
p.k1 = 0.2; p.k2 = 0.3;

// Evaluation the steady state
p.ss.eval;
// print the eigenvalues of the Jacobian matrix
println eigenvalues (p.Jac);

// Output follows:
{    -0.3}
```

6.6 Sensitivity Analysis

Sensitivity analysis at steady state looks at how particular model variables are influenced by model parameters. There are at least two main reasons why it is interesting to examine sensitivities. The first is a practical one. Many kinetic parameters we use in building biochemical models can have a significant degree of uncertainty about them. By determining how much each parameter has an influence on the model's state we can decide whether we should improve our confidence in the particular parameter. A parameter that has considerable influence but at the same time has significant uncertainty is a parameter that should be determined more carefully by additional experimentation. On the other hand a parameter that has little influence but has significant uncertainly associated with it, is relatively unimportant. A sensitivity analysis can therefore be used to highlight parameters that need better precision.

The second reason for measuring sensitivities is to provide insight. The degree to which a parameter can influence a variable tells us something about about the network is responding to perturbations and it responds to the degree it does. Such a study can be used to answer questions about robustness and adaptation. We will delay further discussion of this important topic to part 2 of the book.

How are sensitivities represented? Traditionally there are two way, one based on absolute sensitivities and the second based on relative sensitivities. Absolute sensitivities are simply given by the ratio of the absolute change in the variable to the absolute change in the parameter. That is:

$$S = \frac{\Delta V}{\Delta p}$$

where V is the variable and p the parameter. This equation shows finite changes to the parameter and variable. Unfortunately because most systems are nonlinear, the value for the sensitivity will be a function of the size of the finite change. To make the sensitivity independent of the size of the change, the sensitivity is usually defined in terms of infinitesimal changes:

$$S = \frac{dV}{dp}$$

Although absolute sensitivities are simple they have one drawback namely that the value can be influenced by the units used to measure the variable and parameter. Often in making experimental measurements we won't be able to measure the quantity using the most natural units, instead we may have measurements in terms of fluorescence, colony counts, staining on a gel and so on. Is is most likely that the variable and parameter units will be quite different and each laboratory may have its own way particular units is uses. Absolute sensitivities are therefore quite difficult to compare.

To get round the problem of units, many people will use relative sensitivities, These are simple scaled absolute sensitivities:

$$S = \frac{dV}{dp}\frac{p}{V}$$

The sensitivity is defined in terms of infinitesimal changes for the same reason cited before. The reader may also recall that elasticities are measured in this way. Relative sensitivities are immune from the units we use to measure quantities but also relative sensitivities correspond more closely to how many measurements are made, often in terms of relative or fold changes. In practice steady state relative sensitivities should be measured by taking a measurement at the operating steady state, making a perturbation (preferable a small one), waiting for the system to reach a new steady state then measuring the system again. It is important to be aware that the steady state sensitivities measure how a perturbation in a parameter moves the system from one steady state to another.

6.7 Stability

Figure 6.7 shows a simulation where a species concentration is disturbed and over time relaxes back to the original steady state. This is an example of a stable steady state.

The differential equation for the single floating species, S_1, is given by

$$\frac{dS_1}{dt} = k_1 Xo - k_2 S_1 \tag{6.13}$$

and as we saw before, with a steady state solution

$$S_1 = k_1 Xo/k_2 \tag{6.14}$$

The question we wish to ask here is whether the steady state is stable or not, that is whether perturbation to species recover or not? We can show that the two step model is stable by using the following mathematical argument. The differential equation describing the two step model is given by,

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1$$

If the system is at steady state, let us make a small perturbation to the steady state concentration of S_1, δS_1 and ask how δS_1 changes as a result of this perturbation, that is what is $d(\delta S_1)/dt$? The new rate of change equation is rewritten as follows:

$$\frac{d(S_1 + \delta S_1)}{dt} = k_1 X_o - k_2 (S_1 + \delta S_1)$$

If we insert the steady state solution for S_1 (equation 7.2) into the above equation we are left with:

$$\frac{d\delta S_1}{dt} = -k_2 \delta S_1 \tag{6.15}$$

In other words the rate of change of the **disturbance itself**, δS_1 is negative, that is, the system attempts to reduce the disturbance so that the system returns back to the original steady state. Systems with this kind of behavior are called **stable**. If the rate of change in S_1 had been positive instead of negative however, the perturbation would have continued to diverge away from the original steady state and the system would them be considered **unstable**.

A biochemical pathway is dynamically stable at steady state if small perturbations in the floating species concentrations relax back to the original state.

To continue, let us divide both sides of equation 7.3 by δS_1 and taking the limit, we find that $\partial(dS_1/dt)/\partial S_1$ is equal to $-k_2$. The stability of this simple system can therefore be determined by inspecting the sign of $\partial(dS_1/dt)/\partial S_1$ which can be easily determined by taking the derivatives of the differential equations with respect to the species concentrations.

For larger systems the stability of a system can be determined by looking at all the terms $\partial(dS_i/dt)/\partial S_i$ which are given collectively by the expression:

$$\frac{d(d\mathbf{s}/dt)}{d\mathbf{s}} = \boldsymbol{J} \tag{6.16}$$

where $\boldsymbol{J}$ is called the **Jacobian matrix** containing elements of the form $\partial(dS_i/dt)/\partial S_i$. Equation 7.3 can be generalized to:

$$\frac{d(\delta\mathbf{s})}{dt} = \boldsymbol{J}\delta\mathbf{s} \tag{6.17}$$

where $\boldsymbol{J}$ is given by

$$\begin{bmatrix} \dfrac{\partial S_1/dt}{\partial S_1} & \cdots & \dfrac{\partial S_1/dt}{\partial S_m} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial S_m/dt}{\partial S_1} & \cdots & \dfrac{\partial S_m/dt}{\partial S_m} \end{bmatrix}$$

Equation 7.5 is an example of an unforced linear differential equation and has the general form:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{A}\boldsymbol{x}$$

Solutions to such equations are well known and take the form:

$$x_j(t) = c_1 \boldsymbol{K}_1 e^{\lambda_1 t} + c_2 \boldsymbol{K}_2 e^{\lambda_2 t} + \cdots c_n \boldsymbol{K}_n e^{\lambda_n t}$$

That is the solution to an unforced linear differential equations involves a sum of exponentials, $e^{\lambda_i t}$, constants c_i and vectors, $\boldsymbol{K}_i$. The exponents of the exponentials are given by the eigenvalues (See Appendix C) of the matrix, $\boldsymbol{A}$ and $\boldsymbol{K}_i$ the corresponding eigenvectors. The c_i terms are related to the initial conditions assigned to the problem. It is possible for the eigenvalues to be complex but in general if the real parts of the eigenvalues are negative then the exponents decay (stable) whereas if they are positive the exponents grow (unstable). We can therefore determine the stability properties of a given model by computing the eigenvalues of the Jacobian matrix and looking for any positive eigenvalues. Note that the elements of the Jacobian matrix will often be a function of the species levels, it is therefore important that the Jacobian be evaluated at the steady state of interest.

Example 6.1

The following system:

$$S_1 \to S_2 \to$$

if governed by the following set of differential equations:

$$\frac{dS_1}{dt} = -2S_1$$

$$\frac{dS_2}{dt} = 2S_1 - 4S_2$$

The solution to this system can be derived using Mathematica or by using standard algebraic method for solving linear homogeneous systems. The solution can be found to be:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-2t} + c_2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} e^{-4t}$$

$$S_1 = c_1 e^{-2t}$$

$$S_2 = c_1 e^{-2t} + c_2 e^{-4t}$$

Since the exponents are all negative (-2, -2 and -4), the system is stable to perturbations in S_1 and S_2.

There are many software packages that will compute the eigenvalues of a matrix and there are a small number packages that can compute the Jacobian directly from the biochemical model. For example, the script below is taken from Jarnac, it defines the simple model, initializes the model values, computes the steady state and then prints out the eigenvalues of the Jacobian matrix. For a simple one variable model, the Jacobian matrix only has a single entry and the eigenvalue corresponds to that entry. The output from running the script is given below showing that the eigenvalue is -0.3. Since we have a negative eigenvalue, the pathway must be stable to perturbations in S_1.

```
p = defn model
     $Xo -> S1;  k1*Xo;
      S1 -> $X1; k2*S1;
end;

// Set up the model initial conditions
p.Xo = 1;   p.X1 = 0;
p.k1 = 0.2; p.k2 = 0.3;

// Evaluation the steady state
p.ss.eval;
// print the eigenvalues of the Jacobian matrix
println eigenvalues (p.Jac);

// Output follows:
{    -0.3}
```

Example 6.2

The following system:

$$\rightarrow S_1 \rightarrow S_2 \rightarrow$$

if governed by the following set of differential equations:

$$\frac{dS_1}{dt} = 3 - 2S_1$$

$$\frac{dS_2}{dt} = 2S_1 - 4S_2$$

The Jacobian matrix is computed by differentiating the equations with respect to the steady state values of S_1 and S_2:

$$\boldsymbol{J} = \begin{bmatrix} -2 & 0 \\ 2 & -4 \end{bmatrix}$$

The eigenvalues for this matrix are: -2 and -4 respectively. Since both eigenvalues are negative the system is stable to small perturbations in S_1 and S_2.

The pattern of eigenvalues can tell us a lot about stability but also about the form of the transients that will occur when we perturb the state of the system. In the next section we will investigate this aspect.

6.8 Phase Portraits

The word **phase space** refers to a space where all possible states are presented. For example, in a biochemical pathway with two species, x_1 and x_2, the phase space consists of all possible trajectories of x_1 and x_2 in time. For two dimensional systems the phase space can be very conveniently display on an x/y graph where each axis represents one of the state variables. A visual representation of the phase space is often called a **phase portrait** or **phase plane**. To illustrate phase portrait consider the following simple reaction network:

$$\xrightarrow{v_o} x_1 \xrightarrow{k_1\, x_1} x_2 \xrightarrow{k_2\, x_2}$$

with two linear differential equations:

$$\frac{dx_1}{dt} = v_o - k_1 x_1$$

$$\frac{dx_2}{dt} = k_1 x_1 - k_2 x_2$$

We can assign particular values to the parameters, set up some initial conditions and plot in phase space the evolution of x_1 and x_2. If we replot the solution using many different initial conditions we get something that looks like the plots shown in Figures 7.2 to 7.7.

These plots illustrate a variety of transient behaviors around a steady state. These particular transient behaviors apply specifically to linear differential equations. If we have a nonlinear system and we linearize the system around the steady state, the linearized system will behave in a way suggested by these plots.

A two dimensional linear system of differential equations has solutions of the form:

$$
\begin{aligned}
x_1 &= c_1 k_1 e^{\lambda_1 t} + c_2 k_2 e^{\lambda_2 t} \\
x_2 &= c_3 k_3 e^{\lambda_3 t} + c_4 k_4 e^{\lambda_4 t}
\end{aligned}
$$

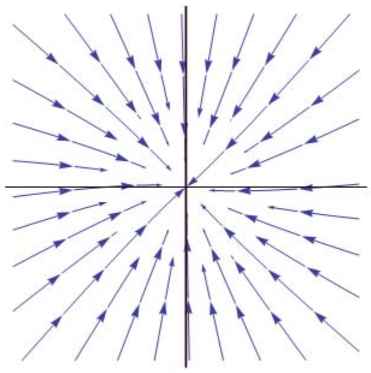


Figure 6.8 Phase portrait for the two species reaction network. Stable node. **Negative Eigenvalues**. Matrix $\boldsymbol{A}$: $a_{11} = -2, a_{12} = 0, a_{21} = -0.15, a_{22} = -2$. Corresponding eigenvalues: $\lambda_1 = -2, \lambda_2 = -2$

The c_i and k_i terms are constants related to the initial conditions and eigenvectors respectively but the λ_i terms, or eigenvalues, determine the qualitative pattern that a given behavior might have. It should be noted that the eigenvalues can be complex or real numbers. In applied mathematics, e raised to a complex number immediately suggests some kind of periodic behavior. Let us consider the different possibilities for the values of the eigenvalues.

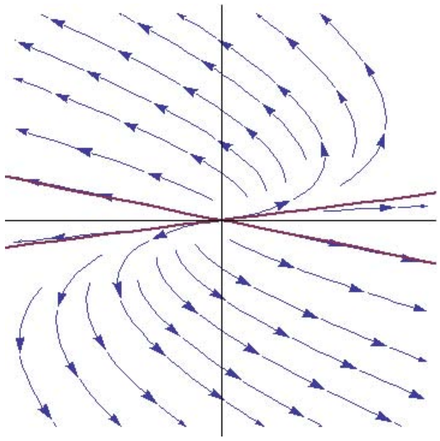


Figure 6.9 Phase portrait for the two species reaction network. Unstable node, also called an improper node. **Positive Eigenvalues**. Matrix $\boldsymbol{A}$: $a_{11} = 1.2, a_{12} = -2, a_{21} = -0.05, a_{22} = 1.35$. Corresponding eigenvalues: $\lambda_1 = 1.6, \lambda_2 = 0.95$

Both Eigenvalues have the same sign, different magnitude but are real. If both eigenvalues are negative the equations describe a system known as a **stable node**. All trajectories move towards the steady state point. If the eigenvalues have the same magnitude and the c_i terms have the same magnitude then the trajectories move to the steady state in a symmetric manner, Figure 7.2. If the k_i values differ then the trajectories will tend to twist, Figure 7.3.

Figure 7.3 shows the case when the two eigenvalues are the same sign (positive in this case) but of different magnitude, When the eigenvalues are positive the trajectories move out from the steady state, an **unstable node**. Such systems are therefore called unstable.

Real Eigenvalues but of opposite sign. If the two eigenvalues are real but of opposite sign we get behavior called a saddle-node. This is where in one direction the trajectories move towards the steady state and in another direction move away. Since trajectories can only move towards the steady

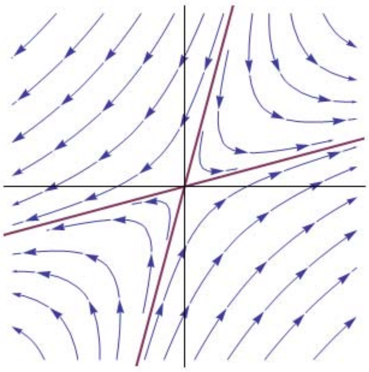


Figure 6.10 Phase portrait for the two species reaction network. Saddle node. **One Positive and One Negative Eigenvalue**. Matrix A: $a_{11} = 2, a_{12} = -1, a_{21} = 1, a_{22} = -2$. Corresponding eigenvalues: $\lambda_1 = -1.73, \lambda_2 = 1.73$

state if they are exactly on the saddle node ridge, but once they reach the node they will diverge. Saddle nodes are therefore unstable.

Complex Eigenvalues Sometimes the eigenvalues can be complex, that is of the form $a + ib$ where i is the imaginary number. It may seem strange that the solution to the differential equations can admit complex eigenvalues. To understand what these mean we have to recall Euler's formula:

$$e^{i\theta} = cos(\theta) + i\ \sin(\theta)$$

Description	Eigenvalues	Behavior
Both Positive	$r_1 > r_2 > 0$	Unstable
Both Negative	$r_1 < r_2 < 0$	Stable
Positive and Negative	$r_1 < 0 < r_1$	Saddle point
Complex Conjugate	$r_1 > r_1 > 0$	Unstable spiral
Complex Conjugate	$r_1 < r_1 < 0$	Stable spiral
Pure Imaginary	$r_1 = r_1 = 0$	Center

Table 6.4 Summary of Node Behaviors

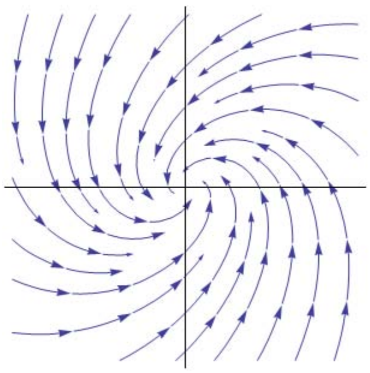


Figure 6.11 Phase portrait for the two species reaction network. Stable spiral node. **Negative Complex Eigenvalues**. Matrix $\boldsymbol{A}$: $a_{11} = -0.5, a_{12} = -1, a_{21} = 1, a_{22} = -1$. Corresponding eigenvalues: $\lambda_1 = -0.75 + 0.97i, \lambda_2 = -0.75 - 0.97i$

or extended to:

$$e^{(a+bi)t} = e^{at}\cos(bt) + i e^{bt}\sin(bt)$$

When the solutions are expressed in terms of sums of sine and cosine terms, the imaginary parts will cancel out, leaving just trigonometric terms with only real parts.

We can show this as follows. Consider the system:

$$x(t) = c_1 z_1 e^{(\lambda + i\mu)t} + c_2 z_2 e^{(\lambda - i\mu)t}$$

where z_1 and z_2 are corresponding conjugate eigenvectors. Using Euler's formula, $e^{i\mu} = \cos(\mu) + i\sin(\mu)$ and that $e^{(\lambda+i\mu)t} = e^{\lambda t}e^{i\mu t}$ we obtain:

$$\begin{aligned} x(t) = c_1 z_1 e^{\lambda t}(\cos(\mu t) + i\sin(\mu t)) \\ + c_2 z_2 e^{\lambda t}(\cos(\mu t) - i\sin(\mu t)) \end{aligned}$$

Writing the conjugate eigenvectors as $z_1 = a + bi$ and $z_2 = a - bi$, we

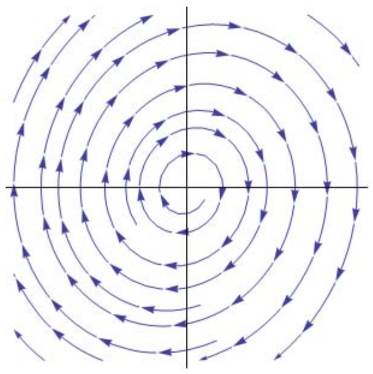


Figure 6.12 Phase portrait for the two species reaction network. Unstable spiral node. **Positive Complex Eigenvalues**. Matrix $\boldsymbol{A}$: $a_{11} = 0, a_{12} = 1.0, a_{21} = -1.2, a_{22} = 0.2$. Corresponding eigenvalues: $\lambda_1 = 0.1 + 1.09i, \lambda_2 = 0.1 - 1.09i$

get:

$$\begin{aligned} x(t) = c_1(a + bi)e^{\lambda t}(\cos(\mu t) + i\sin(\mu t)) \\ +(a - bi)e^{\lambda t}(\cos(\mu t) - i\sin(\mu t)) \end{aligned}$$

Multiply out and separate the real and imaginary parts yields:

$$\begin{aligned} x(t) = e^{\lambda t}\,[c_1(a\cos(\mu t) - b\sin(\mu t) + i(a\sin(\mu t) + b\cos(\mu t))) \\ + c_2(a\cos(\mu t) - b\sin(\mu t) - i(a\sin(\mu t) + b\cos(\mu t)))] \end{aligned}$$

The complex terms cancel leaving only the real parts. If we set $c_1 + c_2 = k_1$ and $(c_1 - c_2)i = k_2$ then:

$$\begin{aligned} x(t) = e^{\lambda t}\,[k_1(a\cos(\mu t) - b\sin(\mu t)) \\ k_2(a\sin(\mu t) + b\cos(\mu t))] \end{aligned}$$

The solution is real when the constants c_1 and c_2 are real. This will only be the case when the eigenvalues are a conjugate pair, $(a \pm ib)$ which is the case we are looking at. Therefore systems that admit a complex pair of conjgate eigenvalues result in periodic real solutions.

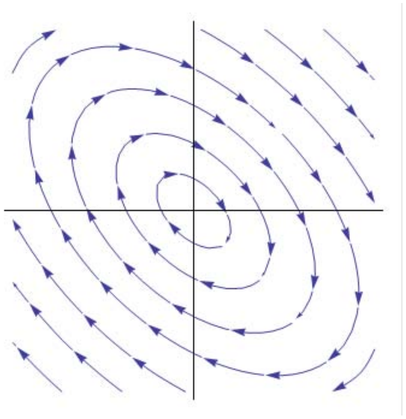


Figure 6.13 Phase portrait for the two species reaction network. Center node. **Complex Eigenvalues, Zero Real Part**. Matrix $\boldsymbol{A}$: $a_{11} = 1, a_{12} = 2.0, a_{21} = -2, a_{22} = -1$. Corresponding eigenvalues: $\lambda_1 = 0 + 1.76i, \lambda_2 = 0 - 1.76i$

This result shows that the appearance of complex numbers in the eigenvalues results in periodic solutions. For this reason solution with complex eigenvalues tend to display trajectories such as those shown in Figure **??** and 7.7. If the real parts are positive then the spiral trajectories move outwards (unstable). If the real parts of the eigenvalues are negative then the spiral trajectory moves into the steady state (stable).

Conjugate Pair

A complex conjugate pair is a complex number of the form: $a \pm bi$. The eigenvalues for a two variable linear system system with matrix $\boldsymbol{A}$, can be computed directly using the relation:

$$\lambda = \frac{\operatorname{tr}(\boldsymbol{A}) \pm \sqrt{\operatorname{tr}^2(\boldsymbol{A}) - 4 \det(\boldsymbol{A})}}{2}$$

where $\operatorname{tr}(\boldsymbol{A}) = a + d$ and $\det(\boldsymbol{A}) = ad - bc$. If the term in the square root is negative, the eigenvalues will always come out as a conjugate pair owning to the $\pm$ term. If $\operatorname{tr}^2(\boldsymbol{A}) - 4\det(\boldsymbol{A}) < 0$ then the solution will be the conjugate pair:

$$\lambda = \frac{\operatorname{tr}(\boldsymbol{A})}{2} \pm \frac{\sqrt{\operatorname{tr}^2(\boldsymbol{A}) - 4\det(\boldsymbol{A})}}{2}$$

Therefore a complex eigenvalue will always be accompanied by its conjugate partner.

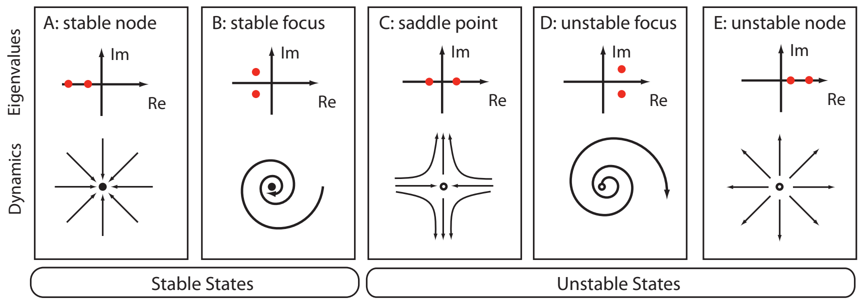


Figure 6.14 Summary of behaviors including dynamics and associated eigenvalues for a two dimensional linear system. Adapted from "Computational Models of Metabolism: Stability and Regulation in Metabolic Networks", Adv in Chem Phys, Vol 142, Steuer and Junker.

6.9 Bifurcation Plots

In its simplest form, a bifurcation plot is just a plot of the steady state value of a system variable, such as a concentration versus a parameter of the system. For example we saw that the steady state solution for the simple system:

$$\frac{dS_1}{dt} = k_1 Xo - k_2 S_1$$

was given by:

$$S_1 = k_1 Xo / k_2$$

We can now plot the steady state value of S_1 as a function of k_2, as shown in Figure 7.9.

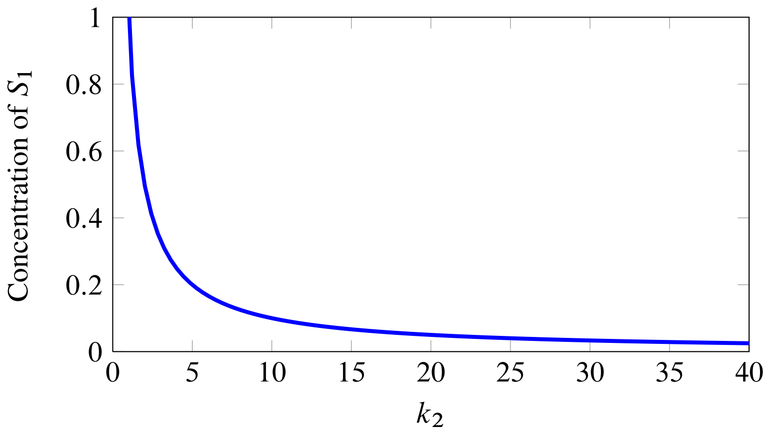


Figure 6.15 Steady state concentration of S_1 as a function of k_2 for the system, $dS_1/dt = k_1 X_o - k_2 S_1$

Of more interest is that bifurcation plots can be used to identify changes in qualitative behavior, particularly systems that have multiple steady states. Consider the system shown 7.10. This shows a simple gene circuit with a positive feedback loop. That is as the transcription factor x accumulates it binds to an operator site on the gene which increases its synthesis.

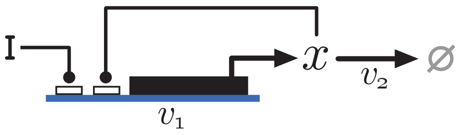


Figure 6.16 System with Positive Feedback

From the circuit diagram we can construct a simple model. This model uses the following kinetic laws for the synthesis and degradation steps.

$$v_1 = k_1 \frac{x^4}{k_2 + x^4}$$
$$v_2 = k_3 x$$

We can plot both rate laws as a function of transcription factor x to obtain the figure shown in Figure 7.11. If we vary the slope of v_2, buy changing k_3, the intersection points will change. We can plot the intersection points as a function of k_3. If we do this we obtain the diagram shown in Figure 7.12.

Figure 7.12 shows that at some value of the parameter k_3, the system has three possible steady states, outside this range only single steady state persists. Bifurcation diagrams are extremely useful for uncovering and displaying such information. Drawing bifurcation diagrams is not easy however. There are some software tools that can help. Figure 7.12 for example was generated using the SBW Auto C# tool, available at http://jdesigner.sourceforge.net/Site/Auto_C.html. Another useful tool for drawing bifurcation diagrams is Oscill8, available from http://oscill8.sourceforge.net/. Both tools can read SBML. Figure 7.12 was generated first by entering the model into Jarnac (Shown in listing 7.1) to generate the SBML. The model was then passed to Auto C# to produce the bifurcation diagram.

Bistability will be discussed in more detail in a separate volume.

```
p = defn cell
```

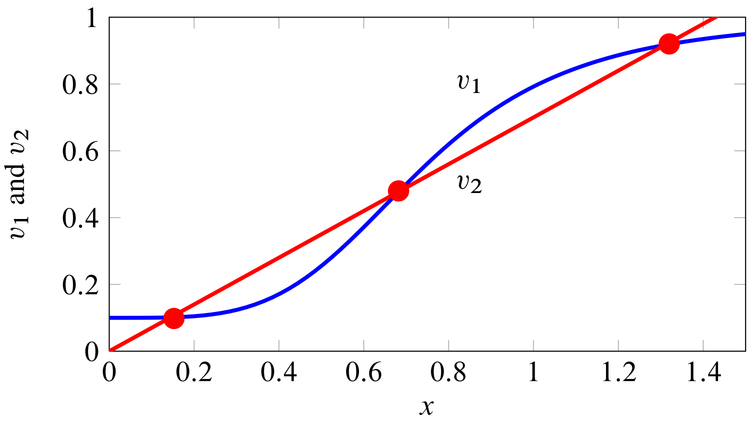


Figure 6.17 Reaction velocities, v_1 and v_2 as a function of x for the system, Figure 7.9. The intersection points marked by fill circles indicate possible steady states.

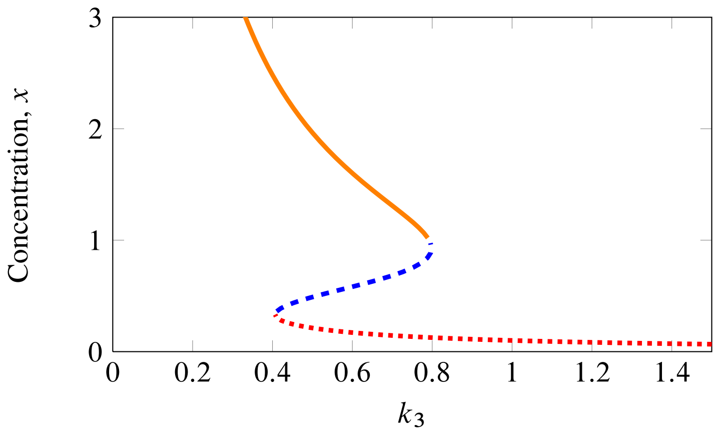


Figure 6.18 Plotting intersection points from Figure 7.11 as a function of k_3. Dotted line marks the lower intersection point, dashed line the middle intersection points, and solid line the upper intersection point.

```
      $Xo -> x; 0.1 + k1*x^4/(k11+x^4);
        x -> $w; k2*x;
end;

// Initialization here
p.k1 = 0.9; p.k11 = 0.3;
p.k2 = 0.7;
// Compute steady state
p.ss.eval;
```

Listing 6.1 Model used to create Figure 7.12

Further Reading

1. Sauro HM (2011) Enzyme Kinetics for Systems Biology. ISBN: 978-0982477311

2. Kipp E, Herwig R, Kowald A, Wierling C and Lehrach H (2005) Systems Biology in Practice, Wiley-VCH Verlag

3. Jarnac web site `http://sbw-app.org/jarnac/`

Exercises

1. Explain what is meant by a stable and unstable steady state.

2. Derive equation 6.5.

3. The steady state of a given pathway is stable. Explain the effect in general terms on the steady state if:

   a) A bolus of floating species is injected into the pathway

   b) A permanent change to a kinetic constant.

Appendix

See http://sbw-app.org/jarnac/ for more details of Jarnac.

```
p = defn cell
      A -> B;  k1*A;
      B -> A;  k2*B;
end;

p.A = 10; p.k1 = 1;
p.B = 0;  p.k2 = 0.5;

m = p.sim.eval (0, 3, 100);
graph (m);
```

Listing 6.2 Script for Figure 4.1

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

p.vo = 1;
p.k1 = 2; p. k2 = 0;
p.k3 = 3;

m = p.sim.eval (0, 6, 100);
graph (m);
```

Listing 6.3 Script for Figure 4.2

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

p.k1 = 0.2;
```

```
p.k2 = 0.4;
p.Xo = 1;
p.S1 = 0.0;

m = p.sim.eval (0, 20, 100, [<p.time>, <p.S1>]);
graph (m);
```

Listing 6.4 Script for Figure 6.4

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

p.k1 = 0.2;
p.k2 = 0.4;
p.Xo = 1;
p.S1 = 0.5;

// Simulate the first part up to 20 time units
m1 = p.sim.eval (0, 20, 100, [<p.time>, <p.S1>]);

// Perturb the concentration of S1 by 0.35 units
p.S1 = p.S1 + 0.35;

// Continue simulating from last end point
m2 = p.sim.eval (20, 50, 100, [<p.time>, <p.S1>]);

// Merge and plot the two halves of the simulation
graph (augr(m1, m2));
```

Listing 6.5 Script for Figure 6.7

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

p.k1 = 0.2;
```

```
p.k2 = 0.4;
p.Xo = 1;
p.S1 = 0.0;

// Simulate the first part up to 20 time units
m1 = p.sim.eval (0, 20, 100, [<p.time>, <p.S1>]);

// Perturb the concentration of S1 by 0.35 units
p.S1 = p.S1 + 0.35;

// Continue simulating from last end point
m2 = p.sim.eval (20, 40, 50, [<p.time>, <p.S1>]);
// Merge the data sets
m3 = augr(m1, m2);
// Do a negative perturbation in S1
p.S1 = p.S1 - 0.35;

// Continue simulating from last end point
m4 = p.sim.eval (40, 60, 50, [<p.time>, <p.S1>]);

// Merge and plot the final two halves of the simulation
graph (augr(m3, m4));
```

Listing 6.6 Script for Figure 6.5

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

p.k1 = 0.2;
p.k2 = 0.4;
p.Xo = 1;
p.S1 = 0.5;

// Simulate the first part up to 20 time units

m1 = p.sim.eval (0, 20, 5, [<p.time>, <p.S1>]);

// Perturb the parameter k1
```

```
p.k1 = p.k1*1.7;

// Simulate from the last point
m2 = p.sim.eval (20, 50, 40, [<p.time>, <p.S1>]);

// Restore the parameter back to ordinal value
p.k1 = 0.2;

// Carry out final run of the simulation
m3 = p.sim.eval (50, 80, 40, [<p.time>, <p.S1>]);

// Merge all data sets and plot
m4 = augr(augr(m1, m2), m3);
graph (m4);
```

Listing 6.7 Script for Figure 6.6

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

p.k1 = 0.2;
p.k2 = 0.4;
p.Xo = 1;
p.S1 = 0.5;

// Simulate the first part up to 20 time units
m1 = p.sim.eval (0, 20, 100, [<p.time>, <p.S1>]);

// Perturb the concentration of S1 by 0.35 units
p.S1 = p.S1 + 0.35;

// Continue simulating from last end point
m2 = p.sim.eval (20, 50, 100, [<p.time>, <p.S1>]);

// Merge and plot the two halves of the simulation
graph (augr(m1, m2));
```

Listing 6.8 Jarnac script used to generate Figure 6.7.

7 Stability

7.1 Stability

The last chapter briefly introduced the concept of stability of biochemical networks. IN this chapter we will delve more deeply into this topic/

Figure 6.7 shows a simulation where a species concentration is disturbed and over time relaxes back to the original steady state. This is an example of a stable steady state.

The differential equation for the single floating species, S_1, is given by

$$\frac{dS_1}{dt} = k_1 Xo - k_2 S_1 \tag{7.1}$$

and as we saw before, with a steady state solution

$$S_1 = k_1 Xo / k_2 \tag{7.2}$$

The question we wish to ask here is whether the steady state is stable or not, that is whether perturbation to species recover or not? We can show that the two step model is stable by using the following mathematical argument.

The differential equation describing the two step model is given by,

$$\frac{dS_1}{dt} = k_1 X_o - k_2 S_1$$

If the system is at steady state, let us make a small perturbation to the steady state concentration of S_1, δS_1 and ask how δS_1 changes as a result of this perturbation, that is what is $d(\delta S_1)/dt$? The new rate of change equation is rewritten as follows:

$$\frac{d(S_1 + \delta S_1)}{dt} = k_1 X_o - k_2(S_1 + \delta S_1)$$

If we insert the steady state solution for S_1 (equation 7.2) into the above equation we are left with:

$$\frac{d\delta S_1}{dt} = -k_2 \delta S_1 \tag{7.3}$$

In other words the rate of change of the **disturbance itself**, δS_1 is negative, that is, the system attempts to reduce the disturbance so that the system returns back to the original steady state. Systems with this kind of behavior are called **stable**. If the rate of change in S_1 had been positive instead of negative however, the perturbation would have continued to diverge away from the original steady state and the system would them be considered **unstable**.

Another way to look this is graphically. Let us plot rate of change, dS/dt as a function of S, Figure 7.1. The steady state is given where the net rate of change, is zero. If the substrate level fall below this value, the net rate goes positive there by increasing the level of S_1. On the other hand if the substrate rises above the steady state level, the graph shows that the net rate of change goes negative so that S_1 is brought back down. The system is therefore stable.

A biochemical pathway is dynamically stable at steady state if small perturbations in the floating species concentrations relax back to the original state.

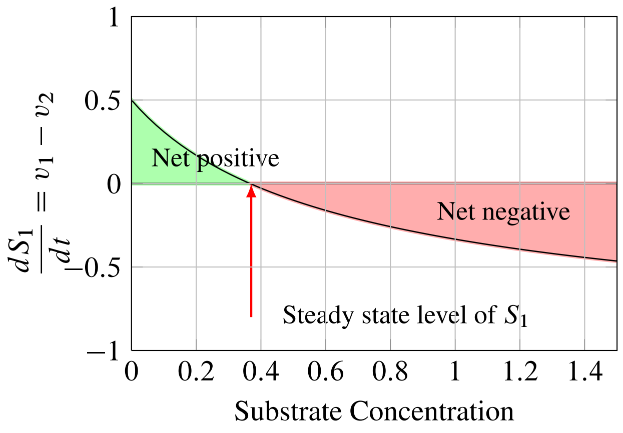


Figure 7.1 Rate of change as a function of S_1, When S_1 is below the steady state value, the net change is positive meaning that S_1 will increase. When S_1 is above the steady state value, the net change is negative meaning that S_1 will decrease. The system is therefore stable.

To continue, let us divide both sides of equation 7.3 by δS_1 and taking the limit, we find that $\partial(dS_1/dt)/\partial S_1$ is equal to $-k_2$. The stability of this simple system can therefore be determined by inspecting the sign of $\partial(dS_1/dt)/\partial S_1$ which can be easily determined by taking the derivatives of the differential equations with respect to the species concentrations.

For larger systems the stability of a system can be determined by looking at all the terms $\partial(dS_i/dt)/\partial S_i$ which are given collectively by the expression:

$$\frac{d(d\mathbf{s}/dt)}{d\mathbf{s}} = \boldsymbol{J} \tag{7.4}$$

where $\boldsymbol{J}$ is called the **Jacobian matrix** containing elements of the form $\partial(dS_i/dt)/\partial S_i$. Equation 7.3 can be generalized to:

$$\frac{d(\delta\mathbf{s})}{dt} = \boldsymbol{J}\delta\mathbf{s} \tag{7.5}$$

where $\boldsymbol{J}$ is given by

$$\begin{bmatrix} \dfrac{\partial S_1/dt}{\partial S_1} & \cdots & \dfrac{\partial S_1/dt}{\partial S_m} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial S_m/dt}{\partial S_1} & \cdots & \dfrac{\partial S_m/dt}{\partial S_m} \end{bmatrix}$$

Equation 7.5 is an example of an unforced linear differential equation and has the general form:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{A}\boldsymbol{x}$$

Solutions to such equations are well known and take the form:

$$x_j(t) = c_1\boldsymbol{K}_1 e^{\lambda_1 t} + c_2\boldsymbol{K}_2 e^{\lambda_2 t} + \cdots c_n\boldsymbol{K}_n e^{\lambda_n t}$$

That is the solution to an unforced linear differential equations involves a sum of exponentials, $e^{\lambda_i t}$, constants c_i and vectors, $\boldsymbol{K}_i$. The exponents of the exponentials are given by the eigenvalues (See Appendix C) of the matrix, $\boldsymbol{A}$ and $\boldsymbol{K}_i$ the corresponding eigenvectors. The c_i terms are related to the initial conditions assigned to the problem. It is possible for the eigenvalues to be complex but in general if the real parts of the eigenvalues are negative then the exponents decay (stable) whereas if they are positive the exponents grow (unstable). We can therefore determine the stability properties of a given model by computing the eigenvalues of the Jacobian matrix and looking for any positive eigenvalues. Note that the elements of the Jacobian matrix will often be a function of the species levels, it is therefore important that the Jacobian be evaluated at the steady state of interest.

Example 7.1

The following system:

$$S_1 \to S_2 \to$$

if governed by the following set of differential equations:

$$\frac{dS_1}{dt} = -2S_1$$

$$\frac{dS_2}{dt} = 2S_1 - 4S_2$$

The solution to this system can be derived using Mathematica or by using standard algebraic method for solving linear homogeneous systems. The solution can be found to be:

$$\begin{pmatrix} S_1 \\ S_2 \end{pmatrix} = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-2t} + c_2 \begin{pmatrix} 0 \\ 1 \end{pmatrix} e^{-4t}$$

$$S_1 = c_1 e^{-2t}$$

$$S_2 = c_1 e^{-2t} + c_2 e^{-4t}$$

Since the exponents are all negative (-2, -2 and -4), the system is stable to perturbations in S_1 and S_2.

There are many software packages that will compute the eigenvalues of a matrix and there are a small number packages that can compute the Jacobian directly from the biochemical model. For example, the script below is taken from Jarnac, it defines the simple model, initializes the model values, computes the steady state and then prints out the eigenvalues of the Jacobian matrix. For a simple one variable model, the Jacobian matrix only has a single entry and the eigenvalue corresponds to that entry. The output from running the script is given below showing that the eigenvalue is –0.3. Since we have a negative eigenvalue, the pathway must be stable to perturbations in S_1.

```
p = defn model
     $Xo -> S1;  k1*Xo;
      S1 -> $X1; k2*S1;
end;

// Set up the model initial conditions
p.Xo = 1;   p.X1 = 0;
```

```
p.k1 = 0.2; p.k2 = 0.3;

// Evaluation the steady state
p.ss.eval;
// print the eigenvalues of the Jacobian matrix
println eigenvalues (p.Jac);

// Output follows:
{    -0.3}
```

Example 7.2

The following system:

$$\rightarrow S_1 \rightarrow S_2 \rightarrow$$

if governed by the following set of differential equations:

$$\frac{dS_1}{dt} = 3 - 2S_1$$

$$\frac{dS_2}{dt} = 2S_1 - 4S_2$$

The Jacobian matrix is computed by differentiating the equations with respect to the steady state values of S_1 and S_2:

$$\boldsymbol{J} = \begin{bmatrix} -2 & 0 \\ 2 & -4 \end{bmatrix}$$

The eigenvalues for this matrix are: -2 and -4 respectively. Since both eigenvalues are negative the system is stable to small perturbations in S_1 and S_2.

The pattern of eigenvalues can tell us a lot about stability but also about the form of the transients that will occur when we perturb the state of the system. In the next section we will investigate this aspect.

7.2 Phase Portraits

The word **phase space** refers to a space where all possible states are presented. For example, in a biochemical pathway with two species, x_1 and x_2, the phase space consists of all possible trajectories of x_1 and x_2 in time. For two dimensional systems the phase space can be very conveniently display on an x/y graph where each axis represents one of the state variables. A visual representation of the phase space is often called a **phase portrait** or **phase plane**. To illustrate phase portrait consider the following simple reaction network:

$$\xrightarrow{v_o} x_1 \xrightarrow{k_1\ x_1} x_2 \xrightarrow{k_2\ x_2}$$

with two linear differential equations:

$$\frac{dx_1}{dt} = v_o - k_1 x_1$$

$$\frac{dx_2}{dt} = k_1 x_1 - k_2 x_2$$

We can assign particular values to the parameters, set up some initial conditions and plot in phase space the evolution of x_1 and x_2. If we replot the solution using many different initial conditions we get something that looks like the plots shown in Figures 7.2 to 7.7.

These plots illustrate a variety of transient behaviors around a steady state. These particular transient behaviors apply specifically to linear differential equations. If we have a nonlinear system and we linearize the system around the steady state, the linearized system will behave in a way suggested by these plots.

A two dimensional linear system of differential equations has solutions of the form:

$$x_1 = c_1 k_1 e^{\lambda_1 t} + c_2 k_2 e^{\lambda_2 t}$$
$$x_2 = c_3 k_3 e^{\lambda_3 t} + c_4 k_4 e^{\lambda_4 t}$$

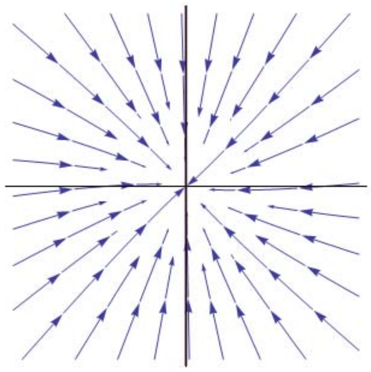


Figure 7.2 Phase portrait for the two species reaction network. Stable node. **Negative Eigenvalues**. Matrix $\boldsymbol{A}$: $a_{11} = -2, a_{12} = 0, a_{21} = -0.15, a_{22} = -2$. Corresponding eigenvalues: $\lambda_1 = -2, \lambda_2 = -2$

The c_i and k_i terms are constants related to the initial conditions and eigenvectors respectively but the λ_i terms, or eigenvalues, determine the qualitative pattern that a given behavior might have. It should be noted that the eigenvalues can be complex or real numbers. In applied mathematics, e raised to a complex number immediately suggests some kind of periodic behavior. Let us consider the different possibilities for the values of the eigenvalues.

Both Eigenvalues have the same sign, different magnitude but are real. If both eigenvalues are negative the equations describe a system known as a **stable node**. All trajectories move towards the steady state point. If the eigenvalues have the same magnitude and the c_i terms have the same magnitude then the trajectories move to the steady state in a symmetric manner, Figure 7.2. If the k_i values differ then the trajectories will tend to twist, Figure 7.3.

Figure 7.3 shows the case when the two eigenvalues are the same sign (positive in this case) but of different magnitude, When the eigenvalues are positive the trajectories move out from the steady state, an **unstable node**. Such systems are therefore called unstable.

Real Eigenvalues but of opposite sign. If the two eigenvalues are real but

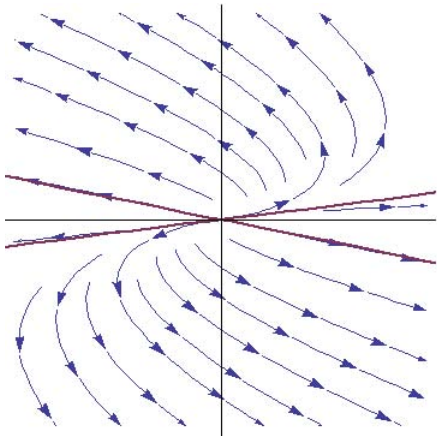


Figure 7.3 Phase portrait for the two species reaction network. Unstable node, also called an improper node. **Positive Eigenvalues**. Matrix $\boldsymbol{A}$: $a_{11} = 1.2, a_{12} = -2, a_{21} = -0.05, a_{22} = 1.35$. Corresponding eigenvalues: $\lambda_1 = 1.6, \lambda_2 = 0.95$

of opposite sign we get behavior called a saddle-node. This is where in one direction the trajectories move towards the steady state and in another direction move away. Since trajectories can only move towards the steady state if they are exactly on the saddle node ridge, but once they reach the node they will diverge. Saddle nodes are therefore unstable.

Complex Eigenvalues Sometimes the eigenvalues can be complex, that is of the form $a + ib$ where i is the imaginary number. It may seem strange that the solution to the differential equations can admit complex eigenvalues. To understand what these mean we have to recall Euler's formula:

$$e^{i\theta} = cos(\theta) + i\,\sin(\theta)$$

or extended to:

$$e^{(a+bi)t} = e^{at}\cos(bt) + i\,e^{bt}\sin(bt)$$

When the solutions are expressed in terms of sums of sine and cosine terms, the imaginary parts will cancel out, leaving just trigonometric terms with only real parts.

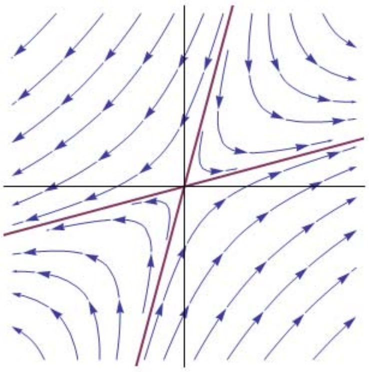


Figure 7.4 Phase portrait for the two species reaction network. Saddle node. **One Positive and One Negative Eigenvalue**. Matrix A: $a_{11} = 2, a_{12} = -1, a_{21} = 1, a_{22} = -2$. Corresponding eigenvalues: $\lambda_1 = -1.73, \lambda_2 = 1.73$

We can show this as follows. Consider the system:

$$x(t) = c_1 z_1 e^{(\lambda + i\mu)t} + c_2 z_2 e^{(\lambda - i\mu)t}$$

where z_1 and z_2 are corresponding conjugate eigenvectors. Using Euler's formula, $e^{i\mu} = \cos(\mu) + i\,\sin(\mu)$ and that $e^{(\lambda+i\mu)t} = e^{\lambda t} e^{i\mu t}$ we obtain:

$$\begin{aligned} x(t) = c_1 z_1 e^{\lambda t}(\cos(\mu t) + i\,\sin(\mu t)) \\ + c_2 z_2 e^{\lambda t}(\cos(\mu t) - i\,\sin(\mu t)) \end{aligned}$$

Description	Eigenvalues	Behavior
Both Positive	$r_1 > r_2 > 0$	Unstable
Both Negative	$r_1 < r_2 < 0$	Stable
Positive and Negative	$r_1 < 0 < r_1$	Saddle point
Complex Conjugate	$r_1 > r_1 > 0$	Unstable spiral
Complex Conjugate	$r_1 < r_1 < 0$	Stable spiral
Pure Imaginary	$r_1 = r_1 = 0$	Center

Table 7.1 Summary of Node Behaviors

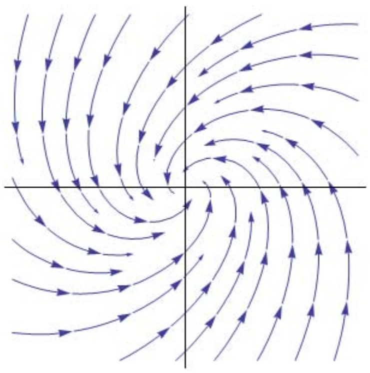


Figure 7.5 Phase portrait for the two species reaction network. Stable spiral node. **Negative Complex Eigenvalues**. Matrix $\boldsymbol{A}$: $a_{11} = -0.5, a_{12} = -1, a_{21} = 1, a_{22} = -1$. Corresponding eigenvalues: $\lambda_1 = -0.75 + 0.97i, \lambda_2 = -0.75 - 0.97i$

Writing the conjugate eigenvectors as $z_1 = a + bi$ and $z_2 = a - bi$, we get:

$$\begin{aligned} x(t) = c_1(a + bi)e^{\lambda t}(\cos(\mu t) + i \sin(\mu t)) \\ +(a - bi)e^{\lambda t}(\cos(\mu t) - i \sin(\mu t)) \end{aligned}$$

Multiply out and separate the real and imaginary parts yields:

$$\begin{aligned} x(t) = e^{\lambda t} \, [c_1(a\cos(\mu t) - b\sin(\mu t) + i(a\sin(\mu t) + b\cos(\mu t))) \\ + c_2(a\cos(\mu t) - b\sin(\mu t) - i(a\sin(\mu t) + b\cos(\mu t)))] \end{aligned}$$

The complex terms cancel leaving only the real parts. If we set $c_1 + c_2 = k_1$ and $(c_1 - c_2)i = k_2$ then:

$$\begin{aligned} x(t) = e^{\lambda t} \, [k_1(a\cos(\mu t) - b\sin(\mu t)) \\ k_2(a\sin(\mu t) + b\cos(\mu t))] \end{aligned}$$

The solution is real when the constants c_1 and c_2 are real. This will only be the case when the eigenvalues are a conjugate pair, $(a \pm ib)$ which is

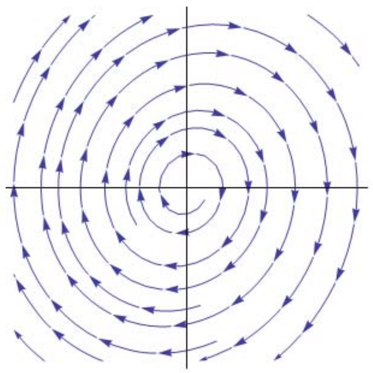


Figure 7.6 Phase portrait for the two species reaction network. Unstable spiral node. **Positive Complex Eigenvalues**. Matrix A: $a_{11} = 0, a_{12} = 1.0, a_{21} = -1.2, a_{22} = 0.2$. Corresponding eigenvalues: $\lambda_1 = 0.1 + 1.09i, \lambda_2 = 0.1 - 1.09i$

the case we are looking at. Therefore systems that admit a complex pair of conjgate eigenvalues result in periodic real solutions.

This result shows that the appearance of complex numbers in the eigenvalues results in periodic solutions. For this reason solution with complex eigenvalues tend to display trajectories such as those shown in Figure **??** and 7.7. If the real parts are positive then the spiral trajectories move outwards (unstable). If the real parts of the eigenvalues are negative then the spiral trajectory moves into the steady state (stable).

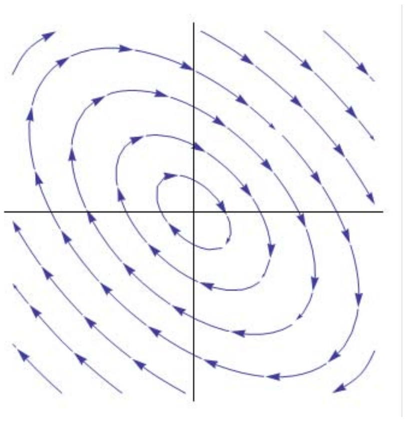


Figure 7.7 Phase portrait for the two species reaction network. Center node. **Complex Eigenvalues, Zero Real Part**. Matrix $\boldsymbol{A}$: $a_{11} = 1, a_{12} = 2.0, a_{21} = -2, a_{22} = -1$. Corresponding eigenvalues: $\lambda_1 = 0 + 1.76i, \lambda_2 = 0 - 1.76i$

Conjugate Pair

A complex conjugate pair is a complex number of the form: $a \pm bi$. The eigenvalues for a two variable linear system system with matrix $\boldsymbol{A}$, can be computed directly using the relation:

$$\lambda = \frac{\text{tr}\,(\boldsymbol{A}) \pm \sqrt{\text{tr}^2(\boldsymbol{A}) - 4\,\det\,(\boldsymbol{A})}}{2}$$

where tr $(\boldsymbol{A}) = a + d$ and det $(\boldsymbol{A}) = ad - bc$. If the term in the square root is negative, the eigenvalues will always come out as a conjugate pair owning to the $\pm$ term. If $\text{tr}^2(\boldsymbol{A}) - 4\,\det(\boldsymbol{A}) < 0$ then the solution will be the conjugate pair:

$$\lambda = \frac{\text{tr}\,(\boldsymbol{A})}{2} \pm \frac{\sqrt{\text{tr}^2(\boldsymbol{A}) - 4\,\det(\boldsymbol{A})}}{2}$$

Therefore a complex eigenvalue will always be accompanied by its conjugate partner.

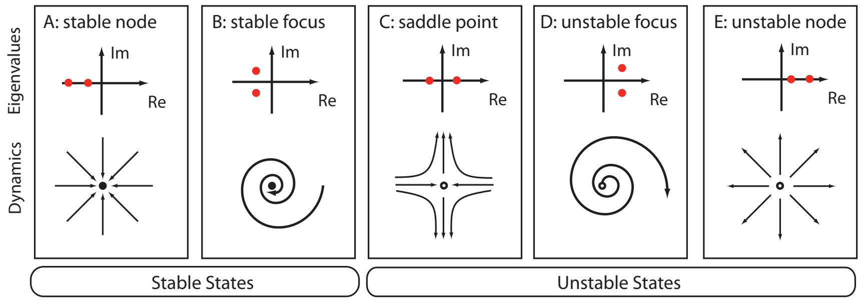


Figure 7.8 Summary of behaviors including dynamics and associated eigenvalues for a two dimensional linear system. Adapted from "Computational Models of Metabolism: Stability and Regulation in Metabolic Networks", Adv in Chem Phys, Vol 142, Steuer and Junker.

7.3 Bifurcation Plots

In its simplest form, a bifurcation plot is just a plot of the steady state value of a system variable, such as a concentration versus a parameter of the system. For example we saw that the steady state solution for the simple system:

$$\frac{dS_1}{dt} = k_1 Xo - k_2 S_1$$

was given by:

$$S_1 = k_1 Xo / k_2$$

We can now plot the steady state value of S_1 as a function of k_2, as shown in Figure 7.9.

Of more interest is that bifurcation plots can be used to identify changes in qualitative behavior, particularly systems that have multiple steady states. Consider the system shown 7.10. This shows a simple gene circuit with a positive feedback loop. That is as the transcription factor x accumulates it binds to an operator site on the gene which increases its synthesis.

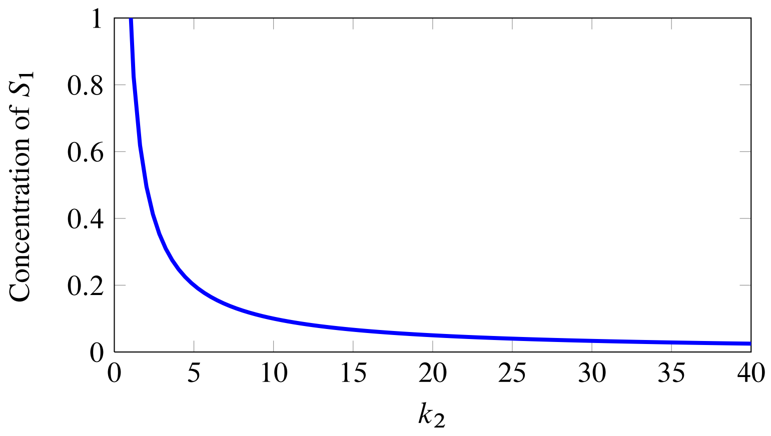


Figure 7.9 Steady state concentration of S_1 as a function of k_2 for the system, $dS_1/dt = k_1 X_o - k_2 S_1$

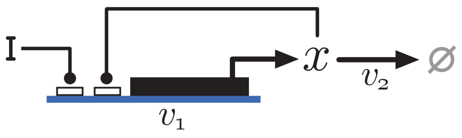


Figure 7.10 System with Positive Feedback

From the circuit diagram we can construct a simple model. This model uses the following kinetic laws for the synthesis and degradation steps.

$$v_1 = k_1 \frac{x^4}{k_2 + x^4}$$
$$v_2 = k_3 x$$

We can plot both rate laws as a function of transcription factor x to obtain the figure shown in Figure 7.11. If we vary the slope of v_2, buy changing k_3, the intersection points will change. We can plot the intersection points as a function of k_3. If we do this we obtain the diagram shown in Figure 7.12.

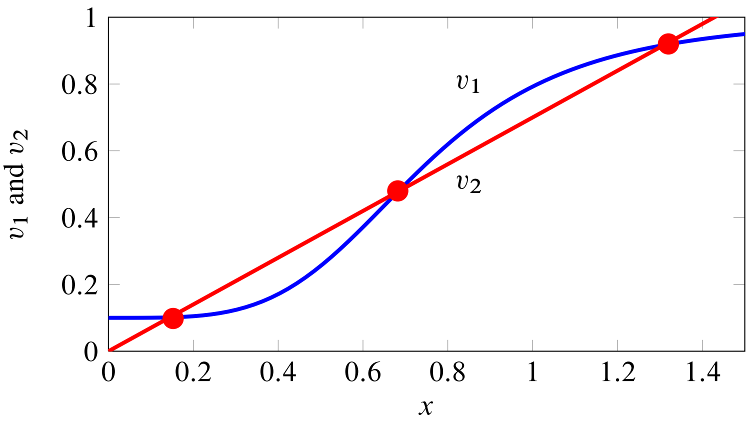


Figure 7.11 Reaction velocities, v_1 and v_2 as a function of x for the system, Figure 7.9. The intersection points marked by fill circles indicate possible steady states. Computed using the SBW AUTO C# Tool.

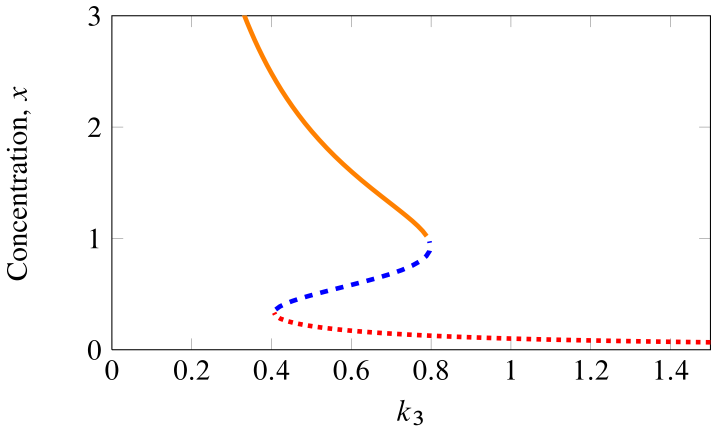


Figure 7.12 Plotting intersection points from Figure 7.11 as a function of k_3. Dotted line marks the lower intersection point, dashed line the middle intersection points, and solid line the upper intersection point.

Figure 7.12 shows that at some value of the parameter k_3, the system has three possible steady states, outside this range only single steady state persists. Bifurcation diagrams are extremely useful for uncovering and displaying such information. Drawing bifurcation diagrams is not easy however. There are some software tools that can help. Figure 7.12 for example was generated using the SBW Auto C# tool, available at http://jdesigner.sourceforge.net/Site/Auto_C.html. Another useful tool for drawing bifurcation diagrams is Oscill8, available from http://oscill8.sourceforge.net/. Both tools can read SBML. Figure 7.12 was generated first by entering the model into Jarnac (Shown in listing 7.1) to generate the SBML. The model was then passed to Auto C# to produce the bifurcation diagram.

Bistability will be discussed in more detail in a separate volume.

```
p = defn cell
      $Xo -> x; 0.1 + k1*x^4/(k11+x^4);
        x -> $w; k2*x;
end;

// Initialization here
p.k1 = 0.9; p.k11 = 0.3;
p.k2 = 0.7;
// Compute steady state
p.ss.eval;
```

Listing 7.1 Model used to create Figure 7.12

Further Reading

1. Steuer R and Junker BH (2009). Computational models of metabolism: stability and regulation in metabolic networks. Advances in chemical physics, 142, 105.

8 Multicompartmental System

8.1 Multicompartment Systems

It is easy to think of a biological cell as a well mixed compartment and base our models around that premise. However, anyone who has looked through a microscope at a drop of pond water and observed swimming protists will quickly realize that many cells are highly structured and compartmentalized. In eukaryotic cells the most obvious compartments are the nucleus, mitochondria, chloroplasts and a wide variety of enclose spaces serving different functions. In all these cases, movement of material occurs from one compartment to another, sometime active (requiring energy) and sometime passive. In addition all the compartments have widely different volumes.

8.2 Simple Diffusion

Let us start be considering the simplest possible example, the reversible and passive diffusion of solute from one compartment of volume V_1 to another compartment of volume V_2 (Figure 8.1).

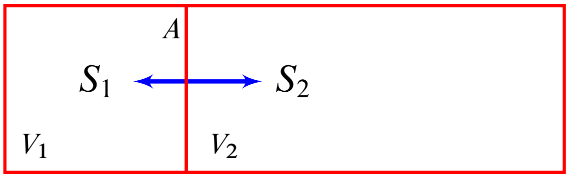


Figure 8.1 Two compartment model with volumes V_1 and V_2. S_1 and S_2 diffusion passively across the membrane with area A.

Let us assume that the volume in compartment two is ten times the volume in compartment one. This means that as mass moves from V_1 to V_2 will be be diluted in the large volume at V_2. This means that even though the concentration of S_1 in V_1 might change by 10%, in volume V_2 it will only change by 1%. We therefore have to account for this when we write the differential equations that describe the change in S_1 and S_2. The key to this is to describe the rates of change in terms of amounts rather than concentrations. Let us define the amount of S_1 and S_2 as follows :

$$S_1 = \frac{n_1}{V_1} \qquad S_2 = \frac{n_2}{V_2}$$

where n_1 and n_2 are the amounts of S_1 and S_2 respectively. According to Fick's first law of diffusion, the flux is proportional to the concentration gradient across the membrane:

$$J_A = -D_A \frac{dS}{dx}$$

The negative sign ensures that the flux is positive when the concentration gradient is negative, that is declining left to right. J_A is the flux in units of moles $\mathrm{l}^{-2}\ \mathrm{t}^{-1}$ (moles per unit area per time), D_A the **diffusion coefficient** has units of $\mathrm{l}^2\ \mathrm{t}^{-1}$ (area per unit time), S is the concentration and dS/dx the concentration gradient in units of moles $\mathrm{l}^{-3}\ \mathrm{l}^{-1}$, that is moles per volume per length.

If the zone of diffusion has a width δ, we can approximate Fick's law:

$$J_A = -D_A \frac{S_{\text{out}} - S_{\text{in}}}{\delta}$$

or

$$\boxed{J_A = P_A(S_{\text{in}} - S_{\text{out}}) \qquad (8.1)}$$

where P_A equals D_A/δ and is called the permeability coefficient with units of length per unit time (often cm t^{-1}). We assume here that the permeability is the same on both sides of the membrane. The units of flux at this stage are moles per unit area per unit time (moles cm^{-2} t^{-1}. To obtain the total amount of mass that moves from one compartment to another we must multiply the flux, J_A, by the cross-sectional area of the membrane area, thus:

$$J = AJ_A$$

where J is the total amount of substance crossing the membrane and A the area of the membrane. If this substance is moving into a volume V, then the rate of change of concentration in the compartment is given by:

$$\frac{dS}{dt} = -\frac{J}{V}$$

The negative sign indicates that mass is leaving the compartment. We can now write the differential equations for the two compartment model:

$$\frac{dS_1}{dt} = -\frac{J}{V_1}; \qquad \frac{dS_2}{dt} = \frac{J}{V_2}$$

where the flux J is given by:

$$J = AP_A(S_1 - S_2)$$

The result should be thermodynamically consistent. To test this we set the rate of change of S_1 to zero, that is:

$$\frac{AP_A}{V_1}(S_1 - S_2) = 0$$

That is $S_1 = S_2$. Since we are dealing with simple diffusion we expect at thermodynamic equilibrium for the two concentrations to be equal, which they are. Note also that the units are consistent, with $U(A) = l^2$, $U(V_1) = l^3$, $U(P_A) = l\ t^{-1}$ and $U(S_x) = \text{mol}\ l^3$.

8.3 Catalytic Reaction across a Membrane

Let us consider a more complex example where a solute S_1 is transported through a protein pore (and hence saturable) and it catalytically transformed into product S_2. Instead of using Fick's law we must consider using a saturable Michaelis-Menten like rate law. Let us assume that the concentration of protein pores on the membrane is given by:

$$e = \frac{n_e}{A}$$

where n_e is the number of protein pores, A the area of the membrane and e the concentration in moles of pores per unit area. The rate of catalysis will be proportional to the concentration of pores on the membrane. If we invoke a saturable rate law we can write that the rate of transformation in moles (amount) per unit area per unit time (the flux J_A) is given by:

$$J_A = e \frac{k_f S_1 - k_r S_2}{1 + S_1/K_{m1} + S_2/K_{m2}}$$

where k_f and k_r are the forward and reverse rate constants such that $k_f/k_r = K_{eq}$. Given the last relation we can write:

$$J_A = e k_f \frac{S_1 - S_2/K_{eq}}{1 + S_1/K_{m1} + S_2/K_{m2}}$$

Given the units for J_A, e and the rate term, the units for k_f are mol $l^3\ t^{-1}$.

The total flux across the membrane, J_A is given as before:

$$J = A J_A$$

The total flux will equal the following:

$$\frac{dn_1}{dt} = -J; \qquad \frac{dn_2}{dt} = J$$

This means that the rate of change of concentration of S_1 and S_2 is given by:

$$\frac{dS_1}{dt} = -\frac{J}{V_1}; \qquad \frac{dS_2}{dt} = -\frac{J}{V_2};$$

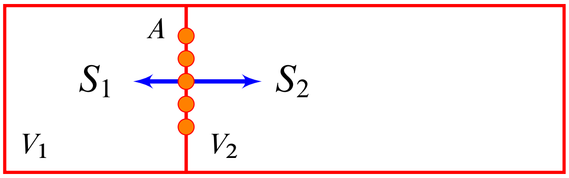


Figure 8.2 Two compartment model with volumes V_1 and V_2. S_1 and S_2 move through saturable protein pores in the membrane and undergo catalytic transformation.

Name	Symbols	Units
Net Flux	J	mol t^{-1}
Flux	J_A	mol l^{-2} t^{-1}
Area	A	l^2
Volume	V	l^3
Concentration	S	mol l^{-3}
Transporter	e	mol l^{-2}
Rate Constant	k_f	mol l^3 t^{-1}

Table 8.1 Units for transporter model. l represents length; S reactant; t time.

Figure 8.1 shows a Jarnac script that models the transporter model. A few things are worth pointing out. By default, Jarnac solves all differential equations in terms of amounts per unit time. This means there is not need to explicitly adjust volume sizes in any equations. Instead we define the compartments we need using the `vol` keyword and then indicate which species is in which compartment. All volume adjustments are then automatic. Jarnac stores levels of species as amounts and converts to concentrations on a need to basis, eg when a concentration is specified in a rate law. This make is quite straight forward to build multicompartment models using Jarnac.

Figure 8.3 shows the results of the simulation. In this case the volume ration is 1 to 10. Notice how the concentration of S_1 starts at 21 but ends up at 1 in the first compartment and 2 in the second compartment. We can check that we've not lost any mass by summing up the mass in each compartment. The total mass at time zero is $21 \times 1 = 21$. The total mass at the end of the run is: $1 \times 1 + 2 \times 10 = 21$. Therefore the mass has been conserved.

```
p = defn cell
      vol V1, V2;
      var S1 in V1, S2 in V2;

      S1 -> S2; A*k*(S1-S2/Keq)/(1 + S1/Km1 + S2/Km2);
end;

p.V1 = 1;  p.V2 = 10;

p.S1 = 21;
p.A = 1; p.k = 1;
p.Km1 = 0.5; p.Km2 = 0.5; p.Keq = 2;

m = p.sim.eval (0, 200, 100);
graph (m);

println "Total Mass = ", p.S1*p.V1 + p.S2*p.V2;
```

Listing 8.1 Script for Multicompartment model with transporter.

8.4 Concentrating Cascade

The final example uses three compartments of decreasing volume. It shows how smaller volumes result in higher and high concentrations. The equilibrium constants for the transport across each membrane is quite high.

```
p = defn cell
      vol V1, V2, V3;
      var S1 in V1, S2 in V2, S3 in V3;
```

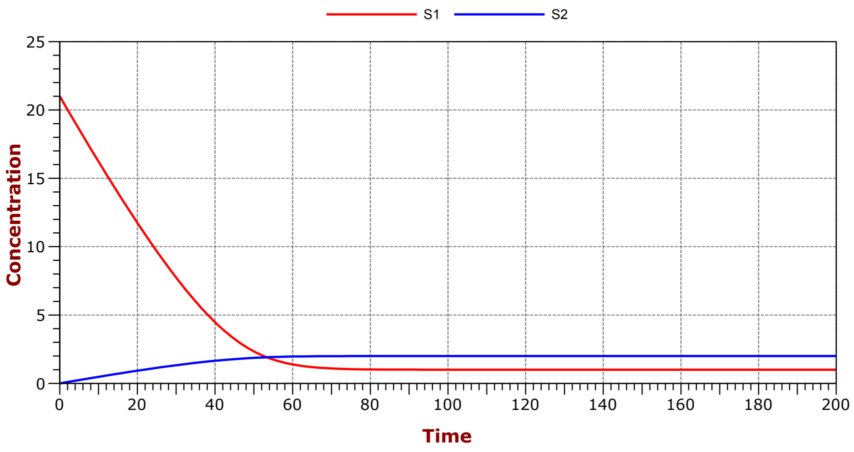


Figure 8.3 Simulation results of a membrane transporter

```
      S1 -> S2; A*k1*(S1-S2/Keq)/(1 + S1/Km1 + S2/Km2);
      S2 -> S3; A*k2*(S2-S3/Keq)/(1 + S2/Km1 + S3/Km2);
end;

p.V1 = 100;
p.V2 = 10;
p.V2 = 1;

p.S1 = 10;
```

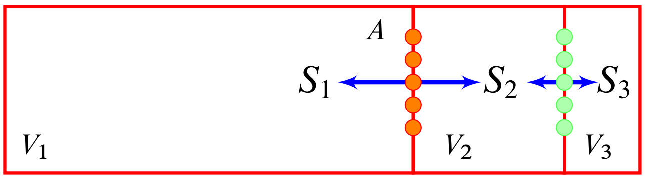


Figure 8.4 Two compartment model with volumes V_1 and V_2. S_1 and S_2 move through saturable protein pores in the membrane and undergo catalytic transformation.

```
p.A = 1; p.k1 = 800; p.k2 = 20;
p.Km1 = 0.5; p.Km2 = 0.5;
p.Keq = 50;

m = p.sim.eval (0, 1000, 200);
graph (m);

println "Total Mass = ", p.S1*p.V1 + p.S2*p.V2 + p.S3*p.V3;
```

Listing 8.2 Script for Multicompartment model with three compartments, each compartment getting progressively smaller.

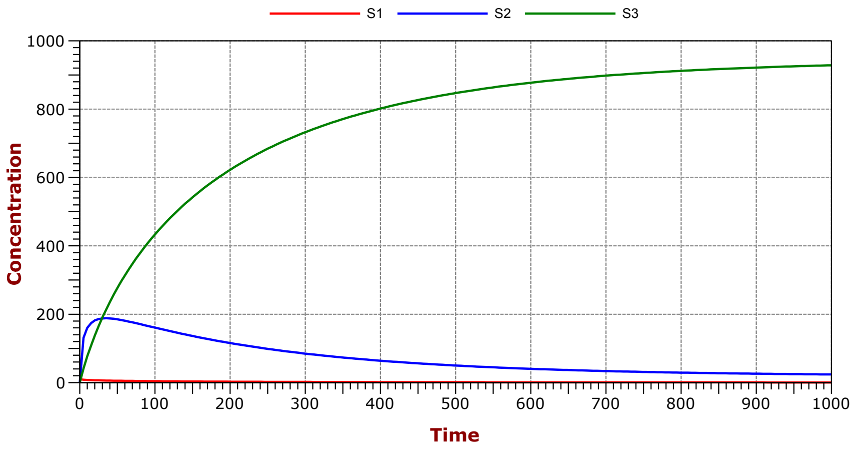


Figure 8.5 Simulation results of a membrane transporter

Further Reading

There are surprisingly few books on compartmental analysis in systems biology. Most books focus on pharmokinetic modeling and its takes some effort to translate the pharmokinetic formalism into a systems biology one. I list three books here, probably the most useful is the one by Neame and Richards which can be obtained with relative ease on the second-hand market.

1. Atkins, GL (1969), Multicompartment models for biological systems, Methuen London, SBN: 416 13820 9 (SBN is not a typo)

2. Jacquez, JA (1985). Compartmental analysis in biology and medicine. Ann Arbor: University of Michigan Press. The third edition (1996) is available from http://www.biomedware.com or directly from http://tinyurl.com/msh54u6.

3. Neame, KD and Richards TG (1972). Elementary kinetics of membrane carrier transport. New York: Wiley. ISBN: 0-470-63078-7

9 Fitting Models

9.1 Introduction

Let us set up an experiment in the lab where a compound S_1 is known to spontaneously react to form S_2 which in turn is converted into a stable compound S_3. The experiment is started by adding an initial amount of S_1 to the reaction vessel. A stop clock is started and the reactions are followed by periodically taking a sample and measuring the levels of S_1, S_2 and S_3. The graph shown in Figure 9.1 shows a typical time series from such an experiment. Note that the data don't follow a smooth curve, this is because of errors made in the measurements. A proposed model for this system might be:

$$S_1 \overset{k_1}{\rightarrow} S_2 \overset{k_2}{\rightarrow} S_3$$

where we hypothesize that both reactions follow simple first-order kinetics. This means we can write down the model equations as:

$$\begin{aligned}\frac{dS_1}{dt} &= -k_1 S_1 \\ \frac{dS_2}{dt} &= k_1 S_1 - k_2 S_2 \\ \frac{dS_3}{dt} &= k_2 S_2\end{aligned} \tag{9.1}$$

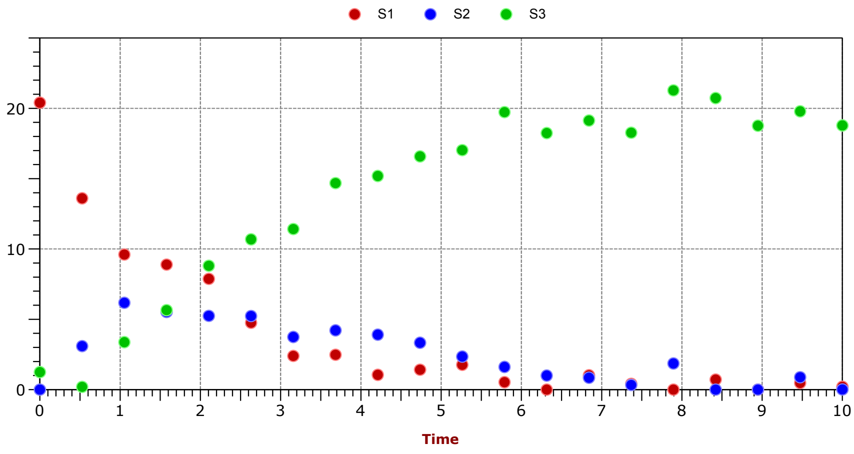


Figure 9.1 Results from a experiment that measures the concentrations of S_1, S_2 and S_3 in time.

The question to consider in this chapter is, given a time series such as the one shown in Figure 9.1 and a proposed model such as 9.1:

1. Can we determine the parameters in the model from the experimental data, e.g. k_1 and k_2?

2. Does the model reasonably represent the known experimental data, that is, is the model a good fit?

3. What confidence do we have in the fitted parameters?

Fitting a model to experimental data is useful for a number of reasons but one obvious application is that we could use the model to make useful predictions. For example, if we started with a different amount of S_1 how long would it take for S_3 to reach a given concentration?

A brute force way to fit a model to some data is to run a simulation of the model many times with random values for the parameters until we find a set of parameters that gives us simulation data that matches the experimental time series curves. One problem with this approach is that we could spend a lot of time coming up with random values for parameters in the hope that at least one set of parameters will match the experimental data. This however is unlikely. Instead, special search algorithms have been devised, called optimization algorithms, that search for the best set of parameters in a systematic way.

Fitting a model means adjusting the parameters of the model until the behavior of the model matches some known experimental data.

To understand how the fitting process works consider a simple model:

$$S_1 \xrightarrow{k_1} S_2$$

Figure 9.2 shows both a solid curve representing a simulation of the model and four experimental data points for the concentration of S_1. The first data point at time zero represents the initial concentration of S_1. Measurements were collected at time points 1, 2, and 3. The e_i terms represent the difference between the experimental data point and the simulation curve. Fitting is the process where we attempt to adjust the parameters of the model (k_1 in this case), such that the difference between the simulation curve and the data points is **minimized**.

Let us indicate the experimental data points using the symbols, x_i, y_i. We will assume there are N data points. We will indicate the model using the symbol $f(x_i; p_1 \ldots p_m)$ where p_i is the ith parameter in the model. That is for a given set of parameters and time point, x_i, the function, f, will

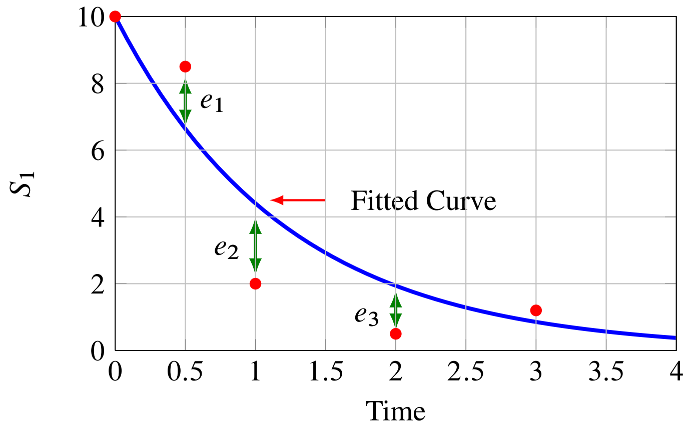


Figure 9.2 Model curve and data plotted on same graph. The solid line is the simulated model, the points represent experimental data. The experimental data has errors, e_i, such that they do not match the model curve exactly. Data fitting attempts to minimize the e_i terms.

return the corresponding model y_i value. If the model is a set of differential equations, we would run a simulation in order to obtain the value of y_i at x_i. The fitting procedure will attempt to minimize the difference between the model f and the data points, that is minimize:

$$y_i - f(x_i; p_1 \ldots p_m)$$

However the difference between a data point and the model could be positive or negative depending on the error in the data point, therefore we take the square of the difference to make the term positive:

$$(y_i - f(x_i; p_1 \ldots p_m))^2$$

This difference however only corresponds to one data point and we should be considering all data points when we're trying to fit the model. Therefore we will form the sum of all the differences and attempt to minimize the sum, that is:

$$\sum_{i=1}^{N}(y_i - f(x_i; p_1 \dots p_m))^2$$

We can take one step further and reason that those data points that are more uncertain should contribute less to the sum than those data points which have been more precisely measured. We therefore weight each difference by the standard deviation, σ, that corresponds to that data point (this assumes that we have some measure of uncertainty, if we don't we set the weight to one):

$$\chi^2 \equiv \sum_{i=1}^{N}\left(\frac{y_i - f(x_i; p_1 \dots p_m}{\sigma_i}\right)^2$$

Sometimes the above equation is also expressed in the following equivalent form that emphasizes the weighing in terms of the variance, σ^2:

$$\chi^2 \equiv \sum_{i=1}^{N}\frac{1}{\sigma_i^2}(y_i - f(x_i; p_1 \dots p_m))^2$$

The equation is called the weighted chi-squared sums of squares and can vary between zero and infinity. If the model is a set of differential equations, the f function is a list of data data points from a simulation run. For example, from the previous model let us set the parameter, k_1 to -0.95. Table 9.1 shows an example of computing the chi-square given some data points and results from a model run.

An important variant on the chi-squared is the **reduced chi-squared** which is used when looking at the quality of the fit and estimating the confidence in the estimated parameter. We will return to this later.

$$\chi^2_{\text{reduced}} \equiv \frac{1}{N-P}\sum_{i=1}^{N}\frac{1}{\sigma_i^2}(y_i - f(x_i; p_1 \dots p_m))^2$$

where N is the number of data points and P the number of parameters to be fitted in the model.

Time	Data Point	Point from Model	Difference	Difference Squared
0	10	10	0	0
0.5	7.9	6.2	-1.68	2.8
1	2.1	3.87	1,77	3.12
2	0.5	1.5	1	1
3	0.6	0.58	-0.02	0.00046

Table 9.1 Calculating chi-squared. Assume we have no variances the data points, therefore weighting is one. χ^2 is the sum of the right most column and equals 6.9. The reduced chi-squared, χ^2_{reduced}, is $6.9/(5/1) = 1.3$.

9.2 Optimization Algorithms

Fitting a model to data is an iterative process. It involves making an initial guess to the parameters, p_i, computing the χ_2 value and using a rule that adjusts the parameter values such that the χ^2 is reduced in the next iteration. This procedure is repeated my times until the χ^2 can no longer be reduced, at which point we have probably fitted the data to the model and the parameter values should be able to reproduce the experimental data when plugged into the model.

One way to imagine this process is to consider a two parameter system where the χ^2 describes a surface. Figure 9.3 shows such a surface, also called a fitness landscape. The z-axis is a measure of the χ^2 and the x and y axes are the two parameters. As the two parameters are varied the χ^2 changes, sometimes to high values, sometimes to low values. The low values are the ones we seek, ideally the lowest possible χ_2 value, called the **global minimum**. We can see that the surface is quite complicated with a number of hills and valleys. This is often the case when fitting a any model of moderate complexity.

To start the optimization we select, possibly at random, values for the two parameters. Let us assume that we started the optimization at the top of the tallest hill. What we seek is the lowest valley on the surface. An obvious strategy to find a valley is to move down hill until we reach the lowest point. However if we did this we wouldn't reach the lowest point but an intermediate low point called a local minimum (most likely point M$_a$). However if we started on the near side of the second peak and moved down the hill we would reach the deepest or global minimum at M_c. Depending on how complex the surface, it can be difficult to find the global minimum. A great variety of approaches have therefore been devised to try to find the global minimum on a fitness landscape. We will describe four common approaches to finding minima on a fitness landscape. To make the notation more compaxt we will refer to χ^2 using the symbol ϵ.

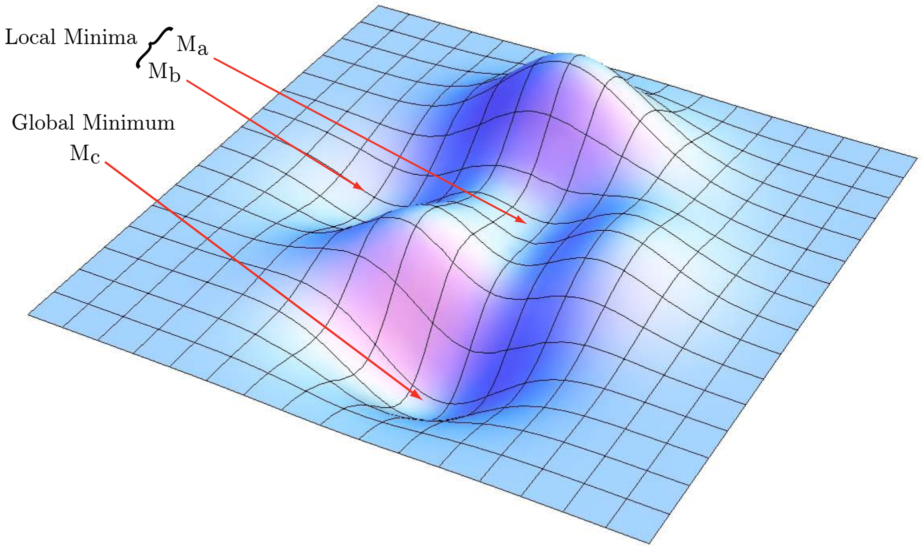


Figure 9.3 Example of a fitness landscape showing multiple minima (M_a and M_b) and a global minimum at M_c. The vertical axis represents χ^2 and the x and y axes the parameters. THe plot shows how χ^2 changes for different parameter values. (Out of interest the function used to plot the surface was: $3(1-x)^2\ \exp(-x^2-(y+1)^2)-10(x/5-x^3-y^5)\ \exp(-x^2-y^2) - 1/3\ \exp(-(x+1)^2 - y^2)$)

Levenburg-Marquardt

In the last section we mentioned an approach that involved moving down the fitness landscape until we reach the minimum. One problem with this method is that in steep sections of the landscape we will tend to move quickly while in more shallow areas we will tend to move slowly. This can be a cause of some problems. If we move too quickly we run the risk of missing the bottom of the valley and overshooting our target (Figure 9.4). On the other hand if we move too slowly when we are near the bottom of the valley we might take too long to reach our target.

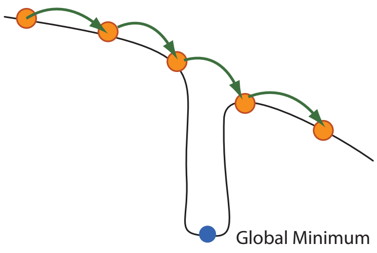


Figure 9.4 Overshoot when using gradient decent. Circles represent points reached during gradient decent. The step size is too big to notice the deep valley and misses it completely.

To help mitigate these potential issues the Levenburg-Marquardt method employs a weighted mixture between two types of searches [70, 85]. The first is a gradient search, and the other is the approximation of the error surface near the minimum as a quadratic function of the parameters. Near the minimum the function can be approximated using Taylor series with two terms, that is:

$$f(x) = f(x_0) + \delta x \frac{df}{dt} + (\delta x)^2 \frac{d^2 f}{dt^2}$$

hence we can approximate the surface near the minimum using a quadratic function. In gradient descent, the approach is to descend down the error surface in a direction opposite (a positive gradient would take us up hill) to the local gradient, i.e. the direction of maximum change. The step size is set to be a constant. For the $(k + 1)$ iteration, p_i is changed according to

$$\mathbf{p}^{k+1} = \mathbf{p}^k - \mu \mathbf{d} \tag{9.2}$$

where[1] the gradient $\mathbf{d} = \frac{\partial \epsilon}{\partial \mathbf{p}}$, and μ is the step size.

As we get closer to the minimum we can approximate the surface using a quadratic function in the parameters. To obtain this approximation we can use the Taylor series to expand about $\mathbf{p}_0$ as follows, where $\delta\mathbf{p} = \mathbf{p} - \mathbf{p}_0$:

$$\epsilon = \epsilon_0 + \delta\mathbf{p}\,\mathbf{d} + \frac{1}{2}\,\delta\mathbf{p}^T\,\mathbf{H}\,\delta\mathbf{p} \tag{9.3}$$

where $\mathbf{d}_i = \partial\epsilon/\partial p_i$. **H** is called the **Hessian** and has elements defined by:

$$H_{ij} = \frac{\partial^2 \epsilon}{\partial p_i\, p_j}$$

and describes the **curvature** of the surface. The minimum of the surface is now found by differentiating expression 9.3 with respect to $\delta\mathbf{p}$, and setting the result to zero. From this we obtain the parameter value, in a single step, as

$$\mathbf{p}^{k+1} = \mathbf{p}^k - \mathbf{H}^{-1}\mathbf{d}. \tag{9.4}$$

The basic approach for the Levenberg-Marquardt method is to merge gradient descent (9.2) with the quadratic approximation (9.4), such that when the error surface is very steep, the gradient descent is chosen. When the surface can be approximated by a quadratic, (9.4) is used which will tend to move rapidly to the target optimum. The combined method can be described by equation 9.5 where the value of μ can be used to move from one strategy to the other:

$$\mathbf{p}^{k+1} = \mathbf{p}^k - \left(\mathbf{H} + \mathbf{I}\frac{1}{\mu}\right)^{-1} \mathbf{d}. \tag{9.5}$$

[1]Bold font lower case, e.g **x** represents a column vector, and bold font upper case, eg. **X** represents a matrix

This, if μ becomes large, the equation approximates to the quadratic function approximation (9.4). If however μ becomes small, then the second term in the inverted expression dominates such that:

$$\mathbf{p}^{k+1} \approx \mathbf{p}^{k} - \mu\mathbf{d}$$

which is the gradient decent method (9.2). One last modification is necessary before we have the full Levenberg-Marquardt method. When using gradient decent we are not using the curvature, **H**. Marquardt suggested that some benefit could be obtained by incorporating the curvature during gradient decent. This makes the step size no longer constant since the curvature changes and hence the algorithm can take longer steps in regions where the gradient is less (such as in a long shallow valley), which is exactly what we would like. In addition the method is less likely to overshoot the minimum. The final Levenberg-Marquardt equation is therefore given by:

$$\mathbf{p}^{k+1} = \mathbf{p}^{k} - \left(\mathbf{H} + \mathrm{diag}(\mathbf{H})\frac{1}{\mu}\right)^{-1}\mathbf{d}. \tag{9.6}$$

The algorithm starts with using gradient descent, and if the error can be reduced, which means that it is successful, it decreases the step size μ, hence the quadratic approximation takes over. This process is continued until the change in the ϵ reduces to a very small number. The Levenberg-Marquardt method has proved to be a very successful approach. It's main drawback is that the method tends to find the nearest minimum which could easily be a local minimum. The method can therefore be sensitive to starting conditions. For the surface shown in Figure 9.3, if we start on the highest hill the Levenberg-Marquardt will most likely find the nearest minimum, M_a which is a local minimum not the global minimum which is what we would like to find. The Levenberg-Marquardt is better suited when combined with other methods. If a good starting point can be found then the Levenberg-Marquardt will rapidly find the minimum.

There are a number of freely available open source implementations of the Levenburg-Marquardt method. There are two GPL licensed solvers, GSL[2] and levmar[3]. To avoid the distribution restriction of the GPL licence

[2] http://www.gnu.org/software/gsl/
[3] http://users.ics.forth.gr/~lourakis/levmar/

the lmfit library[4] is highly recommend (licensed under FreeBSD License). The author has used this library with great success. The author also has free implementations in Pascal and C# which are available upon request. There are a variety of Java versions available on the Web, a search using Levenberg-Marquardt java will located many of them. Scripting languages such a R, Python and Matlab also support implementations of Levenberg-Marquardt.

Simplex or Nelder and Mead

The Levenberg-Marquardt method requires the calculation of derivatives during the iteration, this can be slow and not always easy. The following and remaining methods do not require derivatives which means they are easier to implement. The simplex method, as described by Nelder-Mead [79], is a robust search method (that is it is generally tolerant of noisy data), in which the objective function, in our case ϵ, is computed at several test points, and the test point with the highest value for ϵ, is replaced by another point which has a lower value for ϵ. The replacement of the *worst* point involves some rules (Figure 9.5). In a parameter space of P dimensions, an $P + 1$ dimensional geometrical object is created, called as the *simplex*, with its vertices, initialized to some starting values. The $P + 1$ vertices, of the simplex are the points at which the objective function is evaluated. The simplex then evolves by the following steps:

- The simplex reflects the worst point through the opposite face, to a new point.
- If the reflection above results in a better point, i.e. lower error, it is further stretched in that direction (expansion).
- A contraction of the worst point towards the opposite face of the simplex.
- A contraction along all the faces towards the best point.

[4] http://joachimwuttke.de/lmfit/

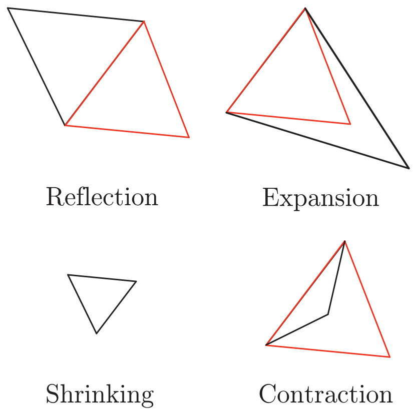


Figure 9.5 In the Nelder and Mead Algorithm, a simplex changes its shape according to four rules, as it does so it makes its way across the fitness landscape.

By successively evolving according to the above steps, the simplex slowly makes its way along the error surface (See https://www.youtube.com/watch?v=HUqLxHfxWqU for an animated example). The shape of the simplex adapts to the landscape, by stretching and contracting. The simplex method can be quite successful unless the initial starting point is a very poor guess. The above steps can be made to converge, either when the simplex size converges to a very small region, or when there is no significant improvement in the error from one iteration to the next.

Implementations of the simplex algorithm are available from a number of sources. The GPL GSL library http://www.gnu.org/software/gsl/ has an implementation. A unrestricted licence version is available from http://www.mikehutt.com/neldermead.html and the author has a C# version available upon request. One advantage of the simplex method is that it is not too difficult to implement (unlike the Levenberg-Marquardt algorithm). Most scripting languages, including Scipy for Python[5] and Java has implementations available.

[5] http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html

Simulated Annealing

The simulated annealing method, derives its name from thermal physics, where the minimization of ϵ, is equivalent to the way a system (such as a metal) reaches its lower state as it is slowly cooled [54]. At a given temperature, the atoms of the metal collide with each other, so that the energy of the system is continually being redistributed. As the temperature is slowly reduced, the atoms begin to form a crystalline structure and eventually reach the minimum energy state. The important point is that the metal has to be cooled slowly, or else, pockets, where the metal is in a higher energy state, than its neighboring regions, can form. For optimization problems, the algorithm works in the following way: Given an initial state i, which in our case would be a set of parameters, the system could jump to another state $i + 1$, with the Boltzmann probability

$$\exp\frac{(\epsilon_i - \epsilon_{i+1})}{T}, \tag{9.7}$$

where T is the **temperature**. For example if ϵ_{i+1} is lower than ϵ_i then the expression will always be greater than one, in this case we will always accept the new solution. However if the new state is bigger then the original then the probability of accepting the new state is less than one and it is possible to actually accept the worse state, effectively we go up hill. Going up hill seems to be counterproductive but it allows the algorithm to potentially jump out from local minima, and eventually find the global minimum. The higher the temperature the more likely the algorithm will move up a hill, therefore the temperature is slowly lowered so that the chance of going up hill reduces.

At a given temperature, the system must be given enough time to sample all the configurations which are accessible using the probability distribution given above. There is no simple way to design a temperature scheduling (temperature as a function of time/iterations), several methods exist and the one that works best depends on the problem at hand. One way way is to follow the algorithm as described in [70, 85] in which the authors consider an adaptation of the simplex method. Our own experience with this approach has been successful.

The basic simulated annealing algorithm is described in the code below:

```
1. Initialize parameter values, $\mathbf{p}$
2. Initialize the temperature, T
3. Calculate the chi-squared, $\epsilon_i$, at the
   current parameter values $\mathbf{p}_i$
4. Make small random changes, $\Delta\mathbf{p}$ to $\mathbf{p}_i$
5. Set $\mathbf{p}_{i+1} = \mathbf{p}_i + \Delta\mathbf{p}$
6. Calculate the new chi-squared, $\epsilon_{i+1}$
7. Calculate $\Delta\epsilon = \epsilon_{i+1} - \epsilon_i$
8. If $\Delta\epsilon \leq 0$ then accept the new state
9. If $\Delta\epsilon > 0$ then
       Generate uniform random number, $u$
       If $u < e^{-\Delta\epsilon/T}$ then
          accept state
       else
          Restore previous state, $\mathbf{p}_i$
20. Reduce the temperature, $T = T - \varepsilon_T$
21. If $T < 0$ or exceeded Max Iterations then
       exit
21. Goto to Step 3.
```

The GPL GSL library[6] has an implementation of simulated annealing and has been used successfully by the author. An unrestricted licensed version in C# is available at[7] and a C version at[8].

Genetic Algorithm

Genetic Algorithms (GA) have been very successful used to solve optimization problems. GA's are special cases of evolutionary algorithms. They are motivated by natural biological processes such as selection, cross over and mutation. The Schema theorem of Holland [38], addresses these intuitive notions, and proves that these operations serve to increase the fitness of a population. In our case we will consider real value coded GA's, where the "gene" is a string of kinetic parameters, which are all real and nonnegative. We start with a random population of individuals where an

[6]http://www.gnu.org/software/gsl/
[7]http://www.codeproject.com/Articles/13789/Simulated-Annealing-Example-in-C
[8] http://www.cs.sunysb.edu/~skiena/algorist/book/programs/

individual is a model with a set of parameters. The fitness of an individual is measured by it's chi-squared value. We monitor ϵ, which decreases as a function of the generation number. General considerations show that the function of crossover, whereby two parent genes, exchange, genetic material is important to preserve the parents' best qualities up into successive generations. They also serve to spread good mutations. This is all due to the fact that only the fittest members are crossed over, and hence their progeny survive into the future generations. There are many variations on what operations occur between each generation, but the following steps could be considered typical:

- Selection: There are various way to rank the fitness of individuals in the population, including elitism, tournament selection or roulette wheel selection. In tournament selection random pairs are made to play a tournament and the winner is decided based on which is fitter. This ensures that even some bad individuals can get selected into the next generation and helps prevent premature convergence. Elitism is where the top 20% or more of the fittest individuals are passed on to the next generation. Roulette selection is where the probability of picking out an individual from the population is based on the fitness of the individual. One or more of these strategies can be used to pick the next generation.

- Crossover: The selected parents are crossed over [42], using an arithmetic mean defined in the following way: Assuming we represent the parents as:

  $$p_1 = (p_1^1, p_1^2, p_1^3, ...) \quad \text{and} \quad p_2 = (p_2^1, p_2^2, p_2^3, ...)$$

  where the p_i^j term is related to the j^{th} parameter in the i^{th} parent. The cross over between p^1 and p^2 will generate two children β_1, β_2 such that:

  $$\beta_1^i = \lambda_i p_1^i + (1-\lambda_i)p_2^i, \qquad (9.8)$$
  $$\beta_2^i = \lambda_i p_2^i + (1-\lambda_i)p_1^i$$

  where λ is a uniform random number between -0.5 and 1.5. Normally arithmetic cross-over maintains the convexity property, but the rule defined above, allows a larger region of parameter space to be explored, since the new points could lie outside the line joining the parents.

- Mutation: For a random number of individuals, one parameter p^i is randomly selected and changed according to

$$p^i = z \;\; p^i_{max}, \tag{9.9}$$

where, $z =$ random $[0, 1]$, is uniformly distributes, and θ^i_{max} is the maximum possible value of the i component of the parameter set.

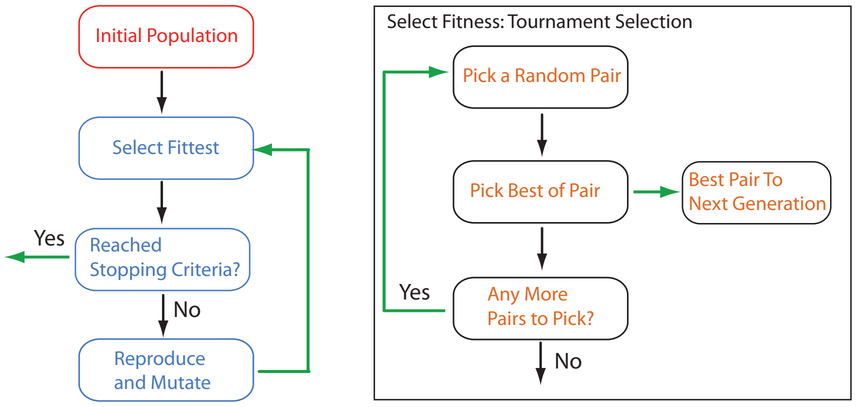


Figure 9.6 Basic genetic algorithm flowchart though many variants exists. There are a number of methods for selecting the fittest individuals, for example tournament selection, roulette selection or elitism.

An example scheme is displayed in Figure 9.6. The operations of cross-over and mutation occur with certain probabilities, which should be adjustable. However the mutation rate is generally a small number < 0.05. Mutations allow the system to explore new regions, whereas crossovers spread these mutations over the population, thereby transferring information about interesting regions in the fitness landscape. Hence if the mutation rate is very high, large regions will be explored, but the members may not survive up into the next generation, since the search is much too exploratory, and not enough information about the landscape has been exploited by the cross-overs. The fitness of the best member in each generation is monitored, and if it turns out that there is little improvement in the fitness, the computation is stopped, and the resulting optimized parameters are examined.

Combining Global and Local search

Some optimization methods are considered local whereas other global. The Levenberg-Marquardt is considered a local method because given a starting point it can usually only find the nearest minimum, hence its search is local. Other methods such as genetic algorithms or simulated annealing are consider global because they tend to search across the fitness landscape, sampling many regions.

Combining a local search within a global search algorithm, is a very attractive possibility. The global search provides the initial seed point, which a local search would use to make a further optimization. For example, for each generation, we take the best two members and use them as initial conditions, for a simplex search. The resulting *fitter* members are replaced back into the population. Hence in every generation, two local searches are performed.

We will now discuss a typical test case that compares the different optimizers. The model we will consider is a simple oscillator, which arises from positive feedback [41]. The model was simulated and random noise was added to the time series to produce noisy data. Four parameters were fitted to the data for the same initial guesses for the parameters, for each of the optimizers. The time taken to reach a good fit was compared including the number of simulations, and the value of ϵ. Levenberg-Marquardt and simplex, were unable to find a fit with the same initial conditions. The data and the fitted curve (bold lines) for the best set of fitted parameters obtained, by using the hybrid optimizer, are displayed in Figure 9.7.

One can see that the GA has the fewest simulations, whereas the simulated annealing takes a lot longer. The hybrid search led to the best optimization, with a modest time, and although the number of simulations are quite large,

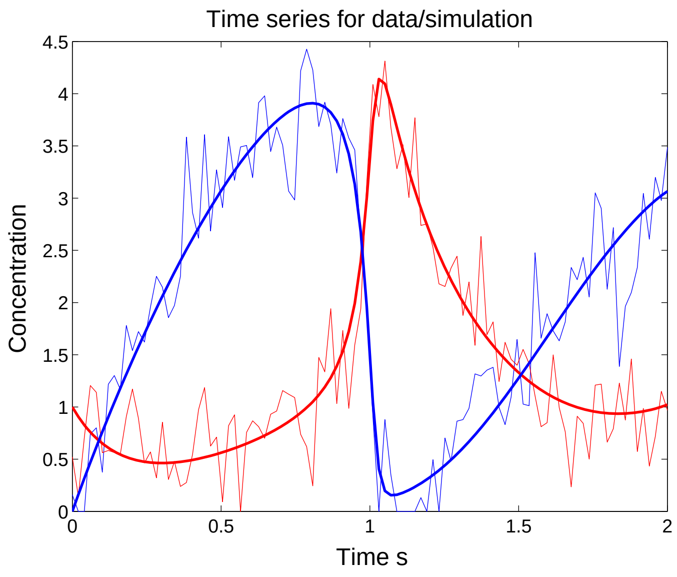


Figure 9.7 A plot comparing the simulation (bold lines), with the data (thin lines), for the concentration time series, for two metabolites for the oscillating model [41].

they are still smaller than the simulated annealing algorithm.

9.3 Is the Model a Good Fit?

If the optimization has been successful, the next question that arises is whether the fit is good or not? There are a number of ways to answer this question which we will cover in the following sections. One possible way is to compare how the data fits **two different models**. One could reason that of the two models, the model that results in the lowest χ^2 is the better fit. However this is not necessarily the case. Imagine a model, m_1, that has ten parameters to fit and another model, m_2, that has only two parameters to fit. Let us assume that the χ^2 for m_1 was 0.5 and the χ^2 for m_2 was 0.8. At first glance it would seem that m_1 is the better fit because it has a lower

Optimizer	Iterations	Simulation time	Simulations	ϵ
GA	500	2 min	4980	3.94
SAsimplex	56	8 min	24959	0.152
GAsimplex	29	4 min	11432	0.104

Table 9.2 Performance comparison for different global optimizers.

χ_2. The danger here is that because m_1 has ten parameters to adjust, there is a lot of freedom such that the model solution could go through every experimental data point resulting in a lower χ^2. This is termed overfitting.

It is much better to compare the reduced χ^2 values since this takes into account the number of parameters to fit. Let us assume that we had ten points to fit the model to. The reduced χ^2 for m_1 will be 0.5/(10 - 9) = 0.5, while the reduced χ^2 for m_2 will be $0.8/(10-2) = 0.1$. Now we can see that after taking into account the number of parameters in each fit, m_2 has the lower χ^2 and therefore we conclude that it is m_2 that is the better fit to the experimental data.

This is a simple check on the plausibility of a given model, simply compare the reduced chi-squared. We will return to other indicators of a bad fir in a model in the following sections.

9.4 Estimating Confidence Intervals

Fitting a model to a set of data generates estimates for the parameters. However since there will inevitably be errors in the experimental data this error will propagate into the parameter estimates. There will therefore be some uncertainty in the values for the parameters. For example, it is important to know whether a fitted K_m with a value of 5.0 has an uncertainty of ± 0.2 or ± 4.8. Knowing these uncertainties becomes important because they influence in turn uncertainties in the predictions made by the model.

We can describe the uncertainty using confidence limits, that is the likelihood for a parameter value to be found within a given confidence limit, for example 95% of the time. Intuitively this means if one were to repeat the same experiment many times and each time fitted the experimental data to the model, we would find that 95% of the time the fitted parameters would lie within the indicated range.

The uncertainty in a parameter p, that is the variance σ_p^2, can be estimated by propagating how each individual data point, x_i influences the parameter through the variance (σ_i^2) of the data point [9]:

$$\sigma_p^2 = \sum \left[\sigma_i^2 \left(\frac{\partial p}{\partial x_i} \right)^2 \right] \tag{9.10}$$

By evaluating the derivative, $\partial p / \partial x_i$, we find that (Details in [9], page 154) the covariance matrix can be obtained from the inverse of the Hessian, $\mathbf{H}$:

$$\text{Cov} = \mathbf{H}^{-1}$$

From which an estimate for the standard deviation in the parameters can be found on the main diagonal of the covariance matrix:

$$\sigma_{p_i} \approx \sqrt{(\mathbf{H})_{ii}^{-1}} \tag{9.11}$$

For a confidence level of 95%, it can be shown that the quoted limits $p_0 \pm \delta p$, are given by:

$$\delta p_i = \pm 1.96 \sqrt{((\mathbf{H})^{-1})_{ii} \frac{\epsilon}{N - P}} \tag{9.12}$$

If should be strongly pointed that the estimates given by equation 9.11 is an approximation and in fact studies indicate that they generally underestimate the actual confidence limits. The reason for the approximation is due to a number of assumptions made, in particular, we assumed that the experimental noise is normally distributed and that the experimentally measured data points must be independent observations. In addition we assume that the number of data points collected is sufficient to give a good random sampling of the uncertainties in the data and that the linear approximation (9.10) when deriving 9.11 holds true. For very nonlinear models and where this is unlikely to be the case. The likelihood therefore of inaccuracies in the uncertainty estimates is therefore quite likely.

Finally, we have said very little about the covariances in the Hessian matrix **H**. The confidence limits are derived from the main diagonal elements of **H**. The off diagonal contain information on the covariances, that is how a change in one parameter can influence the change in another parameter, that is the parameter estimates are not independent of each other. What this means is that parameters are correlated which in turn usually means there is insufficient experimental data (or variety of measurements)to separate the two parameters and identify them individually. We will return to this important topic in another section where we will shows examples of parameter correlation.

An alternative and possibly more trustworthy way to generate confidence limits and one that avoids many of the the problems highlighted above is the use of Monte Carlo simulations [85, 89, 94], which we will now turn to in the following section.

Determining Confidence Intervals from Monte Carlo Simulations

In the last section a description was given on how to estimate the 95% confidence limits on a set of fitted parameters. Intuitively, if we were to repeat the same experiment many times and each time fitted the experimental data to the model we would find that, 95% of the time, the fitted parameters would lie within the indicated range.

Unfortunately the approach used to estimate these confidence limits is riddled with assumptions which may or may not be defensible in many cases. An alternative approach is therefore sought. Going back to the intuitive explanation, if we **could** repeat the experiment many times and fit the data many times we could get many estimates for the parameters, each estimate slightly different due to errors in the experimental data. From the sample of fitted parameters we could then compute a standard deviation and thus obtain a confidence limit (Figure 9.8). Obviously repeating the experiment many times is just not practical but by making two reasonable assumptions we could do the same thing but by carrying out only one real experiment.

The two assumptions we wish to make are:

1. When we repeat an experiment, the underlying biology remains the

same, that is we are measuring the same thing again;

2. What ever errors there are in our measurements, the same kind of error manifests itself each time we repeat the experiment, technically the probability distribution for the errors remains the same, this could be normal, Poisson, or what ever.

If these two assumptions hold then there is nothing stopping us from creating synthetic experimental data sets if we knew the probability distribution for the errors.

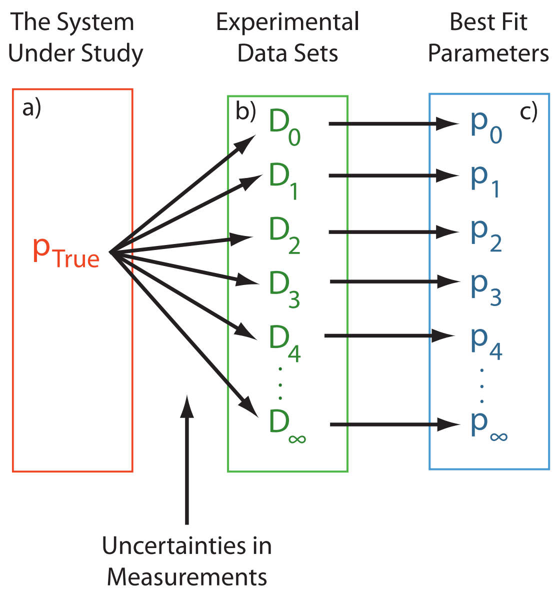


Figure 9.8 a) In the real world we assume our system has a set of "true" parameter values. b) We do experiments which give us experimental data; c) which we use to fit our model to obtain estimates for the parameters. Because each experiment is slightly different due to measurement uncertainty, we will, in turn, generate a set of different but similar fitted parameters.

We need to make one further assertion before we can continue. Let us assume that that the parameter estimates that we obtain from the optimization process are close to the true parameter values. That is let us assume that p_{True} is not far from the out fitted data, p_0. The core concept we will use is that we will generate, using a bootstrap (see next section), new synthetic data sets. The bootstrap will ensure that the new data sets have the same error distribution as the data we collected from the real and only experiment that was done. Each synthetic data set will e fitted to the model, from which we will obtain multiple estimates for the parameters. Once we have a large sample of estimated parameters we can make statements on the uncertainty of our parameter estimates (Figure 9.9). In particular, approximate confidence interval for the parameters can be obtained by using the $\frac{\alpha}{2}$ and 1-$\frac{\alpha}{2}$ sample quantiles from the Monte Carlo estimators of the parameters.

If the experimental uncertainties surrounding measured data are not known, then a proxy can be obtained by relying on the residuals (a discussion residuals is presented in section 9.6) that were produced as a result of the parameter estimation procedure. This is a common approach to take.

Bootstrap

Bootstrapping is a method to infer the statistics of a population by sampling from a sample of the population [28, 85]. It is a means of gaining information about a population when the population itself is not available. The key assumption in a bootstrap is that the sample contains enough information to reconstruct details about the population (An example of a simple bootstrap is given in the appendix).

In practice the bootstrap works as follows. Let us assume we have a sample of observations of size N from our population. We now generate new samples by selecting N random values with replacement. Since we are sampling with replacement, some of the original observations may appear more than once in the new sample sets. We repeat the sampling process until the desired number of "simulated data sets" are generated. The question is what do we sample? One possibility are the residuals that are generated from the initial fit. The residuals are the difference between the fitted data value and the corresponding experimental value, that is:

$$r_i = (y_i) \text{ observed} - (y_i) \text{ predicted}$$

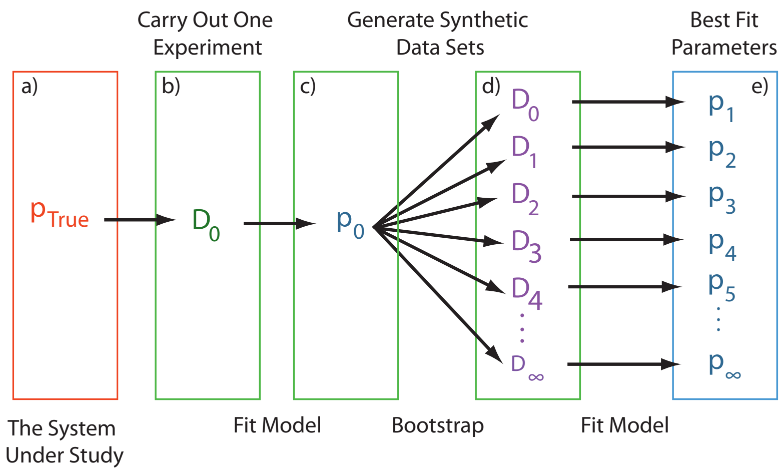


Figure 9.9 b) Generate a set of measurements from one experiment; c) Fit the data to the model to generate parameter estimates, p_0; d) Using a Bootstrap to generate synthetic data sets; e) Fit the synthetic data sets to the model and produce a sample of parameter estimate; Use the sample of parameter estimate to gauge parameter uncertainty.

To generate a synthetic data set, we will sample the residuals and add them to the predicted y_i values. For example assume that after our initial fit of the experimental data to our model the residuals are found to be: $(0.1, -0.5, 0.2)$. To sample this set we pick at random three values from the set, each time we pick a value we also return it to the set (replacement). For example, the following sets are possible samples:

$$(-0.5, 0.2, 0.2), (-0.5, 0.1, 0.2), (0.2, 0.1, 0.1), (-0.5, 0.1, -0.5)$$

Note that due to replacement it is possible to pick the same residual more than once. Once we have our sample residual sets we can now generate the synthetic data sets by adding the sets to the fitted data. That is:

$$(y_i) \text{ synthetic} = (y_i) \text{ predicted} + r_i$$

It is prudent to generate at least 500 to 1000 new synthetic data set in this way. We now take each synthetic data set in turn and fit the data to the model to generate a ‘synthetic’ estimate for the parameters. We will thus generate 500 to 1000 estimates for the model parameters. Using these parameter we can calculate different statistics, for example the standard deviation for each parameter. However, one thing that will be potentially different from the estimated statistics using the Hessian is that whereas the confidence limits from the Hessian will be symmetric, there is not guarantee that the distribution of parameter values will be symmetric. As a result it is best to compute confidence limits using percentile values, although other approaches are possible [103]. For example we could generate 95.5th and 2.5th percentile values.

In the following section we will consider some examples that illustrate some of the ideas presented here.

9.5 Case studies

Test Example

We first discuss an example with simulated data. Consider a linear chain of irreversible uni-molecular reactions, with mass-action kinetics:

$$S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_3 \rightarrow S_4 \rightarrow S_5 \tag{9.13}$$

The noisy concentration data for the six metabolites, which is displayed in Figure 9.10, was simulated with all the kinetic rate coefficients set to 2, and the initial concentration of the first substrate set to 10, and all the others to zero.

The data in Figure 9.10 has 100 points. The noise was assumed to be exponentially distributed and was added to the simulated curves, and presented to the optimizer as the data for fitting the model. The simplex method (the Levenberg-Marquardt gives similar results) to fit the parameters to the data. The parameters were the five rate constants, which were initialized to 0.1. The fit is shown in the same Figure 9.10. A Monte Carlo simulation was then run and for each such data set the parameters were optimized to fit the data. Confidence limits were obtained using 9.12 and are shown in

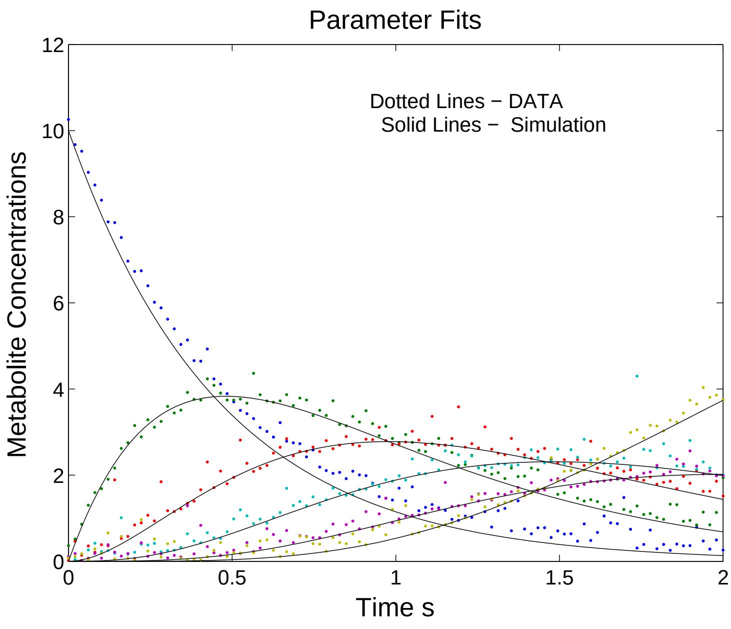


Figure 9.10 Plot of the simulated noisy time series concentration data, and the fitted curves for a linear sequence of of reactions governed by irreversible mass-action kinetics.

Table 9.3. Figure 9.11 shows the distribution of parameter estimates from the Monte Carlo simulations.

The confidence limits, can also be evaluated from the Monte Carlo generated data (Figure 9.11) by choosing limits around the mean values of the parameters, and making sure that 95%, of the points fall between them. The confidence limits constructed in this manner match the ones computed using equation 9.12. In addition the Hessian was found to be well behaved and no significant correlation was found between the parameters (9.11).

Parameter	Value
k_0	2.16 ± 0.05
k_1	1.99 ± 0.042
k_2	1.99 ± 0.083
k_3	1.96 ± 0.183
k_4	1.94 ± 0.199

Table 9.3 The table shows 95% confidence limits for the estimated parameters based on equation 9.12.

Fitting Data to HIV Proteinase

In the second example we fit rate constants to data obtained from a model for irreversible inhibition of HIV proteinase [58]. The data that was analyzed was obtained from `www.biokin.com/dynafit/index.html`, and comprised of two different time courses at different inhibitor concentrations. The parameters to be optimized were five rate constants, and the initial guess values for the parameters, and initial substrate concentrations

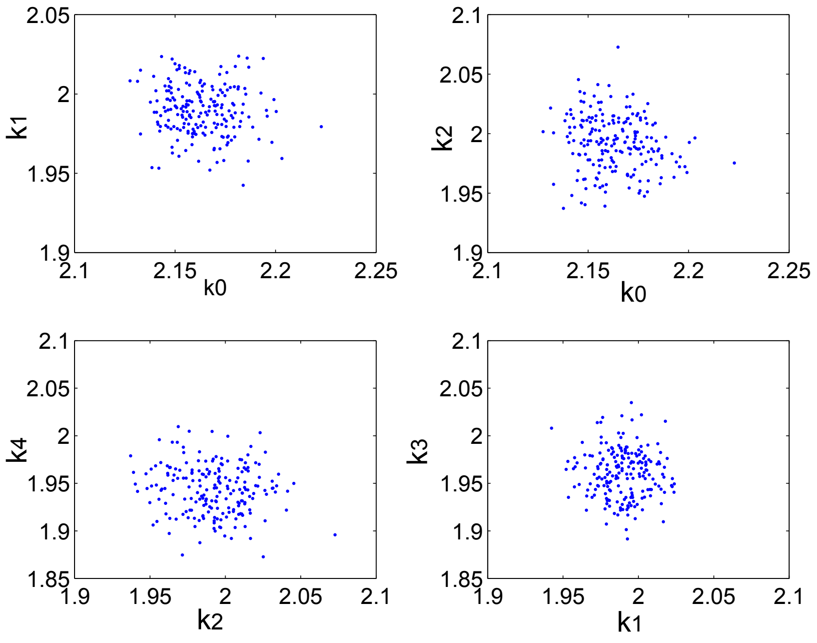


Figure 9.11 Cluster plots for the distribution of fitted parameters for the Monte carlo simulation, for various parameter combinations.

as described in [58][9]. The model is described by the following equations:

$$\begin{aligned} M + M &\underset{k_d}{\overset{k_a}{\rightleftharpoons}} E \qquad (9.14)\\ E + S &\underset{k_s}{\overset{k_{on}}{\rightleftharpoons}} E \\ SE &\overset{k_r}{\rightarrow} E \\ E + P &\underset{k_p}{\overset{k_{on}}{\rightleftharpoons}} EP \\ E + I &\underset{k_i}{\overset{k_{on}}{\rightleftharpoons}} EI \\ EI &\overset{k_{de}}{\rightarrow} EJ \end{aligned}$$

[9]In the example described above we chose the 4th and 5th data sets (Kuzmic 1996, curves D, E of Fig 1, pg 264)), and used initial values of $k_{on} = 100$, $k_d = 0.0001$, $k_a = 0.1$, $I = 0.004$, $E = 0.004$, and $S = 27$. In Kuzmic 1996, some of the species levels are also optimized, but here we are more interested in fitting the parameters

In Figure 9.12 we display the two data sets along with the fits.

Several combinations of optimizers were run, the simulated annealing and simplex took the maximum time, but of the two, the latter gave better results. Running the GA first followed by running the simplex was also successful. As can be seen in Figure 9.12, the fits seem to be quite satisfactory.

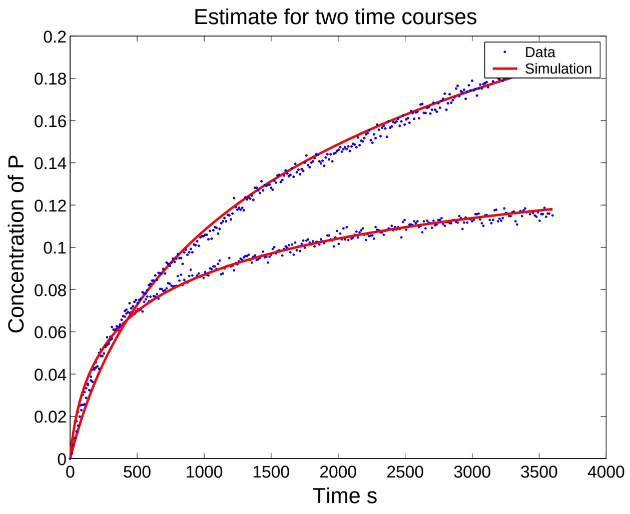


Figure 9.12 The plot showing the two time series concentration data along with the fitted curves.

Using equation 9.12 to compute the parameter confidence limits, Table 9.4 shows the confidence limits for the parameters. The confidence limits on k_r, k_p, k_i and k_{de} are substantially greater than their mean values, which is a clear indication that these numbers cannot be trusted (see [77]). Therefore a Monte Carlo simulation with a bootstrap using the residuals was carried out by generating 500 data sets, and recomputing the statistics for the parameters. Cluster plots for various combinations of parameters are shown in Figure 9.11 and reveal some interesting patterns.

Table 9.5 shows the 95% confidence limits estimated from the cluster plots.

Parameter	Value
k_s	250.9225 ± 0.69
k_r	0.199 ± 0.266
k_p	35.04 ± 109.9
k_i	0.0469 ± 7.46
k_{de}	0.2292 ± 3.76

Table 9.4 The table shows 95% confidence limits for the estimated parameters using the Hessian computed at the end of the optimization.

k_s	k_r	k_p	k_i	k_{de}
± 0.31	± 0.006	± 0.37	± 0.01	± 0.11

Table 9.5 The table shows 95% confidence limits for the estimated parameters, using Monte-Carlo simulation.

These confidence limits are much tighter than those computed using the Hessian (see Table 9.4). As indicated by the arrows, in Figure 9.13, the Monte Carlo simulations uncovered significant correlation between a number of the parameters. This implies that certain combinations of parameters could change, without having any effect on the concentrations of the species. This is also known as the observability problem. For example, consider a simple Michaelis-Menten set of reactions

$$S + E \rightleftharpoons ES \rightarrow P + E, \tag{9.15}$$

where the reversible step between the substrate and complex are k_1, k_{-1}, and the forward reaction rate between complex and product is k_2. In terms of these basic mass action reactions we derive the Michaelis-Menten rate, by assuming that after a very brief transient time the complex forms and after that it remains constant. Under these assumptions, the rate between S and P is

$$\frac{V_{max} S}{K_m + S}, \tag{9.16}$$

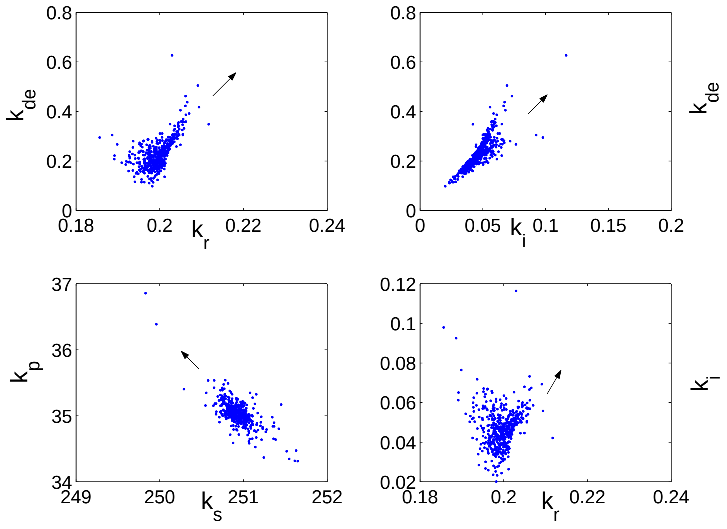


Figure 9.13 Cluster plots for the parameter distributions showing significant correlations for some combinations.

where $V_{max} = k_2 e_0$, e_0 being the enzyme concentration, and $K_m = \frac{k_1}{k_{-1}+k_2}$. Notice that if k_2 is kept constant but k_1, k_{-1} are changed such that K_m remains the same, then the net rate does not change. This is obvious, since we are only changing the final amount of complex generated, not the product. Hence presented with time series data from such a simple model, we will notice a correlation between the spread in the k_1, k_{-1} values. This is generally true for larger models, but it may not be possible to find simple combinations of these parameters which are truly independent. The important point is that the cluster plots show observability of the parameters. It is then a simple step to quantify this, by making the observation that the 2-D cluster plots are sections of the $m \times m$, (where m is the number of parameters) probability distribution of the fitted parameters. The eigenvectors of the Hessian (inverse of the covariance matrix), corresponding to the lowest eigenvalues, are directions along which, if the parameters change, then no significant change in the sums of squares, ϵ results. The eigenvalues of the Hessian are, $\simeq 665, 0.51, 0.0076, 10^{-5}, 10^{-5}$. It is interesting to study the eigenvectors corresponding to the lowest eigenvalues, they are

$$0 = \begin{bmatrix} -0.0019 & 0.0777 \\ -0.0024 & -0.0028 \\ 0.9999 & 0.0043 \\ -0.0102 & 0.9006 \\ 0.0118 & 0.4277 \end{bmatrix} \tag{9.17}$$

Notice that the first eigenvector implies that there is freedom to change k_p, and for the second eigenvector, the combination of k_i, k_{de}, which is seen as correlated in the second subplot in Figure 9.13. Calculating the Hessian is useful since its eigenstructure can often be used to study degeneracy in the model.

9.6 Analysis of Residuals

Parameter estimation procedures we discussed in the earlier sections will produce values based on minimization of the sum of the squares of the residuals. For the minimization procedure to yield best parameter estimates several interrelated assumptions must be satisfied. The first assumption is that the experimental uncertainties in the data are normally distributed with zero mean and constant variance. The second assumption is that the errors are uncorrelated. Any departure from these assumptions means that the residuals contain a structure that is not accounted for by the model. Plotting residuals is therefore an effective way to investigate how well the model fits the data and also to check if the above assumptions are satisfied or not. One simple method to check for the normality assumption is to construct a normal probability plot [76, 102]. The process requires that the residuals are ranked in increasing order such that

$$e_1 < e_2 < e_3 < ... < e_i < ... < e_n$$

where e_i is the i^{th} residual value. If we plot the cumulative probability $P_i = \frac{i-0.5}{n}$, against the residuals values, then in case of normally distributed residuals the resulting points should be in close proximity to a straight line. Any substantial deviations from the straight line indicate that the normality assumption is violated. Figure 9.14 displays normal probability plots for two sets of residuals.

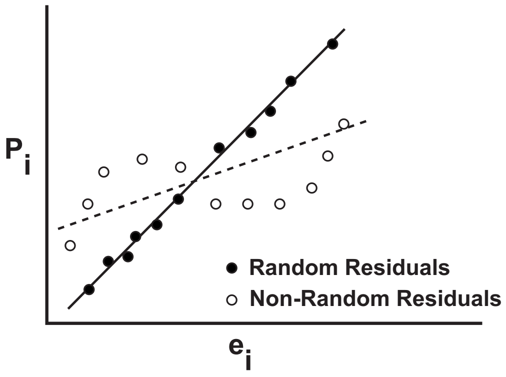


Figure 9.14 Normal probability plots for two sets of residuals.

A plot of residuals e_i versus the corresponding fitted values can be used to check the zero mean and homoscedasticity (constant variance) assumptions. Since the residuals are the deviations of the observations away from fitted values, in an ideal case we would expect the residuals to vary randomly about zero and their spread be almost the same across the plot. If the points in the plot lie on a curve around zero, rather than fluctuating randomly, it is an indication that the zero mean assumption is broken. If the residuals exhibit a pattern, i.e, increase or decrease in magnitude with the fitted values, it is an indication that the constant variance assumption is violated. A plot of residuals against fitted values may sometimes also reveal points with unusually large residuals. These points are potential outliers, that is, data points for which the model is not appropriate. The presence of outliers in the data sets may significantly influence the estimation of model parameter values, it is therefore important to identify those points and correct them (whenever possible) or delete them from the raw data sets. However, in some cases outliers may actually actually aid in improvement of our knowledge about the system under consideration. So one should do a detailed investigation before characterizing and rejecting outliers (see [4]).

Whenever the time sequence in which the data is obtained is known, it is a good idea to generate time series plot of the residuals. The purpose of this is to check there is any correlation between the residuals over time. If the residuals are independent, we expect them fluctuate in a random fashion around a base line centered at zero. Thus time series plot can validate our second assumption of uncorrelated errors.

9.7 χ^2-Goodness of Fit Test

If the standard deviation associated with each data point obtained from the experiment is known, then the variance of the fit between the model and the experimental observations can be characterized by the χ^2 statistic

$$\chi^2 = \sum_{i=1}^{N} \frac{1}{\sigma_i (N - P)} (y_i - f(x_i; p_1 \dots p_m))^2 \tag{9.18}$$

The probability distribution for χ^2 at its minimum can be derived analytically and is equivalent to chi-square distribution with $(N - P)$ degrees of freedom, where N is the number of data points, and P the number of parameters. The value of χ^2 approaches unity for a good fit between the model and data and grows larger as the fit with the data worsens. Of more interest to us is the probability Q that the chi-square will exceed a particular value of χ^2 by chance, even for a correct model. The quantity Q, can be obtained from the tabulated values in statistics books. Small values of Q indicate that either the model is wrong or that there are discrepancies in the measurement errors.

9.8 Caveats in Data Fitting

The experimental data has been collected, a model has been proposed and data fitting has confirmed that the model is able to reproduce the experimental data very well. The next question is what next? The first thing to emphasize is precisely what was implied in the last sentence. That is, the model reproduces the experimental data, and that is all it does. The real test of a model is whether it can now make new non-trivial predictions. There are many models that one could propose that might fit the data adequately, some complex, some simple.

Is it the truth? No!

Can the fitted model make non-trivial predictions?

9.9 Availability in Modeling Applications

There are a number of biochemical modeling applications that support curve fitting of differential equation models. The most popular tools in this category (in alphabetic order) include, COPASI[10], PottersWheel[11], SBSI[12], VCell[13]. COPASI in particular has an extensive set of parameter fitting algorithms. For Matlab users, PottersWheel is probably the choice although it is not entirely clear from the documentation what optimization algorithms are available. There is also a long standing set of software and parameter optimization code by Peter Kuzmic who can provide expert consultancy on parameter fitting.

9.10 Using Python to Fit Data

Python has good support for optimization and data fitting via the SciPy (`www.scipy.org`) extension. SciPy supports the optimize package[14] which in turn implements a number of optimization algorithms including the Nelder and Mean simplex approach. The script showin in listing 9.1 shows a simple example of to fit a Michaelis-Menten equation to some data.

[10]`http://www.copasi.org/`

[11]`http://www.potterswheel.de/`

[12]`http://www.sbsi.ed.ac.uk/`

[13] `http://www.vcell.org/`

[14]`http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html`

```
from scipy import *
from scipy import optimize
x = array([0, 10, 20, 50, 100, 200, 400])
y = array([0, 9, 10, 17, 18, 20, 19])

def residuals (p):
    [vmax,Km] = p
    return y - vmax*x/(Km+x)
output = optimize.leastsq (residuals, [10, 10])
```

Listing 9.1 Python Script to Fit Data

The output variable will contain the values for the fitted $V_{\max}$ and K_m which in this case is 20.745, 15.408. The y data was generate from a curve with $V_{\max} = 20$ and $K_m = 15$ with added noise to simulate experimental error.

One important point work noting, the leastsq routine expects a routine called residuals to return the differences between the data and the model. In order words there is no need to square and sum up the residuals to compute the chi-square directly. In general the residuals routine will compute the following component of the sums of squares:

$$\frac{y_i - f(x_i, p)}{\sigma}$$

We can go further and plot the results of the fit using the code in listing 9.2:

```
def peval(x, p):
    return p[0]*x/(p[1]+x)

Vmax,Km = 20,15
yTrue = Vmax*x/(Km+x)

import matplotlib.pyplot as plt
plt.plot(x, peval (x, output[0]), '--', x, y, 'o', x, yTrue,
    'r', x, residuals(output[0]), 'r^', markersize=10)
plt.title('Least-squares fit to noisy data')
plt.legend(['Fitted Curve', 'Noisy Data',
    'Underlying Function', 'Residuals'], loc=10)
```

```
plt.show()
```

Listing 9.2 Python Script to Fit Data

The Python leastsq uses a modified Levenberg-Marquardt algorithm from the minipack lmdif routine. By calling lsq with an addition argument, `output = lsq (residuals, [10, 10], full_output=1)`, it is possible to obtain the covariance matrix. To obtain confidence limits, the main diagonal elements will need to be multiplied by the reduced chi-square and and the 95% limit, 1.96 (See 9.12). However it may be easier to call curve_fit which will directly return the 68% confidence limits although there is less control over the data that can be used in the fit, for example when there are multiple time courses to fit.

leastsq doesn't directly support Monte Carlo parameter estimation, therefore this will need to be programmed in separately in Python.

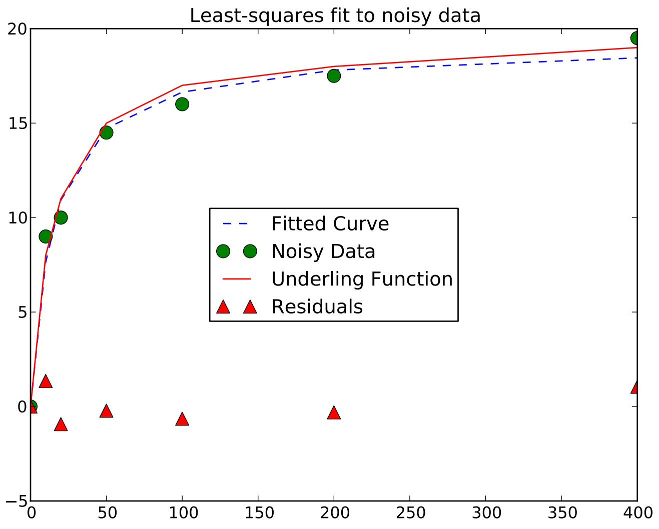


Figure 9.15 Results from Python fitting code, comparing the fitted model to the underling actual model.

Further Reading

1. Johnson ML (1994) Use of Least-Squares Techniques in Biochemistry. Methods in Enzymology, 240, 1-22.

2. Straume M, Johnson ML (1992) Monte Carlo Method for determining complete confidence probability distributions of estimated model parameters. Methods in Enzymology, 210, 117-129

3. Johnson ML, Faunt LM (1992) Parameter estimation by least-squares methods. Methods in Enzymology, 210, 1-37

Appendices

A
Kinetics in a Nutshell

Definition

Reaction Kinetics is the study of how fast chemical reactions take place, what factors influence the rate of reaction and what mechanisms are responsible.

Stoichiometric Amount

This is defined as the number of molecules a particular reactant or product takes part in a reaction. For example:

$$2A + 3B \rightarrow A + 3C$$

In the example above the stoichiometric amount for reactant A is 2, for B is 3. The stoichiometric amount for product A is 1 and for C is 3.

Depicting Reactions

$$aA + bB + \ldots \rightarrow +pP + qQ + \ldots$$

where $a, b, \ldots$ are stoichiometric amounts.

Rates of Change

The rate of change is defined as the rate of change in concentration or amount of a designated molecular species.

$$\text{Rate Of Change} = \frac{dS}{dt}$$

Stoichiometric coefficients

The **stoichiometric coefficient**, c_i, for a molecular species A_j, is the difference between the molar amount of the species on the product side and the molar amount of the species on the reactant side.

$$c_i = \text{Molar Amount of Product} - \text{Molar Amount of Reactant}$$

In the reaction, $2A \longrightarrow B$, the molar amount of A on the product side is zero while on the reactant size it is two. Therefore the stoichiometric coefficient of A is given by $0-2 = -2$. In many cases a particular species will only occur on the reactant or product side and it is not to common to find situations where a species occurs simultaneously as a product and a reactant. As a result, reactant stoichiometric coefficients tend to be **negative** while product stoichiometric coefficients tend to be **positive**.

Reaction Rates

The reaction rate, often denoted by the symbol v, is measured with respect to a given molecular species normalized by the species' stoichiometric coefficient. This definition ensures that no matter which molecular species in a reaction is measures, the reaction rate is uniquely defined for that reaction. More formally the reaction rate for the given reaction:

$$aA + bB + \ldots \rightarrow pP + qQ + \ldots$$

$$v = \frac{1}{c_a}\frac{dA}{dt} = -\frac{1}{c_b}\frac{dB}{dt} \ldots = \frac{1}{c_p}\frac{dP}{dt} = \frac{1}{c_q}\frac{dQ}{dt} \ldots$$

where c_x are the stoichiometric coefficients. Alternative we can write the rate of change in terms of the reaction rate as follows:

$$\frac{dA}{dt} = c_a v \tag{A.1}$$

Elementary mass-action kinetics

An elementary reaction is one that cannot be broken down into simpler reactions. Such reaction will often have simple kinetics called mass-action kinetics. For a reaction of the form

$$aA + bB + \ldots \rightarrow +pP + qQ + \ldots$$

the mass-action kinetic rate law is given by:

$$v = k_1 A^a B^b \ldots - k_2 P^p Q^q \ldots$$

where k_1 and k_2 are the forward and reverse rate constants respectively.

Chemical Equilibrium

In principle all reactions are reversible, meaning transformations can occur from reactant to product or product to reactant. The net rate of a reversible reaction is the difference between the forward and reverse rates. At chemical equilibrium the forward and reverse rates are equal. Chemical equilibrium is then given by:

$$\frac{B}{A} = K_{eq} \tag{A.2}$$

This ratio has special significance and is called the **equilibrium constant**, denoted by K_{eq}. The equilibrium constant is also related to the ratio of the rate constants, k_1/k_2. For a general reversible reaction such as:

$$aA + bB + \ldots \rightleftharpoons pP + qQ + \ldots$$

and using arguments similar to those described above, the ratio of the rate constants can be easily shown to be:

$$K_{eq} = \frac{P^p Q^q \ldots}{A^a B^b \ldots} = \frac{k_1}{k_2} \tag{A.3}$$

where the exponents are the stoichiometric **amounts** for each species.

Mass-action and Disequilibrium Ratio

Although in closed systems, reactions will tend to equilibrium, reactions occurring in living cells are generally out of equilibrium and the ratio of the products to the reactants *in vivo* is then called by the **mass-action ratio**, Γ. The ratio of the mass-action ratio to the equilibrium constant is often called the **disequilibrium ratio**:

$$\rho = \frac{\Gamma}{K_{eq}} \tag{A.4}$$

At equilibrium, the mass-action ratio will be equal to the equilibrium constant and $\rho = 1$. If the reaction is away from equilibrium ($B/A < K_{eq}$) then $\rho < 1$.

For a simple unimolecular reaction it was shown previously that the equilibrium ratio of product to reactant, B/A, is equal to the ratio of the forward and reverse rate constants. Substituting this into the disequilibrium ratio gives:

$$\rho = \Gamma \frac{k_2}{k_1} = \frac{B}{A}\frac{k_2}{k_1}$$

Therefore

$$\rho = \frac{v_r}{v_f} \tag{A.5}$$

That is the disequilibrium ratio is the ratio of the reverse and forward rates. If $\rho < 1$, then the net reaction must in the direction of product formation. If ρ is zero then the reaction is as out of equilibrium as possible with no product present.

Modified Mass-Action Rate Laws

A typical reversible mass-action rate law will require both the forward and the reverse rate constants to be fully defined. Often however, only one rate constant may be known. In these circumstances it is possible to express the reverse rate constant in terms of the equilibrium constant.

For example, given the simple unimolecular reaction, $A \rightleftharpoons B$. it is possible to derive the following:

$$\begin{aligned} v &= k_1 A - k_2 B \\ v &= k_1 A \left(1 - \frac{k_2 B}{k_1 A}\right) \\ \text{Since } K_{eq} &= \frac{k_1}{k_2} \\ v &= k_1 A \left(1 - \frac{\Gamma}{K_{eq}}\right) \end{aligned} \tag{A.6}$$

where Γ is the mass-action ratio. This can be generalized to an arbitrary mass-action reaction to give:

$$v = k_1 A^a B^b \ldots \left(1 - \frac{\Gamma}{K_{eq}}\right) = k_1 A^a B^b \ldots (1 - \rho)$$

where $A^a B^b \ldots$ represents the product of all reactant species, a and b are the **corresponding** stoichiometric amounts, and ρ is the disequilibrium ratio. For example, for the reaction:

$$2A + B \longrightarrow C + 2D$$

where k_1 is the forward rate constant, the modified reversible rate law is:

$$v = k_1 A^2 B \, (1 - \rho)$$

The modified formulation demonstrates how a rate expression can be divided up into functional parts that include both kinetic and thermodynamic components [45]. The kinetic component is represented by the term $k_1 A^a B^b \ldots$ while the thermodynamic component is represented by the expression $1 - \rho$.

We can also derive the modified rate law in the following way. Given the net rate of reaction $v = v_f - v_r$, we can write this expression in the following way:

$$v = v_f \left(1 - \frac{v_r}{v_f}\right)$$

That is:

$$v = v_f (1 - \rho)$$

Further Reading

1. Sauro HM (2012) Enzyme Kinetics for Systems Biology. 2nd Edition, Ambrosius Publishing ISBN: 978-0982477335

B
Enzyme Kinetics in a Nutshell

Enzymes

Enzymes are protein molecules that can accelerate a chemical reaction with changing the equilibrium constant of the reaction of themselves.

Enzyne Kinetics

Enzyme kinetics is a branch of science that deals with the many factors that can affect the rate of an enzyme-catalysed reaction. The most important factors include the concentration of enzyme, reactants, products, and the concentration of any modifiers such as specific activators, inhibitors, pH, ionic strength, and temperature. When the action of these factors is studied, we can deduce the kinetic mechanism of the reaction. That is, the order in which substrates and products bind and unbind and the mechanism by which modifiers alter the reaction rate.

Michaelis-Menten Kinetics

The standard model for enzyme action, describes the binding of free enzyme to the reactant forming an **enzyme-reactant complex**. This complex undergoes a transformation, releasing product and free enzyme. The free enzyme is then available for another round of binding to new reactant.

$$E + S \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} ES \xrightarrow{k_2} E + P \tag{B.1}$$

where k_1, k_{-1} and k_2 are rate constants, S is substrate, P is product, E is the free enzyme, and ES the enzyme-substrate complex.

By either assuming rapid equilibrium between enzyme, substrate and the substrate complex, or assuming a steady state condition on the enzyme substrate complex, an aggregate rate law, often called the Michaelis-Menten equation in the case the rapid equilibrium assumption or the Briggs-Haldane equation when using the steady state assumption is given by:

$$v = \frac{Vm\ S}{K_m + S} \tag{B.2}$$

where V_m is the maximal velocity and KM_m the substrate concentration that yield have the maximum velocity.

Product Inhibition

Reversible Rate laws

An alternative and more realistic model is the reversible form:

$$E + S \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} ES \underset{k_{-2}}{\overset{k_2}{\rightleftharpoons}} E + P \tag{B.3}$$

The aggregate rate law for the reversible form of the mechanism can also be derived and is given by:

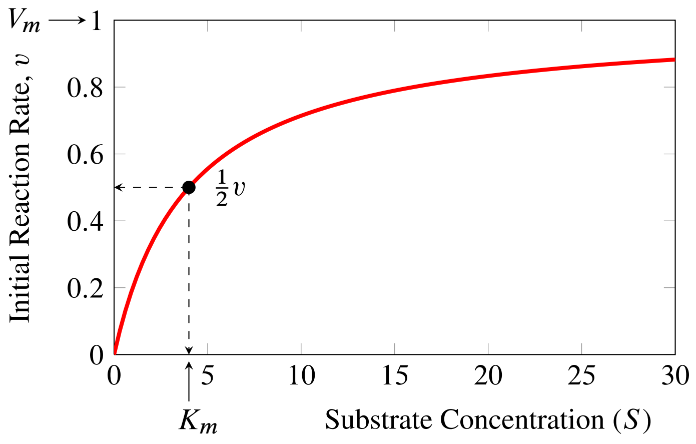


Figure B.1 Relationship between the initial rate of reaction and substrate concentration for a simple Michaelis-Menten rate law. The reaction rate reaches a limiting value called the V_m. K_m is set to 4.0 and V_m to 1.0. The K_m value is the substrate concentration that gives half the maximal rate.

$$v = \frac{V_f \; S/K_S - V_r \; P/K_P}{1 + S/K_S + P/K_P} \tag{B.4}$$

Haldane Relationship

For the reversible enzyme kinetic law there is an important relationship:

$$K_{eq} = \frac{P_{eq}}{S_{eq}} = \frac{V_f \; K_P}{V_r \; K_S} \tag{B.5}$$

and shows that the four kinetic constants, V_f, V_r, K_P and K_S are not independent. Haldane relationships can be used to eliminate one of the kinetic constants by substituting the equilibrium constant in its place. This is useful because equilibrium constants tend to be known compared to kinetic constants. By incorporating the Haldane relationship we can eliminate the reverse maximal velocity (V_r) to yield the equation:

$$v = \frac{V_f/K_S(S - P/K_{eq})}{1 + S/K_S + P/K_P} \tag{B.6}$$

Separating out the terms makes it easier to see that the above equation can be partitioned into a number of distinct terms:

$$v = V_f \cdot (1 - \Gamma/K_{eq}) \cdot \frac{S/K_s}{1 + S/K_S + P/K_P} \tag{B.7}$$

where $\Gamma = P/S$. The first term, V_f is the maximal velocity; the second term, $(1 - \Gamma/K_{eq})$ indicates the direction of the reaction according to thermodynamic considerations and the last terms refers to the fractional saturation with respect to substrate. We thus have a maximal velocity, a thermodynamic term and a saturation term. We will see this breakdown into distinct terms repeatedly as we consider other enzyme kinetic rate laws.

Competitive Inhibition

There are many molecules capable of slowing down or speeding up the rate of enzyme catalyzed reactions. Such molecules are called enzyme inhibitors and activators. One common type of inhibition, called **competitive inhibition**, occurs when the inhibitor is structurally similar to the substrate so that it competes for the active site by forming a dead-end complex.

The kinetic mechanism for a pure competitive inhibitor is shown in Figure B.2(a), where I is the inhibitor and EI the enzyme inhibitor complex. If the substrate concentration is increased, it is possible for the substrate to eventually out compete the inhibitor. For this reason the inhibitor alters the enzyme's apparent K_m but not the V_m.

$$\begin{aligned} v &= \frac{V_m\, S}{S + K_m\left(1 + \dfrac{I}{K_i}\right)} \\ &= \frac{V_m\, S/K_m}{1 + S/K_m + I/K_i} \end{aligned} \tag{B.8}$$

a) Competitive Inhibition

$$\begin{tikzcd} E \arrow[r, shift left] \arrow[d, shift left] & ES \arrow[l, shift left] \arrow[r] & E + P \\ EI \arrow[u, shift left] & & \end{tikzcd}$$

b) Uncompetitive Inhibition

$$\begin{tikzcd} E \arrow[r, shift left] & ES \arrow[l, shift left] \arrow[r] \arrow[d, shift left] & E + P \\ & ESI \arrow[u, shift left] & \end{tikzcd}$$

Figure B.2 Competitive and uncompetitive inhibition. P is the concentration of product, E is the free enzyme, ES the enzyme-substrate complex, and ESI the enzyme-substrate-inhibitor complex.

At $I = 0$, the competitive inhibition equation reduces to the normal irreversible Michaelis-Menten equation. Note that the term $K_m(1 + I/K_i)$ in the first equation more clearly shows the impact of the inhibitor, I, on the K_m. The inhibitor has no effect on the V_m.

The reversible form of the competitive rate law can be derived from equation (**??**) by setting $a \gg 1$ and $b = 0$ and is shown below:

$$v = \frac{\dfrac{V_m}{K_s}\left(S - \dfrac{P}{K_{\text{eq}}}\right)}{1 + \dfrac{S}{K_s} + \dfrac{P}{K_p} + \dfrac{I}{K_i}} \tag{B.9}$$

where V_m is the forward maximal velocity, and K_s and K_p are the substrate and product half saturation constants.

Sometimes reactions appear irreversible, that is no discernable reverse rate is detected, and yet the forward reaction is influenced by the accumulation of product. This effect is caused by the product competing with substrate for binding to the active site and is often called **product inhibition**. Given that product inhibition is a type of competitive inhibition we will briefly discuss it here. An important industrial example of this is the conversion of lactose to galactose by the enzyme β−galactosidase where galactose competes with lactose, slowing the forward rate [37].

To describe simple product inhibition with rate irreversibility, we can set the P/K_{eq} term in the reversible Michaelis-Menten rate law (B.4) to zero. This yields:

$$v = \frac{V_m S}{S + K_m \left(1 + \dfrac{P}{K_p}\right)} \tag{B.10}$$

It is not surprising to discover that equation (B.10) has exactly the same form as the equation for competitive inhibition (B.8). Figure **??** shows how the reaction rate responds to increasing product concentration at a fixed substrate concentration. As the product increases, it out competes the substrate and therefore slows down the reaction rate.

We can also derive the equation by using the following mechanism and the rapid equilibrium assumption:

$$E + S \rightleftharpoons ES \longrightarrow EP \rightleftharpoons E + P \tag{B.11}$$

where the reaction rate, $v \propto ES$.

Cooperativity

Many proteins are known to be oligomeric, that is they are composed of more than one identical protein subunit where each subunit has one or more binding sites. Often the individual subunits are identical.

If the binding of a ligand (a small molecule that binds to a larger macromolecule) to one site alters the affinity at other sites on the same oligomer then this is called **cooperativity**. If ligand binding increases the affinity of subsequent binding events, it is termed **positive cooperativity** whereas if the affinity decreases then it is termed **negative cooperativity**. One characteristic of positive cooperativity is that it results in a sigmoidal response instead of the usual hyperbolic response.

The simplest equation that displays sigmoid like behavior is the Hill equation:

$$v = \frac{Vm\, S^n}{K_d + S^n} \tag{B.12}$$

One striking feature of many oligomeric proteins is the way individual monomers are physically arranged. Often one will find at least one axis of symmetry. The individual protein monomers are not arranged in a haphazard fashion. This level of symmetry may imply that the gradual change in the binding constants as ligands bind, as suggested by the Adair model, might be physically implausible. Instead one might envisage transitions to an alternative binding state that occurs within the entire oligomer complex. The original authors laid out the following criteria for the MWC model:

1. The protein is an oligomer.
2. Oligomers can exist in two states: R (relaxed) and T (tense). In each state, symmetry is preserved and all subunits must be in the same state for a given R or T state.
3. The R state has a higher ligand affinity than the T state.
4. The T state predominates in the absence of ligand S.
5. The ligand binding microscopic association constants are all identical, this is in complete contrast to the Adair model

Given these criteria, the MWC model assumes that an oligomeric enzyme may exist in two conformations, designated T (tensed, square) and R (relaxed, circle) with an equilibrium between the two states with equilibrium constant, $L = T/R$, also called the allosteric constant. If the binding constants of ligand to the two states are different, then the distribution of the R and T forms can be displaced either towards one form or the other. By this mechanism, the enzyme displays sigmoid behavior. A minimal example of this model is shown in Figure B.3.

In the **exclusive model** (Figure B.3), the ligand can only bind to the relaxed form (circle). The mechanism that generates sigmoidicity in this model works as follows. When ligand binds to the relaxed form it displaces the equilibrium from the tense form to the relaxed form. In doing so, additional ligand binding sites are made available. Thus one ligand

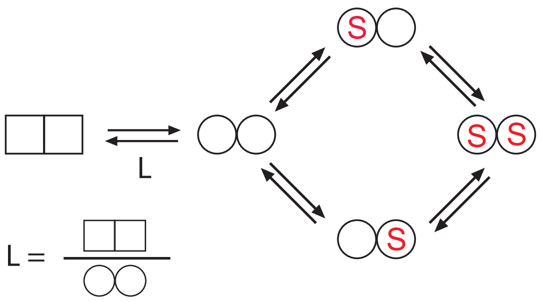


Figure B.3 A minimal MWC model, also known as the exclusive model, showing alternative microscopic states in the circle (relaxed) form. L is called the allosteric constant. The square form is called the tense state.

binding may generate four or more new binding sites. Eventually there are no more tense states remaining at which point the system is saturated with ligand. The overall binding curve will therefore be sigmoidal and will show positive cooperativity. Given the nature of this model, it is not possible to generate negative cooperativity. By assuming equilibrium between the various states it is possible to derive an aggregate equation for the dimer case of the exclusive MWC model:

$$v = V_m \frac{\frac{S}{k_R}\left(1+\frac{S}{k_R}\right)}{\left(1+\frac{S}{k_R}\right)^2 + L}$$

This also generalizes to n subunits as follows:

$$Y = \frac{\frac{S}{k_R}\left(1+\frac{S}{k_R}\right)^{n-1}}{\left(1+\frac{S}{k_R}\right)^n + L} \tag{B.13}$$

For more generalized a reversible rate laws the exhibit sigmoid behavior the reversible Hill equation is a good option to use.

Invoking the rapid-equilibrium assumption we can write the various complexes in terms of equilibrium constants to give:

$$v = \frac{V_f \alpha \, (1-\rho)\,(\alpha+\pi)}{1+(\alpha+\pi)^2}$$

where $\rho = \Gamma / K_{eq}$. For an enzyme with h (using the authors original notation) binding sites, the general form of the reversible Hill equation is given by:

$$v = \frac{V_f \alpha \; (1-\rho) \; (\alpha+\pi)^{h-1}}{1+(\alpha+\pi)^h} \tag{B.14}$$

Allostery

An allosteric effect is where the activity of an enzyme or other protein is affected by the binding of an effector molecule at a site on the protein's surface other than the active site. The MWC model described previously can be easily modified to accomodate allosteric action.

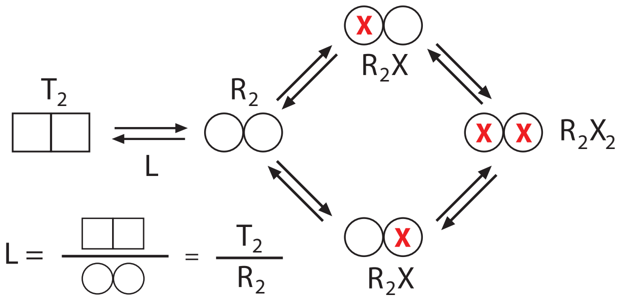


Figure B.4 Exclusive MWC model based on a dimer showing alternative microscopic states in the form of T and R states. The model is exclusive because the ligand, X, only binds to the R form.

The key to including allosteric effectors is the equilibrium between the tense (T) and relaxed (R) states (See Figure B.4). To influence the sigmoid curve, an allosteric effector need only displace the equilibrium between the tense and relaxed forms. For example, to behave as an activator, an allosteric effector needs to preferentially bind to the R form and shift the equilibrium away from the less active T form. An allosteric inhibitor would do the opposite, that is bind preferentially to the T form so that the equilibrium shifts towards the less active T form. In both cases the V_m of the enzyme is unaffected.

The net result of this is to modify the normal MWC aggregate rate law to the following if the effector is an inhibitor:

$$v = V_m \frac{\alpha\,(1+\alpha)^{n-1}}{(1+\alpha)^n + L(1+\beta)^n} \tag{B.15}$$

where $\alpha = S/K_s$ and $\beta = I/K_I$. K_s and K_I are kinetic constants related to each ligand. A MWC model that is regulated by other an inhibitor and an activator is represented by:

$$v = V_m \frac{\alpha\,(1+\alpha)^{n-1}}{(1+\alpha)^n + L\dfrac{(1+\beta)^n}{(1+\gamma)^n}}$$

There are also reversible forms of the allosteric MWC model but is fairly complex. Instead it is possible to modify the reversible Hill rate law to include allosteric ligands.

$$v = \frac{V_f\alpha\left(1 - \dfrac{\Gamma}{K_{eq}}\right)(\alpha+\pi)^{h-1}}{\dfrac{1+\mu^h}{1+\sigma\mu^h} + (\alpha+\pi)^h} \tag{B.16}$$

where

$\sigma < 1$	inhibitor
$\sigma > 1$	activator

Further Reading

1. Sauro HM (2012) Enzyme Kinetics for Systems Biology. 2nd Edition, Ambrosius Publishing ISBN: 978-0982477335

C
Math Fundamentals

C.1 Notation

Sum and Product:

$$a_1 + a_2 + a_3 + \ldots + a_n = \sum_{i=1}^{n} a_i$$

$$a_1 \times a_2 \times a_3 \times \ldots \times a_n = \prod_{i=1}^{n} a_i$$

Vectors and Matrices:

Bold lower case letters indicate vectors, for example: $\boldsymbol{v}$, $\mathbf{s}$

Bold upper case letters indicate matrices, for example: $\mathbf{N}$, $\boldsymbol{X}$.

Derivatives:

One the left is Leibniz's notation and on the right Lagrange's notation:

$$\frac{df}{dx} \equiv f'(x)$$

$$\frac{d^2 f}{dx^2} \equiv f''(x)$$

$$\frac{d^n f}{dx^n} \equiv f^{(n)}(x)$$

C.2 Short Table of Derivatives

$$\frac{d}{dx}[c] = 0 \qquad \frac{d}{dx}[x] = 1$$

$$\frac{d}{dx}[cu] = c\,\frac{du}{dx} \qquad \frac{d}{dx}[u+v] = \frac{du}{dx} + \frac{dv}{dx}$$

$$\frac{d}{dx}[uv] = u\,\frac{dv}{dx} + v\,\frac{du}{dx} \qquad \frac{d}{dx}[u/v)] = \frac{v\,\frac{du}{dx} - u\,\frac{dv}{dx}}{v^2}$$

$$\frac{d}{dx}[u^n] = nu^{n-1}\frac{du}{dx} \qquad \frac{d}{dx}[f(u)] = \frac{df}{du}f(u)\,\frac{du}{dx}$$

$$\frac{d}{dx}[\ln u] = \frac{1}{u}\,\frac{du}{dx} \qquad \frac{de^u}{dx} = e^u\frac{du}{dx}$$

$$\frac{d}{dx}[\sin(u)] = \cos(u)\,\frac{du}{dx} \qquad \frac{d}{dx}[\cos(u)] = -\sin(u)\,\frac{du}{dx}$$

C.3 Logarithms

$$\begin{aligned}
\log(AB) &= \log(A) + \log(B) \\
\log(A/B) &= \log(A) - \log(B) \\
\log(A^n) &= n\ \log(A) \\
x^n \times x^m &= x^{n+m} \\
\frac{x^n}{n^m} &= x^{n-m} \\
(x^n)^m &= x^{n\times m}
\end{aligned}$$

C.4 Partial Derivatives

If the value of a given function depends on two variables then we write this function in the form:

$$u = f(x, y)$$

If it is possible to change x without affecting y then x and y are called independent variables. The rate of change of u with respect to x when x varies but y remains constant is called the **partial derivative** of u with respect to x. Partial derivatives are denoted using the partial symbol, ∂. That is the partial derivative of u with respect to x is:

$$\frac{\partial u}{\partial x}$$

Likewise the partial derivative of u with respect to y is given by:

$$\frac{\partial u}{\partial y}$$

To find a partial derivative, we simply differentiate with respect to the variable of interest while treating the remaining variables as constants. For example, the reaction rate for a given reaction is $v = k_1 S - k_2 P$, where

S is the reactant, P the product and k_1 and k_2 the rate constants. In a controlled environment we should in principle be able to change S and P independently. Therefore we can write down the partial derivatives of the reaction rate with respect to S and P as follows:

$$\frac{\partial v}{\partial S} = k_1$$
$$\frac{\partial v}{\partial P} = -k_2$$

In order to indicate what variables are kept constant in the partial derivative the following notation is sometimes used, particulary in thermodynamics:

$$\left(\frac{\partial v}{\partial S}\right)_P = k_1$$
$$\left(\frac{\partial v}{\partial P}\right)_S = -k_2$$

Or for function with many variables, $x, y, z, \ldots$, the notation would extend to:

$$\left(\frac{\partial u}{\partial x}\right)_{y,z,\ldots}$$

Like derivatives, partial derivatives are defined in terms of a limit. For example the partial derivatives for the function, $f(x, y)$ are defined as:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{h \to 0} \frac{f(x+h,y) - f(x,y)}{h}$$
$$\frac{\partial f(x,y)}{\partial y} = \lim_{h \to 0} \frac{f(x,y+h) - f(x,y)}{h}$$

The graphical interpretation of a partial derivative, $\partial f(x, y)/\partial x$ is that it represents the slope of the function, $f(x, y)$ in the x direction.

C.5 Differential Equations

Differential equations are equations that contain derivatives. For example, the following is a differential equation:

$$\frac{dy}{dx} + y^2 = 0$$

An **ordinary differential equation** is where the derivatives are functions of the same variable. For example, the following equations are ordinary differential equations:

$$\begin{aligned}\frac{dy}{dx} &= ay\\ \frac{dy}{dx} &= 2x + 3y - 8\\ \frac{d^2y}{dx^2} - x\frac{du}{dx} &= 0\end{aligned}$$

A differential equation expressed in terms of the first derivative (dy/dx) is called a first-order differential equation. A differential equation that is expression in terms of second order derivatives (d^2y/dx^2) is called a second-order differential equation. When solving differential equations the objective is to find the function $y(x)$ such that when differentiated gives the original differential equation. For example the solution to:

$$\frac{dy}{dx} = ay$$

is

$$y = y_o e^{ax} \tag{C.1}$$

If we differentiate the solution (C.1) we get back the original differential equation.

Differential equations are used very often to model physical systems where they describe the rate of change of some variable with respect to time, t. The reason why they are used is because we may not explicitly know the solution $y(t)$ but we will often know the rate of change the variable has at any given moment in time, dy/dt. This means we can at least obtain a numerical solution to $y(t)$ even if the analytical solution is unobtainable.

Differential equations can be further classified as autonomous or non-autonomous. Autonomous differential equations are the most common in biochemical models. These equations do not depend on time, that is the right-hand side of the differential equation has no terms relating explicitly to time. For example equation C.2 is autonomous equation C.3 is non-autonomous:

$$\frac{dx}{dt} = x^2 + 10 \tag{C.2}$$

$$\frac{dx}{dt} = x^2 + t - 5 \tag{C.3}$$

A **partial differential equation** is one where the derivatives are functions of more than one derivative. For example the equation is a partial differential equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{\partial p}{\partial x}$$

Note the use of the partial d (∂) in the partial differential equation to indicate that the function u is differentiated with respect to more than one variable. The partial derivative also indicates than when a derivative is made, other other constants are assumed to be held constant.

C.6 Taylor Series

Expressions like $1+2x+6x^2$ and $2+4x+x^2-3x^3$ that consist of the sum of a number of terms raised to a positive power are called polynomials. The only operations allowed in a polynomial are addition, subtraction, multiplication and non-negative integer powers. One of the simplest polynomials is the straight line, $y = a + bx$, termed a polynomial of first degree. The coefficients, a and b can be chosen so that the line will pass through any two points. That is we can express any straight line using $y = a + bx$. Similarly for a polynomial of second degree, $y = a+bx+cx^2$, a parabola, we can choose the constants, a, b and c so that the curve can pass through any three points.

It follows that we can find a polynomial equation of n^{th} degree that will pass through any $n + 1$ points. If the polynomial has an infinite number of terms, then we can imagine that the polynomial can be made to follow any function, $f(x)$ by suitable adjustment of the polynomial coefficients. Although this statement may not always be true, in many cases it is which makes the polynomial series very useful.

A polynomial of infinite degree is called a polynomial series:

$$f(x) = c_o + c_1 x + c_2 x^2 + c_3 x^3 + \ldots$$

The question is, how can we find the polynomial series that will represent a given function, for example $\sin(x)$? To answer this we have to determine the constants, c_o, c_1 etc. in the polynomial equation. Let us assume that we wish to know the value of $\sin(x)$ at $x = 0$ using a polynomial series. At $x = 0$, all terms vanish except for c_o, therefore at $x = 0$:

$$f(0) = c_o$$

We can therefore interpret the first constant, c_o as the value of the function at $x = 0$. What about c_1? Let us take the derivative of the series, that is:

$$f'(x) = c_1 + 2c_2 x + 3c_3 x^2 + \ldots$$

If we set $x = 0$, we find that:

$$f'(0) = c_1$$

That is the second constant, c_1, in the polynomial series is the first derivative of the function. If we take the second derivative we can also show that at $x = 0$, $f''(x0) = 2c_2$, that is $c_2 = f''(0)/2$. For the third derivative we can show that $f'''(0) = 3(2)c_3$, that is $c_3 = f'''(0)/(3!)$. This pattern continues for the remaining terms in the polynomial so that we can now write:

$$f(x) = f(0) + f'(0)x + \frac{f''(0)}{2!}x^2 + \frac{f'''(0)}{3!}x^3 + \ldots$$

Function	Second order approximation
$\frac{1}{1+x}$	$1 + x + x^2$
$\sqrt{1+x}$	$1 + \frac{x}{2} + \frac{x^2}{8}$
$\sin(x)$	x

Table C.1 Examples of common approximations

This series is called the **Maclaurin series** for the function, $f(x)$. It approximates the function around the specific value of $x = 0$. To illustrate the use of the Maclaurin series consider expanding $\sin(x)$ around $x = 0$. $f(0)$ will equal $\sin(0) = 0$. $f'(0) = cos(0) = 1$ and so on. We can therefore write the series as:

$$\sin(x) = 0 + 1x + 0 - \frac{1}{3!}x^3 + 0 + \frac{1}{5!}x^5 - \ldots$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$$

What if we wanted to approximate a function about an arbitrary value of x? To do this we would use the Taylor series which is a generalization of the Maclaurin series. The **Taylor series** is defined by:

$$f(x) = f(x_o) + \frac{\partial f}{\partial x_o}(x - x_o) + \frac{1}{2!}\frac{\partial^2 f}{\partial x_o^2}(x - x_o)^2 + \ldots + \frac{1}{n!}\frac{\partial^n f}{\partial x_o^n}(x - x_o)^n + \ldots \quad \text{(C.4)}$$

where the approximation is now centered on x_o. If we set x_o equal to zero we will obtain the Maclaurin series.

C.7 Total Derivative

Consider the function:

$$f(t) = f(x(t), y(t))$$

The derivative of $f(t)$ with respect to t, is given by the chain rule:

$$\frac{df}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

Note the use of partial derivatives. This equation is often abbreviated to:

$$df = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy$$

where it is called the **total derivative**. Often the variable, t is not specified in the total derivative. Operationally the total derivative computes the change in f, given small changes in x and y.

C.8 Eigenvalues and Eigenvectors

A square matrix such as $\boldsymbol{A}$ can be use to transform a given vector, $\boldsymbol{v}$ in specific ways. For example, if the matrix $\boldsymbol{A}$ is:

$$\begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

then the result of multiplying $\boldsymbol{A}$ into $\boldsymbol{v}$ will yield a vector that is similar to $\boldsymbol{v}$ but where the first element is scaled by 2 and the second element by 4.

For an arbitrary square matrix, if it is possible to find a vector $\boldsymbol{v}$ such than when we multiple the vector by $\boldsymbol{A}$ we get a scaled version of $\boldsymbol{v}$, then we call the vector $\boldsymbol{v}$ the **eigenvector** of $\boldsymbol{A}$ and the scaling value, the **eigenvalue** of $\boldsymbol{A}$. For a matrix of dimension n, there will be at most n eigenvalues and n eigenvectors. In the case of the simple example above the eigenvalues must be 2 and 4 respectively while the two eigenvector will be:

$$\begin{bmatrix} \alpha \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ \alpha \end{bmatrix}$$

The definition of an eigenvector and eigenvalue is often given in the form:

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}$$

We can rearrange this equation as follows:

$$\begin{aligned}\boldsymbol{A}\boldsymbol{v} &= \lambda\boldsymbol{I}\boldsymbol{v}\\ \boldsymbol{A}\boldsymbol{v} - \lambda\boldsymbol{I}\boldsymbol{v} &= 0\\ (\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{v} &= 0\end{aligned}$$

From linear algebra we know that there will be non-zero solutions to $(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{v} = 0$ if $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$. We can use this observation to compute the eigenvalues and eigenvectors of a matrix. For example consider the matrix:

$$\begin{bmatrix}3 & 6\\ 1 & 4\end{bmatrix}$$

Computing $\boldsymbol{A} - \lambda\boldsymbol{I}$ yields:

$$\begin{aligned}\boldsymbol{A} - \lambda\boldsymbol{I} &= \begin{bmatrix}3-\lambda & 6\\ 1 & 4-\lambda\end{bmatrix}\\ \det(\boldsymbol{A} - \lambda\boldsymbol{I}) &= (3-\lambda)(4-\lambda) - 6\\ &= \lambda^2 - 7\lambda + 6\\ &= (\lambda - 6)(\lambda - 1)\end{aligned}$$

The eigenvalues are therefore 6 and 1. With two eigenvalues there will be two eigenvectors. First we consider $\lambda = 6$.

$$\begin{aligned}(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{v} &= 0\\ \left(\begin{bmatrix}3 & 6\\ 1 & 4\end{bmatrix} - \begin{bmatrix}6 & 0\\ 0 & 6\end{bmatrix}\right)\boldsymbol{v} &= 0\\ \begin{bmatrix}-3 & 6\\ 1 & -2\end{bmatrix}\boldsymbol{v} &= 0\end{aligned}$$

By inspection we can see that the eigenvector is:

$$\begin{bmatrix}2\\ 1\end{bmatrix}$$

satisfied this equation. Likewise we can do the same for the other eigenvalue, $\lambda = 1$ where the corresponding eigenvector is found to be:

$$\begin{bmatrix} -3 \\ 1 \end{bmatrix}$$

Further Reading

1. Smail LL (1953) Analytical Geometry and Calculus. Appleton-Century-Crofts ISBN: 978-0982477311

D Statistics Reminder

D.1 Mean

The mean is the sum of values divided by the number of values:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

The mean is not necessarily the middle value but depends on the skewness of the values. The central value is called the **median**.

D.2 Deviation

A measure of deviation of a variable y form its mean is called the **standard deviation**, denoted by σ. A related measure, σ^2, is called the **variance**. Consider the mean of a set of numbers, x_i to be μ. We can compute the deviation each x_i is from the mean as:

$$d_i = x_i - \mu$$

If we take the square of the deviations and compute the average we obtain the variance:

$$\sigma^2 = \frac{1}{N} \sum (x_i = \mu)^2$$

D.3 Standard Error

If we were to sample a population multiple times we could calculate a mean for each sample. The standard deviation for the set of means is called the standard error. The value of the standard error can be calculated using a remarkably simple formula:

$$\text{SE}_{\bar{x}} = \frac{\sigma}{\sqrt{N}}$$

Strictly speaking σ should be the standard deviation of the population but often this is not available and instead the sample standard deviation is used instead. The standard error is also a convenient measure of how precise our measurements are, that is how close a set of measurements are to each other.

D.4 Covariance

If the variability of one variable, x, is influenced by another, y, then this dependence is measured using the covariance, $\text{Cov}(x, y)$. The covariance between two variables is defined by:

$$\text{Cov}(x, y) = \frac{1}{N} \sum \left[(x_i - \mu_x)(y_i - \mu_y) \right]$$

A positive covariance means that the two variables as positively correlated. A covariance of zero means that the two variables are statistically independent.

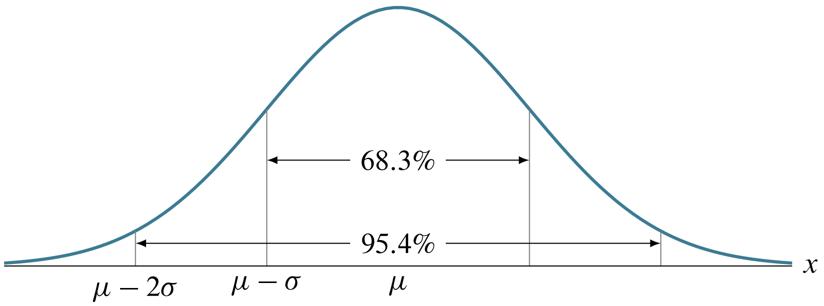


Figure D.1 Normal Distribution: The 68.3% and 95.4% intervals represent one and two standard deviations away from the mean, μ.

D.5 Normal Distribution

The normal or Gaussian distribution is a continuous probability distribution that describes the probability of obtaining a given value, x when the distribution has mean μ and standard deviation σ. The probability in a normal distribution is described by the area under the curve such that the total area is equal to one. The mean corresponds to the peak of the curve (since it is symmetric) and the standard deviation to the width. If a random variable is known to be normally distributed then the Gaussian curve tells us that there is 68.3% chance that the value will lie within one standard deviation from the mean (Figure D.1).

The equation that defines the Gaussian distribution is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

D.6 Confidence Interval

It is often the case that we would like to know the likelihood that a given variable will fall within a specified range. That is we would like some measure of confidence that a measured quantity is likely to fall within a certain range. If someone quoted the statistic that a variable, x, has a 95% confidence interval of $x \pm \Delta x$, that would mean that if we repeatedly measured this variable, that 95% of the time, the measured value would lie between $x + \Delta x$ and $x - \Delta x$. If the distribution of x is normal, then the 95% interval is at 1.96 σ. For example, if we know that a variable has a mean value of 2.5 and a standard deviation of 0.6, then the 95% confidence interval is given by:

$$\bar{x} \pm 1.96\,\sigma = 2.5 \pm 1.96 \times 0.6 = 2.5 \pm 0.3$$

Therefore if we were to take one more measurement, we could state that the value of the measurement will lie between 2.2 and 2.8 95% of the time. We can also say that 1 in 20 (5%) of the time the variable will lie outside this range by chance.

Alternatively we could make not one measurement but obtain an entire new sample of measurements and compute the mean of the new sample. Given a new sample, what can we say about the likely value fot the mean of that sample? Given the original standard deviation we could state that the mean of a new sample will have a confidence limit of:

$$\bar{x}_{95\%} \pm 1.96\ \mathrm{SE}_{\bar{x}}$$

which is a more useful statistic to obtain. That is, the mean of the new sample will have a mean $\pm$ the standard error. For example the mean for a sample of nine data points is 4.0 with a standard deviation of 2.0. Given this information the standard error can be computed to be: $\mathrm{SE}_{\bar{x}} = \sigma/\sqrt{n} = 2/3 = 0.66667$. Therefore the 95% confidence internal on the mean is:

$$\bar{x}_{95\%} \pm 1.96\ \times 0.66667 = 4.0 \pm 1.31$$

That is if we did draw a new sample then 95% of the time the mean would lie within the above range.

D.7 Bootstrapping

Let us assume we wish to estimate the 95% confidence interval for the mean of a population. The problem is we don't have the population, only a sample from the population. We can use bootstrapping to get an estimate for the confidence interval, or more precisely we can use Bootstrapping to generate a distribution that resembles the population from which we can estimate a confidence interval. Bootstrapping is the act of generating the distribution by sampling. Let us assume for argument sake that our sample from the population is:

$$4, 5, 7, 3, 7, 1$$

We will now resample with replacement from the original sample. Replacement means not removing the sampled value from the sample set, this means that it is possible to sample the same value again. An example of a bootstrap sample is:

$$7, 3, 1, 4, 1, 7$$

The new sample should have the same number of elements as the original sample. We now create many hundreds of similar samples, lets say we create 200 samples in this way. For each sample we will compute the mean so that we will have 200 means. That is our bootstrap sample of means. The bootstrap has now essentially finished.

At this point we can compute some interesting statistics from our 200 means. For example, what is the 95% confidence interval for the mean of the original population? If we assume that our sample of means has the same statistical structure as the population then we can use the 200 means to compute the 95% confidence interval. To compute the 95% confidence interval for the means, we must first rank the means in ascending order and then use the 97.5% and 2.5% percentiles as the interval, that is the middle 95% of all bootstrap sample means.

Listing D.1 show some Jarnac code to bootstrap a sample of 30 taken from the normal distribution.

```
sampleSize = 30;
s = vector (sampleSize)
for i = 1 to sampleSize do
    s[i] = stats.gauss(0.2, 1);

 p = []; n = 10000;
 means = vector (n);
 for k = 1 to n do
     begin
     l = vector (sampleSize);
     for i = 1 to sampleSize do
         begin
         r = trunc (rnd (sampleSize)) + 1;
         l[i] = s[r];
         end;
     p.append (l);
     means[k] = mean (l);
     end;
y = hist (means, -1, 1, 50);
graph (y);
```

Listing D.1 Bootstrapping Script

Further Reading

1. Bevington, PR, Data reduction and error analysis for the physical sciences. McGraw-Hill, 1969.

2. Berendsen, Herman JC. A student's guide to data and error analysis. Cambridge: Cambridge University Press, 2011.

3. For something different: David Freedman D, Pisani R, Purves R. Statistics W. W. Norton & Company; 3rd edition, 1998

4. Mandel J. The statistical analysis of experimental data. Dover Publications, 1984.

5. Manly, BFJ. Randomization, Bootstrap and Monte Carlo Methods in Biology. Chapman & Hall, 1997.

E

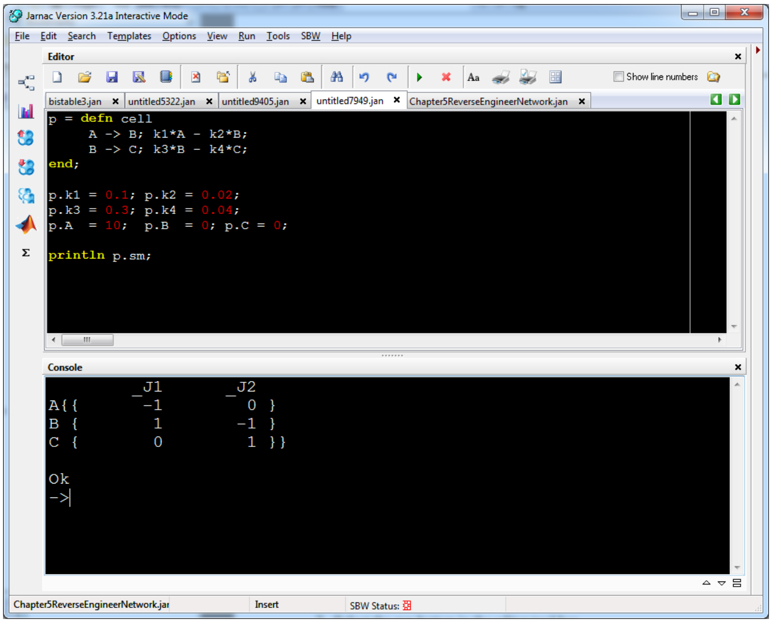
Introduction to Jarnac

Jarnac is a Windows 2000/XP/Vista/7 based interactive language for numerical analysis and building models of reaction networks such as metabolic pathways, signal transduction circuits and gene regulatory networks.

E.1 Quick Start Guide

The main area of user interaction is at the console window, this is located at the top left of the screen (Figure E.1). This is where you enter commands and instructions for Jarnac to obey. The lower panel is the built-in editor where you can write or edit script files – a list of commands or programming constructs for Jarnac to execute.

Operations can be performed with Jarnac in two ways: you can enter commands at the console for immediate execution, or, you can run Jarnac script files. At the console you can type things like, `sin (pi) + log10(1000)`, hit return and Jarnac will return an immediate answer to you. On the other hand one may have a whole set of things to calculate in which case it might be convenient to put all the instructions into a file, a script file, and let Jarnac run the script file.


Jarnac Version 3.21a Interactive Mode
File Edit Search Templates Options View Run Tools SBW Help
Editor
Show line numbers
bistable3.jan
untitled5322.jan
untitled9405.jan
untitled7949.jan
Chapter5ReverseEngineerNetwork.jan
Console
Chapter5ReverseEngineerNetwork.jan
Insert
SBW Status:

```
p = defn cell
     A -> B; k1*A - k2*B;
     B -> C; k3*B - k4*C;
end;

p.k1 = 0.1; p.k2 = 0.02;
p.k3 = 0.3; p.k4 = 0.04;
p.A  = 10;  p.B  = 0; p.C = 0;

println p.sm;
```

```
          _J1         _J2
A{{        -1           0 }
B {         1          -1 }
C {         0           1 }}

Ok
->
```

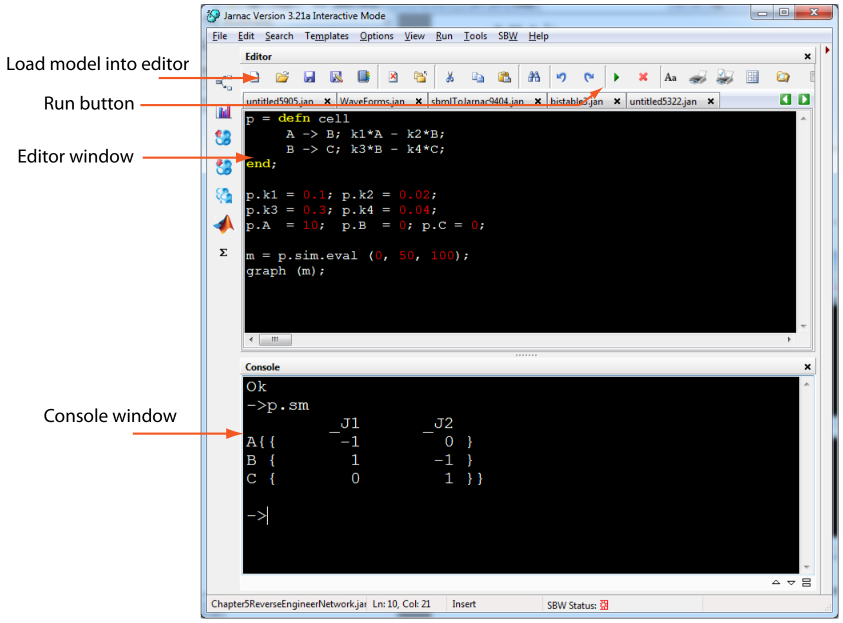
Figure E.1 Screen shot of Jarnac. Lower panel interactive console, upper panel, model editor.

To load and run a model the following instructions can be followed (Figure E.2):

1. Load a model into the editor by using the load button
2. Select the run button in the editor tool bar

For example the following model might be stored in a file called `mymodel.jan`.

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

p.vo = 1;
```


Load model into editor
Run button
Editor window
Console window
Jarnac Version 3.21a Interactive Mode
File Edit Search Templates Options View Run Tools SBW Help
Editor
untitled5905.jan
WaveForms.jan
sbmlToJarnac9404.jan
bistable3.jan
untitled5322.jan
Console
Chapter5ReverseEngineerNetwork.jar Ln: 10, Col: 21 Insert SBW Status:

```
p = defn cell
     A -> B; k1*A - k2*B;
     B -> C; k3*B - k4*C;
end;

p.k1 = 0.1; p.k2 = 0.02;
p.k3 = 0.3; p.k4 = 0.04;
p.A  = 10;  p.B  = 0; p.C = 0;

m = p.sim.eval (0, 50, 100);
graph (m);
```

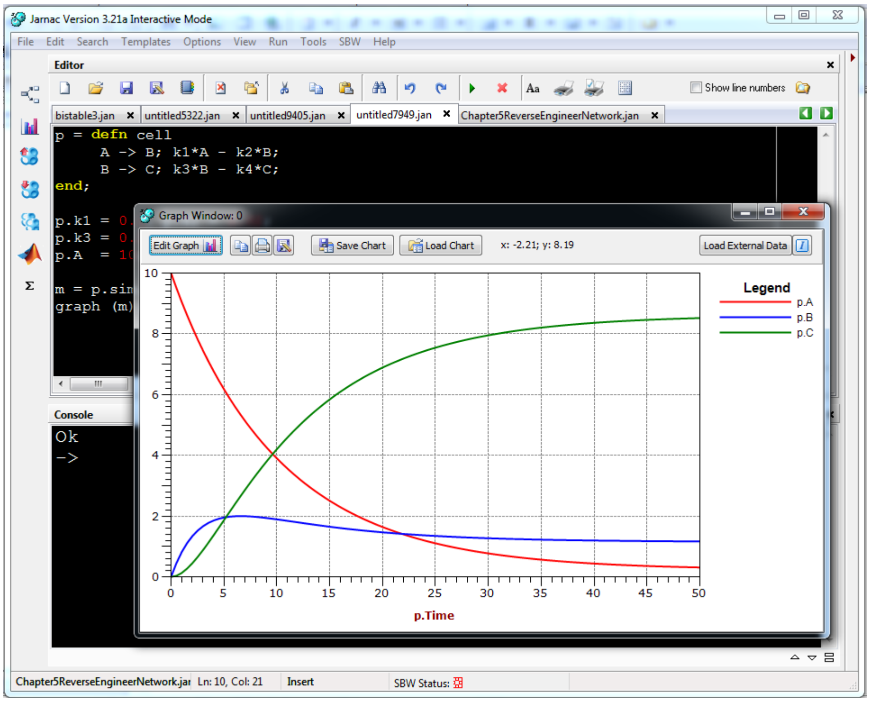
```
Ok
->p.sm
           _J1        _J2
A{{         -1          0 }
B {          1         -1 }
C {          0          1 }}

->
```

Figure E.2 Screen shot of Jarnac with main elements labeled.

```
p.k1 = 2; p. k2 = 0;
p.k3 = 3;

m = p.sim.eval (0, 6, 100);
graph (m);
```

If this file is loaded into the Jarnac editor and the run button selected, the screen shot shown in Figure E.3 will be seen.

It is also possible, though perhaps less convenientl, to load and run models from the console. The following four commands can be executed directly from the console.



```
p = defn cell
     A -> B; k1*A - k2*B;
     B -> C; k3*B - k4*C;
end;

p.k1 = 0.
p.k3 = 0.
p.A  = 10

m = p.sim
graph (m)
```

```
Ok
->
```

Figure E.3 Screen shot of Jarnac with simulation results

`run filename`	Run a script file, eg `run calvin`
`edit filename`	Load the file, filename, into the editor window
`dir`	Display directory listing of current directory, eg `dir`
`cd path`	Change current directory, eg `cd glycolysis`

The `run` command is used to run Jarnac scripts files. Scripts can be written using any editor that can save ascii text files, this can include nodepad, WinEdt or more simply Jarnac's built-in editor.

E.2 Current Directory

Jarnac maintains an internal variable called the current directory. All operations involving external files, for example, running script files, opening export files, saving data objects, etc., will use the current directory as the search path. The current directory can be examined using the predefined variable 'sys.path'. Setting the current directory can be done either by assigning a new path string to the path variable:

```
sys.path="c:\MyModels")
```

or using the system command, 'cd', e.g. `cd c:\MyModels`, note that the command `cd` does not require the path to be put in quotes, this is because `cd` is a console command rather than a Jarnac language statement (See Jarnac Command and Language Statements).

E.3 Simple Examples from the Console Window

In the following console examples, the symbol, `->` is the Jarnac prompt. Everything after the `->` is **user input**, every line without a `->` is **Jarnac output**. It is assumed that at the end of each line the user has hit the return key.

```
->sys.path
c:\MyModels
->sys.path="c:\MyModels\liver"
->sys.path
c:\MyModels\liver
->
```

E.3.1 Numeric handling

```
->1/2 + 10
10.5
->a = 2
->b = 7.5
```

```
->a*b
15
->sin (a*b)
0.650287840157117
->m = {{1,2,3},{7,5,2},{8,7,6}}
->1/m
{{-1.778   -1  1.222}}
 { 2.889    2 -2.111}
 {    -1   -1      1}}
->mt = tr (m)
->inv(mt) = 1/mt
->
->println "pi times 2 is", pi*2;
pi times 2 is 6.2832
```

E.3.2 Type of Data

Jarnac supports a variety of different data types, these can be split into two groups, primitive and object types. Primitive types are generally faster to manipulate compared to object types; however object types posses a much richer array of abilities and representation.

Primitive Types

`Integer`	`signed 32-bit integer eg. 5, 34586`
`Float`	`double precision, eg. 1.23454, 0.45, 3.4e-4`
`Complex`	`double precision complex number, eg 2 + 4i, 7i 1 - 3i`
`String`	`"a string"`
`Boolean`	`True or False`
`Vectors`	`{1,2,3} Elements may be constants or expressions`
`Matrices`	`{{1,2}, {4,5}} Elements may be constants or expressions`
`Code Types`	`<a+b>`

Complex Types

`Lists`	`["XYZ", 1.2, x*y, [i,j,k, 1,2,3,4]]`
`Networks`	`defn cell [J1] S1 -> S2; v1; [J2] S2 -> S3; S2*k1; end`
`i/o Objects`	`File, Console or Printer streams`
`User Functions`	`eg function test (x) return pi*x; end;`
`Graphs`	`Data visualisation objects`

Examples of different data type assignments:

```
a = 1.2345;
i = 99; j = 88; k = 77;
c = 4 + 7.4i;
d = 40i;
m = {{i,2,3}, {j,5,6}, {k,8,9}};
v = vector (1000};
Name = "Jim Smith";
Spock = True;
f = <sin (x) + cos (1.2*y)>;
ShoppingList = ["Bread", "Newspaper", "Olives"];
function MyFunc (x, y) return sin(x)*cos(y); end;
```

E.3.3 Integers

Integers are stored as 32 bit signed values, that is a double word. Integers have the numeric range `-Ű2147483648` to `2147483647`. A number of functions exist to convert integers to and from string format, `StrToInt` and `IntToStr`. If you wish to control the conversion of an integer to it's string representation, use the `Format` function.

E.3.4 Floats

Floats represent the usual scientific floating point number. In Jarnac they are implemented using IEEE standard double format. That is each float requires eight bytes of storage. The range which a float covers is $5.0 \times 10^{-\text{Ű}324}$ to 1.7×10^{308} and each number has 15 to 16 digits of significance.

Jarnac also has built-in two floating point constants, `NAN` and `Inf`, these represent 'Not a Number' and 'Infinity' respectively, as defined in the IEEE standard.

E.3.5 Complex Numbers

The native float type for Jarnac is actually the complex type. A complex number is defined as pair of floating point numbers, a real part and an imaginary part. Both numbers have the same restrictions on their range as a normal floating point number. The imaginary part of a complex number is indicated by a 'i' or 'j' placed immediately after the number, for example

```
0.1i
3.1415j
```

To create a complex number with a nonzero real part, add a floating point number to it, e.g.

```
3+4i
```

Like integers and floats, normal arithmetic can be done with complex numbers, for example:

```
->x = 3+4i + 3-7i
->println x
(6-3i)
->
```

E.3.6 Boolean

Boolean types occupy one byte and can have only one of two states representing true or false. Two built-in constants, `False` and `True` are provided to allow boolean types to be set and tested.

E.3.7 Strings

Literal strings are indicated using the double quote character, `"`, thus, `"Hello World"` is a literal string. If you wish to include the double-quote itself as part of the literal string, use two double-quote characters in succession. For example the string `"""Hello"""` yields `"Hello"`.

The escape character, ‘\’, may be used to insert a limited number of special characters into a literal strings. These are listed below:

`\\`	Yields the character ‘\’
`\n`	Carriage Return/Line Feed (newline)
`\r`	Carriage Return
`\f`	Line Feed
`\t`	Tab character

E.3.8 Code Types

Code types are a way of converting simple algebraic expressions into Jarnac’s internal code representation. Code types start with the symbol < followed by the expression, finishing with a closing >.

For example, the following are code type expressions:

```
f = <a*b + c>
TotalFlux = <p.J1 + p.J2 + p.J3>
FuncToPlot = <sin(x)*2.3 + cos(y)*5.6>
```

When a code type is defined, the `<...>` construct returns a reference to the code, that is a reference to the internal code that Jarnac generates when it parses the expression. This is in contrast to a normal expression which is evaluated immediately and the result returned to the left hand side of the assignment.

Thus the expression, `2+3`, returns the value `5`, while the expression, `<2+3>` returns a reference to the internal Jarnac code which represents the operation `2+3`.

If you wish to evaluate a code type, use the `eval` function. The `eval` function takes a single argument which is the code type expression. The evaluation of the code occurs within the current symbol context. If Jarnac finds that it cannot locate a particular symbol in the code type expression, a runtime error is generated. When evaluating code types within user functions, Jarnac uses the user function symbol context to evaluate the code type.

Like any other data type it is possible to pass code type expressions or variables which reference code types, to user defined functions. Code types are an important data type in a number of numerical analysis situations, in particular they are important during simulation runs and when a user requests the computation of metabolic control analysis coefficients.

```
f = <x*x>;
x = 3.4;
println eval (f);
```

Side Note: The `eval` method will also accept string arguments in addition to code types. This means one can evaluate arbitrary expressions at runtime, for example:

```
function PlotMe (g, f)
   x = 0;
```

```
     for i = 1 to 50 do
         begin
         g.Markat (x, eval (f), 0);
         x = x + 0.1;
         end;
  end;

  g = newgraph;
  PlotMe (g, "sin (x)");
  ReadKey;
  g.close;
```

E.3.9 Matrices

Jarnac has built-in support for Matrices, that is rectangular arrays of numbers with labeled rows and columns. Matrices can be entered in two ways, either by literal declaration or by using one of the built-in matrix functions.

Literal Declaration:

```
m = { {1, 2, 3}, {4,5,6}, {7,8,9} }

m = { {i, i+1}, {j, j+1}, {k, k+1} }

m = {{}}  // Empty matrix
```

Literal declarations my contain constants or expressions which resolve to integers, floats or complex numbers. The comma is used to delimit individual rows and elements and brackets are used to delimit rows. There are also a number of functions that will generate matrices, they include (all number pairs indicate rows and columns):

```
m = matrix (5,5)  // Create an empty 5 by 5 matrix

m = ident (5)     // Create an identity matrix of size 5

m = ones (3,4)   // Create a matrix filled with ones of size 3 by 4
```

The simulation methods:

```
m = p.sim.eval (0, 10, 100);
```

return time course data in the form of a matrix.

There are a few ways to access elements and sections of a matrix. Accessing a single element is done using the indexing syntax, for example, `m[2,3]` means access the value at the second row and third column. **Note that matrix indexing starts from one.**

```
m[1,1] = 1.234;
m[2,2] = m[1,1] + 10;
```

Individual rows and columns or groups or rows and columns can be accessed either using specific methods or slicing. The following methods can be used to extract specific rows and columns from a matrix `m`:

```
x = getColumn (m, 2); // get column 2
x = getRow (m, 4);    // get row 4
```

To extract groups of rows and columns use the following methods:

```
// Extract first and second column
x = getColumns (m, [1,2]);
// Extract 2nd, 4th and 6th columns
x = getColumns (m, [2,4,6]);
// Similar syntax applies to extracting rows,
// extract 5th and 4th rows.
x = getRows (m, [5,4]);
```

It is also possible to use Matlab type slicing of matrices which is less verbose than the methods above. For example:

```
// Extract rows 1 to 2 and columns 3 to 4 (results in a 4 by 4 matrix)
x = m[1:2,3:4]
```

```
// Extract all rows in columns 2, 3 and 4
x = m[:,2:4]
// Extract all rows in column 3 to the last column in the matrix
x = m[:,2:columns (m)]
// Extract all columns in row 2 to the last row in the matrix
x = m[2:rows (m),:]
```

An important distinction between Jarnac matrices and matrices in other programming languages, is that Jarnac matrices are labeled. That is the rows and columns have labels. This means that any manipulations that extracts rows and columns or exchange rows and column will also result in appropriate changes to the row and column labels. For example, consider the matrix:

```
m = {{1,2}, 3,4}};
println m;
       C1     C2
R1{{     1      2}
R2 {     3      4}}
```

This matrix has the default row and column labels. If we transpose this matrix using the `tr()` method we get:

```
m = tr (m);
println m
        R1     R2
C1{{     1      3}
C2 {     2      4}}
```

Column and row labels can be renamed using the `setRowName` and `setColumnName` methods, for example:

```
setRowName {m, 2, 'A New Label');
```

relabels the second row of matrix *m* with a new string.

E.4 Running Script Files

You can write script files using any basic ascii editor but more conveniently you can use the built-in editor that comes with Jarnac. To start a new script file click on the new button in the editor tool bar. Now enter your commands into the lower panel, for example enter the following lines (Note the semicolons at the end of each line):

```
H = 1E-4;
pH = -Log10 (H);
println "The pH is ", pH;
```

Now save the file to some convenient location using the save-as button in the tool bar, and call the file `myfile.jan`. To run this script you can either type the command `run myfile` at the console or click on the run button in the tool bar.

Loading existing script files into the editor is simply a matter of typing `edit myfile` at the console or clicking on the load file button on the editor tool bar. Save any edit by using the save button on the editor tool bar.

Script files should have the extension `.jan`. Commands such as `run` and `edit` will search for file names of the form `x.jan`. This means it is possible to omit the extension and simply use `run x` and Jarnac will implicitly add the extension. Thus to edit a file called `mymodel.jan` one can type the abbreviated form, `edit mymodel` – although `edit mymodel.jan` is equally acceptable – and Jarnac will implicitly add the `.jan` extension for you. This is simply a convenience for the user.

E.4.1 Commenting Code

Like all programming languages it is possible to add comments to a Jarnac script. Two types of comment are supported:

1. C++ style comments as in:

```
// This is a comment
```

2. C style comments as in:

```
/* Use C style comments when you have multiple
   lines to comment.
*/
```

E.5 Reaction Network Models

The main purpose of Jarnac is to make it straight forward to specify complex reaction networks using a familiar chemical reaction notation.

A chemical reaction can be an enzyme catalyzed reaction, a binding reaction, a phosphorylation, a gene expressing a protein or any chemical process that results in the conversion of one of more species (reactants) to a set of one or more other species (products). In Jarnac, reactions are described using the notation:

```
A + ...  -> P + ...
```

where the reactants are on the left side and products on the right side. The left and right are separated by the `->` symbol. For example:

```
A -> B
```

describes the conversion of reactant `A` into product `B`. In this case one molecule of `A` is converted to one molecule of `B`. The following example shows non-unity stoichiometry:

```
2 A -> 3 B
```

which means that two molecules of `A` react to form three molecules of `B`. Bimolecular and other combinations can be specified using the + symbol, that is:

```
2 A + B -> C + 3 D
```

tells us that two molecules of `A` combine with one molecule of `B` to form one molecule of `C` and three molecules of `D`.

To specify species that do not change in time (boundary species), add a dollar character in front of the name, for example:

```
$A + B -> C
```

means that during a simulation `A` is fixed.

Reactions can be named using the syntax `J1:`, for example:

```
J1:  A + B -> C
```

means the reaction has a name, `J1`. Named reaction are useful if you want to refer to the flux of the reaction; Kinetic rate laws come immediately after the reaction specification. If only the stoichiometry matrix is required, then it is not necessary to enter a full kinetic law, a simple `...  -> S1; v;` is sufficient. Here is an example of reaction that is governed by a Michaelis-Menten rate law:

```
A -> B; Vm*A/(Km + A);
```

Note the semicolons.

Here is a more complex example involving multiple reactions:

```
p = defn Branch
   MainFeed:    $X0 -> S1;  Vm*X0/(Km + X0);
   TopBranch:    S1 -> $X1; Vm1*S1/(Km1 + S1);
   BottomBranch: S1 -> $X2; Vm2*S1/(Km2 + S1);
end;
```

Note that there is no need to pre-declare the species names shown in the reactions or the parameters in the kinetic rate laws. Strictly speaking, declaring the names of the floating species is optional, however this feature is for more advanced users who which to define the order of rows that will appear in the stoichiometry matrix. For normal use there is no need to predeclare the species names. To predeclare parameters and variables see the example below:

```
p = defn Branch

      ext Xo, X1, X2; // Boundary species
      var S1;         // Floating species

   MainFeed:    $X0 -> S1;  Vm*X0/(Km + X0);
   TopBranch:    S1 -> $X1; Vm1*S1/(Km1 + S1);
```

```
      BottomBranch: S1 -> $X2; Vm2*S1/(Km2 + S1);
   end;
```

Every model is assigned to a variable, in this case, named ‘p’. To reference model properties and methods, the property or method must be proceeded with the name of the model, eg `p.S1 = 2.3;`

A model declared as in the above example will be converted into a set of differential equations. For example consider the following model:

```
p = defn cell
      $Xo -> S1;  v1;
       S1 -> S2;  v2;
       S2 -> $X1; v3;
end;
```

will be converted into:

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

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

Note that there are no differential equations for X_o and X_1, this is because they are fixed and do not change in time. If the reactions have non-unity stoichiometry than this is taken into account when the differential equations are derived.

E.5.1 Initialization of Model Values

Initializing a model is very simple:

```
  p.X0 = 3.4;
  p.X1 = 0.0;
  p.S1 = 0.1;
```

```
p.Vm = 12; p.Km = 0.1;
p.Vm1 = 14; p.Km1 = 0.4;
p.Vm2 = 16; p.Km2 = 3.4;
```

E.5.2 Time Course Simulation

There are two methods for evaluating the time evolution of a reaction network model, these are, `p.sim.OneStep` and `p.sim.Eval`. `OneStep` is used to compute a single time step between two time points while `Eval` is used to compute the model over a range of time steps. The most commonly used call is the `Eval` method. The simplest way to call it is to write:

```
m = p.sim.eval (0, 10, 100);
```

This call will run a time course simulation starting at time zero, ending at time 10 units and generating 100 points. The results of the run are deposited in the matrix variables m. At the end of the run the m matrix will contain columns corresponding to the time column and all the species concentrations. If more control is required over what the columns of the m matrix should be, the `Eval` method takes an additional argument, a list of terms that will represent the columns of the matrix. For example the following call:

```
m = p.ss.eval (0, 10, 1000, [<p.S1>])
```

will return a matrix 1000 rows deep and one column wide that corresponds to the number of items in the fourth argument. The fourth argument is a list containing the desired outputs. A number of examples include:

```
m = p.ss.eval (0, 10, 1000, [<p.S1>, <p.J1>, <p.J2>, <p.J2>]);
m = p.ss.eval (0, 10, 1000, [<p.S1>, <p.J1+p.J2>]);
m = p.ss.eval (0, 10, 1000,
    [<p.Time>, <p.S1>, <p.S1*Log10 (p.J1)>]);
```

Note that the special variable `Time` is available which represents the independent time variable in the model. The first argument equals the start time point, the second argument the end time point and as mentioned before the third argument represents the required number of output points.

To visualize the output in the form of a graph one simply needs to pass the matrix variable returned by sim.eval to the command, graph. Thus:

```
m = p.ss.eval (0, 10, 1000,
    [<p.Time>, <p.S1>, <p.J1>, <p.J2>, <p.J3>]);
graph (m)
```

or if only the graph is required:

```
graph (p.ss.eval (0, 10, 1000,
      [<p.Time>, <p.S1>, <p.J1>, <p.J2>, <p.J3>]));
```

To selectively graph particular items from the generated matrix use the syntax shown in the following example:

```
graph (m, [2, 3]);
graph (m, [1, 2, 4]);
```

The list argument indicates which columns of *m* to graph. The first column specified in the list is always considered the independent variable (x axis). This means that the statement, `graph (m, [1,2,3,4])` is equivalent to `graph (m)`.

For more control, user can use the `OneStep` method. `OneStep` takes two arguments and returns a double value.

`tNext = p.sim.OneStep (tStart, tStep)`

`OneStep` takes the start time point of the simulation and the step size required to simulate over. For convenience `OneStep` returns the new time value, i.e. `tStart + tStep`. Example:

```
t = 0;
repeat
  t = p.sim.OneStep (t, 0.1);
  println p.S1;
until t > 10;
```

E.5.3 Applying Perturbations to a Simulation

Often in a simulation we may wish to perturb a species or parameter at some point during the simulation and observe what happens. One way to do this in Jarnac is to carry out two separate simulations where a perturbation is made when in between the two simulations. For example, let's say we wish to perturb the species concentration for a simple two step pathway and watch the perturbation decay. The first thing we do is simulate the model for 10 time units, this gives us a transient and then a steady state.

```
p = defn cell
     $Xo -> S1;  k1*Xo;
      S1 -> $X1; k2*S1;
end;

p.Xo = 10; p.k1 = 0.3; p.k2 = 0.15;

m1 = p.sim.eval (0, 10, 50);
```

We then make our perturbation as follow:

```
p.S1 = p.S1 * 1.5;
```

which increases S_1 by 50%. We then carry out a second simulation:

```
m2 = p.sim.eval (10, 20, 50);
```

Note that we set the time start of the second simulation to the end time of the first simulation (10). Once we have the two simulations we can combine the matrices from both simulations using the augment rows method:

```
m = augr (m1, m2)
```

Finally, we plot the results.

```
graph (m);
```

For those who use LaTeXand pgfplot, it is possible to generate pgfplot code by using the commands:

```
str =  exportPGFPlot (m);
copyToClipboard (str);
```

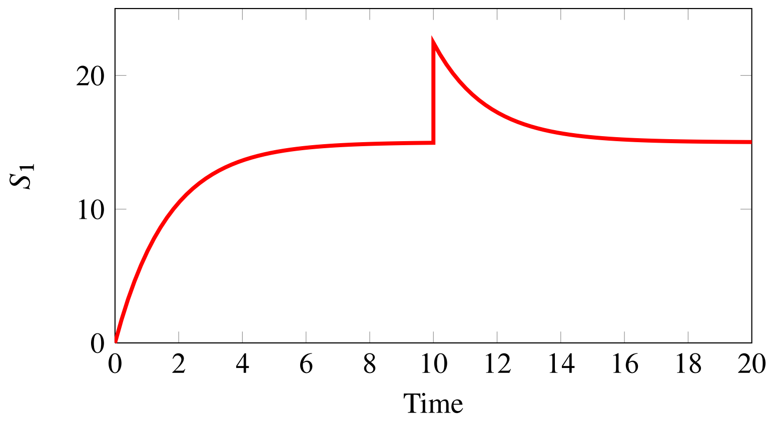


Figure E.4 Plotting a perturbation in S_1.

The copyToClipboard command makes it easy to paste the code into a document.

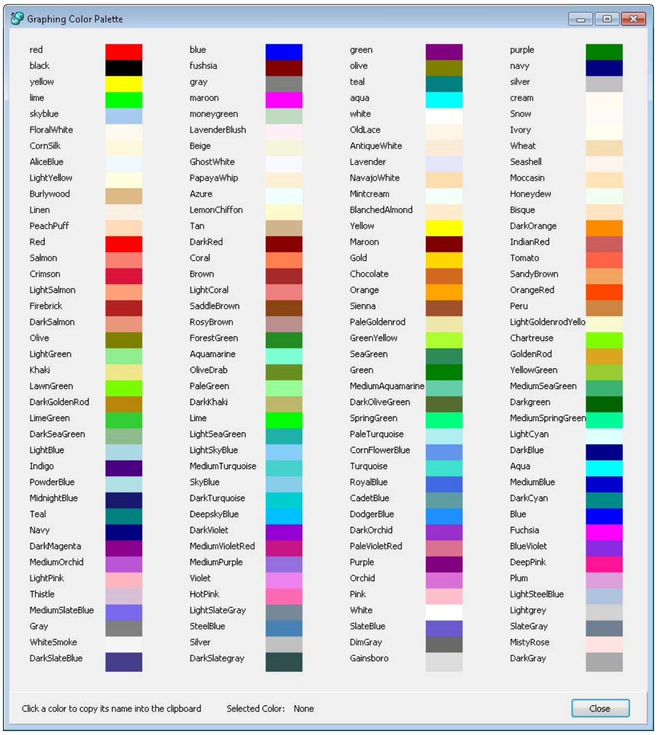
E.5.4 Additional Plotting Commands

In the last section the `graph` function was used to plot data held in a matrix including the ability to selectively plot specific columns from the matrix. Here moer detail will be given on the graph function together with other function that can be used in plotting graphs.

The `graph` function can take a number of arguments. The first argument is always the matrix of data to plot. The next two arguments are optional and are represented as lists. One contains the columns to plot, as we have seen before, for example, `graph (m, [1, 3, 4]);` will plot columns 3 and 4 versus column 1. The second optional argument can be used to specify the color of the individual curves. For example:

```
graph (m, [1,3,4], ["OrangeRed", "LightSeaGreen"]
```

will plot two curves from columns 3 and 4 where data from column 3 will be plotted in OrangeRed and the data from column 4 plotted in LightSea-Green. The complete list of colors is shown in Figure E.5 and can be viewed by looking at View in the main Jarnac menu and selecting Color Palette.



red	blue	green	purple
black	fushsia	olive	navy
yellow	gray	teal	silver
lime	maroon	aqua	cream
skyblue	moneygreen	white	Snow
FloralWhite	LavenderBlush	OldLace	Ivory
CornSilk	Beige	AntiqueWhite	Wheat
AliceBlue	GhostWhite	Lavender	Seashell
LightYellow	PapayaWhip	NavajoWhite	Moccasin
Burlywood	Azure	Mintcream	Honeydew
Linen	LemonChiffon	BlanchedAlmond	Bisque
PeachPuff	Tan	Yellow	DarkOrange
Red	DarkRed	Maroon	IndianRed
Salmon	Coral	Gold	Tomato
Crimson	Brown	Chocolate	SandyBrown
LightSalmon	LightCoral	Orange	OrangeRed
Firebrick	SaddleBrown	Sienna	Peru
DarkSalmon	RosyBrown	PaleGoldenrod	LightGoldenrodYello
Olive	ForestGreen	GreenYellow	Chartreuse
LightGreen	Aquamarine	SeaGreen	GoldenRod
Khaki	OliveDrab	Green	YellowGreen
LawnGreen	PaleGreen	MediumAquamarine	MediumSeaGreen
DarkGoldenRod	DarkKhaki	DarkOliveGreen	Darkgreen
LimeGreen	Lime	SpringGreen	MediumSpringGreen
DarkSeaGreen	LightSeaGreen	PaleTurquoise	LightCyan
LightBlue	LightSkyBlue	CornFlowerBlue	DarkBlue
Indigo	MediumTurquoise	Turquoise	Aqua
PowderBlue	SkyBlue	RoyalBlue	MediumBlue
MidnightBlue	DarkTurquoise	CadetBlue	DarkCyan
Teal	DeepskyBlue	DodgerBlue	Blue
Navy	DarkViolet	DarkOrchid	Fuchsia
DarkMagenta	MediumVioletRed	PaleVioletRed	BlueViolet
MediumOrchid	MediumPurple	Purple	DeepPink
LightPink	Violet	Orchid	Plum
Thistle	HotPink	Pink	LightSteelBlue
MediumSlateBlue	LightSlateGray	White	Lightgrey
Gray	SteelBlue	SlateBlue	SlateGray
WhiteSmoke	Silver	DimGray	MistyRose
DarkSlateBlue	DarkSlategray	Gainsboro	DarkGray

Click a color to copy its name into the clipboard   Selected Color: None

Figure E.5 Plotting Palette.

For plots that display many curves a legend can get in the way, the function `displayLegend (true|false)` can be use to turn on and off the legend display.

By default the plotting panel autoscales the axes. Often this is the appropriate thing to do but if the simulation data starts above the zero axis the autoscaling will also start the y axis above the zero axis. This behavior is sometimes inappropriate. Therefore a set of functions are provided to set the axes limits, these include, individual settings and a built setting function:

```
setYAxisMin (ymin);
setYAxisMax (ymax);
setXAxisMin (xmin);
setXAxisMax (xmax);

// To set axies limit using one call for  he plotting window
setAxes ({0, 10, 0, 20})
```

Is is also possible to switch off autoscaling using the function:

```
setAutoScale (true|false)
```

E.5.5 Multiple Plots

It is possible to use more than one plotting window at any one time. By default one plotting window is assigned and it used all the time unless another window is explicitly created. To create a new plotting window, use the figure() command:

```
n = figure()
```

The command returns the plotting window number. This number can be used later to refer to the window. The default window is given the number zero. To focus all graphing commands to a specific plotting window, include the window number with figure. For example:

```
figure(n)
```

will focus future graph command on window n.

E.5.6 SubPlots

The normal plotting window only plots one set of axes. It is possible however to specify any number of axis in a given plotting window. To do this use the subplot command. For example, the following command will create a plotting window containing six subplots in a two by three array:

```
subpot (2, 3, 1)
```

The third argument in the call specify the current subplot where all graph commands will plot their data. subplots are numbers from one, so that the subplot command `subplot (2,3,1)` means that subsequent calls to graph() will result in plots appearing in the first panel. On the other hand, the command `subplot (2,3,4)` means that subsequent calls to graph() will result in the plot appearing in the fourth subplot. Figure E.6 was generated using the script below:

```
for i = 1 to 6 do
    begin
    subplot (2, 3, i);
    displayLegend (false);
    graph (urndv(50));
    end;
```

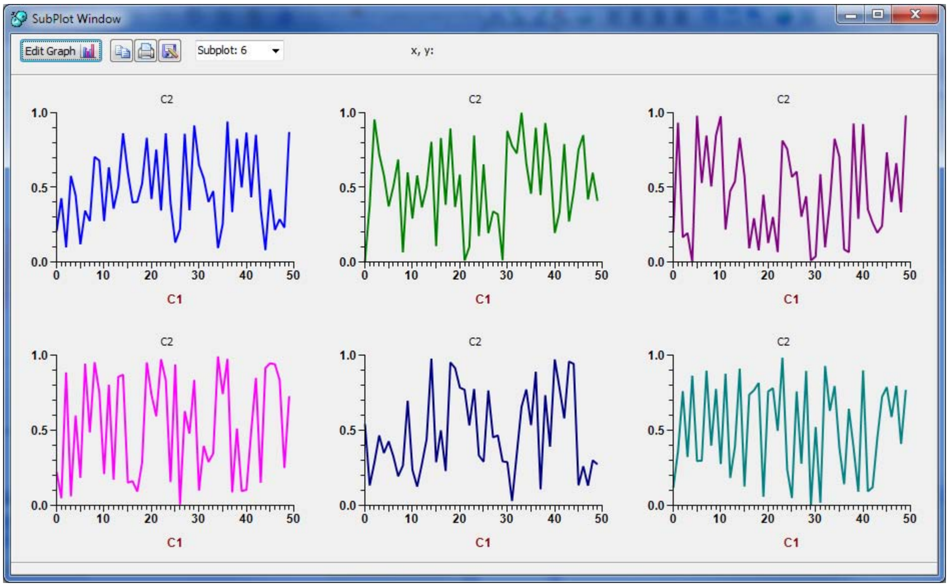


Figure E.6 Screen shot of a subplot window with six subplots.

E.5.7 Steady State and Metabolic Control

To evaluate the steady-state (make sure the model values have been previously initialized) enter the following statement at the console.

```
->p.ss.eval
```

This statement will return a value indicating how effective the computation was, essentially it returns the norm of the rate of change vector (i.e sqrt (Sum of dydt)). The closer this is to zero the better the approximation to steady state.

Once a steady state has been evaluated, the values of the metabolites will be at their steady state values, thus `S1` will equal the steady state concentration of `S1`.

The fluxes through the individual reactions can be obtained by either referencing the name of the reaction (eg `J1`) or via the predefined rate vector `rv[]` which is part of the model object. The advantage to using the rate vector is that the individual reaction fluxes can be accessed by indexing the vector (see example below).

```
->println p.J1, p.J2, p.J3
3.4, ...
->for i = 1 to 3 do println p.rv[i]
3.4, ...
->
```

To compute control coefficients use the statement:

```
cc (Dependent Measure, Independent parameter)
```

The dependent measure is an expression usually containing flux and metabolite references, for example, `p.S1, p.S1+p.S2, p.J1`. The independent parameter must be a simple parameter such as a Vmax, Km, ki, boundary metabolite (X0) or a conservation total such as CSUM1. In the current release it is not yet possible to use a floating metabolite as an independent parameter. Examples include:

```
  p.cc (<p.J1>, p.Vmax1)
  p.cc (<p.J1>, p.Vm1) + p.cc (<p.J1>, p.Vm2)
  p.cc (<p.J1+p.J2+p.J3>, p.Vmax1)
  p.cc (<p.J1>, p.X0)
  p.cc (<p.J1>, p.CSUM1)
```

To compute elasticity coefficients use the statement:

`ee (Reaction Name, Parameter Name)`

For example

```
  p.ee (<p.J1>, p.X0)
  p.ee (<p.J1>, p.S1)
```

Since cc and ee are built-in functions they can be used alone or as part of larger expressions. thus it is easy to show that the response coefficient is the product of a control coefficient and the adjacent elasticity by using:

```
  R = p.cc (<p.J1>, p.X0)
  println R - p.cc (<p.J1>, p.Vm) * p.ee (<p.J1>, p.X0)
```

To obtain the conservation matrix for a model use the model method, `conserve`.

```
->p=defn cycle [J1] E + S1 -> v;
   [J2] ES -> E + S2; v;
   [J3] S2 -> S1; v; end
->p.conserve
    E S1 ES S2
C1{{ 0  1  1  1}
C2 { 1  0  1  0}}
->
```

The result given above indicates that the conservation relations, `S1 + ES + S1` and `E + ES` exist in the model. As a result, Jarnac would generate two internal parameters `CSUM1` and `CSUM2` corresponding to these two relations.

E.5.8 Other Model Properties of Interest

There are however a number of predefined objects associated with a reaction network model which might be of interest. Of particular interest are the stoichiometry matrix, `sm`, the jacobian matrix, `jac` and the rate vector `rv` which we have seen before. These are returned to the user as matrices and can thus be treated like any other matrix type.

```
println p.sm
println p.jac
println p.rv
println p.xml2
println p.xml2("mymodel.xml")
```

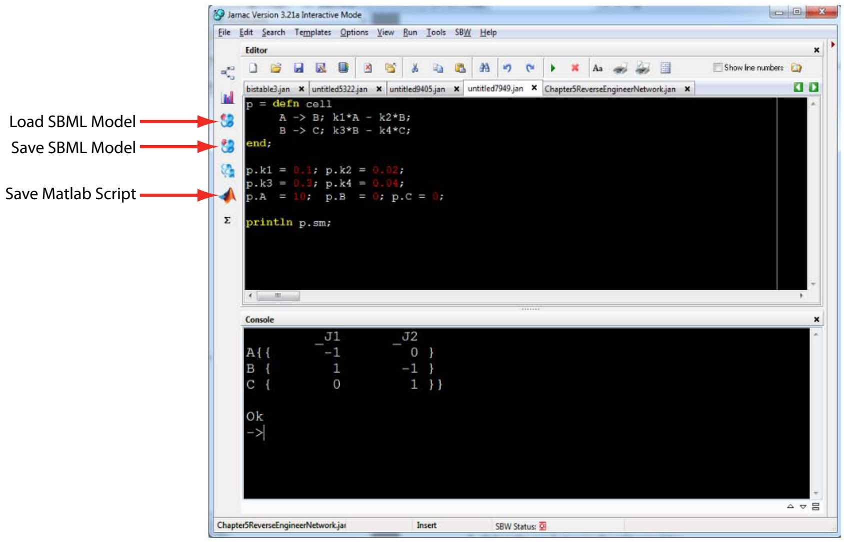
The jacobian is evaluated at the current state of the model, be it in steady state or not. The `xml2` calls can be used to generate standard SBML formatted models (Level 2, version 1). `p.xml2` will display the SBML to the screen and `p.xml2 ("mymodel.xml")` will save the SBML to a file in the current directory.

E.6 Generating SBML and Matlab Files

Jarnac can import and export standard SBML [48] as well as export Matlab scripts for the current model. To load a SBML model, select the load SBML model button (Figure E.7). For saving either SBML or Matlab files, the model **must be run** at least once in order to load the model into memory. Once loaded into memory the save to SBML or save to Matlab buttons can be selected (Figure E.7).

E.7 Exercise

Figure E.8 shows a two gene circuit with a feedforward loop. Assume the following rate laws for the four reactions:


Load SBML Model
Save SBML Model
Save Matlab Script
Jarnac Version 3.21a Interactive Mode
File Edit Search Templates Options View Run Tools SBW Help
Editor
Show line numbers
bistable3.jan
untitled5322.jan
untitled9405.jan
untitled7949.jan
Chapter5ReverseEngineerNetwork.jan
Console
Chapter5ReverseEngineerNetwork.jan
Insert
SBW Status:

```
p = defn cell
      A -> B; k1*A - k2*B;
      B -> C; k3*B - k4*C;
end;

p.k1 = 0.1; p.k2 = 0.02;
p.k3 = 0.3; p.k4 = 0.04;
p.A  = 10;  p.B  = 0; p.C = 0;

println p.sm;
```

```
           _J1        _J2
A{{         -1          0 }
B {          1         -1 }
C {          0          1 }}

Ok
->
```

Figure E.7 Screen shot of Jarnac showing buttons to load and save SBML and Matlab files.

$$v_1 = k_1 X_o$$

$$v_2 = k_2 x_1$$

$$v_3 = k_3 X_o$$

$$v_4 = k_4 x_1 x_2$$

Assume that all rate constants are equal to one and that $X_o = 1$. Assume also that X_o is a fixed species.

1. Create a Jarnac script that could be used to model this system.

2. Run a simulation of the system from 0 to 10 time units.

3. Next change the value of X_o to 2 (double it) and rerun the simulation for another 10 time units from where you left off in the last simulation. Combine both simulations and plot the result, that is time on the x-axis and X_o and x_2 on the y-axis.

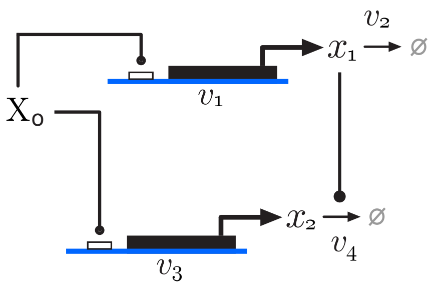


Figure E.8 Two gene circuit with feedfoward loop.

4. What you see?

5. Write out the differential equations for x_1 and x_2.

6. Show algebraically that the steady state level of x_2 is independent of X_o.

References

[1] Alon, U. 2006. *An Introduction to Systems Biology: Design Principles of Biological Circuits (Chapman & Hall/Crc Mathematical and Computational Biology Series)*. Chapman & Hall/CRC.

[2] Aparicio, O., Joseph V Geisberg, and Kevin Struhl. 2004. Curr Protoc Cell Biol **Chapter 17**:Unit 17.7.

[3] Barabási, A. L., and Z N Oltvai. 2004. Nat Rev Genet **5** (2):101–113.

[4] Barnett, V., and Toby Lewis. 1994. *Outliers in statistical data*, vol. 3. Wiley New York.

[5] Bennett, B. D., Elizabeth H Kimball, Melissa Gao, Robin Osterhout, Stephen J Van Dien, and Joshua D Rabinowitz. 2009. Nature chemical biology **5** (8):593–599.

[6] Bergmann, F., and H.M. Sauro. 2006a. title :1637–1645.

[7] Bergmann, F. T., and H. M. Sauro. 2006b. `http://sys-bio.org/sbwWiki`.

[8] Bergmann, F. T., R. R. Vallabhajosyula, and H. M. Sauro. October 2006. Current Proteomics **3**:181–197(17).

[9] Bevington, P. R., and D Keith Robinson. 1969. *Data reduction and error analysis for the physical sciences*, vol. 2. McGraw-Hill New York.

[10] Bhalla, U. S. 2002. Methods Enzymol **345**:3–23.

[11] Biondi, E. G., Sarah J Reisinger, Jeffrey M Skerker, Muhammad Arif, Barrett S Perchuk, Kathleen R Ryan, and Michael T Laub. 2006. Nature **444** (7121):899–904.

[12] Blinov, M. L., J R Faeder, B Goldstein, and W S Hlavacek. 2004. Bioinformatics **20** (17):3289–3291.

[13] Boahen, K. 2005. Scientific American **292** (5):56–63.

[14] Bode, A. M., and Zigang Dong. 2004. Nat Rev Cancer **4** (10):793–805.

[15] BRETT, D., H. POSPISIL, J. VALCARCEL, J. REICH, and P. BORK. 2002. Nature genetics **30** (1):29–30.

[16] Brilli, M., Marco Fondi, Renato Fani, Alessio Mengoni, Lorenzo Ferri, Marco Bazzicalupo, and Emanuele Biondi. 2010. BMC Systems Biology **4** (1):52.

[17] Burns, J. 1969. FEBS Lett **2 Suppl 1**:S30–S33.

[18] Burns, J. A. 1971. *Studies on Complex Enzyme Systems*. PhD thesis University of Edinburgh. `http://www.sys-bio.org/BurnsThesis`.

[19] Cash, J. R., and Alan H. Karp. 1990. ACM Trans. Math. Softw. **16** (3):201–222.

[20] Chance, B. 1943. Journal of Biological Chemistry **151** (2):553–577.

[21] Clarke, B. L. 1980. *Stability of complex reaction networks.*, vol. 42 of *Adv. Chem. Phys.* Wiley, New York.

[22] Cohen, P. 2000. Trends in Biochemical Sciences **25** (12):596–601.

[23] Cohen, S. D., and A. C. Hindmarsh. 1996. Comput. Phys. **10**:138—–143.

[24] de Graauw, M. , (ed.), 2009. *Phospho-Proteomics*, vol. 527 of *Methods in Molecular Biology*. Humana Press.

[25] Deckard, A., F T Bergmann, and H M Sauro. 2006. Bioinformatics **22** (23):2966–2967.

[26] Dickson, R. C., and M. D. Mendenhall. , (ed.), 2004. *Signal Transduction Protocols*, vol. 284 of *Methods in Molecular Biology*. Humana Press, 2nd Edition.

[27] Dormand, J. R., and Peter J Prince. 1980. Journal of computational and applied mathematics **6** (1):19–26.

[28] Efron, B., and Robert Tibshirani. 1986. Statistical science :54–75.

[29] Entus, R., B Aufderheide, and H. M. Sauro. 2007. Systems and Synthetic Biology **10.1007/s11693-007-9008-6**.

[30] Fields, S., and O Song. 1989. Nature **340** (6230):245–246.

[31] Friedland, A. E., Timothy K Lu, Xiao Wang, David Shi, George Church, and James J Collins. 2009. Science **324** (5931):1199–1202.

[32] Gama-Castro, S., Verónica Jiménez-Jacinto, Martín Peralta-Gil, Alberto Santos-Zavaleta, Mónica I Peñaloza-Spinola, Bruno Contreras-Moreira, Juan Segura-Salazar, Luis Muñiz-Rascado, Irma Martínez-Flores, Heladia Salgado, César Bonavides-Martínez, Cei Abreu-Goodger, Carlos Rodríguez-Penagos, Juan Miranda-Ríos, Enrique Morett, Enrique Merino, Araceli M Huerta, Luis Treviño-Quintanilla, and Julio Collado-Vides. 2008. Nucleic Acids Res **36** (Database issue):D120–D124.

[33] Garfinkel, D. 1968. Comput. Biomed. Res. **2**:31–44.

[34] Garfinkel, D., L Garfinkel, M Pring, S B Green, and B Chance. 1970. Annu Rev Biochem **39**:473–498.

[35] Gauges, R., U. Kummer, S. Sahle, and K. Wegner. 2006. Bioinformatics **22** (15):1879–1885.

[36] Gavin, A.-C., Patrick Aloy, Paola Grandi, Roland Krause, Markus Boesche, Martina Marzioch, Christina Rau, Lars Juhl Jensen, Sonja Bastuck, Birgit Dümpelfeld, Angela Edelmann, Marie-Anne Heurtier, Verena Hoffman, Christian Hoefert, Karin Klein, Manuela Hudak, Anne-Marie Michon, Malgorzata Schelder, Markus Schirle, Marita Remor, Tatjana Rudi, Sean Hooper, Andreas Bauer, Tewis Bouwmeester, Georg Casari, Gerard Drewes, Gitte Neubauer, Jens M Rick, Bernhard Kuster, Peer Bork, Robert B Russell, and Giulio Superti-Furga. 2006. Nature **440** (7084):631–636.

[37] Gekas, V., and M. Lopez-Leiva. 1985. Process biochemistry **20** (1):2–12.

[38] Goldberg, D. 1989. Addison Wesley, New York. Eiben AE, Smith JE (2003) Introduction to Evolutionary Computing. Springer. Jacq J, Roux C (1995) Registration of non-segmented images using a genetic algorithm. Lecture notes in computer science **905**:205–211.

[39] Goodyear, C., and G.J. Silverman. 2008. Cold Spring Harbor Protocols **2008** (9).

[40] Hedley, W. J., N. R. Melanie, D. Bullivant, A. Cuellar, Yi Ge, M. Grehlinger, K. Jim, S. Lett, D. Nickerson, P. Nielsen, and H. Yu. 2001. Available via the World Wide Web at `http://www.cellml.org`.

[41] Heinrich, R., S. M. Rapoport, and T. A. Rapoport. 1977. Prog. Biophys. Molec. Biol. **32**:1–82.

[42] Herrera, F., Manuel Lozano, and Jose L. Verdegay. 1998. Artificial intelligence review **12** (4):265–319.

[43] Hindmarsh, A. C. 1983. *In:* Stepleman, R. , (ed.), Scientific Computing, p. 55–64. North-Holland, Amsterdam.

[44] Hoefnagel, M., A Van Der Burgt, DE Martens, J Hugenholtz, and JL Snoep. 2002. Molecular biology reports **29** (1-2):157–161.

[45] Hofmeyr, J. H. 1995. J Bioenerg Biomembr **27** (5):479–490.

[46] Hofmeyr, J. H., and K. J. van der Merwe. 1986. Comp. Appl. Biosci. **2**:243–249.

[47] Hoops, S., S Sahle, R Gauges, C Lee, J Pahle, N Simus, M Singhal, L Xu, P Mendes, and U Kummer. 2006. Bioinformatics **22** (24):3067–3074.

[48] Hucka, M., A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L.

Kasberger, A. Kremling, U. Kummer, N. Le NovÃÆŠÂÂÍre, L. M. Loew, D. Lucio, P. Mendes, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. 2003. Bioinformatics **19**:524–531.

[49] Huerta, A. M., H. Salgado, D. Thieffry, and J. Collado-Vides. 1998. Nucleic Acids Res **26** (1):55–59.

[50] Ito, T., T. Chiba, R. Ozawa, M. Yoshida, M. Hattori, and Y. Sakaki. 2001. Proc Natl Acad Sci U S A **98** (8):4569–4574.

[51] Jeong, H., S. P. Mason, A. L. Barabási, and Z. N. Oltvai. 2001. Nature **411** (6833):41–42.

[52] Karp, P. D., I. M. Keseler, A. Shearer, M. Latendresse, M. Krummenacker, S. M. Paley, I. Paulsen, J. Collado-Vides, S. Gama-Castro, M. Peralta-Gil, A. Santos-Zavaleta, M. I. Peñaloza-Spínola, C. Bonavides-Martinez, and J. Ingraham. 2007. Nucleic Acids Res **35** (22):7577–7590.

[53] Keseler, I. M., Julio Collado-Vides, Alberto Santos-Zavaleta, Martin Peralta-Gil, Socorro Gama-Castro, Luis Muñiz-Rascado, César Bonavides-Martinez, Suzanne Paley, Markus Krummenacker, Tomer Altman, Pallavi Kaipa, Aaron Spaulding, John Pacheco, Mario Latendresse, Carol Fulcher, Malabika Sarker, Alexander G Shearer, Amanda Mackie, Ian Paulsen, Robert P Gunsalus, and Peter D Karp. 2011. Nucleic Acids Res **39** (Database issue):D583–D590.

[54] Kirkpatrick, S., D. Gelatt Jr., and Mario P Vecchi. 1983. science **220** (4598):671–680.

[55] Kitano, H., A Funahashi, Y Matsuoka, and K Oda. 2005. Nat Biotechnol **23** (8):961–966.

[56] Kroeze, W. K., Douglas J Sheffler, and Bryan L Roth. 2003. Journal of Cell Science **116** (24):4867–4869.

[57] Krogan, N. J., Gerard Cagney, Haiyuan Yu, Gouqing Zhong, Xinghua Guo, Alexandr Ignatchenko, Joyce Li, Shuye Pu, Nira Datta,

Aaron P Tikuisis, Thanuja Punna, José M Peregrín-Alvarez, Michael Shales, Xin Zhang, Michael Davey, Mark D Robinson, Alberto Paccanaro, James E Bray, Anthony Sheung, Bryan Beattie, Dawn P Richards, Veronica Canadien, Atanas Lalev, Frank Mena, Peter Wong, Andrei Starostine, Myra M Canete, James Vlasblom, Samuel Wu, Chris Orsi, Sean R Collins, Shamanta Chandran, Robin Haw, Jennifer J Rilstone, Kiran Gandi, Natalie J Thompson, Gabe Musso, Peter St Onge, Shaun Ghanny, Mandy H Y Lam, Gareth Butland, Amin M Altaf-Ul, Shigehiko Kanaya, Ali Shilatifard, Erin O'Shea, Jonathan S Weissman, C. James Ingles, Timothy R Hughes, John Parkinson, Mark Gerstein, Shoshana J Wodak, Andrew Emili, and Jack F Greenblatt. 2006. Nature **440** (7084):637–643.

[58] Kuzmič, P. 1996. Analytical biochemistry **237** (2):260–273.

[59] Le Novère, N., Andrew Finney, Michael Hucka, Upinder S Bhalla, Fabien Campagne, Julio Collado-Vides, Edmund J Crampin, Matt Halstead, Edda Klipp, Pedro Mendes, Poul Nielsen, Herbert Sauro, Bruce Shapiro, Jacky L Snoep, Hugh D Spence, and Barry L Wanner. 2005. Nature biotechnology **23** (12):1509–1515.

[60] Le Novère, N., M. Hucka, H. Mi, S. Moodie, F. Schreiber, A. Sorokin, E. Demir, K. Wegner, M.I. Aladjem, S.M. Wimalaratne, *et al.*. 2009. Nature biotechnology **27** (8):735–741.

[61] Lee, T. I., Nicola J Rinaldi, François Robert, Duncan T Odom, Ziv Bar-Joseph, Georg K Gerber, Nancy M Hannett, Christopher T Harbison, Craig M Thompson, Itamar Simon, Julia Zeitlinger, Ezra G Jennings, Heather L Murray, D. Benjamin Gordon, Bing Ren, John J Wyrick, Jean-Bosco Tagne, Thomas L Volkert, Ernest Fraenkel, David K Gifford, and Richard A Young. 2002. Science **298** (5594):799–804.

[62] Lloyd, C. M., J R Lawson, P J Hunter, and P F Nielsen. 2008. Bioinformatics .

[63] Longabaugh, W. J., Eric H Davidson, and Hamid Bolouri. 2005. Developmental biology **283** (1):1–16.

[64] Luciano, J. S., and R D Stevens. 2007. BMC Bioinformatics **8 Suppl 3**:8 Suppl 3: S3.

[65] Macek, B., F. Gnad, B. Soufi, C. Kumar, J.V. Olsen, I. Mijakovic, and M. Mann. 2008. Molecular & Cellular Proteomics **7** (2):299.

[66] Mangan, S., S Itzkovitz, A Zaslaver, and U Alon. 2006. J Mol Biol **356** (5):1073–1081.

[67] Manninen, T., E Makiraatikka, A Ylipaa, A Pettinen, K Leinonen, and M L Linne. 2006. Conf Proc IEEE Eng Med Biol Soc **1**:2013–2016.

[68] Manning, G., D. B. Whyte, R. Martinez, T. Hunter, and S. Sudarsanam. 2000. Science **298**:1912–1934.

[69] Mardis, E. R. 2007. Nat Methods **4** (8):613–614.

[70] Marquardt, D. 1963. J. Soc. Ind. Appl. Math **11** (2):431–441.

[71] Mattick, J. 2004. Pharmacogenomics J **4**:9–16.

[72] Milo, R., S. Itzkovitz, N. Kashtan, R. Levitt, S. Shen-Orr, I. Ayzenshtat, M. Sheffer, and U. Alon. 2004. Science **303** (5663):1538.

[73] Milo, R., N. Kashtan, S. Itzkovitz, MEJ Newman, and U. Alon. 2003. eprint arXiv: cond-mat/0312028 .

[74] Milo, R., S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. 2002. Science **298**:824–7.

[75] Monod, J., J Wyman, and J. P. Changeux. 1965. J Mol Biol **12**:88–118.

[76] Montgomery, D. C., Elizabeth A Peck, and G Geoffrey Vining. 2012. *Introduction to linear regression analysis*, vol. 821. Wiley.

[77] Müller, T., N Noykova, M Gyllenberg, and J Timmer. 2002. Mathematical Biosciences **177**:147–160.

[78] Myers, C. J., Nathan Barker, Kevin Jones, Hiroyuki Kuwahara, Curtis Madsen, and Nam-Phuong D Nguyen. 2009. Bioinformatics **25** (21):2848–2849.

[79] Nelder, J., and R. Mead. 1965. The Computer Journal **7** (4):308.

[80] Olivier, B., and J.L. Snoep. 2004. Bioinformatics **20** (13):2143–2144.

[81] Pahle, J. 2009. Briefings in bioinformatics **10** (1):53–64.

[82] Park, D. J. M., and B. E Wright. 1973. Comput. Progm. Biomed. **3**:10–26.

[83] Pedersen, M., and G. Plotkin. 2008. *In:* Computational Methods in Systems Biology . Springer,. in press, http://homepages.inf.ed.ac.uk/s0677975/papers/lbs.pdf.

[84] Phizicky, E., P.I.H. Bastiaens, H. Zhu, M. Snyder, and S. Fields. 2003. Nature **422** (6928):208–215.

[85] Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. 1988. *Numerical Recipies in C. The Art of Scientific Computing.* Cambridge University Press, Cambridge.

[86] Ptacek, J., G. Devgan, G. Michaud, H. Zhu, X. Zhu, J. Fasolo, H. Guo, G. Jona, A. Breitkreutz, R. Sopko, *et al.*. 2005. Nature **438** (7068):679–684.

[87] Ptacek, J., and M. Snyder. 2006. Trends in Genetics **22** (10):545–554.

[88] Ramsey, S., David Orrell, and Hamid Bolouri. 2005. Journal of bioinformatics and computational biology **3** (02):415–436.

[89] Rawlings, J. B., and John G Ekerdt. 2002. *Chemical reactor analysis and design fundamentals.* Nob Hill Pub.

[90] Ren, B., F. Robert, J. J. Wyrick, O. Aparicio, E. G. Jennings, I. Simon, J. Zeitlinger, J. Schreiber, N. Hannett, E. Kanin, T. L. Volkert, C. J. Wilson, S. P. Bell, and R. A. Young. 2000. Science **290** (5500):2306–2309.

[91] Sauro, H. M. 2000. *In:* Hofmeyr, J.-H. S., J. M. Rohwer, and J. L. Snoep. , (ed.), Animating the Cellular Map: Proceedings of the 9th International Meeting on BioThermoKinetics, . Stellenbosch University Press.

[92] Sauro, H. M. 2011a. *Enzyme Kinetics for Systems Biology*. Ambrosius Publishing. First Edition.

[93] Sauro, H. M. 2011b. *Enzyme Kinetics for Systems Biology*. Ambrosius Publishing. 2nd Edition.

[94] Sauro, H. M., and J. Barrett. 1995. Molecular and Cellular Biochemistry **145**:141–150.

[95] Sauro, H. M., and D. A. Fell. 1991. Mathl. Comput. Modelling **15**:15–28.

[96] Sauro, H. M., M. Hucka, A. Finney, C. Wellock, H. Bolouri, J. Doyle, and H. Kitano. 2003. OMICS **7(4)**:355–372.

[97] Sauro, H. M., and B. N. Kholodenko. 2004. Prog Biophys Mol Biol. **86**:5–43.

[98] Seshasayee, A. S. N., P. Bertone, G. M. Fraser, and N.M. Luscombe. 2006. Current Opinion in Microbiology **9** (5):511–519.

[99] Shen-Orr, S. S., R. Milo, S. Mangan, and U. Alon. 2002. Nature Genetics **31**:64–68.

[100] Siegal, M. L., Daniel E L Promislow, and Aviv Bergman. 2007. Genetica **129** (1):83–103.

[101] Smith, G. P. 1985. Science **228** (4705):1315–1317.

[102] Straume, M., and ML Johnson. 1992. Methods in enzymology **210**:87.

[103] Straume, M., and Michael L Johnson. 2010. Essential Numerical Computer Methods :55.

[104] Taft, R., and JS Mattick. 2004. Arxiv preprint q-bio.GN/0401020 .

[105] Toledo, F., and Geoffrey M Wahl. 2006. Nat Rev Cancer **6** (12):909–923.

[106] Trafton, A. 2012. http://www.mit.edu/newsoffice/2011/brain-chip-1115.html.

[107] Uetz, P., L. Giot, G. Cagney, T. A. Mansfield, R. S. Judson, J. R. Knight, D. Lockshon, V. Narayan, M. Srinivasan, P. Pochart, A. Qureshi-Emili, Y. Li, B. Godwin, D. Conover, T. Kalbfleisch, G. Vijayadamodar, M. Yang, M. Johnston, S. Fields, and J. M. Rothberg. 2000. Nature **403** (6770):623–627.

[108] Wang, E., R. Sandberg, S. Luo, I. Khrebtukova, L. Zhang, C. Mayr, S.F. Kingsmore, G.P. Schroth, and C.B. Burge. 2008. Nature **456**:470–476.

[109] Wikipedia 2013a. http://en.wikipedia.org/wiki/Analog_computer. [Online; accessed 4-January-2013].

[110] Wikipedia 2013b. http://en.wikipedia.org/wiki/Antikythera_mechanism. [Online; accessed 4-January-2013].

[111] Wikipedia 2013c. http://en.wikipedia.org/wiki/Cellular_automaton. [Online; accessed 4-January-2013].

[112] Wikipedia 2013d. http://en.wikipedia.org/wiki/Difference_engine. [Online; accessed 4-January-2013].

[113] Wikipedia 2013e. http://en.wikipedia.org/wiki/Differential_analyser. [Online; accessed 4-January-2013].

[114] Wikipedia 2013f. http://en.wikipedia.org/wiki/Emergence. [Online; accessed 4-January-2013].

[115] Wikipedia 2013g. http://en.wikipedia.org/wiki/Fractal. [Online; accessed 4-January-2013].

[116] Wikipedia 2013h. http://en.wikipedia.org/wiki/MONIAC-_Computer. [Online; accessed 4-January-2013].

[117] Wikipedia 2013i. http://en.wikipedia.org/wiki/Neural_networks. [Online; accessed 4-January-2013].

[118] Wikipedia 2013j. http://en.wikipedia.org/wiki/Rangekeeper. [Online; accessed 4-January-2013].

[119] Wikipedia 2013k. http://en.wikipedia.org/wiki/Slide_rule. [Online; accessed 4-January-2013].

[120] Wikipedia 2013l. http://en.wikipedia.org/wiki/Tide-_predicting_machine. [Online; accessed 4-January-2013].

[121] Wikipedia 2013m. http://en.wikipedia.org/wiki/V-2a. [Online; accessed 4-January-2013].

[122] Wilkinson, D. J. 2007. `http://www.staff.ncl.ac.uk/d.j.wilkinson/software/sbml-sh/`.

[123] Wilkinson, D. J. 2012. *Stochastic Modelling for Systems Biology*. Chapman & Hall/CRC Press, Boca Raton, Florida, 2nd edition.

History

1. VERSION: 0.9 (PASI 2013 Edition)

   Date: 2013-030-6

   Author(s): Herbert M. Sauro

   Title: Essentials for Biochemical Modeling

   Modification(s): First Edition Release