# Stiff Problems in Numerical Simulation of Biochemical and Gene Regulatory Networks

by

Yihai Yu

(Under the direction of Thiab R. Taha)

## Abstract

With the sequence of genomes of many organisms now available, the major challenge of functional genomics is 'reassembling the pieces'. A chemical reaction network is considered to be a very simple and efficient view of a living system. The goal here is to be able to simulate an arbitrary ensemble of hypothesized biochemical and gene regulatory networks to predict what a cell is doing and compare them with the observed data. Any biological network can be modeled as a system of Ordinary Differential Equations (ODEs) mathematically. There are a number of standard ODE solvers, such as the Euler and Runge Kutta (RK) methods. But from time to time, these methods could be very inefficient for some special systems, called stiff systems. We consider other special ODE solvers, such as Livermore Solver for ODE with General Sparse Jacobian Matrices (LSODES). A simulator KINSOLVER with 4 integration options (Euler, Modified Euler, RK, Adaptive RK-Fehlberg) is now updated by adding LSODES. I also presented the theoretical and numerical work on the stiffness of some biological circuits like *qa* gene cluster, *lac* operon and oregonator. Performance of different methods including MATLAB ODE suite are compared too. The efficiency of computation could be improved by a factor of 10 compared to the standard $4^{th}$ order RK method.

Index words: Ordinary Differential Equation, Biological Circuits, Stiff, Chemical Reaction Network

# Stiff Problems in Numerical Simulation of Biochemical and Gene Regulatory Networks

by

Yihai Yu

B.S., Fudan University, Shanghai, China, 2000

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2004

# Stiff Problems in Numerical Simulation of Biochemical and Gene Regulatory Networks

by

Yihai Yu

Approved:

Major Professor: Thiab R. Taha

Committee: Daniel M. Everett
Jonathan Arnold

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2004

## Acknowledgments

I would like to thank Dr. Thiab Taha to advise me on this interesting project. I would also thank Dr. H. B. Schüttler and Dr. Jonathan Arnold for their advice and kindness.

Thanks for Mr. Boanerges Aleman-Meza who implemented the first version of KINSOLVER and gave me a lot of help for my further work. Thanks for Dr. Jaxk Reeves, Dr. Suchendra Bhandarkar, and Dr. Daniel Everett.

Thanks for my parents, my wife and all my friends!

Table of Contents

# List of Figures

## List of Tables

# Chapter 1

# Introduction

A biological system can be viewed as a chemical reaction network [3]. One of the central problems of functional genomics then becomes the identification of biochemical and gene regulatory networks describing how a living system functions [8]. Validating a biological circuit depends upon our ability to simulate a particular reaction network and to predict how the network responds to various experimental perturbations.

The study of living structures as reaction networks has utilized several well-studied paradigms like the *lac* operon [9], *trp* operon [10, 11], *GAL* gene cluster [12], *qa* gene cluster [13], cell cycle [14], and biological clock [15]. New simulation tools are needed to facilitate our *Computing Life* project at The University of Georgia with these systems.

In order to refine and examine the behavior of a biological circuit in a genomic context, an efficient general purpose simulator is needed to compute life. The KINSOLVER [6] is already available to simulate biological circuits represented by coupled nonlinear differential equations, i.e. compute the time dependant concentrations of each species from a simple interface for specifying and refining the target reaction network. A Monte Carlo ensemble method [4] is proposed to identify the biological circuits statistically, which consists of a probability distribution over the parameter space of possible models. This Monte Carlo method is basically doing "*guess*" the best model thousands of times, and each time a "*guess*" involves solving the ODE once.

But the problem now facing us is that sometimes it just takes too much time to compute a fit due to various reasons, one of which is called stiffness. For a reasonably small biological system such as biological clock for *Neuropora crassa* [18], it takes almost one month to do the ensemble simulation with 40,000 sweeps through the parameter space, hunting for a good fit.

The numerical method for solving ODE systems is a very old topic. The methods discovered in last century are still the basis of the most effective codes[19]. There is a certain class of problems called stiff problems, which are too expensive to solve due to the inherent difficulty they present to conventional methods, no matter how great an improvement in hardware becomes available. Even if you can bear the expense, conventional methods have to take so many steps that the roundoff errors may invalidate the solution. As a matter of fact, most stiff systems are the systems with physical components with greatly different time constants. And most of the biochemical and gene regulatory systems exhibit this property.

There are a number of software packages designed to address stiff problems, such as MATLAB ODE suite (ode15s, ode23s) [20], GEAR and its descendants [21], LSODE and its variants [22]. The KINSOLVER updated by LSODES successfully and efficiently solves many stiff/non-stiff systems as shown in Section 4.4, and is comparable in performance to the widely accepted commercial package MATLAB. The performance can be improved by a factor of 10 compared to the classical 4th order RK method. This improvement is very significant for the MC ensemble method. In other words, the ensemble simulation will take less than one week instead of a month.

Chapter 2 is focused on building mathematical models of the biological systems and the numerical experiments showing the difficulty of stiff problems. Chapter 3 is devoted to the phenomenon of stiffness and why it is important and difficult in details. The method to detect stiffness is in Chapter 4. LSODES and its performance

are reported in Chapter 5. Future work is pointed out in Chapter 6. Chapter 7 gives the conclusion.

# Chapter 2

# Biological Model

## 2.1 Introduction to ODE

The Ordinary Differential Equation (ODE) is a very basic and useful mathematical model in many areas, such as economics, engineering, social science, physics, chemistry, biology, etc.. We focus on initial value problems (IVP), like that for a biochemical reaction network. Here is the canonical first order IVP in general normal form:

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y})$$
$$\mathbf{y}(t_0) = \mathbf{y_0}$$

where

$t$ is the time-like independent variable,

$\mathbf{y}$ is the column N-vector of dependant variables,

N is the size of the vector,

$\frac{d}{dt}$ denotes differentiation with respect to t,

$t_0$ is the initial time and $\mathbf{y_0}$ is the N-vector initial condition, and

$\mathbf{f}$ is a N-vector valued function of $t$ and $\mathbf{y}$.

### 2.1.1 Examples of ODEs

Some examples of ODEs follow:

- $\frac{du}{dt} = F(t)G(u)$, the Growth Equation;

- $\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin(\theta) = F(t)$, the Pendulum Equation;
- $L\frac{d^2Q}{dt^2} + R\frac{dQ}{dt} + \frac{Q}{C} = E(t)$, the LCR Oscillator Equation; and
- $\frac{d^2y}{dt^2} + \varepsilon(y^2+1)\frac{dy}{dt} + y = 0$, the van der Pol Equation;

where t is the independent variable, usually meaning time. $u$, $\theta$, $Q$, $y$ are dependant variables.

### 2.1.2 From high order to first order

The order of a differential equation is the order of the highest derivative in the equation, e.g. the Growth Equation is first order while the Pendulum Equation is a second order equation. But actually only first order ODEs are fundamental and we can construct the high order equations in terms of multiple first order equations.

Here is one example. Suppose we have a $4^{th}$ order ODE:

$$P(t)\frac{d^4y}{dt^4} + Q(t)\frac{d^3y}{dt^3} + R(t)\frac{d^2y}{dt^2} + S(t)\frac{dy}{dt} = f(t)$$

where $P(t)$, $Q(t)$, $R(t)$ and $S(t)$ are arbitrary functions, and $f(t)$ is determined by the actual solution. Suppose the solution is $y(t) = \sin(t)$, then we can immediately decide $f(t)$:

$$f(t) = P(t)\sin(t) - Q(t)\cos(t) - R(t)\sin(t) + S(t)\cos(t)$$

We are looking for 4 differential equations to represent this $4^{th}$ order equation, i.e. we have 4 $t$-dependant variables: $y_1$, $y_2$, $y_3$, $y_4$. And note that:

$$y_1 = y(t) = \sin(t)$$

$$\frac{dy_1}{dt} = \cos(t) = y_2$$

$$\frac{dy_2}{dt} = -\sin(t) = y_3$$

$$\frac{dy_3}{dt} = -\cos(t) = y_4$$

$$\frac{dy_4}{dt} = \sin(t)$$

Then the complete set of ODEs is:

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = y_3$$

$$\frac{dy_3}{dt} = y_4$$

$$\frac{dy_4}{dt} = \frac{f(t) - S(t)y_2 - R(t)y_3 - Q(t)y_4}{P(t)}$$

In other words, even if we pretend we don't know the solution for $f_1$ is $\sin(t)$, we'll find the desired solution if we are provided with the initial condition: $y_1(0) = 0$, $y_2(0) = 1$, $y_3(t) = 0$, $y_4(t) = -1$. So they are equivalent and only $1^{st}$ ODEs need to be considered.

### 2.1.3 Numerical methods v.s. analytical methods

Solving ODEs is a fundamental task in numerical analysis because only a very limited types of ODEs can be solved analytically. Most of the time, we have to refer to approximation methods, i.e. numerical integration. The basic idea is to discretize the IVP and approximate the exact solution. Analytical solutions are also important for checking the accuracy of numerical solution.

## 2.2 How to model biological circuits

A kinetics model is a specification of reactions between hypothesized molecular participants. The diagrammatic representation is like Figure 2.1. The species are represented as boxes. The reactions are represented by circles. The arrows indicate the

directions of reactions. Since any living system can be viewed as a chemical reaction network, a thorough understanding of how these models behave is important.

For one particular equation: $S_1 + S_2 \rightleftharpoons S_3 + S_4$, there are 4 species, and 2 reactions: forward and backward, indicated by $\rightleftharpoons$. Suppose the forward reaction rate is $k_f$ and backward reaction rate is $k_b$. We want to write out the differential equation for the species, e.g. $S_1$. In the forward reaction the species is consumed, therefore: $-k_f[S_1][S_2]$. For the backward reaction, $S_1$ is generated: $+k_b[S_3][S_4]$. Therefore we have:

$$\frac{dS_1}{dt} = -k_f[S_1][S_2] + k_b[S_3][S_4]$$

Similar equations for the other 3 species can be constructed. Usually the backward reaction rate is smaller, sometimes just 0.

### 2.2.1 Simple hydrogen combustion model

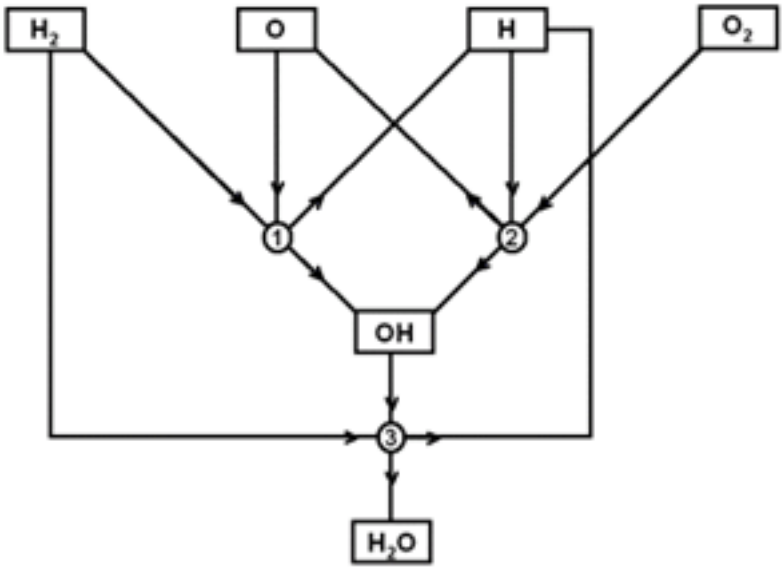


Figure 2.1: Simple Hydrogen Combustion Model.

Almost all of the examples in this paper relate to combustion to produce energy (sometimes for a cell), and so to illustrate the simplest kind of combustion the simple hydrogen combustion model as in Figure 2.1 is introduced, which is referred as Water Model I.

The 3 reactions are:

$$H_2 + O \rightleftharpoons H + OH$$

$$H + O_2 \rightleftharpoons O + OH$$

$$H_2 + OH \rightleftharpoons H_2O + H$$

Based on the method introduced in the beginning of Section 2.2, we can obtain the full multiplicative mass balance kinetics, i.e. the complete set of ODEs:

$$\frac{d[H_2]}{dt} = -k_{f1}[H_2][O] + k_{b1}[H][OH] - k_{f3}[H_2][OH] + k_{b3}[H_2O][OH]$$

$$\frac{d[O]}{dt} = -k_{f1}[H_2][O] + k_{b1}[H][OH] + k_{f2}[H][O_2] - k_{b2}[O][OH]$$

$$\frac{d[O_2]}{dt} = -k_{f2}[H][O_2] + k_{b2}[O][OH]$$

$$\frac{d[H]}{dt} = +k_{f1}[H_2][O] - k_{b1}[H][OH] - k_{f2}[H][O_2] + k_{b2}[O][OH] + k_{f3}[H_2][OH] - k_{b3}[H_2O][OH]$$

$$\frac{d[OH]}{dt} = +k_{f1}[H_2][O] - k_{b1}[H][OH] + k_{f2}[H][O_2] - k_{b2}[O][OH] - k_{f3}[H_2][OH] + k_{b3}[H_2O][OH]$$

$$\frac{d[H_2O]}{dt} = k_{f3}[H_2][OH] - k_{b3}[H_2O][OH]$$

where $k_{fi}$ and $k_{bi}$ are the forward and backward reaction rates respectively for reaction number $i$.

### 2.2.2 QUINIC ACID METABOLISM OF *Neurospora crassa*

The biological circuit in Figure 2.2 from [24] is a kinetic model of DNA, mRNA, and protein involved in carbon metabolism for of *Neurospora crassa* [23]. This *qa* gene

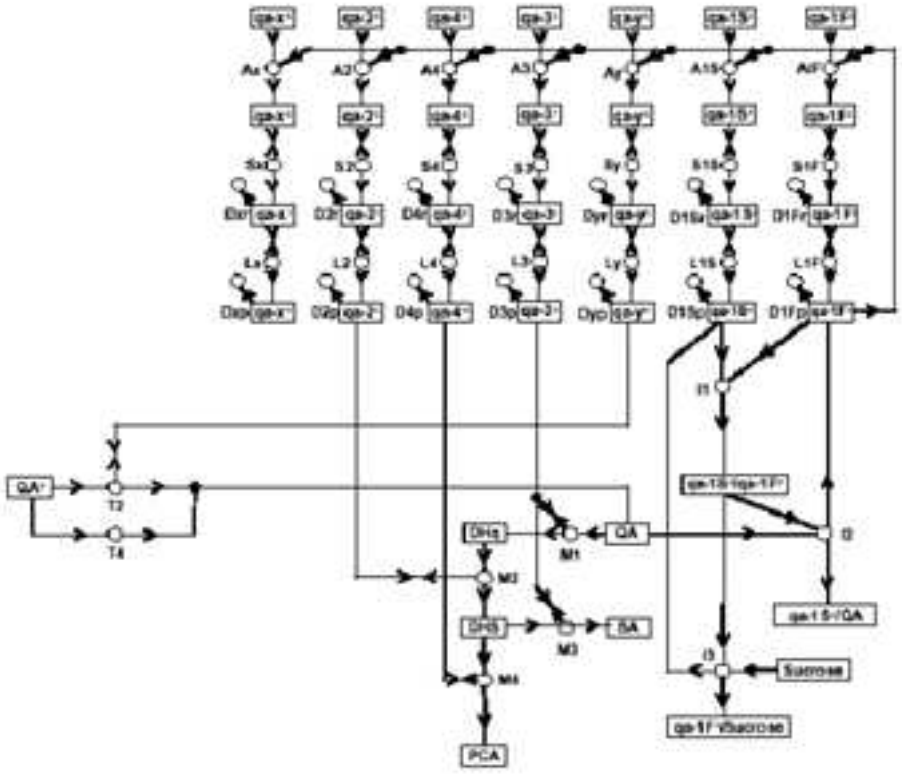


Figure 2.2: quinic acid metabolism of *Neurospora crassa*.

cluster is a good example because of its relative simplicity and because its circuit structure is shared with many other gene regulatory systems.

The first line of figure 2.2 is the set of inactive genes denoted by a superscript 0. These are activated to produce the second line denoted by a superscript 1. All the information is stored in a DNA molecule. Then segments of information are transcribed into mRNA, which is denoted by a superscript $r$. The next step is translation from mRNA to protein. Proteins can form structure or serve as enzymatic role in the cell to affect the system.

If we can identify the initial concentration of all the species and reaction rates of this *qa* cluster, then we can predict how the cell takes quinic acid as a sole carbon source, as food, and lives by solving the differential equations.

## 2.3 Conventional numerical methods for ODE

The numerical methods for ODEs can be categorized in many different ways: one-step v.s. multi-step, explicit method v.s. implicit method, etc.. Here we divide them into 2 groups: stiff methods and non-stiff methods. We talk about 3 conventional non-stiff methods in this section, and stiff methods in Chapter 3. For many biological circuits, they exhibit the property of stiffness and make the conventional methods to be very inefficient.

### 2.3.1 Euler method

The Euler method [25] is the simplest numerical method for solving ODEs which utilizes the first order of Taylor series expansion:

$$y_{n+1} = y_n + hf(t_n, y_n)$$

where $y_n$ is the solution at time $t_n$, and $y_{n+1}$ is the estimate of the solution of time $t_{n+1}$ based on $y_n$ and slope at $t_n$. $h$ is the step size of discretization.

### 2.3.2 Runge-Kutta method

The RK method [26] is a classical higher order method. Here is one example for $4^{th}$ order RK:

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

where

$$k_1 = hf(t_n, y_n)$$

$$k_2 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$$
$$k_3 = hf(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1)$$
$$k_4 = hf(t_n + h, y_n + k_3)$$

RK methods are a family of methods which are designed to approximate Taylor series using a linear combination values of $f(t, y)$ without requiring explicit evaluations of the high derivatives of $f(t, y)$. This method needs 4 derivative function evaluation (DFE)'s for $4^{th}$ RK. We can also make this method to be adaptive, which means the step size is adjusted according to the local error estimation and it requires 6 DFE's per step.

## 2.4 Numerical experiments using MATLAB

The non-stiff numerical methods do not blow up in the presence of stiffness, but they do become very inefficient, and possibly exceed the precision capability of a computer. The most straightforward way to tell a system is stiff or not is measuring the performance.

Here the MATLAB ODE suite is used for the numerical experiments for several reasons. MATLAB is the most popular numerical package ever designed, so the test can be accepted by most of the people in different areas. The ODE Suite has the most complete methods implemented ever including some stiff solvers, so we can test the performance fairly. The 'Eulermethod' is a solver in MATLAB which is an implementation of the classical Euler method. Both ode45 and ode23 are adaptive Runge Kutta methods, but based on Dormand-Prince [40, 41] pair and Bogacki-Shimpine [42, 43] pair, respectively. Ode15s is a stiff solver using numerical differentiation formulas [20] and ode23s is based on a Rosenbrock triple [20].

I tested 5 models: non-stiff/moderately stiff/stiff Water Model I's, van der Pol (vdp) model and the oregonator. Refer to Appendix A for description of all the

models. And the stiffness of these models will be calculated in Chapter 3. '-' means 'no applicable' and '!' means 'fail to solve'. Relative error is calculated relative to the solution of ode15s at (rtol=1E-12 and atol=1E-12). The tests are performed under Windows XP on Pentium III (CPU 930MHz, RAM 192MB).

- Non-stiff Water Model I

| **Method** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|
| | Steps | Time(s) | Steps | Time(s) |
| eulermethod | 24 | 5.0E-3 | 40 | 8.9E-3 |
| ode45 | 65 | 4.0E-2 | 117 | 7.5E-2 |
| ode23 | 19 | 5.1E-2 | 90 | 8.1E-2 |
| ode23s | 12 | 6.0E-2 | 14 | 8.0E-2 |
| ode15s | 14 | 6.5E-2 | 60 | 1.3E-1 |

Table 2.1: Simulate non-stiff Water Model I using MATLAB.

From the above table, we can notice that the performance of Euler method is the best, and RK methods can also solve the problem very efficiently. The stiff methods are worse than conventional methods due to the fact that they need extra computation designed for stiff systems.

- Moderately stiff Water Model I

| **Method** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|
| | Steps | Time(s) | Steps | Time(s) |
| eulermethod | 680 | 7.5E-2 | 40,000 | 3.8 |
| ode45 | 1,593 | 4.8E-1 | 1,625 | 5.1E-1 |
| ode23 | 523 | 3.4E-1 | 546 | 3.5E-1 |
| ode23s | 21 | 9.5E-2 | 397 | 9.2E-1 |
| ode15s | 35 | 8.0E-2 | 146 | 1.7E-1 |

Table 2.2: Simulate moderately stiff Water Model I using MATLAB.

When it comes to low accuracy, all methods perform pretty similarly to each other. The Euler method is even a little better. For high accuracy, the stiff

solver ode15s is preferred, which is much better than the Euler method. This is reasonable because Euler method has lower order of accuracy.

- Stiff Water Model I

| **Method** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|
| | Steps | Time(s) | Steps | Time(s) |
| eulermethod | 499,800 | 5.6E+1 | 550,000 | 6.2E+1 |
| ode45 | ! | $> 3E+2$ | ! | $> 3E+2$ |
| ode23 | ! | $> 3E+2$ | ! | $> 3E+2$ |
| ode23s | 28 | 9.0E-2 | 2,346 | 5.3 |
| ode15s | 50 | 6.9E-2 | 282 | 2.6E-1 |

Table 2.3: Simulate stiff Water Model I using MATLAB.

This is a very challenging stiff model. The Euler method is able to find the solution very expensively, while the other 2 conventional methods fail to find the solution. The stiff solver ode15s is 100 to 1000 faster than the Euler method.

- vdp1000 Model

| **Method** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|
| | Steps | Time(s) | Steps | Time(s) |
| eulermethod | ! | $> 3E+2$ | ! | $> 3E+2$ |
| ode45 | ! | $> 3E+2$ | ! | $> 3E+2$ |
| ode23 | ! | $> 3E+2$ | ! | $> 3E+2$ |
| ode23s | 744 | 1.3 | 65,536 | 1.8E+2 |
| ode15s | 927 | 1.2 | 4,056 | 4.1 |

Table 2.4: Simulate vdp1000 model using MATLAB.

This is a classical stiff problem, and all of the conventional methods fail as expected. The stiff solver ode15s again performs better than ode23s at high accuracy, and almost the same as ode23s at low accuracy, which proves the statement in [20] that we shall use ode23s for low accuracy and use ode15s for high accuracy.

- Oregonator Model

| **Method** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|
| | Steps | Time(s) | Steps | Time(s) |
| eulermethod | 100,000 | 9.8 | ! | $> 3E+2$ |
| ode45 | 46,073 | 1.5E+1 | 65,536 | 7.8E+1 |
| ode23 | 16,563 | 1.0E+1 | ! | $> 3E+2$ |
| ode23s | 7,127 | 1.3E+1 | 37,571 | 7.2E+1 |
| ode15s | 3,331 | 4.3 | 13,919 | 1.4E+1 |

Table 2.5: Simulate Oregonator model using MATLAB.

This is an oscillatory system which exhibits some stiffness as clarified in Section 4.3. Fast oscillation also makes the system difficult to solve. For low accuracy, the Euler method is pretty good, although not as good as ode15s due to stiffness. For high accuracy, ode15 again dominates the performance.

As shown in the tables above, the performance of the conventional methods are very inefficient compared to the stiff methods on stiff problems, though they are better when it comes to the non-stiff problem due to the fact that stiff methods are more expensive computationally. The stiff solver ode15s is the best as it outperformed the ode23s in most of the cases. Now more careful consideration is demanded to detect and solve stiff systems.

# Chapter 3

# The Notion of Stiffness

## 3.1 What is stiff

Before we analyze general stiff systems and give a formal description of stiffness, let us have a look at a simple example. A typical stiff differential equation can be given by:

$$\frac{dy}{dt} = -10^3[y - \exp(-t)] - \exp(-t)$$

$$y(0) = 0$$

where $y$ is a scalar for simplicity. The exact solution is:

$$y(t) = \exp(-t) - \exp(-10^3 t)$$

which consists of two components: $\exp(-t)$ and $\exp(-10^3 t)$. Obviously, $\exp(-10^3 t)$ varies more rapidly than $\exp(-t)$. This system consists of two normal modes: $\tau_1 = 1$ and $\tau_2 = 10^{-3}$. In order to reach equilibrium, we have to consider the time scale $\tau_1$, while we also have to consider $\tau_2$ for the sake of step size of time. Apparently, if we make the step size too big, it will invalidate the conventional numerical methods. Hence the step size has to be small, about the same scale as $\tau_2$. But the total integration time needed to reach equilibrium is much bigger than the step size. As a consequence, we have to wait an unbearably long time.

Let us consider the stiffness more generally in the IVP in general form in Section 2.1. Because each time we make an estimate to approximate the next step $y_{n+1}$, we

make inevitable errors, we must consider the behavior of solutions nearby the exact one. If we make no further errors, then we just follow this new integral curve, and the resulting error depends on this local error. This is sometimes called the Butterfly Effect. Here is one example:

$$\frac{dy}{dt} = A(y - p(t)) + \frac{dp(t)}{dt}$$

$$y(0) = v$$

where $A$ is a constant. The exact solution is:

$$y(t) = (v - p(0)) \exp(At) + p(t)$$

If $A$ is large and positive, the solution curves diverge and we call it unstable. This kind of problem is difficult to solve for any step-by-step numerical methods in general. If $A$ is small in magnitude, the curves are more or less parallel to the exact solution, because the first term doesn't contribute very much to the numerical estimation if the time span isn't unreasonably huge. We call this class of problem as neutrally stable, and they are easy to solve by conventional methods. If $A$ is negative and large in magnitude, all the solution curves are identical to the particular solution $p(x)$. This is called super-stable after a short initial transient depending on the magnitude of $A$. Because the first exponential term is essentially 0.

The behavior of the solutions near a particular solution $\mathbf{g}(t)$ can be studied by Taylor series expansion:

$$\frac{d\mathbf{y}}{dt} = \mathbf{J}(t, \mathbf{g}(t))(\mathbf{y} - \mathbf{g}(t)) + \mathbf{f}(x, \mathbf{g}(t))$$

$$= \mathbf{J}(t, \mathbf{g}(t))(\mathbf{y} - \mathbf{g}(t)) + \frac{d\mathbf{g}(t)}{dt}$$

where the Jacobian matrix **J** has its $(i, j)$ entry as the partial derivative of the $i^{th}$ component of $f$ with respect to the $j^{th}$ component of $y$:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} & \cdots & \frac{\partial f_1}{\partial y_n} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} & \cdots & \frac{\partial f_2}{\partial y_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial y_1} & \frac{\partial f_n}{\partial y_2} & \cdots & \frac{\partial f_n}{\partial y_n} \end{bmatrix}$$

This is actually the general vector form of the previous scalar example, and can be uncoupled into a set of equations of scalar form with the eigenvalues of **J** to be $A$'s.

A formal definition of stiffness was given by L. F. Shampine and C. W. Gear [28]:

> "By a stiff problem we mean one for which no solution component is unstable (no eigenvalue has a real part which is at all large and positive) and at least some component is very stable (at least one eigenvalue has a real part which is large and negative). Further, we will not call a problem stiff unless its solution is slowly varying with respect to the most negative real part of the eigenvalues."

In other words, stiffness is defined in relation to the most negative mode and the time span we are interested in.

## 3.2 How stiffness makes difference

Let's try to integrate the example in Section 3.1 by the Euler method, which is $y_{n+1} = y_n + hf_n = y_n + h\frac{dy_n}{dt}$. Compare this with the general Taylor series expansion:

$$y_{n+1} = y_n + h\frac{dy_n}{dt} + \frac{1}{2}h^2\frac{d^2y_n}{dt^2} + O(h^3),$$

we can get that the local truncation error we make here:

$$\varepsilon = |\frac{1}{2}h^2\frac{d^2y_n}{dt^2}|.$$

At the same time, pretend we know the exact solution and take derivatives twice:

$$\frac{d^2y(t)}{dt^2} = (v - p(0))A^2 \exp(At) + \frac{d^2p(t)}{dt^2}$$

Therefore:

$$h = \left( \frac{2\varepsilon}{|(v - p(0))A^2 \exp(At) + \frac{d^2p(t)}{dt^2}|} \right)^{\frac{1}{2}}.$$

When $t$ is very small and $|\frac{d^2p(t)}{dt^2}|$ is also very small compared to $|A|$:

$$h \doteq \left( \frac{2\varepsilon}{|(v - p(0))A^2|} \right)^{\frac{1}{2}}.$$

When $t$ is large, we have:

$$h \doteq \left( \frac{2\varepsilon}{|\frac{d^2p(t)}{dt^2}|} \right)^{\frac{1}{2}}$$

where the exponential part vanishes for negative $A$. So, initially, the step size is limited by the local truncation error and the eigenvalue $A$. After the initial transient, the step size is independent of the $A$ and eventually becomes large as long as it can satisfy the local error estimate $\varepsilon$ for the adaptive method. There are two main factors affecting the size of the step: accuracy and stability.

### 3.2.1 Accuracy and stability

Accuracy refers to the smallness of the local error, and stability refers to errors not growing in subsequent steps. We shall consider the "test equation": $\frac{dy}{dt} = Ay$. We want to identify the absolute stable area, in other words what is the criterion for $A$ and $h$.

- For the Euler method, the corresponding formula is:

$$y_{n+1} = y_n + Ahy_n$$

$$= (1 + Ah)y_n.$$

Suppose $\{y_n\}$ $(n = 0, 1, 2 \ldots)$ is the solution and $\{y_n + \varepsilon_n\}$ is the corresponding error introduced in each step. Also assume $\varepsilon_0 \neq 0$ and that there is no further error introduced other than this initial error. Then we have:

$$y_{n+1} + \varepsilon_{n+1} = (1 + Ah)(y_n + \varepsilon_n).$$

The difference between the previous 2 equations gives us:

$$\varepsilon_{n+1} = (1 + Ah)\varepsilon_n.$$

Then recursively, we obtain:

$$\varepsilon_n = \varepsilon_0(1 + Ah)^n, \ n = 0, 1, 2 \ldots .$$

This tells us that what is the global error after the $n^{th}$ step due to the only error at the initial point. The criterion for absolute stable area is evidently:

$$|1 + Ah| \leq 1,$$

in other words, $|\varepsilon_{n+1}| \leq |\varepsilon_n|$. If $|1 + Ah| > 1$, then the error will be amplified and eventually invalidate the solution. The absolute stable area is the area of a circle with radius 1 centered at $(-1, 0)$ in the complex surface of $Ah$ as A shown in Figure 3.1. Suppose $A = -18$, $h \leq 0.1$, then, $-1.8 \leq Ah \leq 0$ satisfies the criterion. If the equilibrium time is very big compared to 0.1, the method will just seem very inefficient because we have to maintain this small $h$ at any integration step. As one extreme example, if $A = -10^{16}$, we have to make $h$ at the order of $10^{-16}$ for this Euler method, which could blow out the limit of any computer's precision.

- For $4^{th}$ order RK method, we can apply the test equation $\frac{dy}{dt} = Ay$ too:

$$y_{n+1} = [1 + Ah + \frac{1}{2}(Ah)^2 + \frac{1}{6}(Ah)^3 + \frac{1}{24}(Ah)^4]y_n,$$

and the corresponding recursive relationship for error is:

$$\varepsilon_{n+1} = [1 + Ah + \frac{1}{2}(Ah)^2 + \frac{1}{6}(Ah)^3 + \frac{1}{24}(Ah)^4]\varepsilon_n.$$

Therefore the absolute stable area for $4^{th}$ order RK method is:

$$|1 + Ah + \frac{1}{2}(Ah)^2 + \frac{1}{6}(Ah)^3 + \frac{1}{24}(Ah)^4]| \leq 1,$$

as B shown in Figure 3.1. So, for a stiff problem, the stability restriction dominates the step size outside the transient region instead of the local truncation error control.

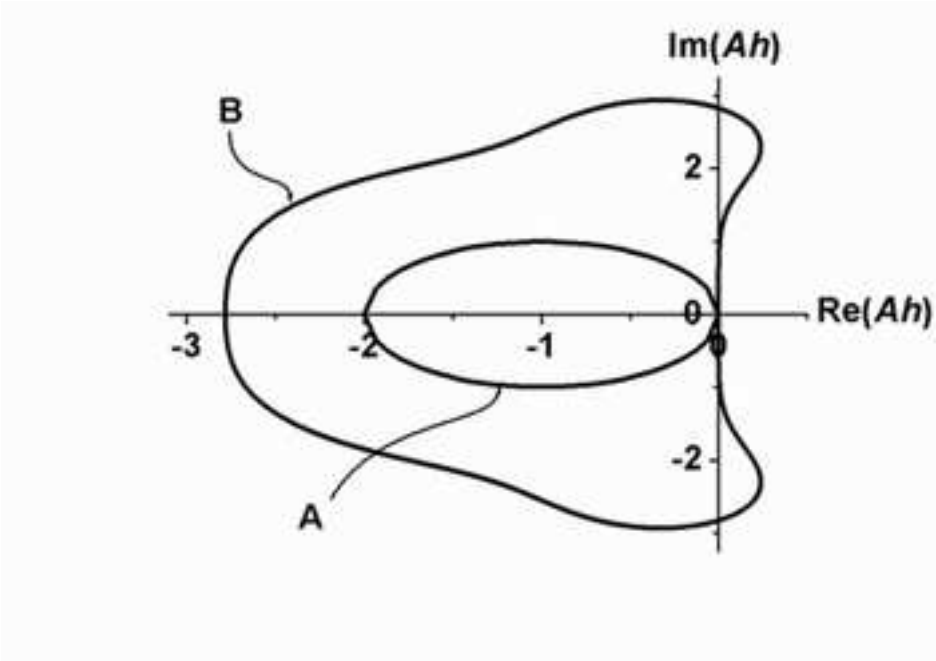


Figure 3.1: Absolute Stability Region for Euler and RK $4^{th}$ methods.

## 3.3 Monte Carlo ensemble method and stiffness

The goal of this *computing life* project is to identify the reaction networks of a living cell's full biochemical and gene regulatory circuitry [8]. A Monte Carlo ensemble

method [4] was proposed, which basically guesses the network thousands of times and gets a distribution of models that fit the data. For each "*guess*", the MC method executes the kinetic solver once. But it takes rather long time to "walk" through the parameter space, e. g. about 4 weeks for the biological clock model as in Appendix A for 40,000 sweeps using RK $4^{th}$ method. The reaction rates and the concentration of species are at very different order of magnitude for most of the actual biological systems, which means stiffness is a very important issue. If we can improve the kinetic solver by a factor of 10, then the time needed for the previous case will be less than a couple of days.

## 3.4 Evolutionary Algorithm method and stiffness

Evolutionary computation (EC) is a very active research area nowadays, which draws inspiration from the process of natural selection. Darwin's theory of evolution, **natural selection and survival of the fittest**, offers an explanation of the biological diversity and its underlying mechanism. EC methods have been used in a wide range of areas, such as electrical engineering, mechanical design, economics, etc. Amazingly, EC also benefits biological science a lot in return.

I have implemented Evolution Strategy (ES), which is one member of the family of evolutionary algorithm (EA), to this chemical reaction network problem. The fitness function is the comparison of the time dependent results out of KINSOLVER with the experimental data for each individual in the population. For more details, you can refer to Appendix B.

Like Monte Carlo ensemble method, ES executes KINSOLVER thousands of times searching for the best solutions. Stiffness could be a big problem again. Any small improvement for the kinetic solver will benefit the overall performance of ES method a lot.

# Chapter 4

# Measure Stiffness

## 4.1 Theoretical analysis of some stiff systems

To analyze the stiffness of the system, we need to come up with a dimensionless index of stiffness for every system:

$$S = \frac{T}{\tau}$$

where $T$ is the time we are interested in, and $\tau = -\frac{1}{Re(\lambda)}$, and $\lambda$ is the eigenvalue with the largest negative real part. Actually $\tau$ varies along the solution. If $S$ is larger than 2000, we say this is stiff system. If it is comparable to 10, we say it's not stiff. All the values between are not well-defined. As mentioned before, stiffness doesn't mean it fails the conventional methods, it just makes the conventional methods inefficient. $S$ just tells us how severe the situation is.

Let's have a look at some actual systems to analyze the stiffness analytically.

- van del Pol (vdp) model

  The ODE form of the vdp model can be written as:

$$f_1 = \frac{dy_1}{dt} = y_2$$

$$f_2 = \frac{dy_2}{dt} = \alpha(1 - y_1^2)y_2 - y_1$$

  Then the Jacobian matrix $\mathbf{J}$ is:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial y_1} & \frac{\partial f_1}{\partial y_2} \\ \frac{\partial f_2}{\partial y_1} & \frac{\partial f_2}{\partial y_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2\alpha y_1 y_2 & \alpha - \alpha y_1^2 \end{bmatrix}$$

Also, suppose the initial condition of the system is: $y_1 = 2, y_2 = 0$.

If $\alpha = 1000$, i.e. vdp1000 model has:

$$\mathbf{J} = \begin{bmatrix} 0 & 1 \\ -1 & -3000 \end{bmatrix}$$

Using the standard procedure to obtain the eigenvalues of the matrix, i.e. $det(\mathbf{J} - \lambda I) = 0$:

$$det \begin{bmatrix} -\lambda & 1 \\ -1 & -3000 - \lambda \end{bmatrix} = 0,$$

yields:

$$\lambda^2 + 3000\lambda + 1 = 0$$

with:

$$\lambda_1 \doteq 0, \lambda_2 \doteq -3000.$$

Obviously, $\lambda_2$ is a very negative value, which indicates that this system could be very stiff. Meanwhile, the time span that we are interested in is the period of one limit cycle, which is about 1500. Armed with the dimensionless index of stiffness defined above, we get:

$$S \doteq 4500000,$$

which is a large number. As you can observe in Section 2.4, the conventional methods perform very badly on this model.

If $\alpha = 1$, i.e. vdp1 model:

$$\mathbf{J} = \begin{bmatrix} 0 & 1 \\ -1 & -3 \end{bmatrix}$$

Then, after similar calculations, we get the eigenvalues:

$$\lambda_1 \doteq -0.38, \lambda_2 \doteq -2.62.$$

This is a very small number, and the period is about 6.7, therefore:

$$S \doteq 17.54,$$

It is far below the stiffness bound 2000.

- Oregonator Model

  Please refer to Appendix A for the details of the model. This is a pretty small system again, with 3 species, so we can analytically manipulate the Jacobian matrix:

$$\mathbf{J} = \begin{bmatrix} -\frac{y}{\varepsilon} + \frac{1}{\varepsilon} - 2\frac{x}{\varepsilon} & \frac{q}{\varepsilon} - \frac{x}{\varepsilon} & 0 \\ -\frac{y}{\delta} & -\frac{q}{\delta} - \frac{x}{\delta} & 2\frac{f}{\delta} \\ 1 & 0 & -1 \end{bmatrix},$$

  of which the 3 eigenvalues are 100, 0.26, -0.76. The period is about 18, and the index of stiffness is about 13.7. The first impression of the system is that it is not a stiff system. But keep in mind, during the course of the interaction, the concentrations of the species vary dramatically, which in turn changes the Jacobian matrix. In other words, a system could be stiff in some region and not stiff in the others. More careful and general numerical effort is shown in the next section.

## 4.2 Stiffness detector

A stiffness detector is implemented using the idea above. Let's consider 10 models: 3 Water Model I's, QA-A, QA-Tr.A, QA-Tr.B, *lac* operon, oregonator, repressilator and the biological clock model. Please refer to Appendix A for the description of all the models.

- Non-stiff Water Model I

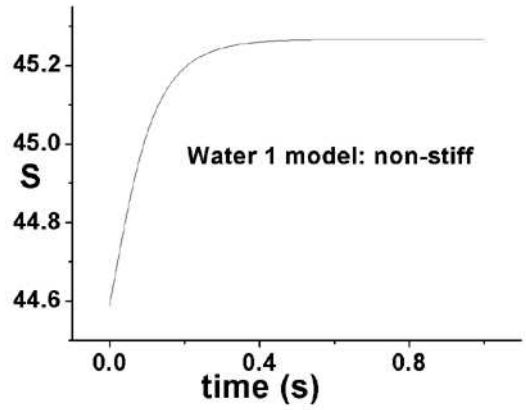


Figure 4.1: Stiffness of non-stiff Water Model I.

The maximum value is about 45.3, which is comparable to 10. Therefore, we can consider this system to be non-stiff. The performance of the MATLAB ODE suite also supports this conclusion as in Section 2.4.

- Moderately stiff Water Model I

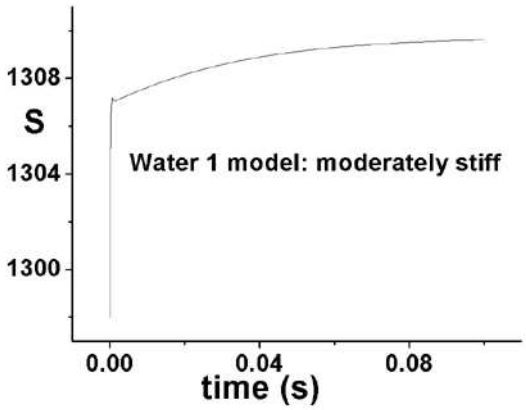


Figure 4.2: Stiffness of moderately stiff Water Model I.

The maximum stiffness is around 1308, which is not well defined. We call it moderately stiff. The performance of stiff methods are a little bit better than the conventional methods.

- Stiff Water Model I

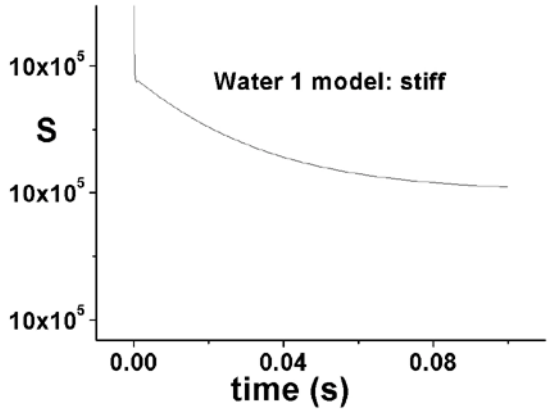


Figure 4.3: Stiffness of stiff Water Model I.

This is a very stiff example, in which the maximum index of stiffness is around 1E+6, and this makes conventional methods perform very badly.

- QA-Tr.B

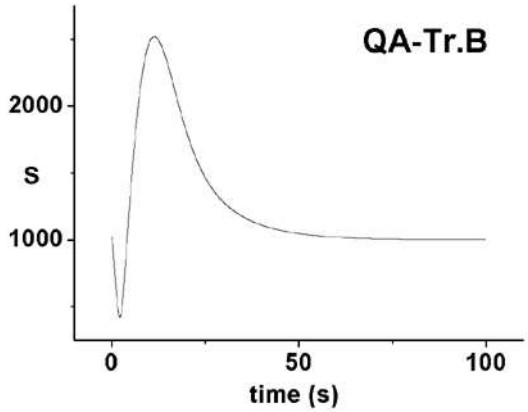


Figure 4.4: Stiffness of QA-Tr.B.

This *qa* gene cluster model has 39 participant species and 45 reactions. The maximum index of stiffness is around 2500. This system is stiff around the particular region, i.e. 10 to 30 seconds.

- QA-A

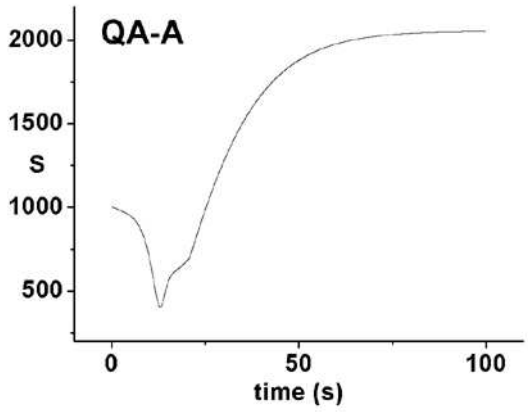


Figure 4.5: Stiffness of QA-A.

After 50 seconds, this system exhibits stiffness, with its 37 participant species and 43 reactions. This is system is not as stiff as the previous one, QA-Tr.B, during certain region, but is stiff a greater fraction of the time.

- QA-Tr.A

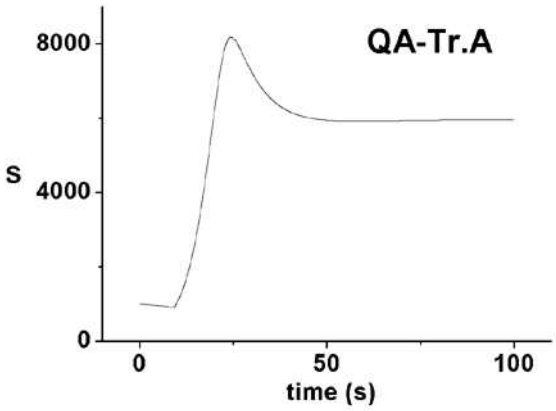


Figure 4.6: Stiffness of QA-Tr.A.

QA-Tr.A has 37 species and 43 reactions. This is obviously a very stiff system, with a peak around 8000, and saturates around 6000.

- *lac* operon

The *lac* operon in *escherichia coli* [9] is one of the first biological circuits explored in detail. The system has 64 species and 69 reactions. The figure below shows that the increasing of the stiffness as time increases will make the conventional methods perfrom worse and worse. The performance of the stiff solver could be better by a factor of 100 to 200 compared to the $4^{th} order$ RK method, while it is improved only by a factor of 10 for the *qa* gene cluster models.

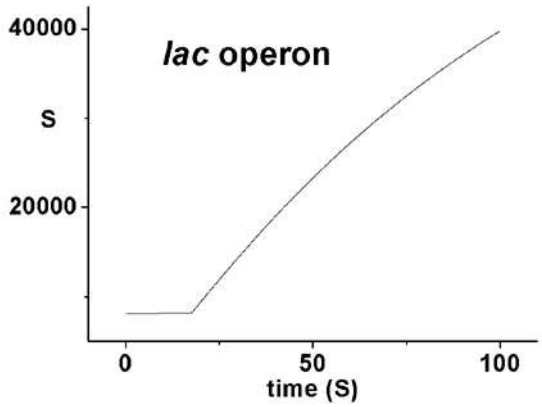


Figure 4.7: Stiffness of *lac* operon.

- Oregonator

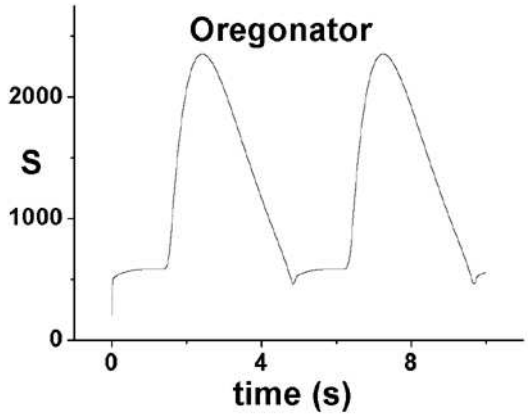


Figure 4.8: Stiffness of oregonator.

This is another famous oscillatory [39] system which also exhibits stiffness. And again, the performance could almost be improved by a factor of 4.

- Repressilator

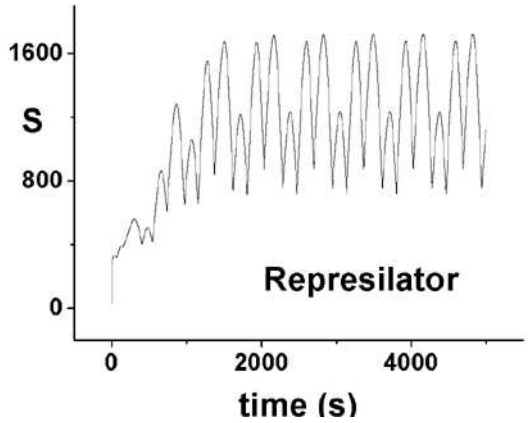


Figure 4.9: Stiffness of repressilator.

Another famous oscillatory system [44] is not very stiff, since the maximum index of stiffness is smaller than 2000. And as we'll see that the stiff method didn't improve, instead it's even worse than the Runge Kutta method.

- Biological clock Model

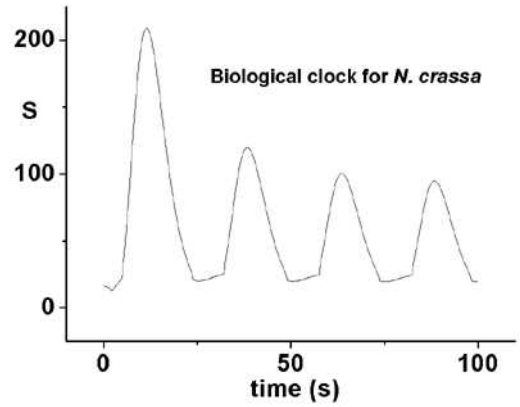


Figure 4.10: Stiffness of biological clock.

This system is even less stiff than the repressilator model, with the peak being around 210. We'll see the similar performance test as the previous model.

# Chapter 5

# Solutions for Stiff Systems

## 5.1 Numerical methods for stiff ODEs

### 5.1.1 Backward Euler Method

Here we introduce a stiff method called the backward Euler method which is a one-step method:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}).$$

This is called an *implicit* method. If any component of the update or intermediate solution vectors appears both as the update on the left hand side of the equation describing the numerical method and the the right hand side as a function evaluation, the such method is said to be *implicit*, otherwise it is *explicit*.

If we use the test equation $\frac{dy}{dt} = Ay$ again, we get:

$$y_{n+1} = y_n + Ahy_{n+1}$$

$$y_{n+1} = \frac{1}{1 - Ah} y_n$$

Then:

$$y_{n+1} + \varepsilon_{n+1} = \frac{1}{1 - Ah}(y_n + \varepsilon_n).$$

And the difference between the previous 2 equations gives us:

$$\varepsilon_{n+1} = \frac{1}{1 - Ah}\varepsilon_n.$$

Then recursively, we obtain:

$$\varepsilon_n = \varepsilon_0 \frac{1}{(1-Ah)^n},\ n = 0, 1, 2 \ldots.$$

So, the propagated error is damped whenever $|\frac{1}{1-Ah}| \leq 1$, which means the whole left half plane is an absolute stable region. This indicates that the step size is not limited too much by the stability issue as the conventional Euler method. As you may notice in the backward Euler equation, the $y_{n+1}$ is used in the right hand side, and this kind of equation is called implicit. Cleve Moler [33] has a very interesting demonstration for how implicit methods work:

> "Imagine you are returning from a hike in the mountain. You are in a narrow canyon with steep walls on either side. An explicit algorithm would sample the local gradient to find the descend direction. But following the gradient on either side of the trail will send you bouncing back and forth from wall to wall, ... You will eventually get home, but it will be long after dark before you arrive. An implicit algorithm would have you keep your eyes on the trail and anticipate where each step is taking you."

### 5.1.2 Backward Differential Formula Method (BDF)

This class of multi-step formulas [25] which are highly effective in solving stiff problems are based on numerical differentiation. We interpolate the previously computed solution values $y_n, y_{n-1}, \ldots, y_{n-p}$ as well as the new one $y_{n+1}$ by a polynomial $P(t)$. The derivative of the solution at $t_{n+1}$ is approximated by $P(t_{n+1})$. The approximation is related to the differential equation by insisting that it satisfies the differential equation at $t_{n+1}$:

$$\frac{dP(t_{n+1})}{dt} = f(t_{n+1}, P(t_{n+1})) = f(t_{n+1}, y_{n+1})$$

Substituting for $\frac{dP(t_{n+1})}{dt}$, we obtain the family of BDF methods:

$$\alpha_0 y_{n+1} + \alpha_1 y_n + \ldots + \alpha_p y_{n+1-p} = h f(t_{n+1}, y_{n+1}),$$

which is also called Gear's formulas because it was popularized by Gear . It needs Newton iteration to solve the nonlinear algebraic equations for $y_{n+1}$ which requires approximating partial derivatives and solving systems of linear equations. The following is the second order BDF:

$$y_{n+1} = \frac{4}{3} y_n - \frac{1}{3} y_{n-1} + \frac{2}{3} f_{n+1}.$$

Let us consider the first order BDF which is also known as the backward Euler method. The most difficult part is to compute $y_{n+1}$ needed for interpolation. The value of $y_{n+1}$ can be computed iteratively as:

$$y_{n+1}^{k+1} = y_n + h f_{n+1}(t_{n+1}, y_{n+1}^k), \;\; k = 0, 1, 2 \ldots$$

But this computation is very expensive. Instead, a variation of Newton's method is considered:

$$F(y_{n+1}) = y_{n+1} - y_n - h f_{n+1} = 0,$$

which leads to:

$$y_{n+1}^{k+1} = y_{n+1}^k - \frac{F(y_{n+1}^k)}{F'(y_{n+1}^k)}, \;\; k = 0, 1, 2 \ldots$$

where:

$$F'(y_{n+1}^k) = I - hbJ(t_{n+1}, y_{n+1}^k),$$

h is the step size, b is 1 for the first order BDF ($\frac{2}{3}$ for the second order BDF), I is the identity matrix of dimension same as the number of species in the system. And $J(t_{n+1}, y_{n+1}^k)$ is the Jacobian matrix same as define in Section 3.1. Therefore, we find:

$$y_{n+1}^{k+1} = y_{n+1}^k - \frac{F(y_{n+1}^k)}{I - hbJ(t_{n+1}, y_{n+1}^k)}, \;\; k = 0, 1, 2 \ldots$$

For convergence, we rewrite it as:

$$M = y_{n+1}^k - y_{n+1}^{k+1} = \frac{F(y_{n+1}^k)}{I - hbJ(t_{n+1}, y_{n+1}^k)},$$

then,

$$(I - hbJ(t_{n+1}, y_{n+1}^k)M = F(y_{n+1}^k)$$

is used for computing $M$ for the sake of matrix inversion operation, although it's still costly as explained in [29]. The values of the vector $M$ are substituted back in the equation to complete one of the $k$ iterations of the Newton's method. The Jacobian matrix $J$ usually varies very slowly, so for a number of time steps, only one evaluation of $J$ is needed to save time.

## 5.2 Numerical solvers for Stiff System

The first notion of stiffness and the first formal methodology for solving stiff ODEs was reported by Curtiss and Hirschelder [34]. They used the term stiff for ODEs because the corresponding servomechanism felt stiff.

C. W. Gear developed a software package that used BDF called DIFSUB [35]. He then revised it to the new code called STIFF in 1969. R. J. Gelinas and A. C. Hindmarsh rewrote STIFF and called the new package GEAR [21] at Lawrence Livermore Natinal Laboratory. Several variants of GEAR has been developed to take advantage of different problem structures and/or computer structures.

The MATLAB ODE suite by L. F. Shampine [20] has 2 stiff solvers: ode15s and ode23s. The solver ode15s uses numerical differentiation formula (NDF) while ode23s uses linearly implicit formula which is a modified Rosenbrock triple. Both of them utilize the relatively fast built-in array operations and linear algebra implemented in MATLAB. These methods are platform dependent though.

G. D. Byrne and A. C. Hindmarsh developed a package called LSODE [22] which uses the LINPACK linear algebra packages, also LSODE treats the Jacobian matrix

as either dense or banded.There are many variants of it, such as LSODA [36], which was written by Petzold, with a novel feature to switch between stiff (BDF) and non-stiff (Adams) methods.

LSODES [5] is another variant of LSODE, which uses the components of the Yale Sparse Matrix Package (YSMP) [30, 31] to take advantage of the sparse Jacobian matrix. This package is incorporated into the KINSOLVER package now.

There are still a lot of other stiff solvers. The above list doesn't mean the unnamed solvers are not worth trying. You can refer to the general survey paper [32] for more possibilities.

## 5.3 Quasi-Static Approximation Method

In a number of areas, especially in biochemical kinetics, we can modify the system to remove the stiffness. The basic idea is that the physical considerations allow some components to be recognized as changing on time scales much shorter than those of other components, in other words, we can divide the system into several subsystems. Imagine we have 2 reactions: one is reacting at the speed of "flying" while the other one is reacting at the speed of "walking". If we are just interested in the very short region of time span, we can simply take the slow reaction as a constant. Such approximations could lead to several sets of non-stiff ODEs coupled together. But it is just too important to ignore the fact that the results of the changed model could be different from the original model as noted in [37].

As an example, the biological clock of *N. crassa* as shown in Figure A.2, the WC-2 protein is in 5-fold molar excess over FRQ and WC-1 protein in the nucleus [38], and hence wc-2 and its products can be treated approximately as constants. The total parameter space can be reduced further by some other simplification of

the model. This reduces the stiffness, and at the same time makes the fixed point analysis [18] possible.

# Chapter 6

# Livermore Solver for ODEs with General Sparse Jacobian Matrices (LSODES)

## 6.1 Introduction to LSODES

LSODES was written by L. F. Shampine and A. H. Sherman [5]. This package uses Backward Differential Formula (BDF) and treats the Jacobian matrix as a general sparse matrix. As mentioned in Section 4.1, the most expensive part is that we need to solve linear systems, which are handled by Yale Sparse Matrix Package (YSMP) in LSODES instead of the brute force Newton iteration. The general form of the problem is:

$$PM = F,\ P = I - hbJ,$$

where M is the correction vector which is the difference between the two consecutive iterations, $I$ is an identity matrix, $h$ is the step size, $b$ is a scalar depending on the current method order and $J$ is the Jacobian matrix. The solution can be expressed in the following several phases:

1. Determine the sparsity structure either by the user directly or by calls to the f routine in the package.

2. Determine the pivot order to maintain maximum sparsity by ODRV module of YSMP.

3. Symbolic LU factorization of the matrix $P$ by CDRV module of YSMP.

4. Construction of $J$.

5. Construction of $P = I - hbJ$.

6. Numerical LU factorization of $P$ by CDRV module of YSMP.

7. Solution of $PM = F$.

For more detailed discussion, please refer to [5].

## 6.2 LSODES in KINSOLVER

LSODES is the $6^{th}$ integration option in KINSOLVER, please refer to [6] or http://gene.genetics.uga.edu/STC about how to use the KINSOLVER and other 5 options.

LSODES is written in Fortran and it specifies the ODE's in the source code. After the biological circuit is specified (i. e. the initial concentrations, the rate constants) in the web page interface, the back-end KINSOLVER starts to manipulate the input file via Java Servlet. If the integration option is 6, which means LSODES, the corresponding Fortran source code for this particular biological circuit is generated. Then the Fortran code is compiled and executed, with output plotted by gnuplot. Finally, the figure is returned to the web page.

## 6.3 Numerical experiments using LSODES

In this section, 8 models are tested. All of them have been measured for their stiffness in Section 4.3. The conventional methods' results for the 3 *qa* gene cluster models and *la* operon were taken from [7]. Relative error is computed relative to the solutions of adaptive RK method using 100,000 time steps. The tests are performed under SunOS 5.8 on SUNFIRE 250G (dual CPU 1.12GHz each, RAM 8GB).

- Non-stiff Water Model I

  In order to get non-trivial measurements, I set the final time to be 10000 instead of 1. The Euler method is the best because this system is very simple, and other methods are more expensive computationally. Adaptive RK method failed to solve the problem because the implementation of this method used static memory allocation which means long time integration requires too large arrays.

| **Method** | **No. of DFEs** | **Relative Error < 1E-6** | |
|---|---|---|---|
| | | Steps | Time(s) |
| Euler | 1 | 22,641 | 0.06 |
| RK4 | 4 | 16,500 | 0.19 |
| Adaptive RK | 6 | ! | ! |
| LSODES | - | 106 | 0.08 |

Table 6.1: Time required to simulate non-stiff Water Model I.

- QA-Tr.B

| **Method** | **No. of DFEs** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|---|
| | | Steps | Time(s) | Steps | Time(s) |
| Euler | 1 | 1,000 | 0.02 | 100,00,000 | ! |
| RK4 | 4 | 700 | 0.07 | 3,163 | 0.31 |
| Adaptive RK | 6 | 651 | 0.12 | 678 | 0.12 |
| LSODES | - | 37 | 0.01 | 229 | 0.03 |

Table 6.2: Time required to simulate QA-Tr.B of *qa* gene cluster.

Euler method is better than other conventional methods at low accuracy, however the Euler method is low order integration method, it is not able to solve the problem at high accuracy within a reasonable amount of time. QA-Tr.B is stiff within a certain region, which makes LSODES better than all other methods.

- QA-A

| **Method** | **No. of DFEs** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|---|
| | | Steps | Time(s) | Steps | Time(s) |
| Euler | 1 | 2,900 | 0.07 | 100,00,000 | ! |
| RK4 | 4 | 800 | 0.08 | 3,163 | 0.30 |
| Adaptive RK | 6 | 634 | 0.12 | 736 | 0.12 |
| LSODES | - | 39 | 0.01 | 180 | 0.03 |

Table 6.3: Time required to simulate QA-A of *qa* gene cluster.

At low accuracy, Euler method is better than other conventional methods again, while it is bad at high accuracy for the same reason as in the previous model. Because QA-A is stiff for most of the time, it is considered stiffer than QA-Tr.B. The LSODES is able to improve the performance much more than other methods.

- QA-Tr.A

| **Method** | **No. of DFEs** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|---|
| | | Steps | Time(s) | Steps | Time(s) |
| Euler | 1 | 3,300 | 0.08 | 100,00,000 | ! |
| RK4 | 4 | 2,300 | 0.22 | 3,163 | 0.30 |
| Adaptive RK | 6 | 1,335 | 0.23 | 2013 | 0.32 |
| LSODES | - | 25 | 0.01 | 197 | 0.03 |

Table 6.4: Time required to simulate QA-Tr.A of *qa* gene cluster.

At low accuracy, LSODES is about 8 times better than the best conventional method: the Euler method. At high accuracy, LSODES is about 10 times better than the RK4 method. The performance of LSODES method for the above 3 *qa* models is almost the same, regardless of the stiffness, and the improvement is about a factor of 10.

| **Method** | **No. of DFEs** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|---|
| | | Steps | Time(s) | Steps | Time(s) |
| Euler | 1 | 103,000 | 3.67 | 1E08 | ! |
| RK4 | 4 | 73,300 | 11.20 | 73,300 | 11.20 |
| Adaptive RK | 6 | - | - | 30,339 | 7.64 |
| LSODES | - | 20 | 0.02 | 102 | 0.05 |

Table 6.5: Time required to simulate *lac* operon.

- *lac* operon

  The *lac* operon is a very stiff system. The performance of LSODES is almost at the same magnitude as the previous 3 *qa* models. This stiff system makes the conventional methods very inefficient. The improvement by LSODES could be around a factor of 200! The '-' for the row of 'Adaptive RK' indicates this model can not be solved at low accuracy using Adaptive RK method.

- Oregonator

| **Method** | **No. of DFEs** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|---|
| | | Steps | Time(s) | Steps | Time(s) |
| Euler | 1 | 100,000 | 0.71 | 1E+8 | ! |
| RK4 | 4 | 11,000 | 0.31 | 25,000 | 0.70 |
| Adaptive RK | 6 | 9,038 | 1.01 | 15,492 | 1.24 |
| LSODES | - | 3,264 | 0.10 | 6,701 | 0.19 |

Table 6.6: Time required to simulate Oregonator model.

This is a stiff and oscillatory system. Usually Euler method is not good for an oscillatory system because an oscillatory system requires rather high accuracy. So in this case a higher order method is preferred. At both low and high accuracy, the method LSODES is better than RK4 method by a factor of 3.

- Repressilator

| **Method** | **No. of DFEs** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|---|
| | | Steps | Time(s) | Steps | Time(s) |
| Euler | 1 | 30,000 | 0.25 | 1E+8 | ! |
| RK4 | 4 | 5,000 | 0.07 | 8,000 | 0.28 |
| Adaptive RK | 6 | - | - | 4,487 | 0.52 |
| LSODES | - | 465 | 0.37 | 1,708 | 0.51 |

Table 6.7: Time required to simulate repressilator model.

The Repressilator is not a stiff system according to the stiffness detection in Section 4.3. Numerical experiments here also proved this. The Euler method is better than LSODES at low accuracy, although the Euler method is not able to solver the system at high accuracy. The RK4 method is the best, because repressilator is again an oscillatory system which means the Euler method didn't outperform RK4.

- Biological Clock for *N. crassa*

| **Method** | **No. of DFEs** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|---|
| | | Steps | Time(s) | Steps | Time(s) |
| Euler | 1 | 10,000 | 0.14 | 1E+8 | ! |
| RK4 | 4 | 200 | 0.01 | 500 | 0.03 |
| Adaptive RK | 6 | 213 | 0.05 | 639 | 0.09 |
| LSODES | - | 481 | 0.02 | 856 | 0.05 |

Table 6.8: Time required to simulate biological clock for *N. crassa*.

The Biological clock is not stiff again as shown in Section 4.3. These measurements here are similar to Repressilator: LSODES is not better than $4^{th}$ order RK method due to the extra computation for the algebraic equations.

## 6.4 ODE15S AND LSODES

The ode15s method is the best ODE solver from MATLAB ODE suite as concluded in Section 2.4. We shall compare ode15s with the method LSODES which is used in the KINSOLVER. Because they are running on different machines: MATLAB is running under Windows XP on Pentium III (CPU 930MHz, RAM 192MB), and KINSOLVER is running under SunOS 5.8 on SUNFIRE 250G (dual CPU 1.12GHz each, RAM 8GB). I used Euler method to solve the simple Water Model I to get a ratio of the 2 machines. The Unix machine is about 50 times faster than the Windows machine.

| **Model** | **Relative Error < 1E-2** | | **Relative Error < 1E-6** | |
|---|---|---|---|---|
| | LSODES (s) | ode15s (s) | LSODES (s) | ode15s (s) |
| Oregonator | 0.10(5) | 4.28 | 0.19(9.5) | 14 |
| vdp1000 | 0.035(1.75) | 1.16 | 0.06(3) | 4.12 |

Table 6.9: Comparison of LSODES and ode15s from MALAB ODE suite.

The number in the parenthesis is the converted time value by a factor of 50 to be compared with the ode15s on the Windows machine. At low accuracy, ode15s is a little bit better than LSODES, while at high accuracy, LSODES is a little bit better than ode15s. Both of the methods have almost the same performance.

# Chapter 7

# Future Work

LSODES has been identified as a very efficient method to solve stiff problems. It is only contained in the KINSOLVER at this stage. We shall certainly incorporate it into the Monte Carlo ensemble method [4], which is going to be used for various biological circuits, to utilize the high efficiency of LSODES. For the *qa* gene cluster model, we can improve about a factor of 10, in other words, the running time will be reduced from one month to about several days.

# Chapter 8

# Conclusion

Numerical solution of system of ODEs, which is the mathematical representation of biochemical and gene regulatory networks, is a very interesting topic. Stiffness of the ODE is one of the reasons to make conventional methods very inefficient. Many actual biological systems present stiffness more or less.

Theoretical analysis of stiff/non-stiff models and stiff/non-stiff methods are done. A dimensionless index of stiffness is defined and a stiffness detector is implemented. The stiffness detection of various models are compatible with the numerical experiments of MATLAB ODE suite and KINSOLVER. ode15s is identified to be the best stiff solver among the MATLAB ODE suite. LSODES is much better than other integration options in KINSOLVER when the problem is stiff, though it is a little bit worse when it comes to non-stiff systems. The performance of LSODES and ode15s are comparable to each other.

All the source code and input files for KINSOLVER and MATLAB ODE suite are available at http://www.physast.uga.edu/ yihyu/stiff.

# Bibliography


[1] Venter, J. C., et. at. (2001) *The Sequence of the Human Genome* Science 291, 1304-1351

[2] International human genome Sequencing Consortium (2001) *Initial sequencing and analysis of the human genome* Nature 409, 860-921.

[3] Beadle, G. W.; and Tatum, E. I.(1941) *Genetic control of biochemical reactions in Neurospora*. Proc. Natl. Acad. Sci. USA 27, 499-506

[4] Battogtokh, D.; Asch, D. K.; Case, M. E.; Arnold, J.; and Schüttler, H. B. (2002) *An ensemble method for identifying regulatory circuits with special reference to the qa gene cluster of Neurospora crassa*. Proc. of the National Academy of Sciences USA 99, 16904-16909.

[5] Hindmarsh, A. C. (1983) *ODEPACK, a sytemized collection of ODE solvers*, in scientific computing, Stepleman, R. S. et al., Eds., 55-64.

[6] Aleman-Meza, Boanerges (2001) *Advances in Numerical Simulation of Kinetics Reaction Equations*. Master Thesis May 2001, Department of Computer Science, The University of Georgia.

[7] Aleman-Meza, Boanerges; Yu, Yihai; schuttler, H. B.; Arnold, J.; and Taha, Thiab R. (2004) *KINSOLVER: A simulator for biochemical and gene regulatory networks*. (to be submitted).

[8] Arnold, J.; Schüttler, H. B.; Logan, D. A.; Battogtokh, D.; Griffith, J.; Arpinar, I. B.; Bhandarkar, S.; Datta, S.; Kochut, K. J.; Kraemer, E.; Miller, J. A.; Sheth, A.;Strobel, G.; Taha, T.; Aleman-Meza, B.; Doss, J.; Harris, L.; and Nyong, A. (2004) *Metabolomics in Ch. 22 of Handbook of Industrial Mycology*. Marcel-Dekker, NY.

[9] Jacob, F.; and Monod, J. (1961) *Genetic regulatory mechanisms in the synthesis of proteins*. J. Mol. Biol. 3, 318-356.

[10] Yanofsky, C.; and Kolter, R. (1982) *Attenuation in amino acid bisynthesis operons*. Ann. Rev. genetic 16, 113-134.

[11] Yanofsky, C. (2001) *Advancing our knowledge in biochemical, genetics, and microbiology through studies of tryptophan metabolism*. Ann. Rev. Biochem. 70, 1-37.

[12] Johnston, M. (1987) *A model fungal regulatory mechanism: the GAL genes of Saccharomyces cerevisiae*. Microbiolog. Rev. 51, 458-476.

[13] Giles, N. H.; Case, M. E.; Baum, J.; Geever, R.; Huiet, L.; Patel, V.; and Tyler, B. (1985) *Gene organization and regulation in the qa (Quinic Acid) gene cluster of Neurospora crassa*. Microbiological Reviews 49, 338-358.

[14] Sveiczer, A.; Csikasz-Nagy, A.; Gyorffy, B.; Tyson, J.J.; and Novak, B. (2000) *Modeling the fission yeast cell cycle: quantized cycle times in wee1- cdc25D mutant cells*. Proc. Natl. Acad. Sci. USA 97, 7865-7870.

[15] Lee, K.; Loros, J.J.; and Dunlap, J.C. (2000) *Interconnected feedback loops in the Neurospora circadian system*. Science 289, 107-110.

[16] Bhalla, U. S.; and Iyengar, I. (1999) *Emergent Properties of Networks of Biological Signaling Pathways*. Science 283, 381-387.

[17] Alves, R.; and Savageau, M.A. (2000) *Systemic properties of ensembles of metabolic networks: application of graphical and statistical methods to simple unbranched pathways.* Bioinformatics 16, 534-547.

[18] Yu, Y.; Altimus, C.; Dudek, L.; Arnold, J.; and Schüttler, H. B. (2004) *A genetic network for the biological clock of Neurospora crassa: what makes the biological clock tick?.* (to be submitted).

[19] Shampine, L. F.; Watts, H. A.; and Davenport, S. M. (1976) *Solving nonstiff ordinary differential equations – The state of the art*, this Review, 18 376-411.

[20] Shampine, L. F.; and Reichelt, M. (1997) *The MATLAB ODE suite* SIAM J. Sci. Comput. vol. 18, No. 1 1-22 January 1997.

[21] Hindmarsh, A. C. (1974) *GEAR: Ordinary Differential Equation System Solver.* Lawrence Livermore Labatory report UCID-30001, Rev. 3 Dec.

[22] Hindmarsh, A. C. (1980) *LSODE and LSSODI, Two New Initial Value Ordinary Differential Equation Solvers.* ACM SIGNUM Newsletter, 15, No. 4, 10-11.

[23] Geever, R. F.; Huiet, L.; Baum, J. A.; Tyler, B. M.; Patel, V. B.; Rutledge, B. J.; Case, M. E.; and Giles, N. H. (1989) *DNA Sequence, organization and regulation of the qa gene cluster of Neurospora crassa.* Journal of Molecular Biology 207, 15-34.

[24] Kochut, K. J.; Arnold, J.; Sheth, A.; Miller, J.A.; Kraemer, E.; Arpinar, I.B.; and Cardoso, J. (2003) *IntelliGEN: a distributed workflow system for discovering protein-protein interation.* Parallel and Distributed Database 13, 43-72.

[25] Dormand, J. R. (1996) *Numerical Methods for Differential Equations – A Computational Approach.* CRC Press

[26] Cheney, E. W.; and Kincaid, D.R. (1999) *Numerical Mathematics and Computing*. Fourth Edition. Brroks/Cole Publishing.

[27] Shampine, L.F.; and Gordon, M.K. (1975) *Computer Solution of Ordinary Differential Equations: the Initial Value Problem*. Freeman.

[28] Shampine, L.F.; and Gear, C.W. (1979) *A User's View of Solving Stiff Ordinary Differential Equations*. SIAM Review, Vol. 21, Issue 1, 1-17.

[29] Burrage, K. (1995) *Parallel and Sequential Methods for Ordinary Differential Equations*. Oxeford Universty Press, New York.

[30] Eisenstat, S. C.; Gursky, M. C.; Schultz, M. H.; and Sherman, A. H. (1977) *Yale Sparse Matrix Package: II. The Nonsymmetric Codes*. Yale University Computer Science Dept. report no. 114

[31] Eisenstat, S. C.; Gursky, M. C.; Schultz, M. H.; and Sherman, A. H. (1982) *Yale Sparse Matrix Package: II. The Symmetric Codes*. Int. J. Num. Math. Eng., 18, 1145-1151

[32] Byrne, G. D.; and Hindmarsh, A. C. (1986) *Stiff ODE Solver: A Review of Current and Coming Attractinons*. UCRL-94297 Lawrence Livermore National Laboratory, for the Journal of Computational Physics.

[33] Moler, Cleve *Moler Cleve Stiff Differential Equations*. Cleve's Corner.

[34] Curtiss, C. F.; and Hirschfelder, J. O. (1952) *Integration of Stiff Equations*. Proc. of Nat. Acad. Sci. USA 38, 235-243.

[35] Gear, C. W. (1968) *The Automatic Integration of Stiff Ordinary Differential Equations*. Proceedings of the 1968 IFIP Congress held in Edinburgh, Scotland, A. J. H. Morrel, Ed., North Holland Publishing Co., Amsterdam, 187-193.

[36] Petzold, L. R. (1983) *Automatic Selection of Methods for solving Stiff and Non-stiff Systems of Ordinary Differential Equations.* SIAM J. SDci. Stat.Comput., 4, 136-148.

[37] Lapidus, L.; Aiken, R. C.; and Liu, Y. A. (1974) *The occurrence and numerical solution of physical and chemical systems having widely varying time constants, Stiff Differential Systems.* Willoughby, R. A., ed., Plenum Press, NY, 187-200.

[38] Denault, D. L.; Loros, J. J.l and Dunlap, J.C. (2001) *WC-2 mediates WC1-FRQ interaction with the PAS protein-linked circadian feedback loop of Neurospora.* EMBO. J. 20, 109-117.

[39] Murray, J.D. (1993) *Mathematical Biology.* Sptringer-Verlag, NY.

[40] Brankin, R. W.; Gladwell, I.; and Shampine, L. F. (1972) *RKSUITE: A suite of Runge-Kutta codes for the initial value problem for ODEs*, Tech. Report 92-S1, Math. Dept., Southern Methodist Univ., Dallas, TX.

[41] Brayton, R. K.; Gustavson, F. G.; and Petzold, L. R. (1972) *A new efficient algorithm for solving differential-algebrai systems using implicit backward differentiation formulars*, Proc. IEEE, 60, 98-108.

[42] Bader, G.; and Deulfhard, P. (1981) *A semi-implicit mid-point rule for stiff systems of ordinary differential equations*, Tech. Report 114, Institut fur Angewandte Mathematik, Universitat, Heidelberg, Germany.

[43] Bogacki, P.; and Shampine, L. F. (1989) *A 3(2) pair of Runge-Kutta formulars.* Appl. Math. Lett., 2, 1-9.

[44] Elowitz, M. B.; and Leibler, S. (2000) *A synthetic oscillatory network of transcriptional regulator.* Nature 403, 335-338.

[45] Holland, J. H. (1992) *Adaption in Natural and Artificial Systems*. MIT Press, Cambridge, MA. (1st edition, 1975 The University of Michigan Press, Ann Arbor).

[46] Beyer, H. G.; Shwefel, H. P. (2002) *Evolution strategies: A comprehensive introduction*. Natural Computing, 1:1 3-52.

[47] Fogel, D. B. (1995) *Evolutionary Computation*. IEEE Press.

[48] Fogel, D. B.; Owens, A. J.; and Walsh, M. J. (1966) *Artificial Intelligence through Simulated Evolution*. Wiley, Chichester, UK.

[49] *Introduction to Evolutionary Computing*. (2003 )Eiben and Smith. Springer-Verlag, New York.

[50] Goldberg, D. E. (1989) *Genetic Algorithm in Search, Optimization and machine Learning*. Addison-Wesley.

[51] Glover, F. Tabu (1996) *Search and adaptive memory programming - advances, applications, and challenges*. In: Barr, R. S., Helgason, R. V., and Kennington, J. L. Eds., Interfaces in Computer Science and Operations Research. Kluwer Academic Publishers, Norwell, MA, 1-75.

[52] Hansen, P.; and Mladenovic, N. (1998) *An introduction to variable neighborhood search*. In: S. Vob, S. Martello, I. H. Osman, C. Roucairol, Eds., Meta-Heuristics: Advances and Trends in Local Search paradigms for Optimization. Proceedings of MIC 97 Conference, Kluwer Academic Publishers, Dordrecht, The Netherlands.

[53] http://www.agner.org/random/.

# Appendix A

## Description of the models

The input files for all the models are available at http://www.physast.uga.edu/ yihyu/stiff.

- Water Model I

  There are 3 models used: non-stiff, moderately stiff and stiff. The diagrammatic representation is shown in Figure 2.1. For non-stiff model, the setup of reaction rates are:

  $$k_{f1} = 1.3, k_{b1} = 1.2, k_{f2} = 1.4, k_{b2} = 0.6, k_{f3} = 1.0, k_{b3} = 1.3,$$

  and the initial concentration of the species are:

  $$[H2] = 10, [O2] = 3, [O] = 4, [H] = 5, [OH] = 6, [H2O] = 7.$$

  For moderately stiff model, the only change is that initial concentration of [H2] is 10000. For stiff model, besides initial concentration of [H2] becomes 10000, the $k_{f1}$ is changed to be 1000.3 as well.

- van del Pol (vdp) model

  vdp system is a nonlinear oscillator. This is a classical stiff problem in the field of electrical engineering, which was first reported in the September 1927 issue of *Nature* by Balthazar van der Pol and his colleague van der Mark. The mathematical representation of this system is:

  $$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = \alpha(1 - y_1^2)y_2 - y_1$$

The stiffness can be controlled by changing the parameter $\alpha$. I used $\alpha = 1$ (vdp1) for non-stiff case, and $\alpha = 1000$ (vdp1000) for stiff case, with the same initial conditions: $y_1 = 2$ and $y_2 = 0$.

- *qa* gene cluster model

  3 *qa* gene cluster models are discussed: QA-Tr.B with 39 species and 45 reactions, QA-A with 37 species and 43 reactions and QA-Tr.A with 37 species and 43 reactions [6].

- *la* operon model

  *lac* operon in *escherichia coli* [9] is one of the first biological circuit explored in detail, which has 64 species and 69 reactions [6].

- Oregonator model

  This oscillatory reaction network of oregonator involves $x = [hbrO_2]$, $y = [Br^-]$ and $z = [Ce^{4+}]$, which can be approximated by its dimensionless form [39]:

$$\varepsilon\frac{dx}{dt} = qy - xy + x(1 - x)$$

$$\delta\frac{dy}{dt} = -qy - xy + 2fz$$

$$\frac{dz}{dt} = x - z$$

  where we set $\varepsilon = \delta = 0.01, q = 0.005$, and $f = 0.3$.

- Repressilator model

  Figure A.1 is taken from [7]. The *lacI* gene product represses the *tetR* gene, whose gene product in turn represses the *cI* gne, whose gene product closes the loop by repressing the *lacI* gene [44].

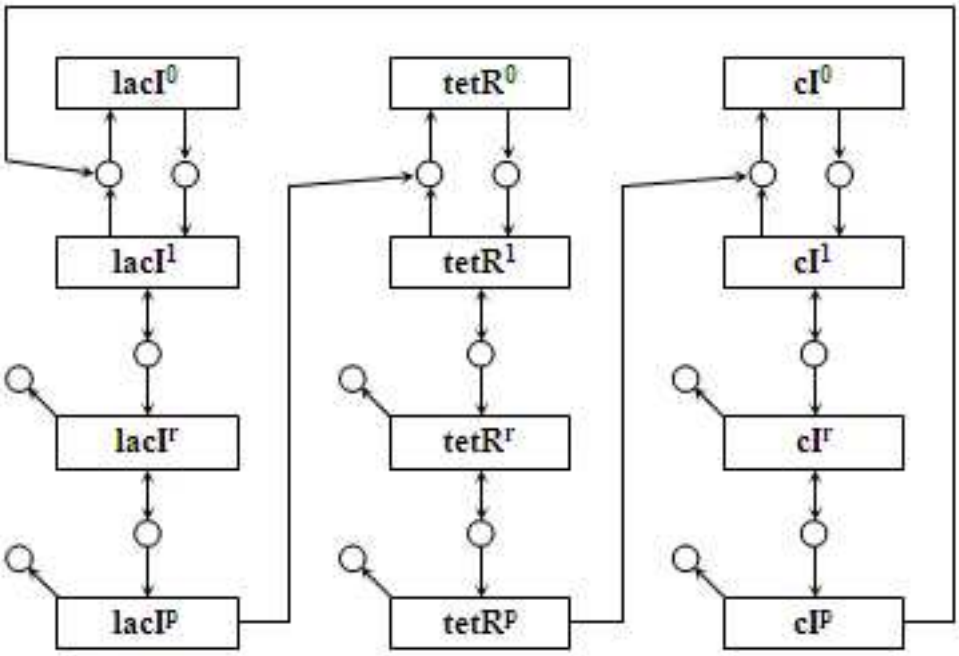


Figure A.1: Biological circuit of the repressilator.

- Biological clock for *Neurospora crassa* model

  As in Figure A.2 from [18], the biological clock of *Neurospora crassa* is another important and interesting example. This system involves 16 species and 25 reactions. But the concentrations of some species are much high than others, e.g. the gene of $wc2^1$ is much higher than gene of $frq^1$, which could lead to very serious stiff problem. The other method we can use is called quasi-static approximation which is discussed in Section 4.5. We can divide the system into 2 different subsystems, and take some of the species to be just constant since they barely change in the time span of other species. If we input the

parameters carefully, we can make the system to oscillate at the period of 24 hours., which is called circadian oscillation.

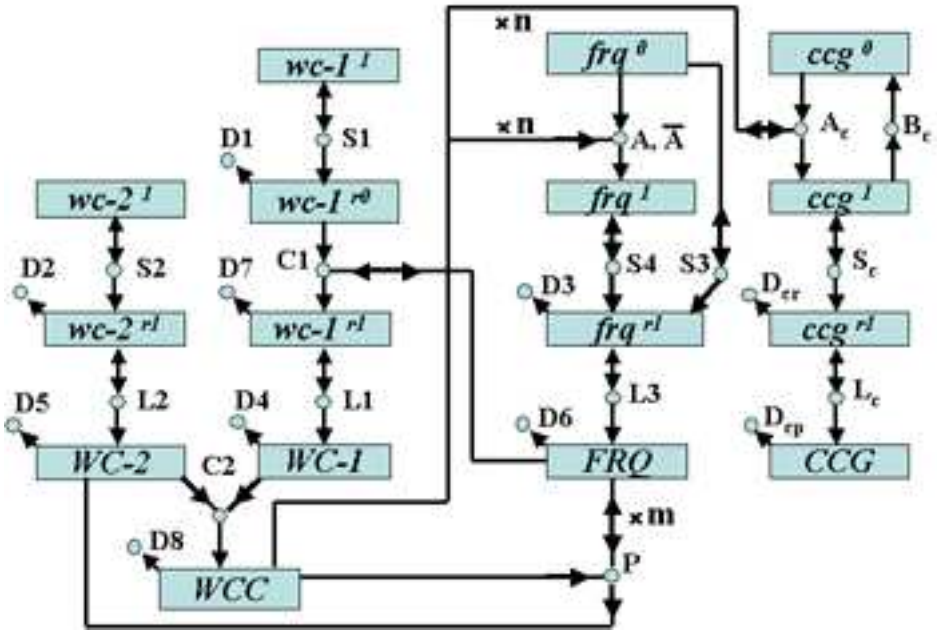


Figure A.2: Biological Clock of *Neurospora crassa*.

# Appendix B

## Solve Chemical Reaction Network Using Evolution Strategy

### B.1 Introduction

An efficient general purpose simulator, KINSOLVER, from [6] is already available to simulate chemical reaction network, i.e. compute the time dependent concentration of each species. This simulator is appropriate only for deterministic models when we know all the reaction parameters. The reality is that most of the parameters are unknown, such as the reaction rates. My idea is to use Evolution Strategy (ES) to solve this problem.

As a very first trial, I only consider the simple Hydrogen Combustion Model I.

### B.2 Evolution Strategy (ES)

The family of evolutionary algorithm (EA) includes genetic algorithm (GA) [45], evolution strategy (ES) [46], evolutionary programming (EP) [47] [48], genetic programming (GP) [49], etc. The common idea behind all of the variants of EA is that: given a population of individuals (or potential solutions to an optimization problem), the environmental pressure causes natural selection, which causes a rise in the fitness of the population. The components of any typical EA are:

- Representation of individuals
- Evaluation function or fitness function for selecting a solution

- Population
- Parent selection mechanism
- Variation operators (recombination and mutation)
- Survivor selection mechanism
- Initialization procedure
- Termination criterion

The first step is usually to find an efficient way to represent a solution or individual. For each particular EA, it has its own representation. For example, GA usually involves using binary vector to represent individuals, while ES uses real-valued vectors. The most crucial component is the fitness function, which is a quality function for a proposed solution needs to be either maximized or minimized. Then we shall maintain a population and each individual in the population is associated with a fitness value. Iteration starts after initialization. For each iteration, a certain number of parents are selected, which are used to generate a number of children using some specific variation operators. For binary vector representation, the simplest operator we can use is to flip one single bit of the vector that is the representation of one particular individual solution. The termination criterion could be the total iterations you want to spend, or total computing time. A more useful scheme is probably to stop the program if the fitness of the population doesn't improve over a certain number of iterations.

Figure B.1 shows the performance of 3 general methods from a global perspective: random search, problem tailored method and EA. For any problem, random search is always the worst, while EA clearly outperforms random methods. But a problem tailored method could be much better than EA and random search, however, it is

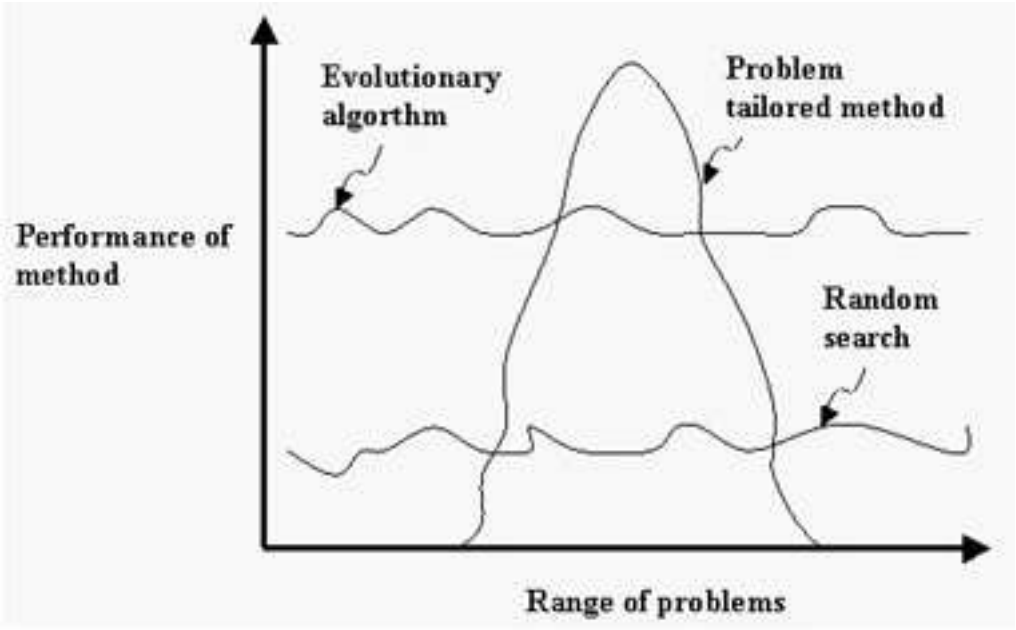


Figure B.1: EA performance from Goldberg [50].

just designed for a specific class of problem. To be more precise, sometimes, EA even performs worse than random search. When we want to get global optimization, the EA may be stuck in some local optimum due to the environmental pressure or bias we introduced into the process. Any local search algorithm has this same problem. A number of methods have been proposed to solve this problem [51] [52]. However the theoretical basis behind these algorithms is still in development.

We can consider the problem tailored methods to be very greedy methods, while random search is non-greedy method. Greed helps to speed up the computation just as in real world, greed could possibly lead to success. But if the situation is not clear or the system is just too complex for us to understand thoroughly, greed could lead us to local optimum or even wrong conclusions due to wrong pressure we

introduced, which is the case for most of the problems facing us now. We can make ES be either very greedy (close to problem tailored methods) or non-greedy (close to random method). The essence of a good EA is to find the best point between them considering a trade-off. The most significant difference between EA and other local search method, such as the Monte Carlo method, is that EA maintains diversity.

ES was first invented in the early 1960s by Rechenberg and Shwefel when they were working on optimization of shape [45]. The standard implementation is sketched in Table B.1:

| | |
|---|---|
| **Representation** | Real-value vectors |
| **Recombination** | Discrete or intermediary |
| **Mutation** | Gaussian perturbation |
| **Parent selection** | Uniform random |
| **Survivor selection** | ( $\mu$, $\nu$ ) or ( $\mu + \nu$ ) |
| **Specialty** | Self-adaption of mutation sizes |

Table B.1: Sketches of ES from [49].

For each problem, we first figure out how to represent individuals , which is called genotype. For ES, we use a vector, with each element to be a real number. Then we shall maintain a population with a reasonable size. The aim is to mutate and recombine the genotype representations of the selected individuals (parents) to give us the next generation (children). Then we decide who survives using either ( $\mu$, $\nu$ ) or ( $\mu + \nu$ ). The first one means we select $\mu$ survivors only from $\nu$ children, while the second one means we select survivors from the union of parents and children. ES uses Gaussian perturbation to do the mutation with step size is adjusted as the iteration goes on.

## B.3 The Algorithm and Implementation

The genotype is taken as the unknown rate constants in a vector of real values. The fitness function is the results out of the KINSOVLER compared with experimental results, e.g. the total square error. We have to assume we know the topology of the network and we have a certain amount of data measured. For each particular individual of the population, we feed the reaction rates into the KINSOLVER. The time-dependent concentrations of each species are computed and compared with the experimental data. Fitness for this is just the summation of total discrepancies over all observations. I used Gaussian perturbation to do the mutation. Use uniform random selection for the parent selection to avoid local optima and maintain better diversity. Use ( $\mu$, $\nu$ ) for survivor selection. I chose $\mu$ =15, $\nu$ =100. The perturbation method is uncorrelated n-dimensional step sizes (depending on the number of unknowns):

$$\sigma_i^{'} = \sigma_i exp(\tau^{'} N(0,1)) + \tau N_i(0,1),$$

$$x_i^{'} = x_i + \sigma_i N_i(0,1)$$

where $\tau^{'} \propto 1/\sqrt{2n}$ and $\tau \propto 1/\sqrt{2\sqrt{n}}$, $N(0,1)$ denotes a random number drawn from the Normal distribution with 0 mean and standard deviation 1, $N_i(0,1)$ is the same definition for its corresponding element in the n-dimensional unknown parameter vector.

The following is the overall flow of the algorithm:

1. Randomly initialize $\mu$ individuals.

2. Uniformly and randomly select $\nu$ breeding parents and apply Gaussian mutation to get $\nu$ children. I used [53] for normal distribution random number generator.

3. Use KINSOLVER to compute the fitness of each child.

4. Select the best $\mu$ out of the $\nu$ as new generation. We want to minimize the fitness.

5. Stop when stopping criterion is met. Use the total number of iterations as the stopping criterion because the fitness varies depending on how many experimental data points. Or you can choose if after some certain number of iterations, the fitness doesn't change.

I implemented the algorithm using C++. And the plotting tool is Origin 7.0. Experiments are taken on atlas.cs.uga.edu. All the source codes are available at http://www.physast.uga.edu/ yihyu/es.

## B.4 Experimental Results

### B.4.1 Experimental Setup

Since I don't have any experimental data for this Water Model, I did a virtual experiment that was just basically one run of a specified kinetic model using KINSOLVER with initial concentrations, rate constants in the input file and get the real-time results for all the 6 species as shown in Figure 5, 6, 7, 8, 9, 10. To be more realistic, I also added some random noise to them. For each species, suppose we have 11 experimental data points. Therefore, we have 66 data points in total. The dots in the figures are the actual "experimental" data points.

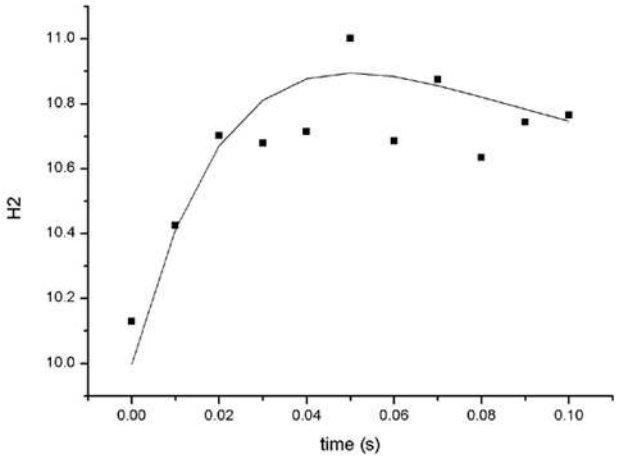


Figure B.2: [H2] v.s. time.

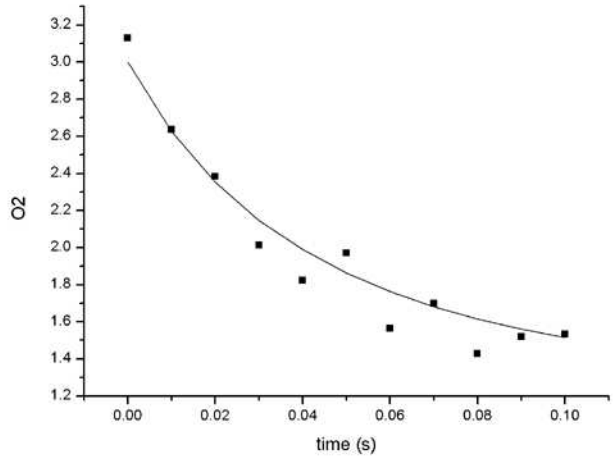


Figure B.3: [O2] v.s. time.

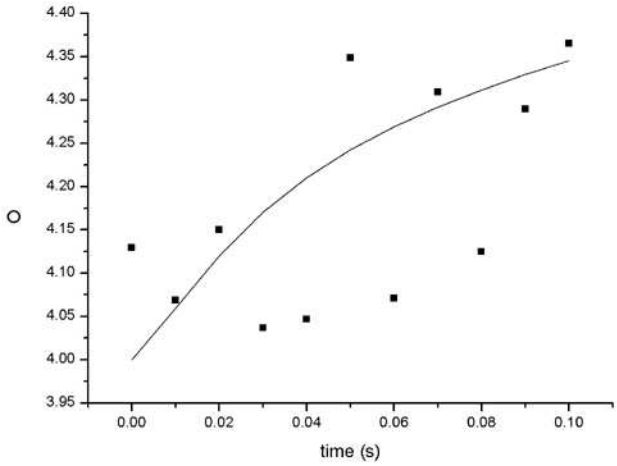


Figure B.4: [O] v.s. time.

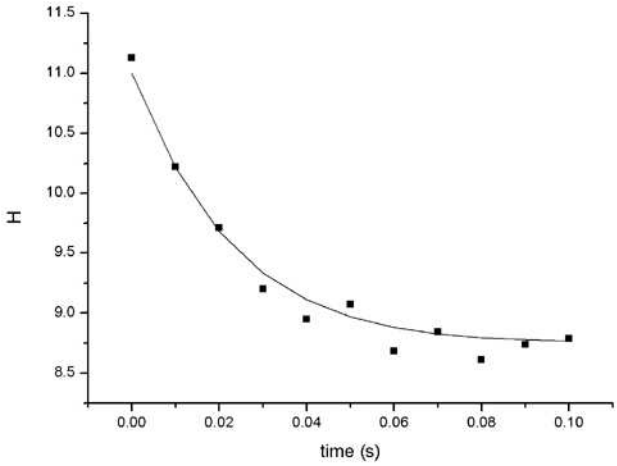


Figure B.5: [H] v.s. time.

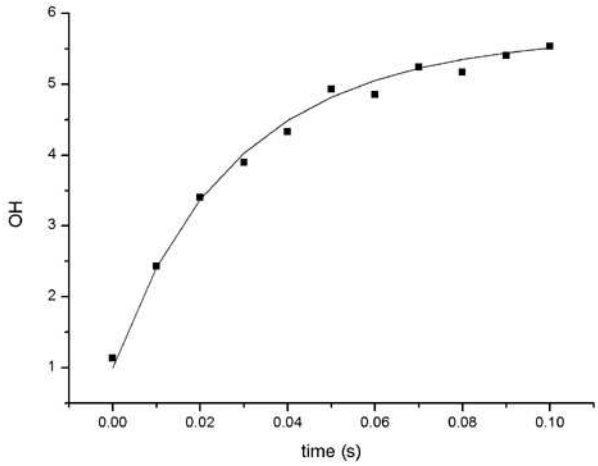


Figure B.6: [OH] v.s. time.

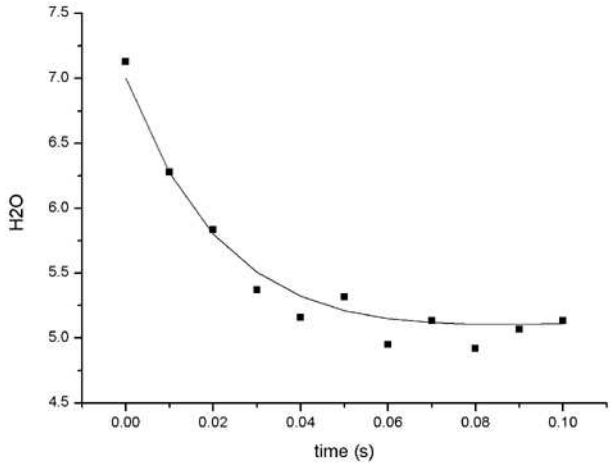


Figure B.7: [H2O] v.s. time.

Table B.2 below shows the actual reaction rates we want to find out.

| **Reaction** | **Value** |
|---|---|
| **kf1** | 1.3 |
| **kb1** | 1.2 |
| **kf2** | 1.4 |
| **kb2** | 0.6 |
| **kf3** | 1.0 |
| **kb3** | 1.3 |

Table B.2: Actual reaction rates for Water Model I.

### B.4.2 Identify the reaction rates and the real time result

- **5 known and 1 unknown.**

This is the simplest case. The first thing we may be interested in is the figure of merit: the fitness versus the iteration number as in B.8. The fitness drops very quickly to about 5, and doesn't improve further for a long time. fk1 converges to 1.3 as in Figure B.9. Both B.8 and B.9 start the plateau at about iteration number 25. The real time concentration of O2 is plotted in B.10 together with the experimental data for kf1 being 1.3029, which is the 'guess' corresponds to the smallest fitness value. Similar properties can be achieved and plotted for the other 5 species.

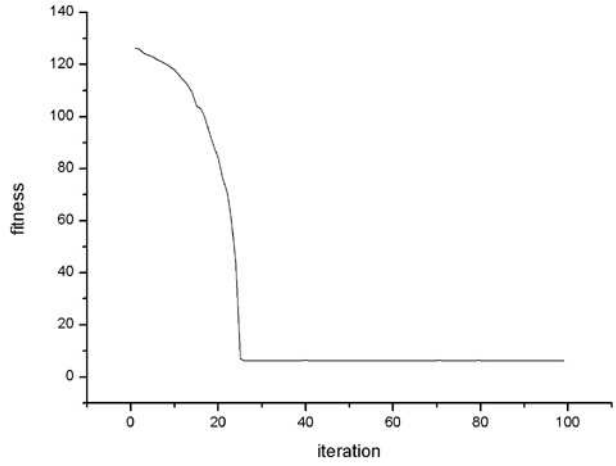


Figure B.8: Fitness v.s. iteration number.

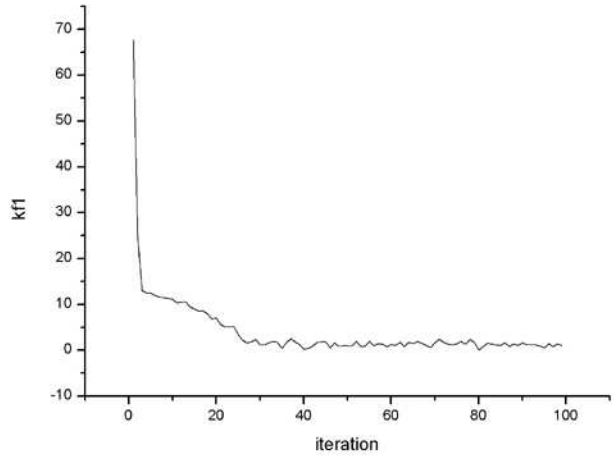


Figure B.9: kf1 v.s. iteration number.

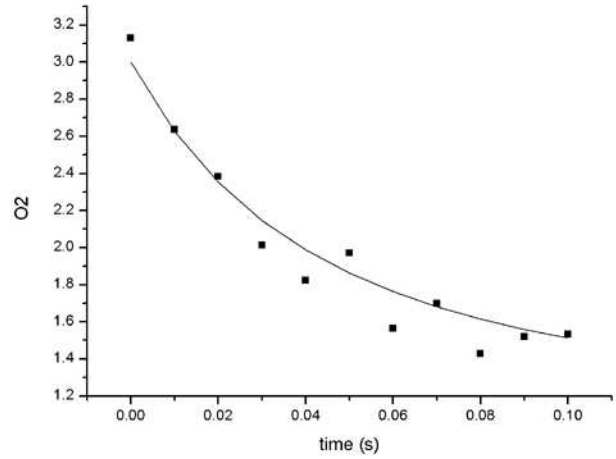


Figure B.10: [O2] v.s. time.

- **4 known and 2 unknown.**

  This is a little bit more complicated due to the dimension of parameter space increased. B.11 shows the decrease of the fitness versus iteration number. More iterations are needed to get to the plateau. Plateau occurs at about 25 for the previous case while it is about 50 for this case. This is reasonable because the complexity of the unknown parameter space is increased. B.12 shows the O2 versus real time again for the best fitness. It is still a pretty good fit. The best solutions for kf1 and kb1 are 1.41795 and 1.16662, respectively.

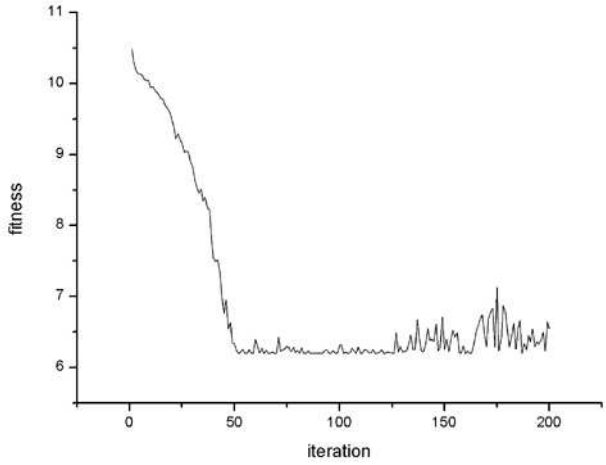


Figure B.11: Fitness v.s. iteration number.

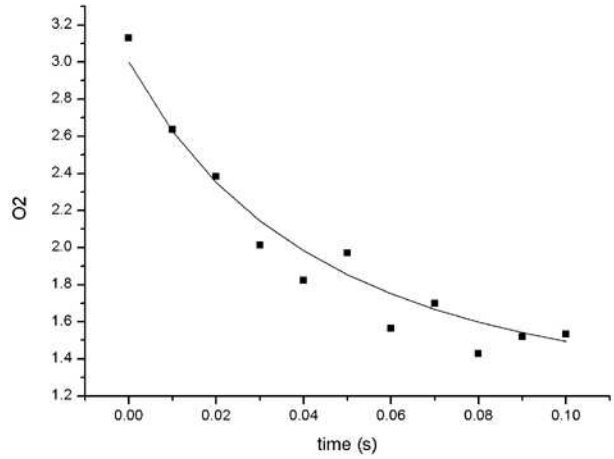


Figure B.12: [O2] v.s. time.

- **All 6 are unknown.**

  As we can see in Figure B.13, the fitness decreases again as iteration number increases again. This time about 200 iterations are needed to get to the start of the smallest fitness plain. Figure B.14 shows that fitting the O2 for the virtual experimental data with the best reaction rates. The reaction rates vector is 1.44572 1.50547 1.42727 0.518262 1.11856 1.22016, which is pretty close to the actual values. As the dimension of the unknown parameter space increases, there could be a lot of local minima for the fitness. We should be careful about the situation, we could get as stuck in some local minima. ES is pretty nice for this problem because it could possibly exert a 'big' perturbation on the previous solution.

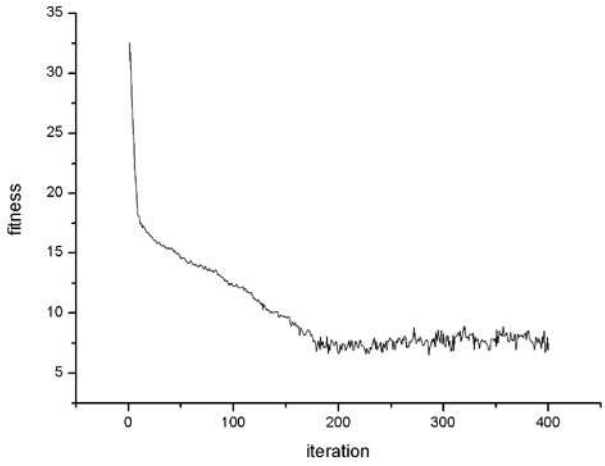


Figure B.13: Fitness v.s. iteration.

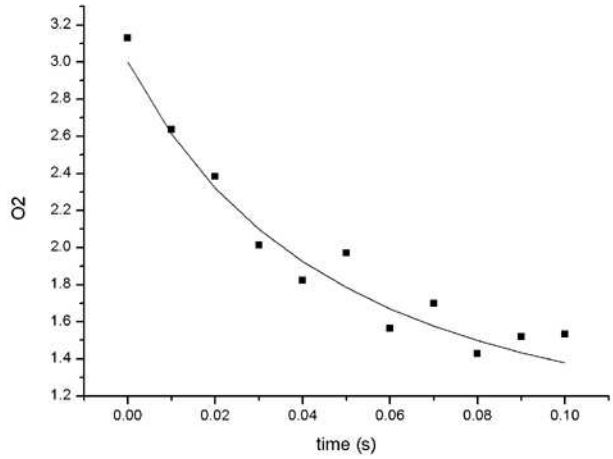


Figure B.14: [O2] v.s. time.

### B.4.3 Predictions for species

We can design the virtual experiment such that only 5 species' data are available, in other words, we shall suppose that measuring H2O is not applicable for some reason. Now, we only have 5*11 = 55 data points now. We can test the prediction ability of this method. Figure B.15 shows the fitness versus iteration. Figure B.16 shows the prediction of H2O together with the actual noised experimental data. The prediction is not as good as earlier example, but it shows the trend of the change of the concentration of H2O. Consider the fact that we have totally no idea about the experimental information about this species at the beginning. This prediction could give experimenters some idea of the magnitude of the concentration of H2O. I also doubt we can further get better fitting because it seems in Figure B.15 that the fitness doesn't change a lot for a long time. I think we can call it "degeneration" since we

don't have information of H2O, there could be many "same" good solutions. How to jump out of the local minima is always an active topic for evolutionary computation. We should further improve the algorithm by making it to be less greedy to solver this problem.

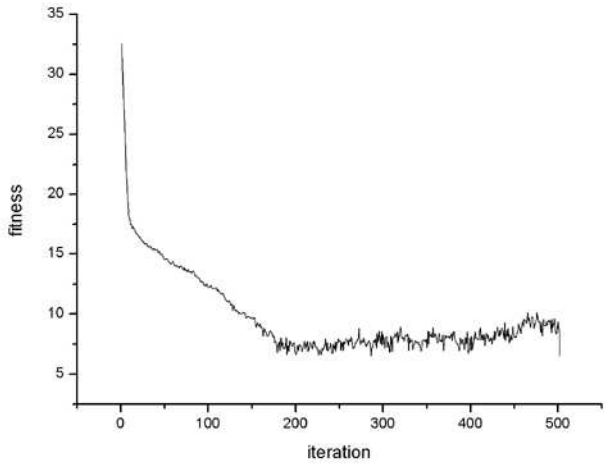


Figure B.15: Fitness v.s. iteration number.

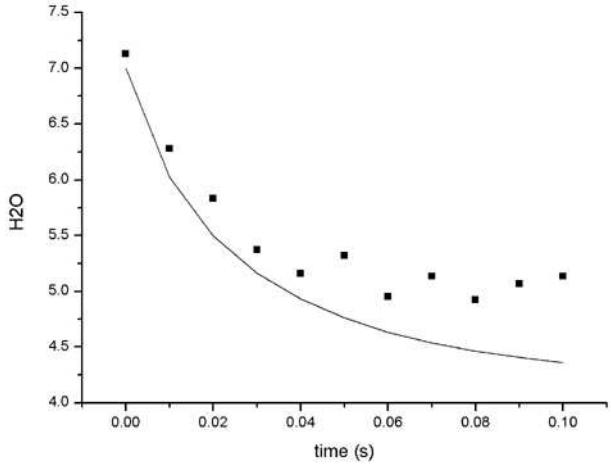


Figure B.16: [O2] v.s. time.

### B.4.4 Future Work

1. We can push the method to more critical situation, such as put more noise to the experimental data, or use fewer data points. Less greedy algorithm could be better for high dimension systems.

2. We can also try more sophisticated ES, such as correlated methods. Since the reaction rates could be correlated to each other. But it's still an open question.

3. Apply this code to other realistic models, such as the *qa* gene cluster, biological clock, repressilator.

4. We can also think of combining MC method and ES method together, exploiting the pros of the 2 methods, such as the power of MC dealing with high dimensional system.