Numerical Solutions of Models for Glucose and Insulin Levels in Critically Ill Patients

A thesis submitted

by

Anh Thai Nhan (B.Sc.)

to

The School of Mathematics, Statistics and Applied Mathematics,
National University of Ireland, Galway

in fulfilment of the requirements for the degree of

Master of Science

September 2011

Thesis Supervisor: Dr. Niall Madden

Head of School: Dr. Ray Ryan

Abstract

This thesis is concerned with the numerical solution of mathematical models for glucose and insulin levels in critically ill patients. These models present several difficulties that make computing accurate solutions efficiently a nontrivial challenge. These difficulties include that the differential equations are stiff, have discontinuous data, and have highly non-uniform dynamics. The numerical methods we use are all examples of classical one-step Runge-Kutta methods. We show how to choose from these methods the ones that are most suited to problems with discontinuous data. We then show how so-called implicit methods can be implemented to generate stable solutions to stiff problems. Finally, in the context of using a Dynamic Bayesian Network to simulate glucose and insulin levels in intensive care unit patients where the glucoregulatory system are very fast and unpredictable, we propose a time stepping control algorithm that allows the stepsize to adapt to changes in the model dynamics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

This thesis is a report on my work on designing and analysing numerical methods for simulating solutions to certain initial values problems in such that way that they can be easily incorporated into an expert system that then tunes the model parameters to a particular case. It is part of a larger project "Nonlinearity and Uncertainty in Drug Modelling" funded by Science Foundation Ireland. It is a strongly interdisciplinary, collaborative project, involving Computer Science, Mathematics and Applied Mathematics, with an application in Medical Informatics, as well as having input from clinicians in University Hospital Galway. There are six core members of the group. Liam O'Callaghan and Petri Piiroinen in Applied Mathematics are constructing new mathematical models based on initial value differential equations. Catherine Enright and Michael Madden in Information Technology incorporate these models into a Dynamic Bayesian Network (DBN) software system that is used to adapt model parameters to individual patients. However, in order to run the DBN, our colleagues need suitable numerical schemes. My role has been to design and implement various numerical methods to solve the models in a way that can be incorporated into the DBN sufficiently.

Ordinary differential equations, particularly *initial value problems* (IVPs), are the most commonly used mathematical tool for modelling of the biological processes (see, e.g. [2, 19]). These models can represent a huge array of phenomena such as population dynamics, infectious diseases, physiological processes in humans, and the growth of cancerous tumours. Of primary interest in this thesis are models for the interactions between glucose and insulin in critically ill patients. The models we study in this thesis are expressed as systems of initial value problems.

In the field of Artificial Intelligence, a *Bayesian Network* is a type of expert system in which a graphical probabilistic model is used to determine the most probable state of variables in a network when all the information required to determine their true state is not available [15, Chap. 14]. It uses a directed acyclic

graph to represent the interactions between its variables. When a Bayesian Network is used in a time series, it is called a *Dynamic* Bayesian Network (DBN). In the context of differential equations, the DBN generates tens of thousands of "particles" that represent different possible values for all the differential equation's parameters, and then solves the differential equation for each of these. When an observation is available, which corresponds to knowing the value of the solution a particular point in time, the DBN then uses Bayes' theorem to determine the probability that a given particle was the "true" one. The DBN again generates a new family of particles and restarts the simulation until a new observation is available. In this way the DBN can recalibrate the model's parameters. Obviously, the DBN cannot solve differential equations exactly. Numerical solutions are therefore required. Furthermore, the computational cost in implementing the DBN is very high since it has to numerically solve for each of the thousands of particles.

Recently, a DBN approach has been used for modeling insulin levels in patients in intensive care unit [5]. As described in [5], the DBN-based framework has been shown to be successful in tackling problems in which several sources of uncertainty are dominant. More specifically, that paper maps the *Intensive Care Unit-Minimal Model* (ICU-MM) developed by Van Herpe et al [20] into a DBN in order to predict the glucose and insulin interaction levels in ICU patients and also to recalibrate the model parameters from the population levels. However, there are still some difficulties that need to be overcome.

Firstly, as mentioned above, the implementation of a DBN is expensive. The DBN model sets up the relationships between every factor that comes from the ICU model and uses a conditional distribution to predict their values over time. Therefore, as shown in [5] at each time slice, a huge number of computations takes place. As a result, the DBN requires algorithms for simulating solutions to IVPs that are accurate but, moreover, highly efficient. In this thesis we consider several ways of doing this. These include the use of high order schemes, the use of implicit methods which gives a stable solution for even long time steps, and adaptive schemes to optimise the time step. As we will see, of these the adaptive method is most suited to the DBN.

Secondly, as discussed in [4]:

> The dynamics of a glucoregulatory system are very fast. A DBN which aims to capture these dynamics would have to be run with time steps of less than 1 minute.

Since the dynamics of the model are not uniform and not known in advance, we are again motivated to design a technique that can adapt the stepsize to capture the corresponding dynamics efficiently. In other words, we need an algorithm that automatically allows the use of large steps during periods of slow change, and small steps during periods of fast change. In this way, significant run time improvements can be obtained. With the challenges from the DBN implementation and the dynamics of glucoregulatory system, a need to design a simple and efficient algorithm has been our main goal during the project.

1.2 Thesis Outline

In the very first stage of my research, my task was to study the numerical solution for the ICU-MM originally proposed in [1, 10, 20]. This mathematical model is expressed as a system of four nonlinear initial value differential equations. Therefore, the fundamental and classical numerical methods for solving IVPs, such as Euler's method and Runge-Kutta methods, form the core of Chapter 2. In this chapter, we give a careful analysis of the errors in numerical solutions generated by one-step methods, with particular attention given to Euler's method. This is because this error analysis is necessary in order to develop more sophisticated algorithms in later chapters.

Most classical examples of initial value problems have smooth coefficients. However the IVPs that come from real world models are based on observed data, which may be discontinuous, or perhaps only piecewise continuous. This difficulty can lead to reduced accuracy and rates of convergence of some one-step methods. This scenario is also addressed in Chapter 2 using several examples. By investigating these, we suggest some numerical methods that should be used for the problems that have discontinuous data. Furthermore, we recommend a simple non-uniform stepsize technique that fully recovers the order of accuracy for some methods in which the rate of convergence is adversely affected by the discontinuity.

In Chapter 3 we investigate the ICU-MM model with real patient data, and address some of the difficulties that arise when solving it numerically. In the second half of the chapter, we introduce a technique to control the uniform stepsize of the one-step methods based on a *prescribed tolerance*. The improvement in terms of computational cost is shown by applying this algorithm to the ICU-MM. It is considered as a simple application of the error analysis in Chapter 2. This idea is developed for another, more sophisticated but efficient algorithm in Chapter 5.

In the middle stage of my 18 month Masters project, our team members from Applied Mathematics proposed an improved model for the glucose and insulin levels in critically ill patients. However, this new model is much more difficult to solve numerically. For example, the new model divides the total simulation period into several subperiods, and *all* of the model parameters may change between periods. Another considerable challenge is the *stiffness* of the model. The numerical schemes of Chapter 2 are all explicit, and as we explain, not appropriate for stiff problems in which the computed solution can be highly unstable. Thus, Chapter 4 deals with the property of stiffness, and focuses on an *implicit scheme* that is stable for stiff problems. The efficiency of the implicit method is shown through the numerical results we present, in particular the result for the mathematical model of our colleagues from Applied Mathematics. However, implementing the implicit method requires the solution of a system of nonlinear equations, so we briefly discuss the iterative methods that are used in our implementation, such as Newton's method and Broyden's method.

As mentioned above, in order to optimise the DBN implementation of glucose models that can take days to run, a technique that can control the stepsize is needed. In addition, the dynamics of the new model are not uniform over time, i.e. derivatives change rapidly in some intervals and slowly in other

intervals. Based on the uniform technique of controlling the stepsize discussed in Chapter 3 and the error analysis of Euler's method in Chapter 2, we develop the adaptive non-uniform time stepping algorithm in Chapter 5. It does not requires a fixed stepsize as with the algorithm in Chapter 3. Instead, this technique chooses a suitable stepsize at each step depending on the dynamics of the systems. Applying this scheme, we carefully investigate the numerical results by calibrating the user-chosen parameters in the adaptive scheme for the latest model that our project members have provided. Furthermore, we also apply this algorithm to a model based on the classic van der Pol oscillator that presents non-uniform dynamics over different intervals. Although originally proposed for modelling electrical circuits, it has a wide range of applications in biology, e.g. [2, Chap. 16]. Chapter 5 concludes with the application of the adaptive algorithm to the glucose insulin model, which, as described in the recently submitted research article [4], has been successfully incorporated into the DBN.

1.3 Experimental Data

In our research group's project, the models and methods have been validated by comparison with experimental data from real patients in the ICU department of University Hospital Galway (UHG). In this thesis, we give results based on some of this data. In Chapter 3 we use a data set from study [5] we refer to as *Patient 23*. This data was collected from the data base archives of UHG. Permission for collecting this data was given by the Galway Research Ethics Committee, UHG. All records were anonymised and stored on encrypted drives.

In Chapter 4 and Chapter 5, we use data that was directly collected for this research project, and which we refer to as *Patient 102*. This data was measured by two colleagues, Dr Brian Harte and Ms Anne Mulvey, in June 2010 in UHG. For this specific research, not only glucose levels, but other factors that affect glucose levels were also measured. These other factors had to be taken into account since DBN can recalibrate these parameters to an individual patient from the general population-level values. Again permission for collecting this data was given by the Galway Research Ethics Committee, UHG.

1.4 Acknowledgements

This thesis is based on works supported by the Science Foundation Ireland under Grant No 08/RFP/CMS1254. I am grateful to group members Catherine Enright and Michael Madden who have helped me understand the DBN framework and operation. I am also grateful to other group members Liam O'Callaghan and Petri Piiroinen for their thorough explanation of their glucose insulin model and parameters. I am also indebted to my supervisor Dr. Niall Madden for his constant encouragement, motivation and helpful advice.

Chapter 2

Numerical Methods for Initial Value Problems

In this chapter we briefly review some theory about initial value differential equations, and the classical one-step Runge-Kutta (RK) methods used to solve them numerically. We state the important theoretical results concerning these methods that relate to their accuracy and convergence in Section 2.2.4. Along with these, we give numerous worked examples. A very compact Matlab code to implement them is presented in Section 2.2.8. In Section 2.3, we show that some of these methods are not appropriate for problems with discontinuous coefficients: the accuracy and the rate of convergence is less than that theoretically possible for problems with discontinuous data. Finally, in Section 2.3, we describe a nonuniform step algorithm with which we successfully recover the maximum rate of convergence of some numerical methods.

2.1 Introduction to Initial Value Problems

For many years mathematical models based on differential equations have been used as effective and successful tools in order to represent the biological processes in nature (e.g., [2, 13, 19]). One of the simplest such models is concerned with predator and prey populations. Suppose we can observe the number of wolves and number of rabbits in a forest. These values will change over time due to interactions between the wolves and rabbits. Let $x(t)$ be the number of rabbits and $y(t)$ be the number of wolves at the time t. At the time t_0, suppose the number of rabbits is $x(t_0) = x_0$, and the number of wolves is $y(t_0) = y_0$ and without predators, assume the prey population grows and, without prey, the predator population declines. Then, this interacting process can be modeled as

$$\begin{aligned} \frac{dx}{dt} &= (a_1 - b_1 y)x, \\ \frac{dy}{dt} &= (-a_2 + b_2 x)y, \end{aligned} \tag{2.1}$$

where a_1, a_2, b_1 and b_2 are positive parameters relating to the interaction of wolves and rabbits. The system of differential equations (2.1) and conditions $x(t_0) = x_0$, $y(t_0) = y_0$ on the interval $[t_0, t_N]$ are called an *Initial Value Problem (IVP).*

We may suppose that the general form of the differential equation is written as

$$\frac{dy}{dt} = f(t, y), \quad \text{for } t > t_0, \tag{2.2}$$

with the value of the function $y(t)$ given at t_0, i.e., $y(t_0) = y_0$. In many cases, such an initial value problem as (2.2) cannot be solved exactly. Therefore, a numerical solution is needed. The basic idea for solving (2.2) numerically is to divide the interval of interest into discrete steps of fixed length and find approximations to the function y at those values of t. That is, we find numerical solutions at $t_1, \ldots, t_N$ with $t_{i+1} - t_i = h > 0$.

Before considering numerical schemes for initial value problems, we briefly recall some facts from the theory of ordinary differential equations. The presentation here is based on [17, Chap. 12].

Definition 2.1. A function f satisfies a ***Lipschitz Condition*** (with respect to its second argument) in the rectangular region D if there is a positive real number L such that

$$|f(t, u) - f(t, v)| \leq L|u - v|, \tag{2.3}$$

for all $(t, u) \in D$ and $(t, v) \in D$.

Example 2.1. The function $f(t, y) = (t - y)/3$ satisfies the Lipschitz condition in $\mathbb{R}^2$, because with $(t, u), (t, v) \in \mathbb{R}^2$, we have

$$|f(t, u) - f(t, v)| = \left| \frac{t-u}{3} - \frac{t-v}{3} \right| = \left| \frac{-u+v}{3} \right| \leq \frac{1}{3}|u - v|.$$

So the function $f(t, y) = (t - y)/3$ satisfies the Lipschitz condition with $L = 1/3$.

Example 2.2. [17, Exer. 12.1] The function

$$f(t, y) = \frac{2y}{1 + y^2} \left(1 + \frac{1}{e^{|t|}} \right), \tag{2.4}$$

satisfies a Lipschitz condition; this may be shown as follows. Let us take any $x \in \mathbb{R}$, then

$$\begin{aligned} |f(t, u) - f(t, v)| &= \left| \frac{2u}{1 + u^2} \left(1 + \frac{1}{e^{|t|}} \right) - \frac{2v}{1 + v^2} \left(1 + \frac{1}{e^{|t|}} \right) \right|, \\ &= \left| \left(1 + \frac{1}{e^{|x|}} \right) \left(\frac{2u}{1 + u^2} - \frac{2v}{1 + v^2} \right) \right|. \end{aligned}$$

Since $\left(1 + \frac{1}{e^{|x|}} \right) \leq 2$ for all $x \in \mathbb{R}$, hence

$$\begin{aligned} |f(t, u) - f(t, v)| &\leq 2 \left| \frac{2u(1 + v^2) - 2v(1 + u^2)}{(1 + u^2)(1 + v^2)} \right| \\ &= 4 \left| \frac{1 + uv}{(1 + u^2)(1 + v^2)} \right| |u - v|. \end{aligned}$$

In addition, $\dfrac{1+uv}{(1+u^2)(1+v^2)} \leq 1$, so we get

$$|f(t,u) - f(t,v)| \leq 4|u-v|.$$

Therefore, (2.4) satisfies the Lipschitz condition with Lipschitz constant $L = 4$.

The following theorem states sufficient conditions on f for the existence of a solution to the ordinary differential equation (2.2).

Theorem 2.1 (Picard's Theorem)**.** *Suppose that the real-valued function $f(t,y)$ is continuous for $t \in [t_0, t_M]$ and $y \in [y_0 - C, y_0 + C]$; that $|f(t,y_0)| \leq K$ for $t_0 \leq t \leq t_M$; and that f satisfies the Lipschitz condition (2.3). If*

$$C \geq \frac{K}{L}\left(e^{L(t_M - t_0)} - 1\right),$$

then (2.2) with initial value $y(t_0) = y_0$ has a unique solution on $[t_0, t_M]$. Furthermore,

$$|y(t) - y(t_0)| \leq C, \quad t_0 \leq t \leq t_M.$$

A very detailed proof can be found in [17, p.312]. We now apply this theorem to some examples.

Example 2.3. Consider the following initial value problem:

$$\frac{dy}{dt} = f(t,y) = 2y - 5, \quad \text{with,} \;\; y(0) = y_0 = 3.$$

In order to apply Picard's Theorem to show that it has a unique solution on $[0, \infty)$, we need to show that there exists a constant C that satisfies

$$C \geq \frac{K}{L}\left(e^{L(t_M - t_0)} - 1\right).$$

Indeed, the function $f(t,y) = 2y - 5$ satisfies the Lipschitz condition with $L = 2$ as the Lipschitz constant:

$$\begin{aligned} |f(t,u) - f(t,v)| &= |(2u-5) - (2v-5)|, \\ &= |2(u-v)| \leq 2|u-v|. \end{aligned}$$

So, $L = 2$ and $f(t,y_0) = 1 = K$. Therefore, with any $t_M \in (0,\infty)$ we can choose C large enough to get:

$$C \geq \frac{K}{L}\left(e^{L(t_M - t_0)} - 1\right) = \frac{1}{2}\left(e^{2t_M} - 1\right).$$

Since t_M is arbitrary in $(0,\infty)$, by Picard's Theorem, Example 2.3 has a unique solution in $[0,\infty)$. The exact solution is $y(t) = (e^{2t} + 5)/2$.

The above example is somewhat simple because it is an *autonomous* differential equation. That means the independent variable does not appear in the right hand side of the differential equation. Now, consider another more complicated example of a non-autonomous differential equation.

Example 2.4. Show that

$$\frac{dy}{dt} = \frac{3y-1}{t}, \quad y(1) = 1,$$

has a unique solution on $[1,\infty)$.

First, we find the Lipschitz constant L on $[1, \infty)$ by:

$$|f(t,u) - f(t,v)| = |\frac{3u-1}{t} - \frac{3v-1}{t}| = |\frac{3}{t}(u-v)|,$$
$$\leq \frac{3}{|t|}|u-v| \leq 3|u-v|.$$

Furthermore,

$$|f(t,y_0)| = \left|\frac{3y_0-1}{t}\right| = \left|\frac{2}{t}\right| \leq 2, \quad \forall t \in [1,\infty).$$

Now, we have $L = 3$ and $K = 2$, independent of C, and for any $t_M \in [1, \infty)$, C can be chosen by

$$C \geq \frac{K}{L}\left(e^{L(t_M - t_0)} - 1\right) = \frac{2}{3}\left(e^{2t_M} - 1\right).$$

By the Picard's Theorem, Example 2.4 has a unique solution on $[1, \infty)$. Its exact solution is $y(t) = \frac{2}{3}t^3 + \frac{1}{3}$.

2.2 One-step methods for IVPs

2.2.1 Euler's Method

Suppose the initial value problem (2.2) has a unique solution. In many cases, however, we can not find the exact solution $y = y(t)$ in explicit form. Nevertheless, we can define the approximate values of the solution $y_1, y_2, \ldots, y_N$ corresponding to the set of N given points $t_1 < t_2 < \ldots < t_N$. The set of values $\{y_1, y_2, \ldots, y_N\}$ are called the "numerical solution". We will briefly describe some numerical schemes in this section.

Euler's Method is the simplest method of approximating the solution of the differential equation $y' = f(t,y)$ starting by treating the function f as a constant, $f(t_0, y_0)$, and replacing the derivative y' by the forward difference quotient. Let y_i denote the approximation for $y(t)$ at $t = t_i$ (i.e., $y_i \approx y(t_i)$). Then $y_1 = y_0 + hf(t_0, y_0)$, where $h = (t_N - t_0)/N$. In general, Euler's method is written as:

$$y_{i+i} = y_i + hf(t_i, y_i), \qquad i = 0, 1, \ldots, N-1. \tag{2.5}$$

There are several other ways to justify the formula in (2.5), which we now describe.

Geometrically, Euler's Method corresponds to using the line tangent to the solution curve $y(t)$ at (t_0, y_0) to find y_1. Using the value $y' = f(t_0, y_0)$ as the slope for the tangent line, the equation of the tangent line at the point (t_0, y_0) is

$$y - y_0 = f(t_0, y_0)(t - t_0).$$

At the point $t = t_1 = h + t_0$, we have

$$y_1 = y_0 + f(t_0, y_0)(t - t_0) = y_0 + hf(t_0, y_0).$$

Continue to use the approximate value $f(t_1, y_1)$ as the slope for next tangent line. This gives Euler's Method (2.5).

Algebraically, Euler's Method can be found by replacing the derivative y' by the forward difference approximation; at the first step

$$y' = f(t_0, y_0),$$

becomes

$$\frac{y_1 - y_0}{t_1 - t_0} \cong f(t_0, y_0),$$

so

$$y_1 = y_0 + hf(t_0, y_0).$$

Therefore, we again arrive at Euler's scheme (2.5).

Taylor's Theorem gives the most rigorous approach to deriving Euler's formula. Suppose that y'' is differentiable, using a Taylor expansion for y at $t_{i+1} = t_i + h$, we get

$$y(t_{i+1}) = y(t_i) + hy'(t_i) + \frac{h^2}{2}y''(\xi), \tag{2.6}$$

for $\xi \in [t_i, t_{i+1}]$. By taking $y(t_{i+1}) = y_{i+1}$, $y(t_i) = y_i$ and $y'(t_i) = f(t_i, y_i)$, we obtain Euler's method again. This also suggests the local discretization error at each step is $\mathcal{O}(h^2)$.

Example 2.5. Consider the initial value problem from [18, p. 285]

$$y' = ty + t^3, \quad y(0) = 1, \quad t \in (0, 1].$$

This has the exact solution $y(t) = 3e^{t^2/2} - t^2 - 2$. Figure 2.1 shows Euler's method applied to the problem with $N = 8$ steps. The error at each step is quantified by the distance along the vertical axis between the true solution and the approximate solution. We can observe the error accumulation after every step in this figure (this concept of error accumulation is made more rigorous in Definition 2.2). Table 2.1 shows the errors in the approximate solutions generated by applying Euler's method to Example 2.5 with different numbers of steps. In this table, we can see that the errors reduce by a factor of 2 when the number of steps is doubled. This suggests that, although the local discretization error may be proportional to h^2, as implied by (2.6), the accumulated error is proportional to the stepsize h.

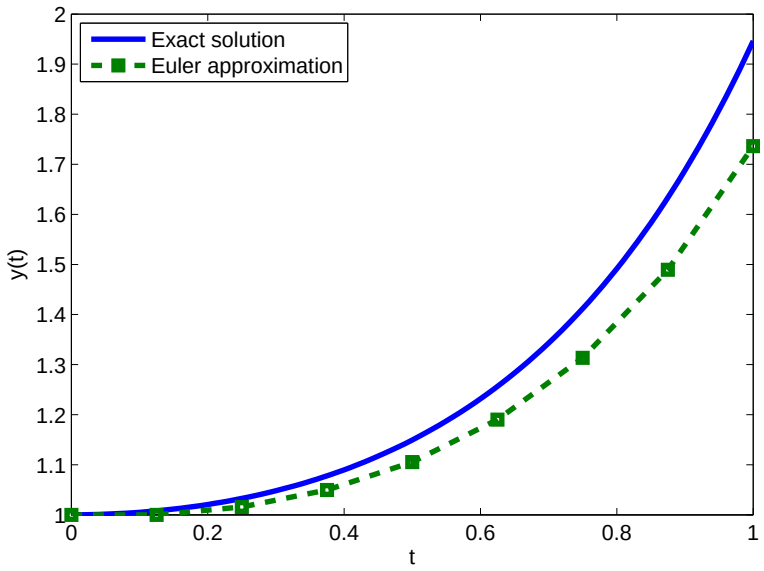


Figure 2.1: True solution and Euler's method with $N = 8$ for Example 2.5.

N	$\max_{i=0,1,\ldots,N} \lvert y(t_i) - y_i \rvert$
2	6.337e-001
4	3.789e-001
8	2.101e-001
16	1.111e-001
32	5.720e-002
64	2.903e-002
128	1.463e-002

Table 2.1: Errors in the numerical solution obtained by applying Euler's method to Example 2.5

.

2.2.2 Euler's method for systems of IVPs

In practice, most real world problems are complex and involve multiple interacting components, and so are modelled by *systems* of differential equations. Euler's method (2.5) can be applied to systems of ODEs. In the following, we consider a system of two first-order ODEs in detail:

$$y_1' = f_1(t, y_1, y_2),$$
$$y_2' = f_2(t, y_1, y_2),$$

with initial values: $y_1(t_0) = y_{1,0}$ and $y_2(t_0) = y_{2,0}$. Then we have the approximation:

$$y_{1,i+1} = y_{1,i} + hf_1(t_i, y_{1,i}, y_{2,i}),$$
$$y_{2,i+1} = y_{2,i} + hf_2(t_i, y_{1,i}, y_{2,i}).$$

From this, we can easily generalize Euler's Method to the case of a system of m equations:

$$y_{j,i+1} = y_{j,i} + hf_j(t_i, y_{1,i}, \ldots, y_{m,i}), \quad \text{for } j = 1, 2, \ldots, m.$$

We now employ Euler's Method to a simple coupled system where we know the exact solution.

Example 2.6. Let us consider the system

$$\begin{aligned} \frac{dy_1}{dt} &= -1 + e^t + y_1 y_2, \\ \frac{dy_2}{dt} &= -1 - e^{-t} + y_1 y_2, \end{aligned}$$

on the interval $(0, 1]$ where $y_1(0) = 1$ and $y_2(0) = 1$. The exact solution is

$$y_1(t) = e^t, \qquad y_2(t) = e^{-t}.$$

Therefore, we can easily calculate the errors for each component that occurs when using Euler's Method. These are shown in Table 2.2. As with the scalar example above, the error of each component is proportional to $h = 1/N$.

N	$\max_{i=0,1,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,1,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$
2	5.1892e-001	2.9614e-001
4	3.4399e-001	2.2045e-001
8	2.1269e-001	1.4786e-001
16	1.2274e-001	8.9559e-002
32	6.6926e-002	5.0146e-002
64	3.5122e-002	2.6685e-002
128	1.8018e-002	1.3787e-002

Table 2.2: Errors in the numerical solution obtained by applying Euler's method to Example 2.6

.

Figure 2.2 shows the graph of the true solution and Euler's approximation with the number of steps $N = 8$. Two dash lines marked with squares are the approximate solutions. We again see the gradually increasing accumulated error. We will study this phenomenon in the next section.

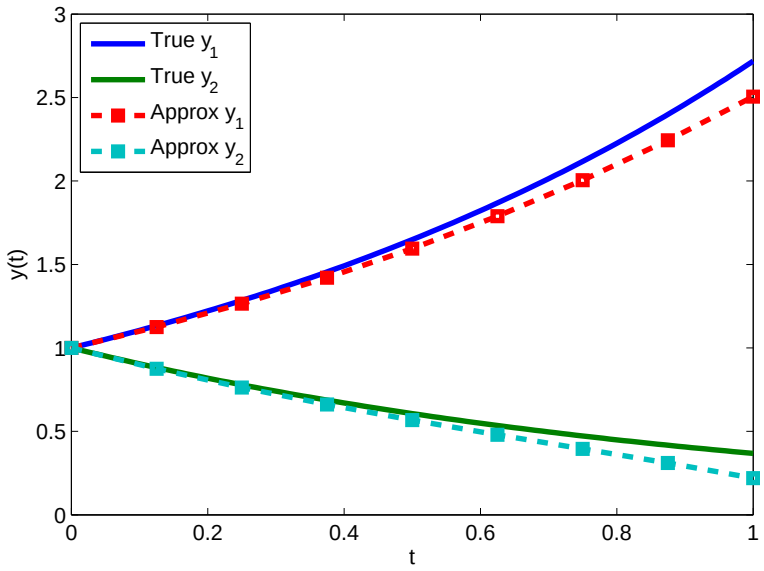


Figure 2.2: True solution and Euler's method with $N = 8$ for Example 2.6.

2.2.3 Analysis of Euler's Method

In this section, we will have a closer look at the error that occurs when using Euler's Method. This is the basis of much of the theoretical developments in later chapters. By understanding this error carefully, we can design and develop algorithms to control it.

The Euler's Method is a very special case of so-called *one-step methods* that have the general form:

$$y_{i+1} = y_i + h\phi(t_i, y_i; h). \tag{2.7}$$

In the method of Euler, one has $\phi(t_i, y_i; h) = f(t_i, y_i)$.

In order to assess the accuracy of the one-step method (2.7), we first introduce some definitions.

Definition 2.2. The difference between the approximate solution and the true solution of the initial value problem at step i is the *global error*:

$$\varepsilon_i = y(t_i) - y_i.$$

The global error quantifies the accumulated error from the previous steps together with the new error from the Euler approximation at this step. In addition, suppose the quantity

$$\frac{\varepsilon_{i+1} - \varepsilon_i}{h} = \frac{y(t_{i+1}) - y_{i+1} - (y(t_i) - y_i)}{h} = \frac{y(t_{i+1}) - y(t_i) - h\phi(t_i, y(t_i); h)}{h}$$

shows us how well the one-step method approximates the exact solution on each interval $[t_i, t_i + h]$. This leads naturally to the next definition.

Definition 2.3. The *truncation error* is

$$T_i := \frac{y(t_{i+1}) - y(t_i)}{h} - \phi(t_i, y(t_i); h). \tag{2.8}$$

For Euler's Method, which has $\phi(t_i, y_i; h) = f(t_i, y_i)$, we substitute the Taylor expansion (2.6) into (2.8) to get:

$$T = \max_{i=0,\ldots,N} |T_i| \leq \frac{h}{2} \max_{t_0 \leq t \leq t_N} |y''(t)|. \tag{2.9}$$

Since $y''(t)$ is independent of h, we can write (2.9) as $T \leq Ch$ for some constant C independent of h. This can also be expressed succinctly as

$$T = \mathcal{O}(h). \tag{2.10}$$

The next theorem relates the global error and the truncation error [17, Theorem 12.2]:

Theorem 2.2. *Consider the general one-step method* (2.7)*, where ϕ is a continuous function of its arguments and satisfies a Lipschitz condition with respect to its second argument. Then,*

$$|\varepsilon_N| \leq T \left(\frac{e^{L(t_N - t_0) - 1}}{L} \right),$$

where $T = \max_{i=1,2,\ldots,N} |T_i|$.

From (2.9), the global error for Euler's Method can be bounded by

$$|\varepsilon_N| \leq \frac{h}{2} \max |y''(t)| \left(\frac{e^{L(t_N - t_0) - 1}}{L} \right).$$

Since L, y'', t_N and t_0 depend on the problem data, we can write this more succinctly as $|\varepsilon_N| \leq hK$. That is we can rewrite $|\varepsilon_N| = \mathcal{O}(h)$, which emphasises that the error is directly proportional to the stepsize h.

2.2.4 Consistency and Convergence

We would like to know if our one-step method is reliable. In other words, we wish to examine the convergence behavior as $h \to 0$ of an approximation solution furnished by a one-step method, and hope to show that

$$\lim_{h \to 0} y_N = y(t_N).$$

Equivalently,

$$\lim_{h \to 0} \varepsilon_N = 0.$$

This issue is the crucial part of this section. First, we need some definitions.

Definition 2.4. The **order of accuracy** (also called the *order of convergence*) of a numerical method is $\mathfrak{p}$ if there are positive constants K and h_0 such that

$$|T_N| \leq Kh^{\mathfrak{p}}, \quad \text{for all } h \leq h_0.$$

From (2.10), we see that Euler's method is 1^{st}-order.

Definition 2.5. A one-step method $y_{i+1} = y_i + h\phi(t_i, y_i; h)$ is **consistent** with the differential equation $y'(t) = f\big(t, y(t)\big)$ if $f(t, y) \equiv \phi(t, y; 0)$.

As mentioned, the Euler's method has $\phi(t, y; h) = f(t, y)$, hence it is consistent.

Definition 2.6. We say that the one-step method is **convergent** if

$$\lim_{h \to 0} |\varepsilon_N| = 0.$$

In particular, the global error for Euler's method can be bounded as $\varepsilon_N \leq Ch$. Therefore, Euler's method converges for small enough h. With arguments similar to those used to prove Theorem 2.2, one can prove the convergence of the general one-step method [17, Theorem 12.3].

Corollary 2.1. *Consider the initial value problems (2.2) and one-step method (2.7). Let ϕ be continuous and satisfy the Lipschitz condition and suppose this one-step method has*

$$|T| \leq Ch^{\mathfrak{p}}.$$

Then,

$$|\varepsilon_N| \leq Ch^{\mathfrak{p}}. \tag{2.11}$$

Remark 2.1. *In Section 2.3, we will consider a case where ϕ is not continuous, so this theory does not hold.*

From Corollary 2.1 and Definition 2.4, we can see that if a one-step method has order $\mathfrak{p}$ then there exists a constant K such that

$$|\varepsilon_N| \leq Kh^{\mathfrak{p}}.$$

Note that the above inequality only gives a *lower bound* for the rate of convergence, $\mathfrak{p}$. That is, the scheme may actually have a higher rate of convergence in practice than is given by the theory. Therefore, it is common to numerically verify if this given rate is *sharp*; that is, not only do we have that $|\varepsilon_N| \leq Kh^{\mathfrak{p}}$, but actually $|\varepsilon_N| \approx Kh^{\mathfrak{p}}$. A standard approach is the use that

$$\mathfrak{p} \approx \log_2 \left(\frac{|\varepsilon_N|}{|\varepsilon_{2N}|} \right).$$

To justify this, let ε_N be the global error at with respect to the number of steps N and the corresponding stepsize be h. We obtain

$$|\varepsilon_N| \approx Kh^{\mathfrak{p}}.$$

When the number of steps is doubled to $2N$, then the new stepsize is $h/2$ and,

$$|\varepsilon_{2N}| \approx K \left(\frac{h}{2} \right)^{\mathfrak{p}}.$$

Therefore,

$$\frac{|\varepsilon_N|}{|\varepsilon_{2N}|} \approx 2^{\mathfrak{p}}.$$

and,

$$\mathfrak{p} \approx \log_2 \left(\frac{|\varepsilon_N|}{|\varepsilon_{2N}|} \right).$$

In Table 2.3 we again give the computed errors for various N for Example 2.5, as in Table 2.1, but now with the computed rate of convergence. We can see that the method is indeed 1^{st}-order convergent.

N	$\max_{i=0,1,\ldots,N} \lvert y(t_i) - y_i \rvert$	**Rate**
2	6.337e-001	
4	3.789e-001	0.7418
8	2.101e-001	0.8510
16	1.111e-001	0.9191
32	5.720e-002	0.9577
64	2.903e-002	0.9783
128	1.463e-002	0.9890

Table 2.3: Rate of convergence of Euler's method for Example 2.5

.

2.2.5 Runge-Kutta 2

Runge-Kutta 2 (RK2) is a simple approach to improve upon the accuracy we can obtain from Euler method. The general RK2 is written as:

$$\begin{aligned} k_1 &= f(t_i, y_i), \\ k_2 &= f(t_i + \alpha h, y_i + \beta h k_1), \\ y_{i+1} &= y_i + h(A_1 k_1 + A_2 k_2). \end{aligned}$$

If A_1 and A_2 are chosen so that $A_1 + A_2 = 1$, then the method is consistent. Moreover, if we take $\alpha = \beta$ and $A_2 = 1/(2\alpha)$, then the method is "2^{nd} order accurate":

$$|y(t_i) - y_i| \leq Ch^2,$$

It is very useful and convenient to display the coefficients as a Butcher tableau [3, §232], as in Table 2.4 below. Let us suppose a Runge-Kutta method has $\mathfrak{s}$ stages. Then, the tableau is defined by the square matrix $\boldsymbol{\beta} = (\beta_{ij}) \in \mathbb{R}^{\mathfrak{s}\times\mathfrak{s}}$ and the two vectors $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{\mathfrak{s}})^T \in \mathbb{R}^{\mathfrak{s}}$, $\boldsymbol{A} = (A_1, \ldots, A_{\mathfrak{s}})^T \in \mathbb{R}^{\mathfrak{s}}$.

$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$
	$\boldsymbol{A}^T$

Table 2.4: Butcher tableau form.

With the Butcher tableau above, the general scheme is

$$\begin{aligned} k_1 &= f(t_i + \alpha_1 h, y_i + \beta_{11} h k_1 + \beta_{12} h k_2 + \ldots + \beta_{1s} h k_s), \\ k_2 &= f(t_i + \alpha_2 h, y_i + \beta_{21} h k_1 + \beta_{22} h k_2 + \ldots + \beta_{2s} h k_s), \\ \vdots &= \vdots \qquad\qquad \vdots \\ k_s &= f(t_i + \alpha_s h, y_i + \beta_{s1} h k_1 + \beta_{s2} h k_2 + \ldots + \beta_{ss} h k_s), \\ y_{i+1} &= y_i + h(A_1 k_1 + A_2 k_2 + \ldots + A_s k_s). \end{aligned} \tag{2.12}$$

In this section, we only focus on numerical methods that are *explicit*. That is where k_i in the right hand side of (2.12) is not dependent of k_j for $j \geq i$. Therefore, the matrix $\boldsymbol{\beta}$ is *strictly lower triangular*. So the zero elements of the matrix $\boldsymbol{\beta}$ can be omitted for the usual notational convention.

Remark 2.2. *Notice that if $\boldsymbol{\beta}$ is not strictly lower triangular, then to compute k_i we need $k_1, k_2, \ldots, k_s$. That is, in general, such methods are* implicit*, and are much more difficult to implement in practice. We will return to implicit methods later in Chapter 4.*

We now consider the Butcher tableau for specific explicit Runge-Kutta second order methods. In particular, one can choose $\alpha = \beta = 1, A_1 = A_2 = 1/2$ and get the *Improved Euler Method* or can take $\alpha = \beta = 1/2, A_1 = 0, A_2 = 1$ and get the *Modified* (*Midpoint*) *Euler Method*. Table 2.5 shows these methods in tableau form.

0		
1	1	
	1/2	1/2

0		
1/2	1/2	
	0	1

Table 2.5: The Improved (left) and Modified (right) Euler Methods

If we apply the Modified method to a system of m first-order ODEs, we can formulate the methods as follows: for $j = 1, 2, \ldots, m$,

$$\begin{aligned} k_{j,1} &= f_j(t_i, y_{1,i}, \ldots, y_{m,i}), \\ k_{j,2} &= f_j(t_i + \frac{h}{2}, y_{1,i} + \frac{h}{2} k_{1,1}, \ldots, y_{m,i} + \frac{h}{2} k_{m,1}), \\ y_{j,i+1} &= y_{j,i} + k_{j,2}. \end{aligned}$$

With the Improved method, the general formula for the system is:

$$\begin{aligned} k_{j,1} &= f_j(t_i, y_{1,i}, \ldots, y_{m,i}), \\ k_{j,2} &= f_j(t_i + h, y_{1,i} + h k_{1,1}, \ldots, y_{m,i} + h k_{m,1}), \\ y_{j,i+1} &= y_{j,i} + \frac{h}{2}(k_{j,1} + k_{j,2}). \end{aligned}$$

We now apply the Improved and Modified methods to our previous Example 2.5. The results are shown in Table 2.6. For this table, when the number of steps is doubled, the error is reduced by the factor of four compared with the factor of two in Table 2.3.

N	Improved		Modified	
	ε_N	p_N	ε_N	p_N
2	1.282e-002		1.128e-001	
4	4.846e-003	1.4035	3.418e-002	1.7225
8	1.610e-003	1.5894	9.417e-003	1.8598
16	4.440e-004	1.8587	2.469e-003	1.9312
32	1.174e-004	1.9193	6.319e-004	1.9662
64	3.014e-005	1.9618	1.598e-004	1.9833
128	7.635e-006	1.9808	4.019e-005	1.9917

Table 2.6: Errors in the solutions to Example 2.5 using the Improved and Modified Euler's Methods.

Example 2.7. The model we want to study in Chapter 3 is a system of four equations. We now consider an example of a system of 4 equations,

$$\begin{aligned}
\frac{dy_1}{dt} &= y_1 + y_2^2 + y_4^2 - t, \\
\frac{dy_2}{dt} &= y_4, \\
\frac{dy_3}{dt} &= y_3 y_4, \\
\frac{dy_4}{dt} &= -y_2,
\end{aligned}$$

on the interval $(0, 1]$ with initial values $\boldsymbol{y}(\mathbf{0}) = (1, 0, 1, 1)^T$. The exact solution is

$$\boldsymbol{y} = \begin{pmatrix} e^t + t \\ \sin(t) \\ e^{\sin(t)} \\ \cos(t) \end{pmatrix}.$$

With $N = 4$ intervals, the stepsize $h = (t_M - t_0)/N = 0.25$. The approximate and exact solution are graphed in Figure 2.3.

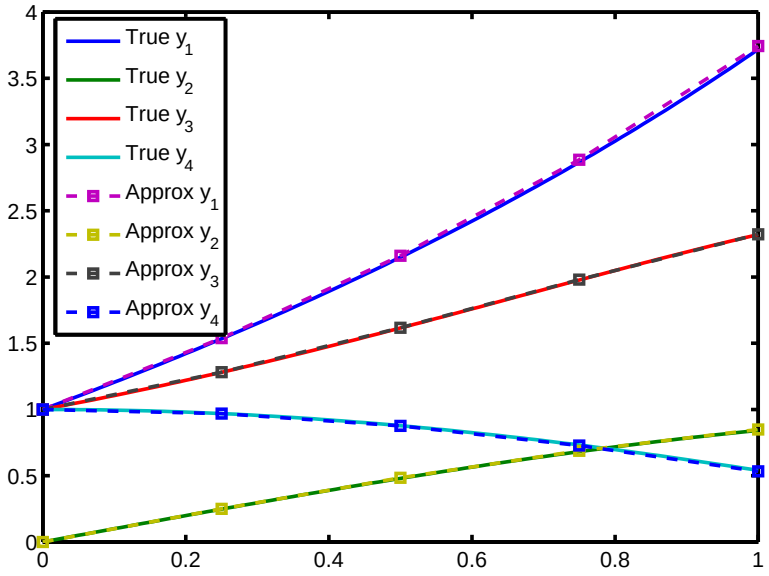


Figure 2.3: The graph of Example 2.7 using Improved Euler Method.

The errors of each component can be seen in Table 2.7. When N is doubled, a reduction by a factor of four is seen in all four components. Therefore, the order of convergence of these methods is $\mathfrak{p} = 2$.

N	$\max_{i=1,\ldots,N}\lvert y_1(t_i) - y_{1,i}\rvert$	$\max_{i=1,\ldots,N}\lvert y_2(t_i) - y_{2,i}\rvert$	$\max_{i=1,\ldots,N}\lvert y_3(t_i) - y_{3,i}\rvert$	$\max_{i=1,\ldots,N}\lvert y_4(t_i) - y_{4,i}\rvert$	Rate
2	9.7148e-002	3.3529e-002	2.5682e-002	2.4677e-002	
4	2.5688e-002	7.1313e-003	2.5625e-003	7.5865e-003	1.9191
8	6.4471e-003	1.6160e-003	2.7768e-004	2.0514e-003	1.9944
16	1.6035e-003	3.8510e-004	7.4759e-005	5.3083e-004	2.0075
32	3.9901e-004	9.3816e-005	2.8635e-005	1.3486e-004	2.0067
64	9.9467e-005	2.3140e-005	8.4156e-006	3.3980e-005	2.0041
128	2.4828e-005	5.7465e-006	2.2618e-006	8.5276e-006	2.0023

Table 2.7: Errors in the numerical solution obtained by applying the Improved Method to Example 2.7

.

2.2.6 Runge-Kutta 3

The general form for an explicit third order Runge-Kutta method is

$$\begin{aligned} k_1 &= f(t_i, y_i), \\ k_2 &= f(t_i + \alpha_2 h, y_i + \beta_{21} h k_1), \\ k_3 &= f(t_i + \alpha_3 h, y_i + \beta_{31} h k_1 + \beta_{32} h k_2), \\ y_{i+1} &= y_i + h(A_1 k_1 + A_2 k_2 + A_3 k_3). \end{aligned}$$

The tableau is shown in Table 2.8.

0			
α_2	β_{21}		
α_3	β_{31}	β_{32}	
	A_1	A_2	A_3

Table 2.8: Butcher tableau for general RK3.

The scheme's coefficients are chosen so that the method has consistency and third order convergence. The conditions are

$$\begin{aligned} A_1 + A_2 + A_3 &= 1, \\ A_2\alpha_2 + A_3\alpha_3 &= \frac{1}{2}, \\ A_2\alpha_2^2 + A_3\alpha_3^2 &= \frac{1}{3}, \\ A_3\beta_{32}\alpha_2 &= \frac{1}{6}. \end{aligned}$$

The parameters for four well-known third order methods are shown in Table 2.9 (see, e.g. [7, Section 12.2]). They are the Nystrom, Nearly Optimal, Classical and Heun Methods:

(a) Nystrom

0			
2/3	2/3		
2/3	0	2/3	
	2/8	3/8	3/8

(b) Nearly optimal

0			
1/2	1/2		
3/4	0	3/4	
	2/9	3/9	4/9

(c) Classical

0			
1/2	1/2		
1	−1	2	
	1/6	4/6	1/6

(d) Heun

0			
1/3	1/3		
2/3	0	2/3	
	1/4	0	3/4

Table 2.9: Four Runge-Kutta 3 methods.

Now we return to Example 2.5 again and apply the Heun and Classical Methods. The errors are shown in Table 2.10. We note that they are converging at a rate that is proportional to N^{-3}.

N	Heun		Classical	
	ε_N	p_N	ε_N	p_N
2	1.921e-002		4.477e-003	
4	2.936e-003	2.7098	8.422e-004	2.4105
8	4.027e-004	2.8664	1.293e-004	2.7037
16	5.258e-005	2.9371	1.793e-005	2.8501
32	6.712e-006	2.9696	2.361e-006	2.9246
64	8.477e-007	2.9851	3.030e-007	2.9622
128	1.065e-007	2.9926	3.837e-008	2.9811

Table 2.10: Errors in the numerical solution to Example 2.5 using the Heun and Classical Methods.

2.2.7 Runge-Kutta 4

The general form of explicit Runge-Kutta fourth order method is shown in the Butcher Tableau in Table 2.11:

0				
α_2	β_{21}			
α_3	β_{31}	β_{32}		
α_4	β_{41}	β_{42}	β_{43}	
	A_1	A_2	A_3	A_4

Table 2.11: General RK4 Method.

Once again, in order to guarantee the consistency and convergence of these method, the conditions on the coefficients are

$$\begin{aligned}
A_1 + A_2 + A_3 + A_4 &= 1,\\
A_2\alpha_2 + A_3\alpha_3 + A_4\alpha_4 &= \frac{1}{2},\\
A_2\alpha_2^2 + A_3\alpha_3^2 + A_4\alpha_4^2 &= \frac{1}{3},\\
A_3\beta_{32}\alpha_2 + A_4\beta_{42}\alpha_2 + A_4\beta_{43}\alpha_3 &= \frac{1}{6},\\
A_2\alpha_2^3 + A_3\alpha_3^3 + A_4\alpha_4^3 &= \frac{1}{4},\\
A_3\alpha_3\beta_{32}\alpha_2 + A_4\alpha_4\beta_{42}\alpha_2 + A_4\alpha_4\beta_{43}\alpha_3 &= \frac{1}{8},\\
A_3\beta_{32}\alpha_2^2 + A_4\beta_{42}\alpha_2^2 + A_4\beta_{43}\alpha_3^2 &= \frac{1}{12},\\
A_4\beta_{43}\beta_{32}\alpha_2 &= \frac{1}{24}.
\end{aligned}$$

The road to establish these conditions can be found in many standard textbooks (e.g. [3, §235]).

The most commonly used Runge-Kutta method is the fourth-order method, which uses a linear combination of four function evaluations:

$$\begin{aligned} k_1 &= f(t_i, y_i), \\ k_2 &= f(t_i + \frac{h}{2}, y_i + \frac{h}{2}k_1), \\ k_3 &= f(t_i + \frac{h}{2}, y_i + \frac{h}{2}k_2), \\ k_4 &= f(t_i + h, y_i + hk_3). \end{aligned}$$

These four equations, together with the recursion equation

$$y_{i+1} = y_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

make up the method. Table 2.12 shows the Butcher tableau for this RK4 method.

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	2/6	2/6	1/6

Table 2.12: The classic RK4 Method.

Example 2.8. Now let us compare four methods for a single equation, with continuous data:

$$y(0) = 1, \quad y' = \frac{1}{2}\big(t - y(t)\big).$$

Let $E_N = \max_{i=0,\dots,N} |y(t_i) - y_i|$, $p_N = \log_2(E_N/E_{2N})$, the numerical result is shown in Table 2.13 and can be visualised in Figure 2.4 by using a log-log plot. As discussed, the error of p^{th}-order method is proportional to $h^p = N^{-p}$. Therefore, when we plot the log of the number of steps N versus the log of the corresponding error, we obtain the line with the slope $-p$ for various methods, as seen in Figure 2.4.

N	Euler		Modified		Heun		RK4	
	E_N	p_N	E_N	p_N	E_N	p_N	E_N	p_N
2	1.32×10^{-1}		1.14×10^{-2}		7.24×10^{-4}		3.65×10^{-5}	
4	6.11×10^{-2}	1.11	2.60×10^{-3}	2.14	8.18×10^{-5}	3.15	2.05×10^{-6}	4.15
8	2.94×10^{-2}	1.05	6.20×10^{-4}	2.07	9.73×10^{-6}	3.07	1.22×10^{-7}	4.07
16	1.45×10^{-2}	1.02	1.52×10^{-4}	2.03	1.17×10^{-6}	3.04	7.42×10^{-9}	4.04
32	7.17×10^{-3}	1.01	3.76×10^{-5}	2.01	1.46×10^{-7}	3.02	4.58×10^{-10}	4.02
64	3.57×10^{-3}	1.01	9.31×10^{-6}	2.01	1.82×10^{-8}	3.01	2.84×10^{-11}	4.01

Table 2.13: Errors in the computed solution obtained by applying classical methods to Example 2.8

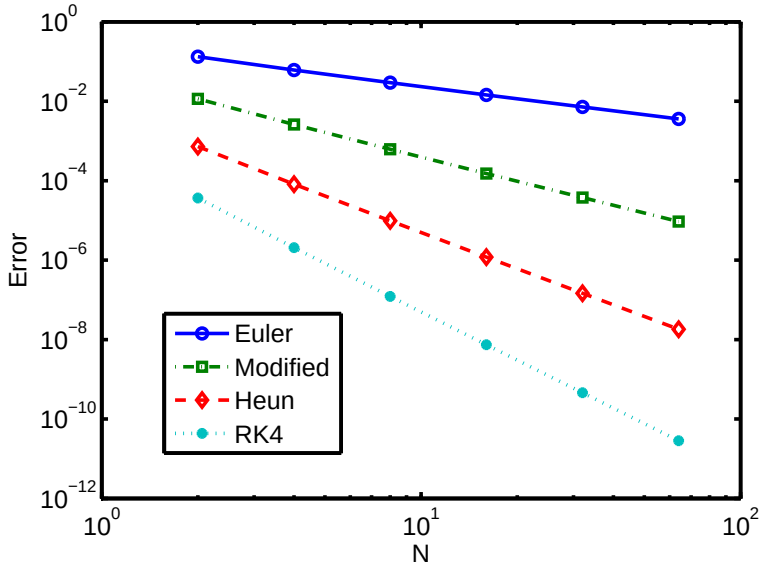


Figure 2.4: A log-log plot of the errors in numerical solution for various N.

2.2.8 Implementation of Runge-Kutta Methods in Matlab

Matlab is the most commonly used computer system for implementing numerical methods, see, e.g. [7, 11]. All of the numerical results, tables and figures presented in this thesis were generated by Matlab programmes that I wrote. As an example, I now give a short segment of compact Matlab code that implements a general explicit Runge-Kutta method, expressed as a Butcher Tableau, for a system of m equations.

Algorithm 2.1 Matlab Implementation of a general explicit Runge Kutta Method

```
for n = 1:N
    K=zeros(length(y0), length(A));
    for i=1:Order
        K(:, i) = f(t(n) + alpha(i)*h, Y(:, n) + h*K(:, 1:i-1)*beta(i, 1:i-1));
    end
    Y(:, n+1) = Y(:,n) + h*K*A';
end
```

Notation:

- The **alpha**, **beta**, and **A** are from the Butcher Tableau.
- N: number of steps.
- Order: the order of Runge-Kutta method. Note that: Order=length(**A**).

2.3 Problems with discontinuities

As we have seen, the classical numerical methods like Euler's and Runge-Kutta work very well for initial value problems (2.2) in which the function f is continuous and satisfies a Lipschitz condition. In this section, we investigate some issues that occur when f is not continuous and try to understand why this presents difficulties for certain classical numerical schemes. We also propose a non-uniform stepsize technique to overcome these difficulties.

2.3.1 Preliminary examples

Example 2.9. Consider the following initial value problem

$$y'(t) = \begin{cases} 2e^{2t}, & t \leqslant 1/2, \\ -2e, & t > 1/2, \end{cases}$$

on the interval $t \in (0, 1]$ and $y(0) = 1$.

The graph of $f = y'(t)$ on the interval $[0, 1]$ is shown in Figure 2.5 (left). We can see that it is discontinuous at the point $t = 1/2$. The graph on the right shows its exact solution:

$$y(t) = \begin{cases} e^{2t}, & t \leqslant 1/2 \\ -2et + 2e, & t > 1/2. \end{cases}$$

We see that it is only piecewise smooth.

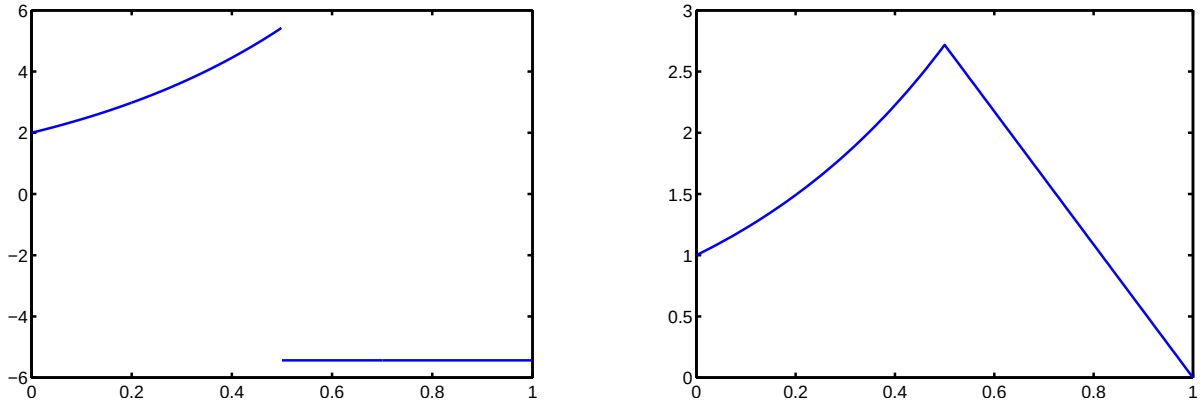


Figure 2.5: The graph of $y'(t)$ (left) and $y(t)$ (right) on $[0, 1]$.

We will see that, for this problem, the accuracy and rate of convergence of some numerical methods is less than that theoretically possible when f is smooth. If we can understand why this occurs, then we can choose which method is best. For example, we first employ the Improved Euler Method and the Classical RK3 methods which can be summarized in the tableaux below. We notice that for both these methods, one of the coefficients of the vector $\boldsymbol{\alpha}$ in the Butcher tableau is $\mathbf{1}$. It is highlighted in bold.

0		
1	1	
	1/2	1/2

0			
1/2	1/2		
1	−1	2	
	1/6	4/6	1/6

Using Improved and Classical RK3 methods, we obtain the numerical results in Table 2.14.

N	Improved		Classical	
	ε_N	p_N	ε_N	p_N
2	2.577e+000		9.055e-001	
4	1.323e+000	0.8666	4.530e-001	0.9992
8	6.706e-001	0.9363	2.265e-001	0.9999
16	3.375e-001	0.9690	1.133e-001	1.0000
32	1.693e-001	0.9847	5.663e-002	1.0000
64	8.481e-002	0.9924	2.832e-002	1.0000
128	4.244e-002	0.9962	1.416e-002	1.0000

Table 2.14: Errors and rate of convergence for Example 2.9 using Improved Euler and Classical Methods.

According to theory, the Improved Euler and Classical RK3 methods have the order of convergence $\mathfrak{p} = 2$ and $\mathfrak{p} = 3$ respectively. However, in Table 2.14 we see that they are both only 1^{st}-order.

Now consider the Modified and Heun methods, which are shown below. There we see that all the coefficients in vector $\boldsymbol{\alpha}$ are less than 1.

0		
1/2	1/2	
	0	1

0			
1/3	1/3		
2/3	0	2/3	
	1/4	0	3/4

Numerical results for these methods applied to Example 2.9 are displayed in Table 2.15. Note that the rates of convergence are $\mathfrak{p} = 2$ and $\mathfrak{p} = 3$ for the Modified Euler and Heun methods respectively. Figure 2.6 shows the rates of convergence for each of these four methods applied to Example 2.9. The Improved method and Classical RK3 have the same 1^{st}order of convergence, but the Modified and Heun methods can maintain their orders of convergence. The differences in results between these methods lies in having the parameter $\alpha < 1$. Since at step i of the one-step methods, the function f is evaluated at $t = t_i + \alpha h$, $\alpha_k < 1$ for $k = 1, 2, \ldots, p$ ensures that is it not evaluated at the point of discontinuity. Therefore, we prefer to use the numerical schemes such as the Modified and Heun methods for certain problems where the right-hand side is defined in a piecewise fashion.

N	Modified		Heun	
	ε_N	p_N	ε_N	p_N
2	6.956e-002		7.481e-003	
4	1.777e-002	1.9689	9.711e-004	2.9457
8	4.467e-003	1.9921	1.230e-004	2.9803
16	1.118e-003	1.9980	1.547e-005	2.9921
32	2.796e-004	1.9995	1.938e-006	2.9965
64	6.992e-005	1.9999	2.425e-007	2.9984
128	1.748e-005	2.0000	3.033e-008	2.9992

Table 2.15: Errors and rate of convergence for Example 2.9 using the Modified Euler and Heun Methods.

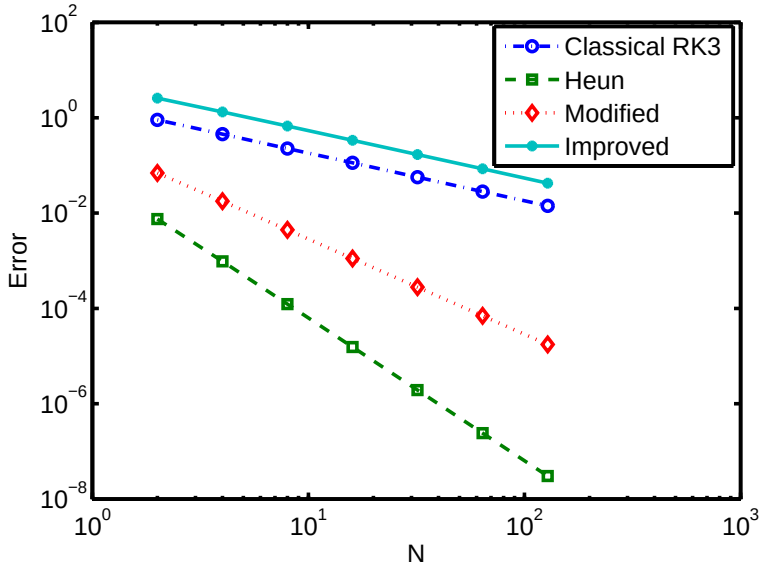


Figure 2.6: The rates of convergence in Example 2.9.

2.3.2 Recovering the rate of convergence

In order to cope with the reduction in the rates of convergence that appear in the Improved and Classical RK3 Methods as seen in Table 2.14, we introduce a simple nonuniform step technique. According to the theory, the Improved Method is 2^{nd}-order. By (2.11), we have

$$|T_i| \leq Ch^2 \sim CN^{-2}.$$

This due to the fact that the truncation error is $\mathcal{O}(h^2)$, which is deduced from Taylor's Theorem. However Taylor's Theorem assumes the function f must be differentiable at all points in the interval. Furthermore, the Corollary 2.1 that relates the truncation error to the global error uses that f is Lipschitz.

Suppose the discontinuity occurs at t_i and so the above assumptions are not satisfied. As we observed

in Table 2.14, the error for the Improved method is now just 1^{st}-order at t_i, i.e,

$$|\epsilon_i| \leq CN^{-1}.$$

Then the accumulated error of next step remains in 1^{st} order

$$|\epsilon_{i+1}| \leq |\epsilon_i| + CN^{-2} \sim CN^{-1}.$$

Therefore, to recover the rate of convergence, we can add an extra step with the length of $h = 1/N^2$ before the point where the discontinuity occurs.

Returning to Example 2.9, the discontinuity occurs at the point of $t = 1/2$. With uniformly spaced time steps, the Improved Euler's and Classical RK3 methods will evaluate f at that point, leading to a reduced rate of convergence at that point, which is accumulated to give the reduced rate of convergence at all subsequent points. As suggested by the above discussion, we add an extra step with the length of $h = 1/N^2$ for the Improved Method and $h = 1/N^3$ for the Classical RK3 respectively before $t = 1/2$. With this approach, we recover the maximum rate of accuracy successfully for Improved and Classical RK3 methods. The results are shown in Table 2.16.

N	Improved		Classical	
	ε_N	p_N	ε_N	p_N
3	1.323e+000		2.264e-001	
5	3.171e-001	2.0612	2.829e-002	3.0008
9	7.707e-002	2.0410	3.537e-003	2.9994
17	1.907e-002	2.0145	4.423e-004	2.9996
33	4.755e-003	2.0042	5.529e-005	2.9998
65	1.188e-003	2.0011	6.912e-006	2.9999
129	2.969e-004	2.0003	8.641e-007	2.9999

Table 2.16: Errors and rate of convergence applying the Improved and Classical Methods with the adjusted stepsize to Example 2.9.

Chapter 3

The Intensive Care Unit-Minimal Model

In this chapter we investigate the numerical solution of the well-known Intensive Care Unit-Minimal Model (ICU-MM) of Van Herpe [10] that models the glucose and insulin levels in critically ill patients. In practical cases, certain coefficients of the model are discontinuous. As noted above in Section 2.3, this can lead to a reduction of the rate of convergence of some one-step numerical methods. We use those observations to select the best scheme for this problem. Furthermore, in Section 3.2, we propose a simple time step selection algorithm for Runge-Kutta methods. The numerical results for the ICU-MM using this algorithm are also shown in Section 3.2.4. This will serve as an introduction to the more sophisticated time step controlling method that we will develop in Chapter 5.

3.1 The ICU-MM

We first introduce the Intensive Care Unit-Minimal Model (see, e.g. [5, 10]) which we studied in the first half of the project. We also apply the classical numerical one-step methods that we described in Section 1.1 to the ICU-MM model in Section 3.1.2.

3.1.1 A Mathematical Model

It is important to clinicians to be able to control efficiently the range of glycaemia (i.e., serum glucose levels) in critically ill patients in an intensive care unit (ICU). This is done by giving a glucose infusion if the level is too low, and insulin if the level is too high. However, responses to these therapies can vary greatly between patients, and getting the right balance of infusion levels is very difficult. As discussed in [5], some studies have noted that control of serum glucose may lead to improved outcomes for ICU

patients, while others have found that attempting to keep serum glucose levels within certain ranges may actually increases mortality rates. Also, as stated in [9],

> "*Two European multicenter studies [8] were stopped because of the high incidence of hypoglycemia as a consequence of the introduction of a protocol to calculate the appropriate insulin dose based on frequent glucose measurements*".

Therefore, the prediction of the individual response of a patient to glucose and insulin infusions is very valuable in a hospital setting. To aid with this, the *Minimal Model* for controlling glucose tolerance was proposed by Bergman et al. [1]. The model was further developed by Van Herpe et al. [20, 10] for the case of critically ill patients, leading to the ICU-MM. This model formed the basis of an approach using a *Dynamic Bayesian Network* [5]. The ICU-MM is presented as the following system of ordinary differential equations:

$$\begin{aligned}
\frac{dG(t)}{dt} &= \big(P_1 - X(t)\big)G(t) - P_1 G_b + \frac{F_G(t)}{V_G}, \\
\frac{dX(t)}{dt} &= P_2 X(t) + P_3(I_1(t) - I_b), \\
\frac{dI_1(t)}{dt} &= \alpha \max(0, I_2(t)) - n(I_1(t) - I_b) + \frac{F_I(t)}{V_I}, \\
\frac{dI_2(t)}{dt} &= \beta\gamma(G(t) - h) - nI_2(t),
\end{aligned} \tag{3.1}$$

where the coefficients and terms are briefly described in the Table 3.1 below. We refer readers to [5, 10] for a more detailed discussion.

G	The glucose concentration in blood plasma.
X	The effect of insulin on net glucose disappearance. X is proportional to the insulin in the remote compartment.
I_1	The insulin concentration in blood plasma.
I_2	The remote insulin.
G_b	The basal value of plasma glucose.
I_b	The basal value of plasma insulin.
F_I and F_G	The intravenous rate of insulin and glucose are the two input variables to the model.
V_G	The glucose distribution space.
V_I	The insulin distribution volume.
P_1	The glucose effectiveness when insulin remains at basal level.
P_2	The fractional rate of net remote insulin disappearance.
P_3	The fractional rate of insulin-dependent increase.
γ	The proportion by which endogenous insulin is released when the endogenous insulin is produced.
$\hbar$	The glucose threshold.
n	The time constant for insulin disappearance.
β	An additional model coefficient to keep units correct.
α	A scaling factor for the second insulin variable I_2.

Table 3.1: Description of parameters in the ICU-MM

For the system (3.1), the most interesting terms are F_G and F_I. They are the two input variables of the intravenous rate of glucose and insulin; since the amount of these substances that are being prescribed for a patient are reviewed every few hours, and changed instantaneously, the corresponding coefficients in the differential equations are discontinuous. Furthermore, the glucose values G are observed and updated every few hours. Table 3.2 shows the glucose record of sample Patient 23 mentioned in Section 1.3 over an interval of 100 hours. These values are also graphed in Figure 3.1. Furthermore, that figure shows the glucose and insulin infusion rates. Since these are taken to change instantaneously, the corresponding coefficients F_G and F_I in (3.1) are discontinuous.

Time(min)	**New Value**	**Time(min)**	**New Value**
0	172.8	2100	118.8
120	162.0	2820	135.0
300	147.6	3240	158.4
480	144.0	3480	140.4
660	140.4	3720	131.4
900	133.2	3840	140.4
1080	147.6	4020	111.6
1140	124.2	4200	156.6
1320	124.2	4320	151.2
1560	138.6	4680	135.0
1740	124.2	4980	185.4

Table 3.2: Observed Glucose Levels

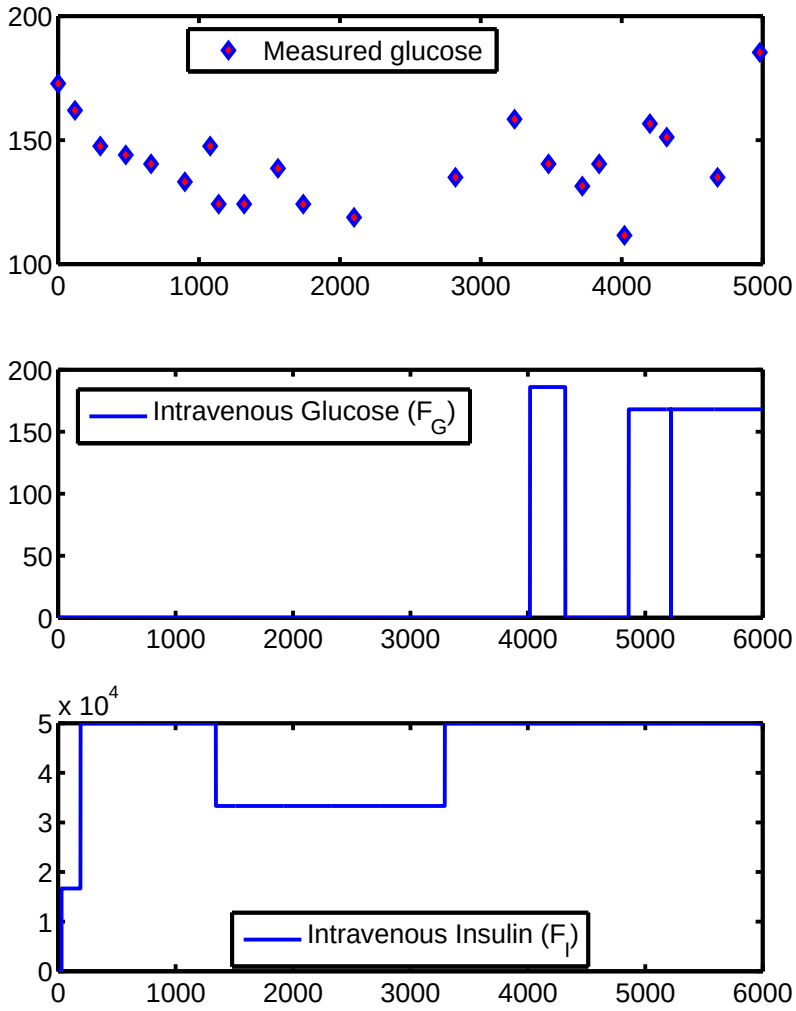


Figure 3.1: Observed glucose (top), the intravenous rate of glucose (middle) and the intravenous rate of insulin (bottom).

3.1.2 Numerical Solution of the ICU-MM

It is not possible find an exact solution to the ICU-MM (3.1), so we must use a numerical method such as Euler's method, Runge-Kutta 2 or Runge-Kutta 4. As we have seen in the previous section, Euler's, RK2 and RK4 methods have the order $p = 1, p = 2$ and $p = 4$ respectively. However, the specific data from Patient 23 are discontinuous, so as discussed in Section 2.3, some classical methods may not achieve their expected order of convergence. The Table 3.4 shows the numerical results and the rate of convergence for some of the classical methods described above in Section 2.2 where we have used RK4 with a very large number of steps to create a benchmark solution. We denote the maximum error at any given point by:

$$E_N = \max_{i=1,\ldots,N} |y(t_i) - y_i|,$$

where $y(t_i)$ is actually taken from the benchmark solution. The term N in the first column is the number of steps per minute. The Table 3.3 shows initial values and values of parameters that we use in this simulation. Figure 3.2 represents a sample benchmark solution by RK4 method.

Vars and coefficients	**Value**	**Units**
G	172.8	mg/dl
X	0.0001	1/min
I_1	9.5	μU/ml
I_2	1.49	μU/ml
G_b	135	mg/dl
I_b	10.7	μU/ml
F_I	0	μU/min
F_G	0	mg/min
V_G	120	dl
V_I	9000	ml
P_1	-0.0371	1/min
P_2	-0.0224	1/min
P_3	0.000025	ml/(min2μU)
γ	0.00014001	
$\hbar$	107.4	mg/dl
n	0.2623	1/min
β	1	1/min
α	0.35	1/min

Table 3.3: Initial values and the values of parameters in the ICU-MM

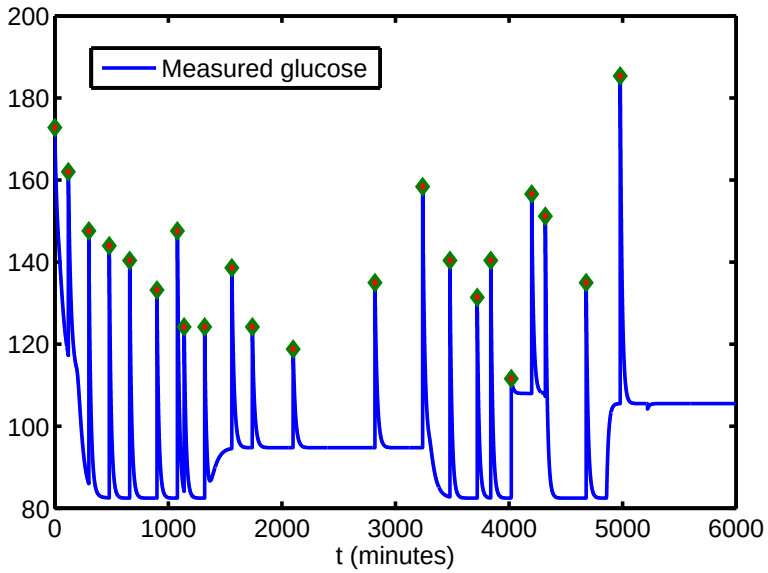


Figure 3.2: A sample benchmark solution of component G with updated values.

N	Euler		Improved		Modified		RK4	
	E_N	Rate	E_N	Rate	E_N	Rate	E_N	Rate
1	9.156×10^{-1}		6.89×10^{-1}		6.895×10^{-1}		6.895×10^{-1}	
2	4.519×10^{-1}	1.018	4.611×10^{-3}	7.224	4.619×10^{-3}	7.221	2.218×10^{-4}	11.602
4	2.245×10^{-1}	1.009	1.139×10^{-3}	2.017	1.143×10^{-3}	2.014	5.516×10^{-5}	2.007
8	1.119×10^{-1}	1.004	2.831×10^{-4}	2.008	2.839×10^{-4}	2.009	1.375×10^{-5}	2.004
16	5.586×10^{-2}	1.002	7.054×10^{-5}	2.004	7.090×10^{-5}	2.001	3.432×10^{-6}	2.002
32	2.791×10^{-2}	1.001	1.765×10^{-5}	1.998	1.766×10^{-5}	2.005	8.568×10^{-7}	2.002
64	1.395×10^{-2}	1.000	4.408×10^{-6}	2.002	4.411×10^{-6}	2.001	2.135×10^{-7}	2.004
128	6.973×10^{-3}	1.000	1.101×10^{-6}	2.001	1.103×10^{-6}	2.000	5.274×10^{-8}	2.017

Table 3.4: Errors in the numerical solution to the ICU-MM with the data for Patient 23.

The results seem to suggest that the RK4 method has a lower rate of convergence than for a problem with continuous coefficients. Once again, this is because of the discontinuities in the data leads to a reduction in the order of convergence of the numerical scheme. Therefore, we suggest the Modified Method is a suitable choice of 2^{nd}-order method for such types of problems.

3.2 Adaptive Uniform Stepsize Methods for Solving ODEs

3.2.1 Introduction

Consider an initial value problem:

$$y'(t) = f(t, y(t)), \text{ for } t \in (0, T] \text{ and } y(0) = y_0. \tag{3.2}$$

In practice we usually need to find an approximate solution with a given accuracy. Therefore, the problem of choosing an adequate stepsize h arises. A classical way to increase the accuracy is to reduce the stepsize by a factor of two after every step until the error is less than some desired bound. In the following section, we develop a simple uniform time step selection algorithm that computes the adequate stepsize so that the one-step method has the error that satisfies a given tolerance.

3.2.2 Uniform stepsize selection algorithm

We will describe this technique for Euler's method. It is easily extended to Runge-Kutta and other methods. Suppose we have an initial value problem of the form (3.2), and we are interested in approximating y at the point $t = T$. As mentioned in Section 2.2.1, using Taylor's expansion the local error (2.6) is:

$$y(t_k + h) - y_{k+1} = Ch^2.$$

In order to control this error, we need to choose a sufficiently large number of steps so that resulting y_n approximates $y(T)$ with a desired accuracy. In other words, we need to devise a sequence of stepsizes $\{h^{(i)}\}$ and compute the corresponding value of $y_{n_i}^{(i)}$ by using the relations $y_0^{(i)} = y_0$ and

$$y_{k+1}^{(i)} = y_k^{(i)} + h^{(i)} f(t_k^{(i)}, y_k^{(i)}). \tag{3.3}$$

Here we are using n_i to denote the number of time steps when $h = h^{(i)}$, and so $y_{n_i}^{(i)}$ is the approximation for $y(T)$. Let us define the global error of each step:

$$\epsilon_k^{(i)} = |y(t_k) - y_k^{(i)}| = Ch^2.$$

Then we are looking for $|\epsilon_{n_i}^{(i)}| < \epsilon$ to hold where ϵ is a user-chosen tolerance. Let us assume that the error factor C is constant for all h; this is not strictly true, but for h small enough is reasonable. Then, the accumulated error when calculating $y_{n_i}^{(i)}$ is:

$$\epsilon_{n_i}^{(i)} = n_i C(h^{(i)})^2 = \frac{T}{h^{(i)}} C(h^{(i)})^2 = TCh^{(i)} = Kh^{(i)},$$

where K is also a constant. Consider two iterations with stepsize $h^{(1)}$ and $h^{(2)}$ where $h^{(1)}$ can be chosen arbitrarily and $h^{(2)} < h^{(1)}$, typically we can choose $h^{(2)} = h^{(1)}/2$. We have

$$y(t) - y_{n_1}^{(1)} = \epsilon_{n_1}^{(1)} = Kh^{(1)},$$

$$y(t) - y_{n_2}^{(2)} = \epsilon_{n_2}^{(2)} = Kh^{(2)}.$$

Subtracting these equations yields

$$y_{n_2}^{(2)} - y_{n_1}^{(1)} = K(h^{(1)} - h^{(2)}).$$

Therefore,

$$K = \frac{(y_{n_2}^{(2)} - y_{n_1}^{(1)})}{(h^{(1)} - h^{(2)})}.$$

Thus, the condition for the next stepsize $h^{(3)}$ to be within the desired range is

$$\epsilon > |\epsilon_{n_3}^{(3)}| = |Kh^{(3)}| = \frac{h^{(3)}|y_{n_2}^{(2)} - y_{n_1}^{(1)}|}{|h^{(1)} - h^{(2)}|}$$

$$\Rightarrow \frac{\epsilon|h^{(1)} - h^{(2)}|}{|y_{n_2}^{(2)} - y_{n_1}^{(1)}|} > h^{(3)}.$$

Therefore,

$$h^{(3)} = \mathsf{q}\epsilon\frac{|h^{(1)} - h^{(2)}|}{|y_{n_2}^{(2)} - y_{n_1}^{(1)}|}. \tag{3.4}$$

for some coefficient $\mathsf{q} < 1$, that is chosen by the user. A typical value is $\mathsf{q} = 0.9$. We can sumarise the algorithm as follows:

Algorithm 3.1 The uniform stepsize selection algorithm

Step 1 Choose an initial stepsize $h^{(1)}$, compute the approximate solution $y_{n_1}^{(1)}$.

Step 2 Choose the stepsize $h^{(2)}$, typically $h^{(2)} = h^{(1)}/2$, compute the approximate solution $y_{n_2}^{(2)}$.

Step 3 Compute the stepsize $h^{(3)} = \mathsf{q}\epsilon\frac{|h^{(1)}-h^{(2)}|}{|y_{n_2}^{(2)}-y_{n_1}^{(1)}|}$, and compute $y_{n_3}^{(3)}$.

Note: Analogously, if the numerical method is convergent of order p, the formula at step 3 will be:

$$h^{(3)} = \mathsf{q}\left(\frac{|(h^{(1)})^p - (h^{(2)})^p|\epsilon}{|y_{n_2}^{(2)} - y_{n_1}^{(1)}|}\right)^{(1/p)}.$$

3.2.3 Comparison with Euler's method by halving the stepsize

A more standard method of choosing a suitable stepsize is to pick some initial stepsize $h^{(0)}$ and compute approximations with $h^{(i)} = h^{(0)}/2^i$, for $i = 1, 2, \ldots$ until the error in corresponding solution is less than the desired tolerance. We will compare the efficiency of this scheme and Algorithm 3.1 by applying both of them to the simplest initial value problem:

$$y'(x) = y; \quad y(0) = 1, \tag{3.5}$$

on the interval $(0, 1]$. The goal is to approximate the value $y(1)$ with an error less than $\epsilon = 10^{-3}$. The exact solution is $y(x) = e^x$. Then $y(1) \approx 2.7183$. We first find the value of N that ensures the error at $t = 1$ is less than $\epsilon = 10^{-3}$ simply by doubling N until the error is reached. Let us choose the initial number of steps as $N = 8$ and the values of $y_{n_i}^{(i)}$ for the sequence $\{h^{(i)}\}$ defined by

$$y_8^{(1)} \approx 2.6131, \qquad y_{128}^{(5)} \approx 2.7116,$$

$$y_{16}^{(2)} \approx 2.6651, \qquad y_{256}^{(6)} \approx 2.7149,$$

$$y_{32}^{(3)} \approx 2.6916, \qquad y_{512}^{(7)} \approx 2.7166,$$

$$y_{64}^{(4)} \approx 2.7049, \qquad y_{1024}^{(8)} \approx 2.7174,$$

with the error at $y_{1024}^{(8)} = 8.3887 \times 10^{-4}$. Therefore, 8 iterations are needed and the cost of computing the approximation is

$$8m + 16m + \cdots + 1024m = 8(2^8 - 1)m = 2040m, \tag{3.6}$$

where m is the computational cost of one step of (3.3), which is constant for our implementation. Now let us keep the initial number of steps with $N^{(1)} = 8, N^{(2)} = 16$, but here we use Algorithm 3.1 with the coefficient $\mathtt{q} = 0.95$. Then the first two approximations to $y(1)$ are the same:

$$y_8^{(1)} \approx 2.6131, \quad y_{16}^{(2)} \approx 2.6651.$$

Because the allowed error is $\epsilon = 10^{-3}$, we have

$$h^{(3)} = 0.95\frac{(0.125 - 0.0625) \times 0.001}{|2.6651 - 2.6131|} \approx 0.0011. \tag{3.7}$$

Then $N^{(3)} = \left\lceil \frac{1}{h^{(3)}} \right\rceil = 877$. As expected, we have the error 9.7945×10^{-4}. Thus, when our simple uniform stepsize selection method is used, the number of iterations needed is only 3 with the total cost of computation

$$8m + 16m + 877m = 901m. \tag{3.8}$$

which is around half of the cost of computing with doubling the number of steps.

3.2.4 Applying the algorithm to the ICU-MM

We now apply Algorithm 3.1 using Euler's method to the model (3.1) and the same values of parameters as discussed in Section 3.1. We choose two initial calculations with number of steps per minute $N = 1$ and $N = 2$, then $h^{(1)} = 1$ and $h^{(2)} = 0.5$. The corresponding approximate solutions are 134.82 and 134.37 respectively. Suppose we want the allowed error to be $\epsilon = 10^{-2}$, with $\mathtt{q} = 0.9$. Then we get

$$h^{(3)} = 0.9\frac{(1 - 0.5) \times 10^{-2}}{|134.37 - 134.82|} = 0.01.$$

Therefore, $N^{(3)} = \left\lceil \frac{1}{h^{(3)}} \right\rceil = 100$. The result obtained by this algorithm after 3 steps is summarised in Table 3.5. Comparing with the error of Euler's method in Table 3.4, the algorithm shows its efficiency. Indeed, with the given tolerance $\epsilon = 10^{-2}$, the algorithm helps us compute the step length in which the error is met after just three iterations. On the other hand, if we use the "stepsize halving" algorithm, we need to compute the solution for each of $N = 1, 2, \ldots, 128$.

N	Euler
1	9.156×10^{-1}
2	4.519×10^{-1}
100	9.802×10^{-3}

Table 3.5: Adaptive uniform algorithm for Patient 23 with a tolerance of $\epsilon = 10^{-2}$.

3.3 Conclusion

We have discussed the numerical solution of the Intensive Care Unit-Minimal Model by using classical numerical methods for IVPs. For this model, the difficulties arising from real measured data are also

addressed. This causes a reduction of order of convergence of several numerical schemes. For such problems with discontinuous coefficients, we suggest which methods are the best choices. We also proposed a uniform stepsize selection technique to pre-estimate the stepsize for which the computed error is satisfied.

As stated in [4]:

> In a typical busy ICU, a patient's plasma glucose may only be measured every 1-4 hours. Since these sparse measurements are used as evidence in the DBN, there seems little point in running inference on a DBN with steps of length 1 minute or less.

This fact motivates the need to develop another algorithm to cope with this issue; this we do in later chapters.

Chapter 4

Numerical Solution of Stiff Differential Equations

The property of "stiffness" is a very common phenomenon associated with many differential equations. A wide range of such problems comes from the study of vibrations, chemical reactions, electrical circuits and biological processes (see, e.g. [6, Section 5.11]). Of particular interest for this study is a system of differential equations for modelling the glucose and insulin levels in critically ill patients that has been proposed by colleagues in Applied Mathematics; the overarching goal of my Masters project has been to design a suitable numerical scheme for this model.

As discussed earlier, severely ill patients in an Intensive Care Unit (ICU) may stay there for several days. Controlling the interaction between blood glucose and insulin during that period of time is very significant, and so accurate mathematical models are useful. The ICU-MM model of [20], which we introduced in Chapter 3 above, is one approach. However, it often fails to capture important system dynamics. To address this, our colleagues from Applied Mathematics, Petri Piiroinen and Liam O'Callaghan, have proposed a new model, and have fitted the model parameters to observed data for a particular patient over the period of several days.

This new model seems to be far more responsive than the simpler ICU-MM. However, this improved modelling comes at a cost: it presents a much more difficult problem to solve numerically. In particular, the model is quite stiff. As we discuss below, the standard explicit methods we mentioned in Section 2.2 tend to generate wildly inaccurate computed solutions for such problems. In this chapter, we briefly discuss the stiffness phenomenon in differential equations in Section 4.1 and also investigate the application of *implicit schemes* in Section 4.3. These schemes seem to be a suitable choice for stiff differential equations. However, they require that a nonlinear system be solved at each time-step; we show how this can be done for the given problem in Section 4.2. Finally, the chapter culminates with the robust solution of the complex model (4.8) using an implicit scheme.

4.1 Stiff Differential Equations

The numerous classes of initial value problems for which the error can grow very large when using explicit methods, like Euler's and the explicit Runge-Kutta methods, are called *stiff equations* [6, Section 5.11]. As we have seen in Section 2.2.3, one-step methods for approximating solutions to initial value problems lead to error terms that involve higher derivatives of the solution. The stiffness phenomenon can occur when the derivative magnitude increases sharply but the solution does not. As noted by Iserles [12, p. 56]

> Several attempts at a rigorous definition of stiffness appear in the literature, but it is perhaps more informative to adopt an operative designation. Thus, we say that an ODE system
>
> $$\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y}), \quad t \geq t_0, \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0,$$
>
> is *stiff* if its numerical solution by some methods requires (perhaps in a portion of the solution interval) a significant depression of the stepsize to avoid instability.

This means that for most stiff problems, the generated solution can be very unstable unless an extremely small stepsize is used. We will consider a simple example in order to understand the behaviour of the stiff problem.

Example 4.1. The initial value problem:

$$y' = -20y, \quad y(0) = 1, \tag{4.1}$$

has the exact solution $y = e^{-20t}$. Therefore, the approximate solution must tend to zero in the long term. The following analysis is based on [6, Chap. 5]. Let $h = T/N$, where T is the final time for the simulation, and $t_i = ih$, $i = 0, 1, \ldots, N$. Applying Euler's method to this problem, we have $y_0 = 1$, and,

$$y_{i+1} = y_i + h(-20y_i) = (1 - 20h)y_i = (1 - 20h)(1 - 20h)y_{i-1} = \ldots$$

so

$$y_{i+1} = (1 - 20h)^{i+1}, \quad \text{for } i = 0, 1, \ldots, N - 1.$$

We will now see the issue with applying Euler's method to this problem. Because the exact solution is $y = e^{-20t}$, the error at the i^{th} step is

$$|y(t_i) - y_i| = |e^{-20ih} - (1 - 20h)^i|.$$

But as i increases, the exact solution $y = e^{-20t}$ tends to zero, however, the approximate solution $y_i = (1 - 20h)^i$ will have this property only if $|(1 - 20h)| < 1$. That is, the method will yield a reasonable solution only if $h < 1/10$.

More generally, suppose we have the initial value problem

$$y' = \lambda y, \quad y(0) = y_0, \quad \text{and } \lambda < 0. \tag{4.2}$$

Using an analogous argument, the condition for the acceptable stepsize is $|1 + \lambda h| < 1$, which implies that $h < 2/\lambda$.

For the initial value problem (4.2), it is not only Euler's method that may yield an unstable solution, but also general one-step methods. Following the approach of [6, §5.11], we express a general method as a function of h and λ:

$$y_{i+1} = Q(h\lambda)y_i,$$

and note that the accuracy of the scheme is determined by how well $Q(h\lambda)$ approximates $e^{h\lambda}$. Since $\lambda < 0$, $e^{h\lambda}$ tends to zero, and the approximate solution has this property when $|Q(h\lambda)| < 1$. This leads to the following definition (see [6, Definition 5.25]):

Definition 4.1. The *region of absolute stability* of a one-step method is

$$R = \{h\lambda \in \mathbb{C} \mid |Q(h\lambda)| < 1\}.$$

Definition 4.2. A numerical method is said to be *A-stable* if its region of absolute stability contains the entire left half plane; that is it is stable for all h. If it is stable only when for $h \leq h_{\min}$, for some $h_{\min}$ that depends on the method and the problem data, then we say it is *conditionally stable.*

We see that Euler's method above is not A-stable—it is only conditionally stable. Later in this chapter, we will show a method that is A-stable for the problem (4.2). Before coming up with that appropriate method for stiff problems, we consider their numerical solution in the following examples.

4.1.1 Examples

Example 4.2. Consider a system of initial value problems, see [6, page 335]:

$$\begin{aligned}
\frac{dy_1}{dt} &= 9y_1 + 24y_2 + 5\cos(t) - \frac{1}{3}\sin(t), \quad y_1(0) = \frac{4}{3}, \\
\frac{dy_2}{dt} &= -24y_1 - 51y_2 - 9\cos(t) + \frac{1}{3}\sin(t), \quad y_2(0) = \frac{2}{3},
\end{aligned}$$

which has the unique solution

$$\begin{aligned}
y_1(t) &= 2\exp(-3t) - \exp(-39t) + \frac{1}{3}\cos(t), \\
y_2(t) &= -\exp(-3t) + 2\exp(-39t) - \frac{1}{3}\cos(t).
\end{aligned}$$

Here, the transient term e^{-39t} in the solution causes this system to be stiff. Table 4.1 and Figure 4.1 show the errors when Euler's method is used with the stepsize $h = 0.1$. We can see the maximum error in each component is greater than 4.2077×10^4. The huge oscillations also appear in Figure 4.1.

t	$\lvert y_1(t_i) - y_{1,i}\rvert$	$\lvert y_2(t_i) - y_{2,i}\rvert$
0.1	0.0003×10^4	0.0005×10^4
0.2	0.0009×10^4	0.0017×10^4
0.3	0.0024×10^4	0.0049×10^4
0.4	0.0071×10^4	0.0142×10^4
0.5	0.0205×10^4	0.0410×10^4
0.6	0.0595×10^4	0.1190×10^4
0.7	0.1725×10^4	0.3450×10^4
0.8	0.5003×10^4	1.0006×10^4
0.9	1.4509×10^4	2.9018×10^4
1.0	4.2077×10^4	8.4153×10^4

Table 4.1: Errors in the numerical solution obtained by applying Euler's method to Example 4.2.

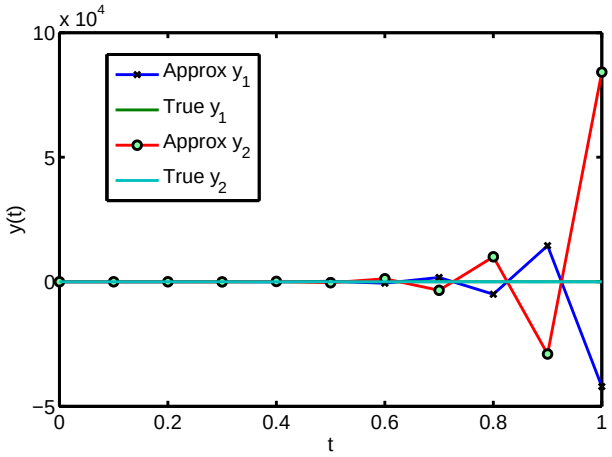


Figure 4.1: The graph of Example 4.2 using Euler's method.

As we have seen above, the explicit Euler method is not sufficient for such problems. A family of stable methods to deal with the property of stiffness is the so-called "implicit" methods which we mentioned in Remark 2.2. However, implicit methods lead to nonlinear systems of equations that must be solved. Therefore, we first introduce some iterative methods for solving such problems.

4.2 Nonlinear Functions of Several Variables

As mentioned in [7, Chap. 1], there is a long history of finding the numerical solutions to nonlinear problems, starting with the Babylonians over 3500 years ago. The fundamental methods like Fixed-Point, Newton's and Secant methods are carefully discussed in many standard textbooks on Numerical Analysis (see, e.g. [17, 7]). These methods can be extended to nonlinear problems in several variables.

We here discuss Newton's, Secant and Broyden's methods which we use when applying an implicit scheme to stiff problems.

In general, the system of n nonlinear equations in n unknowns has the form

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0, \\ f_2(x_1, x_2, \ldots, x_n) &= 0, \\ \vdots \qquad & \qquad \vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0, \end{aligned} \tag{4.3}$$

where each function f_i can be considered as mapping a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ in the n-dimensional space $\mathbb{R}^n$ onto the real line $\mathbb{R}$. By defining a function $\boldsymbol{F}$ mapping $\mathbb{R}^n$ into $\mathbb{R}^n$

$$\boldsymbol{F}(x_1, x_2, \ldots, x_n) = \big(f_1(x_1, x_2, \ldots, x_n), f_2(x_1, x_2, \ldots, x_n), \ldots, f_n(x_1, x_2, \ldots, x_n)\big)^T,$$

and using vector notation, the system (4.3) can be rewritten more compactly

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}. \tag{4.4}$$

The functions f_1, f_2, …, f_n are called the *coordinate functions* of $\boldsymbol{F}$.

The matrix

$$J(\boldsymbol{x}) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1}(\boldsymbol{x}) & \dfrac{\partial f_1}{\partial x_2}(\boldsymbol{x}) & \ldots & \dfrac{\partial f_1}{\partial x_n}(\boldsymbol{x}) \\ \dfrac{\partial f_2}{\partial x_1}(\boldsymbol{x}) & \dfrac{\partial f_2}{\partial x_2}(\boldsymbol{x}) & \ldots & \dfrac{\partial f_2}{\partial x_n}(\boldsymbol{x}) \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial f_n}{\partial x_1}(\boldsymbol{x}) & \dfrac{\partial f_n}{\partial x_2}(\boldsymbol{x}) & \ldots & \dfrac{\partial f_n}{\partial x_n}(\boldsymbol{x}) \end{pmatrix},$$

is called the *Jacobian* of $\boldsymbol{F}$ at $\boldsymbol{x}$. In the iterative methods we will discuss below, the computation of Jacobian is required.

4.2.1 Iterative methods for solving systems of nonlinear equations

We first recall Newton's method. With a scalar problem, suppose we want to solve $f(x) = x$ where $f : \mathbb{R} \to \mathbb{R}$. The idea of Newton's method is to choose an initial guess $x^{(0)}$ and then compute the sequence

$$x^{(k+1)} = x^{(k)} - f(x^{(k)})\lambda(x^{(k)}),$$

where we choose λ so that $x^{(k+1)}$ is the point where the line through $(x^{(k)}, f(x^k))$ with slope $f'(x^{(k)})$ cuts the x-axis. Therefore, the formula for the iteration is

$$x^{(k+1)} = x^{(k)} - f(x^{(k)})/f'(x^{(k)}).$$

This is a form of relaxation method:

$$x^{(k+1)} = g(x^{(k)}) = x^{(k)} - \phi(x)f(x^{(k)}).$$

By choosing $\phi(x) = 1/f'(x)$, we get the Newton's method.

Using a similar approach in the n-dimensional case, the function $\boldsymbol{G}$ can be defined by

$$\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{x} - J^{-1}(\boldsymbol{x})\boldsymbol{F}(\boldsymbol{x}),$$

where $J(\boldsymbol{x})$ is the Jacobian of $\boldsymbol{F}$, and the functional iteration procedure evolves from selecting $\boldsymbol{x}^{(0)}$ and generating, for $k = 1, 2, \ldots$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{G}(\boldsymbol{x}^{(k)}) = \boldsymbol{x}^{(k)} - J^{-1}(\boldsymbol{x}^{(k)})\boldsymbol{F}(\boldsymbol{x}^{(k)}).$$

The following theorem tells us about the convergence of Newton's method [6, Chap. 10].

Theorem 4.1. *Let $\boldsymbol{p}$ be a solution of $\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{x}$. Suppose a number $\delta > 0$ exists such that*

1. $\dfrac{\partial g_i}{\partial x_j}(\boldsymbol{x})$ *is continuous on* $N_\delta = \{\boldsymbol{x} | \|\boldsymbol{x} - \boldsymbol{p}\|_\infty < \delta\}$ *for each* $i = 1, 2, \ldots, n$ *and* $j = 1, 2, \ldots, n$;
2. $\dfrac{\partial^2 g_i}{\partial x_j \partial x_k}(\boldsymbol{x})$ *is continuous and* $\left|\dfrac{\partial^2 g_i}{\partial x_j \partial x_k}(\boldsymbol{x})\right| \le M$ *for some constant* M, *whenever* $\boldsymbol{x} \in N_\delta$, *for each* $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, n$ *and* $k = 1, 2, \ldots, n$;
3. $\dfrac{\partial g_i}{\partial x_k}(\boldsymbol{p}) = 0$ *for each* $i = 1, 2, \ldots, n$ *and* $k = 1, 2, \ldots, n$.

Then there exists a number $\widehat{\delta} \le \delta$ such that the sequence generated by $\boldsymbol{x}^{(k)} = \boldsymbol{G}(\boldsymbol{x}^{(k-1)})$ converges quadratically to $\boldsymbol{p}$ for any choice of $\boldsymbol{x}^{(0)}$, provided that $\|\boldsymbol{x}^{(0)} - \boldsymbol{p}\| < \widehat{\delta}$. Moreover,

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{p}\|_\infty \le \frac{n^2 M}{2}\|\boldsymbol{x}^{(k-1)} - \boldsymbol{p}\|_\infty^2,$$

for each $k \ge 1$.

In practice, it is very expensive to compute $J^{-1}(\boldsymbol{x})$. To avoid this, we could rewrite the scheme as

$$J(\boldsymbol{x}^{(k)})\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}^{(k)}), \tag{4.5}$$

and,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \boldsymbol{y}.$$

So, (4.5) is a linear system that can be solved by Gaussian elimination.

We next investigate the numerical results obtained by applying this method to the following example.

Example 4.3. Consider the nonlinear system

$$6x_1^3 + x_1 x_2 - 3x_2^3 - 4 = 0$$
$$x_1^2 - 18x_1 x_2^2 + 16x_2^3 + 1 = 0.$$

It has a solution at $\widetilde{\boldsymbol{x}} = (1, 1)^T$. We will use Newton's method to obtain an approximation solution when initial guess is $\boldsymbol{x}^{(0)} = (2, 2)^T$ and

$$\boldsymbol{F}(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))^T,$$

where

$$f_1(x_1, x_2) = 6x_1^3 + x_1x_2 - 3x_2^3 - 4,$$
$$f_2(x_1, x_2) = x_1^2 - 18x_1x_2^2 + 16x_2^3 + 1.$$

The Jacobian matrix for the system is

$$J(x_1, x_2) = \begin{pmatrix} 18x_1^2 + x_2 & x_1 - 9x_2^2 \\ 2x_1 - 18x_2^2 & -36x_1x_2 + 48x_2^2 \end{pmatrix},$$

and

$$\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{pmatrix} = \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \end{pmatrix} - \begin{pmatrix} y_1^{(k)} \\ y_2^{(k)} \end{pmatrix},$$

where $\boldsymbol{y}^{(k)} = (y_1^{(k)}, y_2^{(k)})^t$ is the solution of linear system $J(\boldsymbol{x}^{(k)})\boldsymbol{y}^{(k)} = \boldsymbol{F}(\boldsymbol{x}^{(k)})$. The results of applying this iterative procedure are shown in Table 4.2. As seen in Table 4.2, the approximate solution approaches the exact solution quickly after just 6 iterations. This is because, providing the initial guess is close enough to the true solution, Newton's method has *quadratic convergence*. That means the error at the i^{th} step is roughly the square of the error at the $(i-1)^{\text{th}}$ step.

k	$x_1^{(k)}$	$x_2^{(k)}$	$\|\boldsymbol{x}^{(k)} - \widetilde{\boldsymbol{x}}\|_\infty$
0	2	2	
1	1.3726	1.3403	5.046e-001
2	1.0784	1.0538	9.507e-002
3	1.0053	1.0027	5.989e-003
4	1.0000	1.0000	4.047e-005
5	1.0000	1.0000	1.260e-009
6	1.0000	1.0000	2.483e-016

Table 4.2: The convergence of Newton's method applied to Example 4.3.

Two surfaces $z = f_1(x_1, x_2) = 6x_1^3 + x_1x_2 - 3x_2^3 - 4$, and $z = f_2(x_1, x_2) = x_1^2 - 18x_1x_2^2 + 16x_2^3 + 1$, in the range $0 \le x_1, x_2 \le 2$ are shown in Figure 4.2. The intersection of two surfaces and the plane $z = 0$ shows us the solution of above system. In other words, the zero of this system is the intersection of two curves $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = 0$.

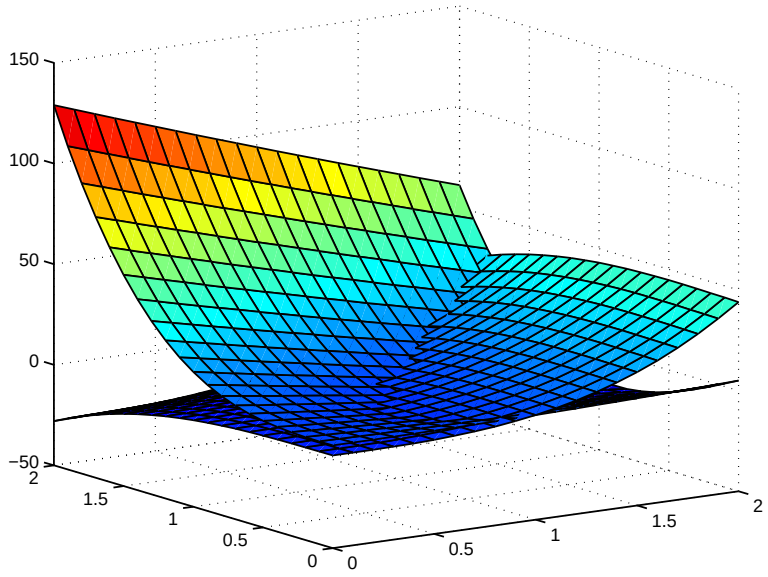


Figure 4.2: The surfaces $z = f_1(x_1, x_2)$ and $z = f_2(x_1, x_2)$.

Newton's method shows its promise. However, in most situations, especially for mathematical models, the exact evaluation of partial derivatives is inconvenient. As we know in the one dimensional case, we depart from Newton's method and replace the first derivative by the approximation

$$f'(x_1) \approx \frac{f(x_1) - f(x_0)}{x_1 - x_0},$$

to obtain the *Secant Method*. In the n-dimensional case, a similar approach is employed for partial derivative approximation

$$\frac{\partial \boldsymbol{F}}{\partial x_i}(x_1, x_2, \ldots, x_n) \approx \frac{1}{h}\big(\boldsymbol{F}(x_1, \ldots, x_i + h, \ldots, x_n) - \boldsymbol{F}(x_1, \ldots, x_n)\big).$$

We now apply the Secant method to Example 4.3 with two initial guesses $\boldsymbol{x}^{(0)} = (0.9, 0.9)^T$ and $\boldsymbol{x}^{(1)} = (0.8, 0.8)^T$. The numerical results are shown in Table 4.3. We can see that the Secant method is fast, but not as fast as Newton's since the ratio of two successive errors obtained by applying the Secant method is just the golden ratio $\alpha = (1 + \sqrt{5})/2 \approx 1.618$.

k	$x_1^{(k)}$	$x_2^{(k)}$	$\|\boldsymbol{x}^{(k)} - \widetilde{\boldsymbol{x}}\|_\infty$
0	0.9000	0.9000	1.414e-001
1	0.8000	0.8000	2.828e-001
2	0.9774	0.9100	9.279e-002
3	1.6287	2.7124	1.824e+000
4	2.2535	2.3055	1.810e+000
5	0.9870	1.2701	2.704e-001
6	1.0166	1.2134	2.141e-001
7	1.0369	1.1022	1.087e-001
8	1.0118	1.0397	4.144e-002
9	1.0037	1.0110	1.162e-002
10	1.0005	1.0015	1.577e-003
11	1.0000	1.0001	7.173e-005
12	1.0000	1.0000	4.808e-007
13	1.0000	1.0000	1.503e-010

Table 4.3: The convergence of the Secant method applied to Example 4.3.

Another modified form of Newton's method that does not require computation of the Jacobian is *Broyden's method* [7, §7.1.2]. The method is described as follows: for $k = 1, 2, \ldots$, take

$$\begin{aligned} x_{k+1} &= x_k - A_k^{-1} F(x_k), \\ s_k &= x_{k+1} - x_k, \\ y_k &= F(x_{k+1}) - F(x_k), \\ A_{k+1} &= A_k + \frac{(y_k - A_k s_k)(s_k)^T}{(s_k)^T s_k}. \end{aligned} \tag{4.6}$$

The initial matrix A_0 is usually taken to be either $J(x_0)$ (the Jacobian of F at x_0) or a finite difference approximate to it.

Returning to Example 4.3, but applying Broyden's method with initial guess $\boldsymbol{x}^{(0)} = (0.9, 0.9)^T$. The numerical results are shown in Table 4.4. Note that the approximate solution approaches the true solution quite quickly after just six steps with the error of 2.97×10^{-10}.

k	$x_1^{(k)}$	$x_2^{(k)}$	$\|\boldsymbol{x}^{(k)} - \widetilde{\boldsymbol{x}}\|_\infty$
0	0.9000	0.9000	1.414e-001
1	1.0090	1.0071	1.148e-002
2	0.9991	0.9997	9.397e-004
3	1.0000	0.9999	7.741e-005
4	1.0000	1.0000	1.628e-005
5	1.0000	1.0000	8.080e-008
6	1.0000	1.0000	2.970e-010

Table 4.4: The convergence of Broyden's method applied to Example 4.3.

4.3 The Implicit Euler Method

We now introduce a numerical scheme that is stable for stiff problems no matter how large the stepsize is. The Explicit Euler's method (2.5) is reframed as an *Implicit Euler method* if the right-hand side is evaluated at t_{i+1} instead of t_i:

$$\boldsymbol{y}_{i+1} = \boldsymbol{y}_i + h\boldsymbol{f}(t_{i+1}, \boldsymbol{y}_{i+1}). \tag{4.7}$$

But note that $\boldsymbol{y}_{i+1}$ appears on both sides (hence the name "implicit"). This scheme, expressed as a Butcher tableau, is in Table 4.5.

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

Table 4.5: Butcher tableau for the Implicit Euler method.

As mentioned, (4.7) leads to a nonlinear equation $\boldsymbol{F}(\boldsymbol{x}) = 0$ that must be solved. The nonlinear system of equations can be solved by iterative methods such as Newton's method introduced in Section 4.2.

To understand more about the implicit Euler method and how it works, we reconsider (4.1). The scheme now leads to

$$y_{i+1} = y_i + hf(t_{i+1}, y_{i+1}) = y_i - 20hy_{i+1}.$$

Hence

$$y_{i+1} = \frac{1}{1+20h} y_i = \frac{1}{(1+20h)^{i+1}}.$$

So now, for any $h > 0$, the term $\dfrac{1}{(1+20h)^{i+1}}$ is less than 1. Therefore, the approximate solution tends to the equilibrium solution 0. We next investigate the numerical solutions of some stiff problems. First, we begin with a simple single equation.

Example 4.4. Consider the following initial value problem consisting of a single equation:

$$y' = 10(1-y), \quad y(0) = 1/2,$$

on the interval [0,10]. The solution is easy to determine: $y(t) = 1 - e^{-10t}/2$. For $t > 1$, the solution has already reached its equilibrium 1 within 4 decimal places, and it never moves any farther away from 1. The differential equation is stiff due to the term e^{-10t}. Figure 4.3 shows the contrast between the explicit and implicit Euler methods with the number of steps $N = 32$. Note that with the explicit Euler method, the maximum error is greater than 10^{10}. We also see that there are many oscillations in the left of Figure 4.3.

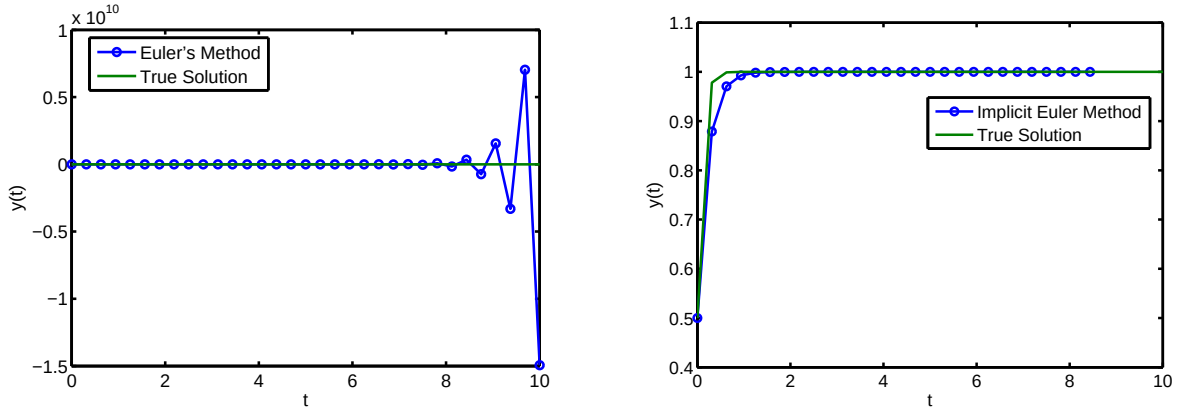


Figure 4.3: Example 4.4 using Explicit Euler Method (left) and Implicit Euler Method (right).

We will see the *efficiency* of the Implicit Euler Method by applying this scheme to Example 4.2 with the stepsize $h = 0.1$. The results are in Table 4.6 and Figure 4.4. We note that the result shown in Table 4.6 is far more accurate than shown in Table 4.1. Moreover, compared with Figure 4.1 there is no oscillation in the approximate solution as seen in Figure 4.4.

t	$\lvert y_1(t_i) - y_{1,i}\rvert$	$\lvert y_2(t_i) - y_{2,i}\rvert$
0.1	0.1280	0.3399
0.2	0.0429	0.0406
0.3	0.0866	0.0304
0.4	0.0937	0.0440
0.5	0.0894	0.0440
0.6	0.0809	0.0401
0.7	0.0710	0.0353
0.8	0.0609	0.0303
0.9	0.0514	0.0256
1.0	0.0429	0.0213

Table 4.6: Errors in the numerical solution generated by the Implicit Euler Method applied to Example 4.2.

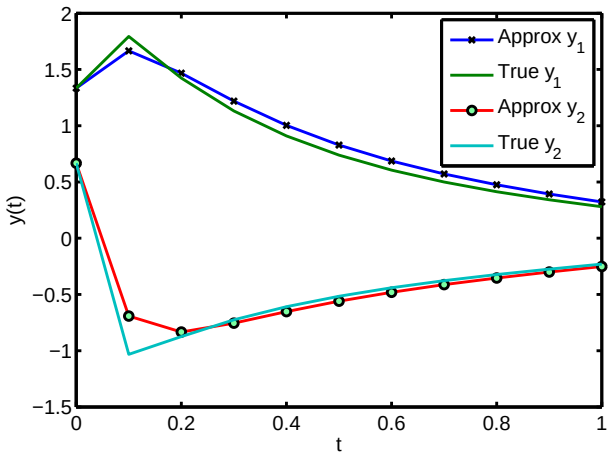


Figure 4.4: The graph of the computed solution to Example 4.2 using the Implicit Euler Method.

Returning to the Problem (4.2), we now show that the Implicit Euler Method is A-stable. Employing the Implicit Euler Method for $y' = \lambda y$ with $y(0) = y_0$ and $\lambda < 0$, we get

$$y_{i+1} = y_i + hy_{i+1},$$

then,

$$y_{i+1} = \frac{y_i}{1 - h\lambda} = \ldots = \frac{y_0}{(1 - h\lambda)^{i+1}},$$

Obviously, $\left|\frac{1}{1 - h\lambda}\right| < 1$ for all $h > 0$. Hence

$$Q(h\lambda) < 1, \forall h\lambda < 0,$$

so the Implicit Euler Method is A-stable.

4.4 A New Mathematical Model for Glucose and Insulin Levels

In a recent development, Liam O'Callaghan and Petri Piiroinen [14] have proposed an improved model based on the Intensive Care Unit-Minimal Model (3.1) to predict blood glucose (i.e., glycemia) levels of critically ill patients who are receiving glucose and insulin infusions.

$$\begin{aligned}
v_1\frac{dG_1}{dt} &= \frac{k_1\big(G_2(t) - G_1(t)\big)}{k_{m1} + G_1(t) + G_2(t)} - \frac{\big(k_2 + k_0 I_3(t)\big)G_1(t)}{k_{m0} + G_1(t)} + F_G(t),\\
v_2\frac{dG_2}{dt} &= \frac{k_1\big(G_1(t) - G_2(t)\big)}{k_{m1} + G_1(t) + G_2(t)} + \frac{k_3 gly}{1 + \exp\big(k_{31}(I_3(t) - b_1)\big)} - \frac{k_3 G_2(t)}{1 + \exp\big(k_{31}(b_2 - I_3(t))\big)},\\
v_3\frac{dI_1}{dt} &= k_5\big(I_{\max} - I_1(t)\big) - \frac{k_6 I_1(t)}{1 + \exp\big(k_{61}(c_1 - G_1(t))\big)},\\
v_4\frac{dI_2}{dt} &= \frac{k_6 I_1(t)}{1 + \exp\big(k_{61}(c_1 - G_1(t))\big)} - \frac{k_7 I_2(t)}{k_{m7} + I_2(t)} - k_8 I_2(t) + F_I(t),\\
v_5\frac{dI_3}{dt} &= \frac{k_7 I_2(t)}{k_{7m} + I_2(t)} - k_9 I_3(t),
\end{aligned} \tag{4.8}$$

where

- G_1 is plasma glucose concentration–this is the main output of interest from the model;
- G_2 is the concentration of glucose in hepatocytes (intracellular);
- I_1 is the concentration of insulin in pancreatic β-cells;
- I_2 is the plasma insulin concentration;
- I_3 is concentration of bound insulin;
- and the v_i are the volumes of the respective compartments;
- $F_G(t)$ and $F_I(t)$ are the instantaneous glucose and insulin infusion rates, respectively. These are the primary inputs for the model.

An explanation of other terms can be found in [14].

Remark 4.1. *The model* (4.8) *was proposed in December 2010 and formed the basis for the investigations in this chapter. In March 2011 a further modification of the model was given. We will use that more recent model in Chapter 5. However, both models present the same challenges for numerical solution, so we work with the original model here.*

The Figure 4.5 below shows the solutions of the five components for typical values of the parameters. The diagram also suggests that the system (4.8) may be stiff. For example, the second subgraph shows how quickly the solution G_2 changes over a short period of time.

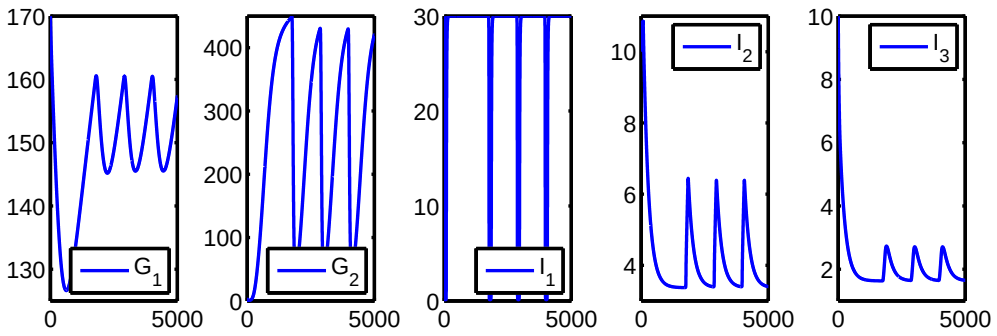


Figure 4.5: The solution of five components.

4.4.1 Numerical Solution using the Explicit Euler Method

Table 4.7 and Figure 4.6 show the results for Euler's method for (4.8) with the real data of Patient 102 [14]. For this calculation, we have used the built-in differential equation solver `ode45` in Matlab to create a benchmark solution. Because of the stiffness of the model, Euler's method needs a very large number of

steps, $N \approx 65536$, to compute a stable solution. When N is smaller, the computed solution is unstable, and has wild oscillations. Indeed, some values cannot be represented properly in Matlab and are returned as `NaN`, which stands for "Not a Number". We can visualize the Table 4.7 using a log-log plot. Note that all three components G_2, I_1 and I_2 return `NaN` with $N < 65536$; and are not plotted.

N	G_1	G_2	I_1	I_2	I_3
4096	NaN	NaN	NaN	NaN	4.65×10^{1}
8192	1.83×10^{1}	NaN	NaN	NaN	4.46×10^{1}
16384	1.02×10^{1}	NaN	NaN	NaN	4.22×10^{1}
32768	1.36×10^{1}	NaN	NaN	NaN	4.33×10^{1}
65536	1.41×10^{-1}	1.43×10^{1}	2.24×10^{1}	3.71×10^{-1}	2.57×10^{-2}
131072	7.06×10^{-2}	7.13×10^{0}	6.90×10^{0}	1.15×10^{-1}	1.28×10^{-2}
262144	3.53×10^{-2}	3.57×10^{0}	2.52×10^{0}	4.18×10^{-2}	6.43×10^{-3}
524288	1.76×10^{-2}	1.78×10^{0}	1.15×10^{0}	2.07×10^{-2}	3.21×10^{-3}
1048576	8.82×10^{-3}	8.92×10^{-1}	5.48×10^{-1}	1.04×10^{-2}	1.61×10^{-3}

Table 4.7: Errors in the numerical solution generated by the explicit Euler method to (4.8).

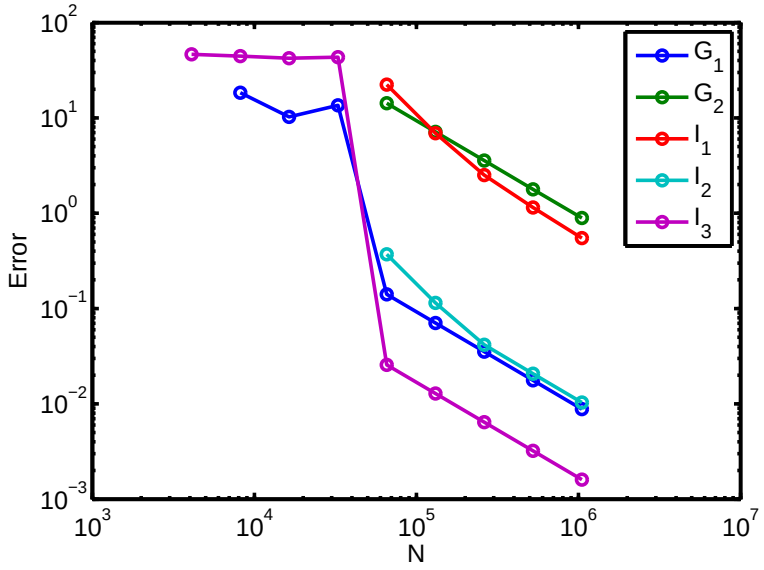


Figure 4.6: A log-log plot of the errors shown in Table 4.7.

4.4.2 Numerical Solution using the Implicit Euler Method

As we can see from previous examples, the implicit Euler method is adequate for stiff problems. In particular, when using the explicit Euler method, the model (4.8) needs a very large number of steps to achieve a meaningful result as we see in Table 4.7. Because errors in real world models are often dominated by errors in measured data, it can be more important that the numerical solution is stable, than being highly accurate. So it is very useful to have a scheme that can produce a reasonable solution, even

using large time steps. We now employ the implicit Euler method. In our experiments for this problem, Newton's Method for solving the system of nonlinear equations has proved successful, but expensive. We instead use Broyden's method (4.6). The numerical result for the glucose-insulin model (4.8) is shown in Table 4.8. We emphasize that this result is much more accurate than that shown in Table 4.7 when N is small. For this computation, the tolerance for nonlinear solver is set by 10^{-5}. Figure 4.7 shows that the method tends to be convergent even with a quite small number of steps $N = 4096$. Note that when N is large enough so that the explicit Euler method is stable, it is just as accurate as the implicit Euler method.

N	G_1	G_2	I_1	I_2	I_3
4096	2.26e+000	2.10e+002	2.97e+001	1.51e+000	4.10e-001
8192	1.13e+000	1.12e+002	2.80e+001	8.99e-001	2.07e-001
16384	5.66e-001	5.69e+001	2.15e+001	5.54e-001	1.03e-001
32768	2.81e-001	2.84e+001	1.27e+001	3.13e-001	5.13e-002
65536	1.44e-001	1.45e+001	6.85e+000	1.66e-001	2.62e-002

Table 4.8: Errors in the numerical solution obtained by applying the Implicit Euler Method to (4.8).

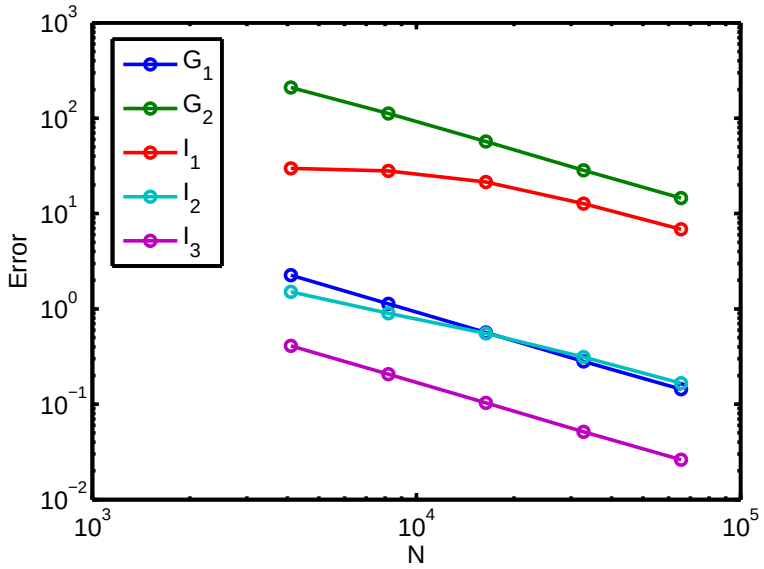


Figure 4.7: A log-log plot of the errors shown in Table 4.8.

4.5 Conclusion

The stiffness phenomenon is very common in ordinary differential equations, especially in the problems that come from real-world models. The explicit schemes discussed in Chapter 2 are not appropriate for such problems, but the implicit schemes studied in this chapter are. The numerical result for the new model proposed by colleagues where the stiffness is dominant is achieved by using the robust implicit

framework. Unfortunately, however, it is not feasible to introduce a nonlinear solver into the DBN. In the next chapter we will introduce a method that is feasible. However, to evaluate the accuracy of that method, it is important that we be able to generate a robust benchmark solution. The implicit scheme that we have described above—and more sophisticated Matlab implementations of high-order implicit schemes—are used to do this.

Chapter 5

An Adaptive Time Stepping Algorithm

We introduce an adaptive *non-uniform* time stepping algorithm in this chapter. The fundamental difference between this framework and the algorithm we presented in Section 3.2 is that it does not require a fixed time step. This technique allows the stepsize to vary flexibly to adapt to the dynamics of the system. Using it we can compute accurate solutions to our main model problem in a highly efficient manner, by using short time steps only in regions of the domain where the solution is highly variable. We describe the adaptive time-stepping algorithm and its user-chosen parameters in Section 5.1. Following that, we apply the technique to a variant of the well-known van der Pol oscillator, whose solution transitions between periods of rapid change and others of relative stability. We apply this algorithm to the van der Pol example since this equation has many of the same qualities as the glucose insulin model in Chapter 4, but is simpler. Also, it is widely used in the literature as an example of a model of physiological systems [2], and as a classical example of a stiff problem [3] . We show how to tailor the parameters in the algorithm to optimise the numerical result for this specific problem. Finally, in Section 5.3 we apply the scheme to the latest model developed for the glucose insulin problem and show that the results obtained are far more accurate than the equally spaced stepsize scheme. But first, we now describe the adaptive algorithm.

5.1 The Adaptive Time Stepping Algorithm

The key idea of a variable stepsize method is to prescribe a tolerance that the error at each step should not exceed, and to monitor the error to see if the tolerance has been met. If the tolerance is exceeded, then the algorithm repeats the step with a reduced stepsize. If the error tolerance is met, it accepts the step and chooses a new stepsize that should be appropriate for the next step. A good local error estimator is needed to drive the algorithm. For example, for Euler's method we can estimate the local

discretization error (2.6) at each step by using the fact that:

$$|\varepsilon_i| \leq \frac{h_i^2}{2} y''(\xi),$$

where $\xi \in [t_{i-1}, t_i]$. Using the backward difference formula, we get:

$$\begin{aligned} |\varepsilon_i| &\leq \frac{h_i^2}{2} \left| \frac{y'(t_i) - y'(t_{i-1})}{h_{i-1}} \right| \\ &= \frac{h_i^2}{2} \left| \frac{f(t_i, y_i) - f(t_{i-1}, y_{i-1})}{h_{i-1}} \right| = \frac{h_i^2}{2} \Delta. \end{aligned}$$

where $\Delta = \left| \dfrac{f(t_i, y_i) - f(t_{i-1}, y_{i-1})}{h_{i-1}} \right|$.

Let `Tol` denote the prescribed tolerance. Following [3, §202], we want to ensure that $|\varepsilon_i| \leq$ `Tol`, so if the tolerance is not met, the step is repeated with a new possible stepsize h_{new} which can be determined as follows:

$$h_{\text{new}} \approx \mathtt{q}\sqrt{\frac{2\mathtt{Tol}}{\Delta}},$$

where `q` $\in (0, 1)$ is set as a "safety factor" for the successful step. In order to maintain efficiency, if the estimated error is significantly less than `Tol`, the stepsize is increased with a scaling based on a user-chosen parameter for the next step.

5.1.1 Algorithm pseudo-code and user-chosen parameters

The adaptive algorithm is shown below. It features four user-specified parameters. They are

- The error tolerance, `Tol`. The algorithm attempts to ensure that the local discretization error (2.6) introduced at each step is no more than this value. The actual accumulated error will be larger, typically $\sum |\varepsilon_i| \approx N \max |\varepsilon_i|$. Also, since it is based on an estimate for the error, it is not exact.
- The "safety factor" `q`. In lines 6 and 13 of the algorithm below, the term $\sqrt{\dfrac{2\mathtt{Tol}}{\Delta}}$ is an estimate for the stepsize h required to achieve the desired truncation error. But if it overestimates the correct value even by a tiny amount we will need an extra step. To avoid this, we choose `q` $\in (0, 1)$ and set $h = \mathtt{q}\sqrt{\dfrac{2\mathtt{Tol}}{\Delta}}$.
- The maximum stepsize ratio `M1`. When the solution switches from a region in which its derivative changes rapidly to a region in which its derivative changes slowly, the algorithm will increase the stepsize. How quickly this happens depends on `M1`. However, we know that having too large a change in stepsize can be bad for the error. Therefore, `M1` controls how quickly the stepsize increases. So we should always have `M1` > 1, but not too large.
- The minimum stepsize ratio `M2`. The purpose of the while loop from lines 11 to 17 is to reduce the stepsize until the tolerance is achieved. Taking `M2` < 1 ensures that h will always be reduced at each iteration of the loop, and so that it will eventually converge. This is likely to occur when the solution moves from a region where it varies slowly to where it varies rapidly.

Algorithm 5.1 The adaptive time step algorithm

while $(t_{step} \leq t_N)$ **do**
  if $(step = 1)$ **then**
    $h = h_0$ (Initial guess for time step)
  else
    $\Delta = \frac{1}{h}\max\left(|f(t_{step}, y_{step}) - f(t_{step-1}, y_{step-1})|\right)$ (Approximate of y'')
    $h = \min\left(\mathbf{q}\sqrt{\frac{2Tol}{\Delta}}, h \times \mathbf{M1}\right)$
  end if
  $t_{step+1} = t_{step} + h$
  $y_{step+1} = y_{step} + hf(t_{step}, y_{step})$;
  $\Delta = \frac{1}{h}\max\left(|f(t_{step+1}, y_{step+1}) - f(t_{step}, y_{step})|\right)$ (estimate the local error)
  while $\frac{\Delta h^2}{2} > Tol$ **do**
    $Rejections = Rejections + 1$;
    $h = \min\left(\mathbf{q}\sqrt{\frac{2Tol}{\Delta}}, h \times \mathbf{M2}\right)$ (reduce the step length)
    $t_{step+1} = t_{step} + h$
    $y_{step+1} = y_{step} + hf(t_{step}, y_{step})$;
    $\Delta = \frac{1}{h}\max\left(|f(t_{step+1}, y_{step+1}) - f(t_{step}, y_{step})|\right)$
  end while
end while

5.2 Van der Pol's Equation

5.2.1 Overview

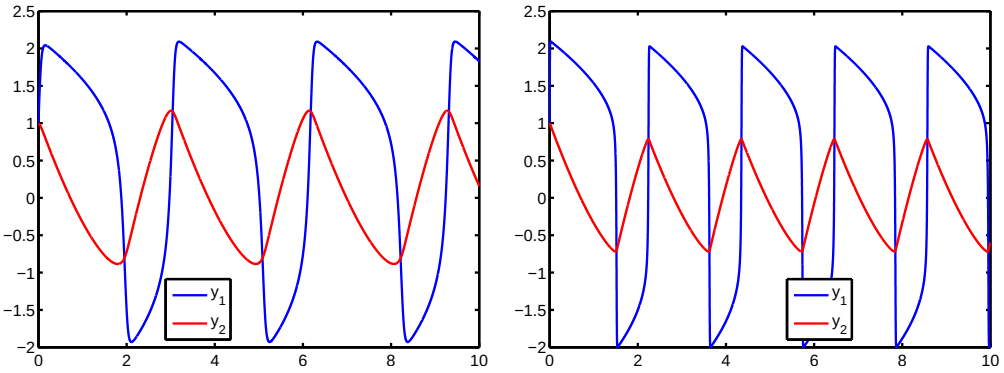
To demonstrate the benefits of the adaptive non-uniform algorithm, we investigate a model based on the classic van der Pol oscillator (see, e.g. [3, §105]). Although originally proposed for modelling electrical circuits, it has found many applications. Most relevant to this study are the wide variety of applications in biology, e.g. [2, Chap. 6]. Here we present the van der Pol equation as a system of two initial value problems:

$$\begin{aligned} \frac{dy_1}{dt} &= \frac{1}{\epsilon}\left(y_2 - \frac{y_1^3}{3} + y_1\right), \quad y_1(0) = 1, \\ \frac{dy_2}{dt} &= a - y_1, \quad y_2(0) = 1. \end{aligned} \tag{5.1}$$

where a and ϵ are parameters. This is a typical example of a stiff problem. Furthermore, since its dynamics are non-uniform and not known in advance, it is a suitable test problem for our proposed algorithm.

Figure 5.1(a) below shows the behavior of the solution on the interval $[0, 10]$ with $a = 0.5$ and $\epsilon = 0.1$.

Since we do not know the exact solution of this problem, we use the Runge-Kutta 4 method (Section 2.2.7) with a very large number of steps, $N = 2^{19}$, to create a benchmark solution. From our perspective, the most important term is ϵ. For smaller values of ϵ the problem is stiff. In particular, 5.1(b) shows the solution when $\epsilon = 0.01$. Notice that the solution changes more rapidly for the smaller value of ϵ.



(a) $\epsilon = 0.1$ (b) $\epsilon = 0.01$

Figure 5.1: Solutions to (5.1)

5.2.2 Numerical Result

We apply Euler's method to (5.1) as shown in Table 5.1. Note that the error of the first component y_1 is quite large even with $N = 279$, the largest number of steps in Table 5.1. In particular, the computed solution obtained with $N = 165$ steps of equal size is shown in Figure 5.2. Here we use $N = 165$ steps because that corresponds to the number of steps taken by the adaptive algorithm with `Tol` $= 2^{-4}$, and permits us to do a direct comparison. Even though the computed solution for y_2 cannot be distinguished from the benchmark, we can observe many oscillations in the approximate solution to y_1. The numerical method does not successfully capture the model dynamics. For example, from the time $t = 5$ onward, a significant lag is observed.

N	$\max_{i=0,1,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,1,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$
117	3.87e+000	1.84e+000
134	3.68e+000	1.58e+000
165	3.26e+000	9.92e-001
216	2.93e+000	7.14e-001
279	2.66e+000	5.55e-001

Table 5.1: Errors in the numerical solution to (5.1) obtained using Euler's method with uniform steps.

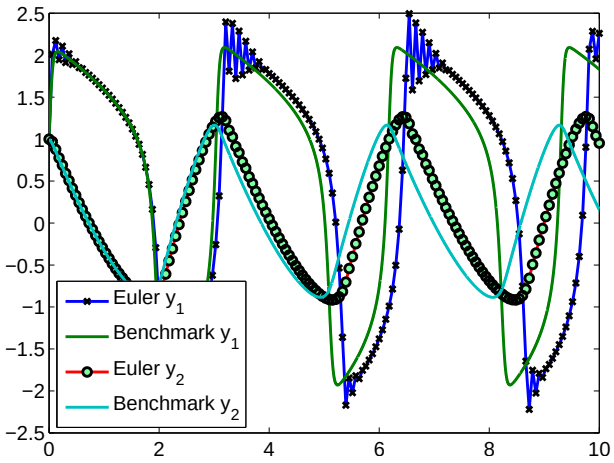


Figure 5.2: Euler's Method with uniform steps applied to (5.1) with $N = 165$.

Now we compute an approximate solution using the adaptive method with the same numbers of steps as in Table 5.1. These result from different tolerances as shown in Table 5.2, along with the corresponding estimated errors. For this computation we took `q` $= 0.9$, `M1` $= 2$ and `M2` $= 0.5$ as the parameters in the adaptive algorithm. We emphasize that these results are more accurate than the results in Table 5.1. For example, with $N = 279$ the errors in y_1 and y_2 are 7 times smaller and 12 times smaller, respectively.

In Figure 5.3 we show the graph of the solution obtained with a tolerance of 2^{-4}. The resulting value of N is 165, so we can compare with the result shown in Figure 5.2. Notice that now the solution dynamics are captured quite well, and there are no obvious oscillations or delay.

`Tol`	N	$\max_{i=0,1,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,1,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$
2^{-2}	117	3.34e+000	1.00e+000
2^{-3}	134	2.42e+000	4.88e-001
2^{-4}	165	1.07e+000	1.81e-001
2^{-5}	216	4.87e-001	7.60e-002
2^{-6}	279	3.86e-001	4.49e-002

Table 5.2: Errors in the computed solution obtained by the adaptive algorithm applied to (5.1).

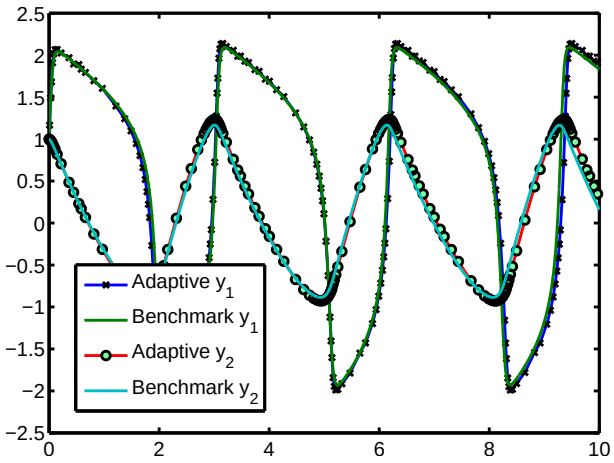


Figure 5.3: The numerical solution to (5.1) obtained by the adaptive algorithm with $N = 165$.

5.2.3 Tuning the parameters

The algorithm we have given above is intended for use in a DBN. To ensure the algorithm efficiency, we need to investigate the best choice of the user-chosen parameters, `q`, `M1` and `M2`. In Table 5.3, we show the results of running the adaptive scheme with various values of the tolerances, and typical values of `q` $= 0.9$, `M1` $= 2$, and `M2` $= 0.5$. Since the stepsizes in adaptive algorithm are allowed to vary, the sixth and seventh columns in Table 5.3 show the corresponding maximum and minimum stepsize together with their ratio in the second last column. The last column shows the run time in seconds. The third column is the number of rejections that occurs when applying the adaptive algorithm. Recalling line 11 of Algorithm 5.1, we see that if the proposed stepsize gives a computed solution that does not satisfy the error tolerance, then that step is rejected, and a new smaller step taken. The *Rejections* counter enumerates the number of times this happens. The smaller this number, the more efficient the algorithm is.

`Tol`	N	*Rejections*	$\max_{i=0,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$	$\max h_i$	$\min h_i$	Ratio	Time
2^{-6}	279	145	3.86e-001	4.49e-002	0.2193	0.0056	39.45	0.0385
2^{-7}	361	158	2.78e-001	3.17e-002	0.1613	0.0042	38.46	0.0341
2^{-8}	454	122	2.89e-001	3.50e-002	0.1144	0.0034	33.27	0.0401
2^{-9}	562	24	3.44e-001	4.30e-002	0.0811	0.0026	31.49	0.0418
2^{-10}	791	28	2.50e-001	3.15e-002	0.0575	0.0021	27.41	0.0591
2^{-11}	1114	32	1.79e-001	2.25e-002	0.0408	0.0016	25.83	0.0827
2^{-12}	1567	29	1.27e-001	1.59e-002	0.0289	0.0013	22.89	0.1168
2^{-13}	2208	27	8.91e-002	1.12e-002	0.0205	0.0009	22.53	0.1619
2^{-14}	3115	25	6.29e-002	7.89e-003	0.0145	0.0006	22.53	0.2291
2^{-15}	4395	16	4.41e-002	5.52e-003	0.0103	0.0005	22.48	0.3214

Table 5.3: Errors in the computed solution obtained by applying the adaptive algorithm to (5.1) with `q` $= 0.9$, `M1` $= 2$ and `M2` $= 0.5$.

It is easier to visualize the errors by using a log-log scale plot as in Figure 5.4. Because of the stiffness

of the problem, we see that the errors increase at first. Clearly, for a small enough value of the tolerance (i.e. large number of steps) the errors reduce, and method is convergent for large enough N.

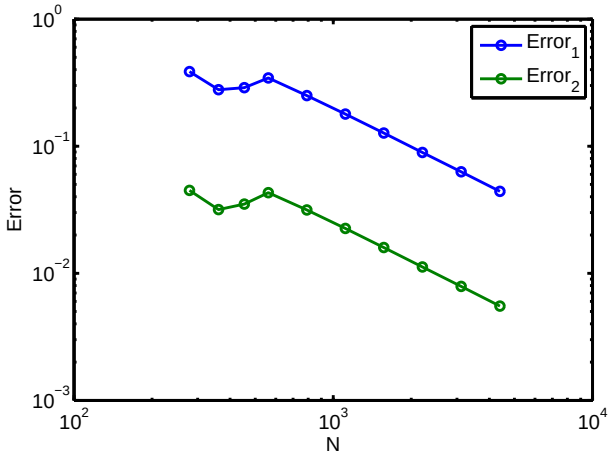


Figure 5.4: Errors in the computed solution using the adaptive algorithm for (5.1).

Recall that the parameter `q` is the "safety factor": its purpose is to adjust the stepsize to something just less than the maximum recommended. We first investigate its optimal value by fixing `Tol` $= 2^{-10}$, `M1` $= 2$ and `M2` $= 0.5$. The results are shown in Table 5.4. From this, we see that a choice of `q` $= 0.9$ gives the best result in terms of the total number of steps computed (i.e., the sum of the number of accepted and rejected steps). However, any value in the range $[0.75, 0.9]$ appears to give acceptable results.

`q`	N	Rejections	$N + Rej$	$\max_{i=0,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$	$\max h_i$	$\min h_i$	Ratio
0.10	6979	1	6980	2.76e-002	3.45e-003	0.006	0.0003	22.49
0.20	3489	1	3490	5.48e-002	6.84e-003	0.013	0.0006	22.54
0.30	2326	1	2327	8.16e-002	1.02e-002	0.019	0.0009	22.47
0.40	1744	1	1745	1.08e-001	1.34e-002	0.026	0.0011	22.64
0.50	1395	1	1396	1.34e-001	1.66e-002	0.032	0.0014	22.68
0.60	1164	3	1167	1.62e-001	2.01e-002	0.038	0.0017	22.57
0.65	1077	7	1084	1.77e-001	2.20e-002	0.042	0.0018	22.63
0.70	1001	8	1009	1.90e-001	2.36e-002	0.045	0.0020	22.79
0.75	936	11	947	2.06e-001	2.58e-002	0.048	0.0021	22.84
0.80	882	18	900	2.23e-001	2.80e-002	0.051	0.0023	22.75
0.90	791	28	819	2.50e-001	3.15e-002	0.058	0.0021	27.41
0.95	874	270	1144	1.90e-001	2.44e-002	0.061	0.0017	36.66
0.99	1080	733	1813	2.97e-001	3.90e-002	0.063	0.0014	44.56

Table 5.4: Errors in the computed solution obtained by using the adaptive algorithm to (5.1) with various `q`.

Now recall that the purpose of `M1` is to control the maximum increased stepsize. We then fix `q` $= 0.9$, `M2` $= 0.5$ and `Tol` $= 2^{-10}$, and investigate how the algorithm behaves for different values of `M1`. We observe that there are no significant changes of the results in Table 5.5, so the range of suitable values of `M1` for this example is [1.2,2]. Typically, we will pick `M1` $= 1.5$.

M1	N	*Rejections*	$N + Rej$	$\max_{i=0,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$	$\max h_i$	$\min h_i$	Ratio
1.10	831	12	843	1.82e-001	2.14e-002	0.0544	0.0020	27.843
1.20	796	14	810	2.34e-001	2.92e-002	0.0565	0.0024	24.009
1.40	790	19	809	2.47e-001	3.10e-002	0.0573	0.0019	30.197
1.50	790	22	812	2.50e-001	3.13e-002	0.0578	0.0019	31.156
1.60	791	26	817	2.52e-001	3.16e-002	0.0579	0.0020	28.368
1.75	790	26	816	2.51e-001	3.15e-002	0.0577	0.0022	26.814
2.00	791	28	819	2.50e-001	3.15e-002	0.0575	0.0021	27.412
3.00	791	28	819	2.50e-001	3.15e-002	0.0575	0.0021	27.407
5.00	791	28	819	2.50e-001	3.15e-002	0.0575	0.0021	27.407
10.00	791	28	819	2.50e-001	3.15e-002	0.0575	0.0021	27.407
20.00	791	28	819	2.50e-001	3.15e-002	0.0575	0.0021	27.407

Table 5.5: Errors in the computed solution obtained by using the adaptive algorithm to (5.1) with various values of M1.

Finally, we investigate the range of choices for M2—whose role is to control the minimum stepsize decrease. It gives a lower bound for a new stepsize to avoid a hugely sudden reduction in two successive steps. We fix q = 0.9, M1 = 1.5 and Tol = 2^{-10}, and let M2 vary from 0.01 to 20. Table 5.6 shows a sample of the results. From these we suggest M2 = 0.9 is the best choice in terms of the accuracy and computational cost.

M2	N	*Rejections*	$N + Rej$	$\max_{i=0,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$	$\max h_i$	$\min h_i$	Ratio
0.10	824	14	838	2.54e-001	3.20e-002	0.0577	0.0004	140.333
0.25	799	14	813	2.54e-001	3.20e-002	0.0573	0.0011	50.042
0.40	796	21	817	2.50e-001	3.15e-002	0.0575	0.0015	37.681
0.50	790	22	812	2.50e-001	3.13e-002	0.0578	0.0019	31.156
0.60	788	27	815	2.50e-001	3.13e-002	0.0574	0.0024	24.391
0.75	784	26	810	2.47e-001	3.09e-002	0.0577	0.0025	22.732
0.80	783	24	807	2.47e-001	3.08e-002	0.0576	0.0025	22.684
0.85	783	26	809	2.46e-001	3.06e-002	0.0577	0.0025	22.708
0.90	782	22	804	2.44e-001	3.04e-002	0.0576	0.0025	22.667
0.95	782	22	804	2.44e-001	3.04e-002	0.0576	0.0025	22.667

Table 5.6: Errors in the computed solution obtained by using the adaptive algorithm to (5.1) with various values of M2.

We now regenerate results corresponding to those in Table 5.3, but now with the selected set of parameters q = **0.9**, M1 = **1.5** and M2 = **0.9**, and show the results in Table 5.7. The corresponding log-log plot is shown in Figure 5.5. Note that the errors decrease for all values of N. Furthermore, the number of rejections in this case is less than the number of rejections in Table 5.3.

Tol	N	*Rejections*	$\max_{i=0,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$	$\max h_i$	$\min h_i$	Ratio	Time
2^{-6}	233	97	4.56e-001	9.22e-002	0.2197	0.0096	22.80	0.0353
2^{-7}	304	113	4.20e-001	5.00e-002	0.1494	0.0070	21.47	0.0290
2^{-8}	406	95	3.76e-001	4.58e-002	0.1136	0.0050	22.75	0.0361
2^{-9}	556	24	3.37e-001	4.19e-002	0.0810	0.0036	22.73	0.0482
2^{-10}	782	22	2.44e-001	3.04e-002	0.0576	0.0025	22.67	0.0589
2^{-11}	1102	24	1.74e-001	2.17e-002	0.0410	0.0018	22.68	0.0830
2^{-12}	1557	25	1.24e-001	1.55e-002	0.0289	0.0013	22.58	0.1169
2^{-13}	2198	22	8.79e-002	1.10e-002	0.0205	0.0009	22.60	0.1645
2^{-14}	3105	20	6.21e-002	7.76e-003	0.0145	0.0006	22.54	0.2328
2^{-15}	4390	19	4.39e-002	5.49e-003	0.0102	0.0005	22.47	0.3264

Table 5.7: Errors in the computed solution obtained by using the adaptive algorithm to (5.1) with q = 0.9, M1 = 1.5 and M2 = 0.9.

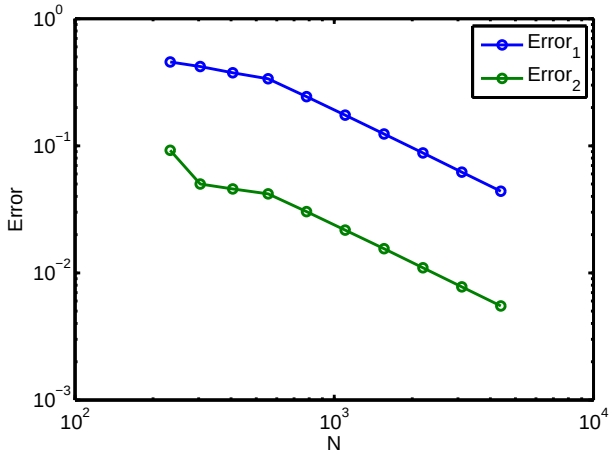


Figure 5.5: A log-log plot of the errors shown in Table 5.7.

Finally, let us compare the results in Table 5.7 with Euler's Method (uniform stepsize) using the same number of steps as in Table 5.8 to see the advantage of the adaptive algorithm. Note that, for $N = 4390$ we achieve an accuracy in y_1 and y_2 of 2.89×10^{-1} and 3.56×10^{-2} respectively in 0.1449 seconds. The adaptive algorithm can surpass this using only $N = 782$ and in 0.0589 seconds.

N	*Rejections*	$\max_{i=0,\ldots,N} \lvert y_1(t_i) - y_{1,i} \rvert$	$\max_{i=0,\ldots,N} \lvert y_2(t_i) - y_{2,i} \rvert$	Stepsize	Time
233	0	2.85e+000	6.61e-001	0.0429	0.1148
304	0	2.57e+000	5.09e-001	0.0329	0.0636
406	0	2.22e+000	3.83e-001	0.0246	0.0136
556	0	1.85e+000	2.80e-001	0.0180	0.0311
782	0	1.44e+000	2.00e-001	0.0128	0.0583
1102	0	1.08e+000	1.42e-001	0.0091	0.0366
1557	0	7.91e-001	1.00e-001	0.0064	0.0515
2198	0	5.70e-001	7.11e-002	0.0045	0.0727
3105	0	4.07e-001	5.03e-002	0.0032	0.1031
4390	0	2.89e-001	3.56e-002	0.0023	0.1449

Table 5.8: Errors in the computed solution to (5.1) obtained by using Euler's Method using uniform steps.

With the `Tol` $= 2^{-8}$, `M1` $= 1.5$, `q` $= 0.9$ and `M2` $= 0.9$, the Figure 5.6 shows the stepsizes on the interval $[0, 10]$. The top figure shows the approximate solution of each components. The middle figure presents the corresponding stepsize used. We can see that when the solution, and particularly the first component, is changing rapidly, the algorithm proposed very small time steps. Then, as the solution transitions to varying more slowly, the stepsizes increase. The third figure shows the corresponding errors. We can observe that the error is larger where the solution changes dramatically.

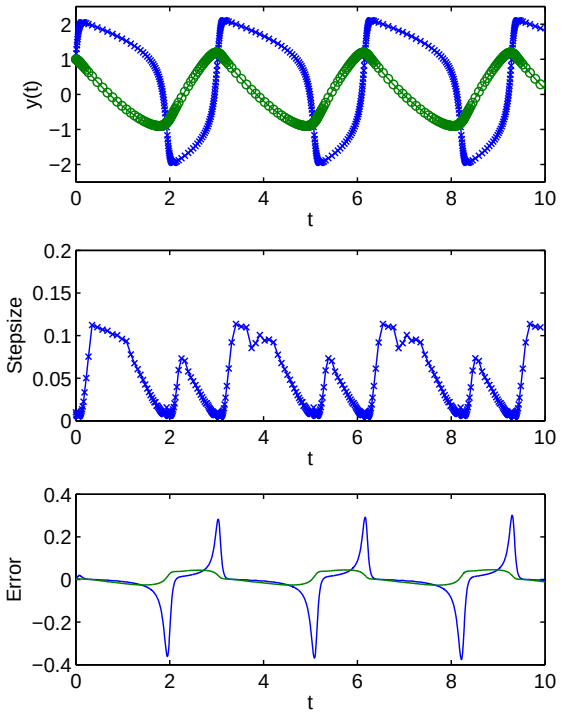


Figure 5.6: The approximate solution (top), the stepsizes (middle) and the corresponding errors (bottom) using the adaptive algorithm.

5.3 The Updated Model

As mentioned earlier in Remark 4.1, our colleagues Liam O'Callaghan and Petri Piiroinen have recently produced the following model [14]. Our task is to design an efficient algorithm that can compute a reliable solution efficiently and that can be incorporated into the DBN. The model is:

$$
\begin{aligned}
v_1\frac{dG_1}{dt} &= \frac{k_1\big(G_2(t)-G_1(t)\big)}{k_{m1}+G_1(t)+G_2(t)} - \frac{\big(k_2+k_0I_3(t)\big)G_1(t)}{k_{m0}+G_1(t)} + F_G(t),\\
v_2\frac{dG_2}{dt} &= \frac{k_1\big(G_1(t)-G_2(t)\big)}{k_{m1}+G_1(t)+G_2(t)} + \frac{k_3gly}{1+\exp\big(k_{31}(I_3(t)-b_1)\big)} - \frac{k_3G_2(t)}{1+\exp\big(k_{31}(b_2-I_3(t))\big)},\\
v_3\frac{dI_1}{dt} &= k_5\big(I_{\max}-I_1(t)\big) - \boldsymbol{k_{62}I_1} - \frac{k_6I_1(t)}{1+\exp\big(k_{61}(c_1-G_1(t))\big)},\\
v_4\frac{dI_2}{dt} &= \frac{k_6I_1(t)}{1+\exp\big(k_{61}(c_1-G_1(t))\big)} + \boldsymbol{k_{62}I_1} - \frac{k_7I_2(t)}{k_{m7}+I_2(t)} - k_8I_2(t) + F_I(t),\\
v_5\frac{dI_3}{dt} &= \frac{k_7I_2(t)}{k_{7m}+I_2(t)} - k_9I_3(t).
\end{aligned}
\tag{5.2}
$$

Compared with (4.8), the terms in bold have been added to the new model.

5.3.1 Parameters and Initial Conditions

As discussed in [14], there are many things that effect the body's behavior and response to glucose and insulin infusions, such as an individual's evolving hormonal situation. Therefore, it is inevitable that model parameters are subject to change even over a short period of time. In particular, to fit the observed data for Patient 102, five subintervals have been chosen, each with a different set of parameter values. The subintervals (in minutes) are $[663, 903]$, $[903, 1320]$, $[1320, 2700]$, $[2700, 3611]$ and $[3611, 4680]$. Their specific values are presented in next section. In [14], the values for v_i and the initial values given in Table 5.9 are proposed.

Parameter	Value	Parameter	Value
v_1	120	G_1	172.8
v_2	480	G_2	23.383
v_3	5	I_1	3.24845
v_4	80	I_2	44.2727
v_5	172	I_3	9.67814

Table 5.9: Initial values for (5.2).

The values of F_G, the rate of infusion of glucose, and F_I, the rate of infusion of insulin, can be changed over time. Their values are given in Table 5.10.

	Value	Time
F_G	0	$663 \leq t \leq 1170$
F_G	130	$1170 < t \leq 4680$
F_I	50	$663 \leq t \leq 1763$
F_I	100	$1763 < t < 3522$
F_I	50	$3522 \leq t \leq 4680$

Table 5.10: Values of F_G and F_I in (5.2).

Table 5.11 shows values of other parameters for each of the five intervals as recommended. For more detailed discussion, we refer readers to [14].

	$663-903$	$903-1320$	$1320-2700$	$2700-3611$	$3611-4680$
k_1	7.86956	1252.37000	168.49800	97.41470	211.40400
k_{1m}	94.60890	7.31446	166.22300	84.17190	146.01000
k_2	2.94199	144.28500	153.46600	30.24230	255.24000
k_0	1.92455	2107.18000	146.45200	26.30920	3.16033
k_{0m}	23.35690	99.46770	30.99500	11.06300	13.84840
k_3	25.58330	1.24518	1.87608	10.83570	45.22780
k_{31}	9.75146	1.17483	1.84850	2.19118	1.02463
b_1	1.05747	231.69200	5.98174	74.76230	163.70300
b_2	116.51500	468.50300	136.79000	541.33200	1186.60000
gly	447.43800	14.58550	160.70800	2.91558	38.32260
k_5	47.21230	23.01750	1.89866	2.55946	29.28230
I_{max}	1.71010	28.27270	4.17840	1.74772	8.30618
k_6	1.71618	3.27457	2.59920	14.20370	115.27400
k_{61}	1.55723	1.49683	3.78112	2.04527	3.58975
k_{62}	5.56218	2.80181	4.00191	1.15245	1.49800
c_1	538.54600	149.33100	76.51730	1086.64000	131.55200
k_7	1.48927	3.04619	15.59210	88.86110	55.17500
k_{7m}	48.58910	41.79710	1.68236	14.34170	3.95976
k_8	4.11877	5.58924	49.75270	9.70765	1.08533
k_9	2.15535	1.79874	8.84896	4.04624	1.54755

Table 5.11: Values of other parameters in (5.2).

5.4 Numerical Results

5.4.1 Some observations

In this section, we study the numerical solution to (5.2) for specific data corresponding to Patient 102. The duration of the simulation is from $t = 663$ minutes to $t = 4680$ minutes. We will mostly focus on

the results for Euler's Method and the adaptive algorithm. We also suggest a refinement of the adaptive scheme to improve the efficiency for this particular problem.

As discussed in the previous section, the sets of values of parameters are given in five separate intervals, shown in Table 5.11. In addition, we also have observed glucose values from Patient 102. In Figure 5.7, we show the computed solution obtained by using `ode15s`, a built-in differential equation solver in Matlab. It is based on multistep method with variable order, and is designed to be particularly suitable for stiff problems [16]. Therefore, it is a suitable for generating a benchmark solution. The coloured diamonds are the observed glucose values.

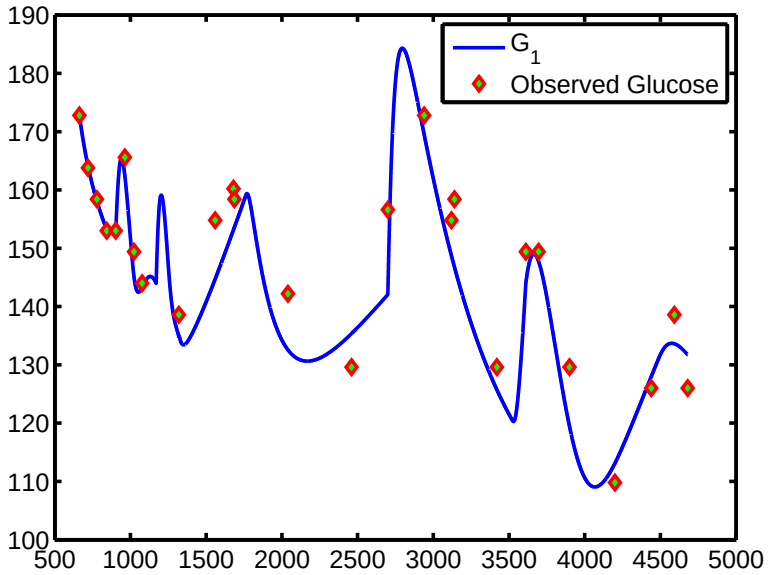


Figure 5.7: Predicted and observed blood glucose levels for $t \in [663, 4680]$.

The new model and the changes of parameter values over time present some numerical difficulties. In fact, the model (5.2) is solved on a long interval of 4017 minutes. This in turn is divided into five separate subintervals corresponding to different sets of values of parameters. The values of parameters vary very widely; that can adversely effect numerical methods. For example, Figure 5.8 shows the computed G_1 using Euler's Method with $N = 20443$ uniform steps (blue line). Note that it does not suggest a meaningful result for G_1.

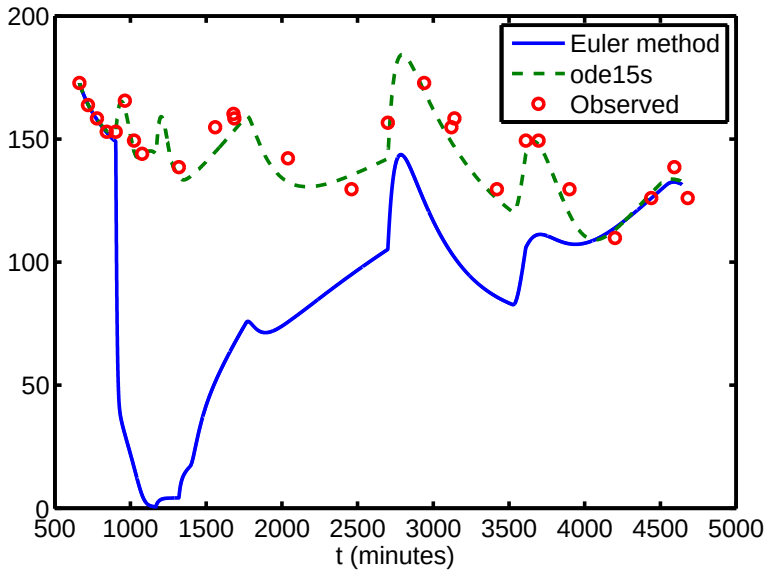


Figure 5.8: The G_1 obtained by using Euler's Method with $N = 20443$ uniform steps.

In Figure 5.9 we show the result obtained using the same number of steps but with the adaptive algorithm. We see that this gives a numerical solution that is indistinguishable from the benchmark solution generated using Matlab's `ode15s`.

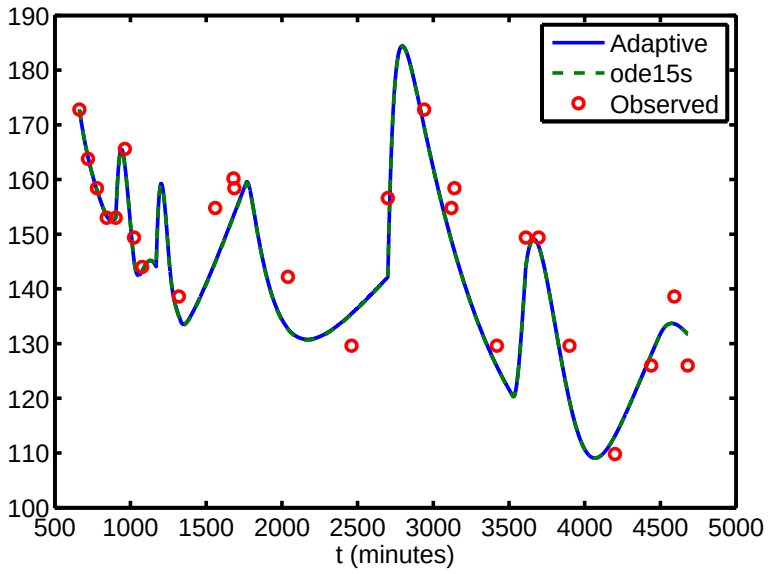


Figure 5.9: The G_1 obtained by using the adaptive algorithm with $N = 20443$.

Another complication is that the magnitude of the five components are very different from each other. For example, G_1, which is measured in mg/dl ranges from 108.6 to 183.2, while the value of G_2, also measured in mg/dl, increases from 23 to over 4465. Figure 5.10 shows the solution for each component. This fact suggests some refinements can be made to further improve the adaptive algorithm for this particular problem.

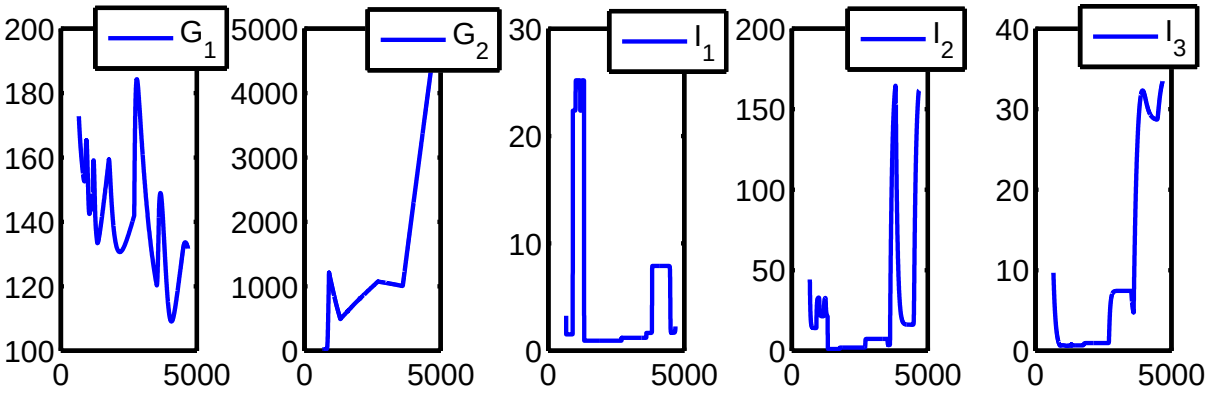


Figure 5.10: The graph of solutions of each component in (5.2).

We apply Euler's method to (5.2); the errors are shown in Table 5.12 and visualized in Figure 5.11. Note that, because of the some difficulties of (5.2) as discussed in previous section, one must take N very large to obtain meaningful results. Indeed, for $N < 70740$, the method fails to compute a physically meaningful solution.

No. of Steps	No. of Rej	G_1	G_2	I_1	I_2	I_3
17223	0	1.73e+002	1.04e+003	NaN	NaN	4.06e+001
20443	0	1.56e+002	1.02e+003	NaN	NaN	5.63e+000
26913	0	2.16e+000	2.42e+000	NaN	NaN	1.40e+000
36443	0	1.86e+000	1.78e+000	NaN	NaN	1.14e+000
50358	0	3.57e+001	3.32e+000	NaN	NaN	6.93e+000
70740	0	1.65e-001	9.19e-001	1.48e+000	7.49e-001	8.14e-003
99774	0	1.17e-001	6.52e-001	1.00e+000	4.89e-001	5.34e-003

Table 5.12: Errors in the numerical solution to (5.2) using Euler's Method with uniform steps.

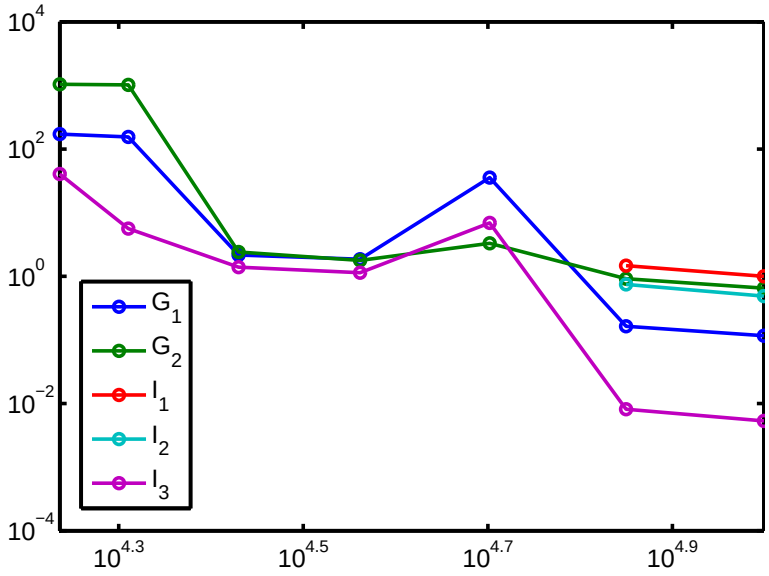


Figure 5.11: A log-log plot of the errors shown in Table 5.12.

5.4.2 Turning the user-chosen parameters

In Section 5.2.3 we showed how to carefully tune the parameters `q`, `M1`, and `M2` for the van der Pol example (5.1). We repeated the same procedure for the glucose-insulin model (5.2). However, for the sake of brevity, we don't present all the tables that we generated for that investigation, and just present our findings. For this particular problem, we suggest their values are `q` $= 0.9$, `M1` $= 1.1$ and `M2` $= 0.9$. We present results obtained using the adaptive algorithm with these parameters in Table 5.13 and Figure 5.12. Comparing with Table 5.12 we see that they are much more accurate than when Euler's method is applied with uniform steps. Indeed, physically meaningful results are obtained with a relatively small number of time steps (compared with the explicit Euler method); and we observe typical 1^{st}-order convergence. As we also see in Figure 5.12, the adaptive algorithm gives acceptable results even with a small number of steps, say $N = 17219$.

No. of Steps	No. of Rej	G_1	G_2	I_1	I_2	I_3
17219	110	5.02e-001	4.09e+000	1.18e+000	2.88e+000	3.09e-002
20440	123	4.50e-001	3.80e+000	1.12e+000	2.71e+000	2.97e-002
26910	96	4.00e-001	3.47e+000	1.04e+000	2.51e+000	2.76e-002
36439	67	3.51e-001	3.07e+000	9.32e-001	2.24e+000	2.47e-002
50355	53	2.97e-001	2.59e+000	7.93e-001	1.90e+000	2.10e-002
70736	38	2.10e-001	1.83e+000	5.62e-001	1.34e+000	1.49e-002
79686	*37*	*1.86e-001*	*1.62e+000*	*4.98e-001*	*1.19e+000*	*1.32e-002*
99771	34	1.48e-001	1.29e+000	3.98e-001	9.51e-001	1.06e-002

Table 5.13: Errors in the computed solution to (5.2) by the adaptive algorithm.

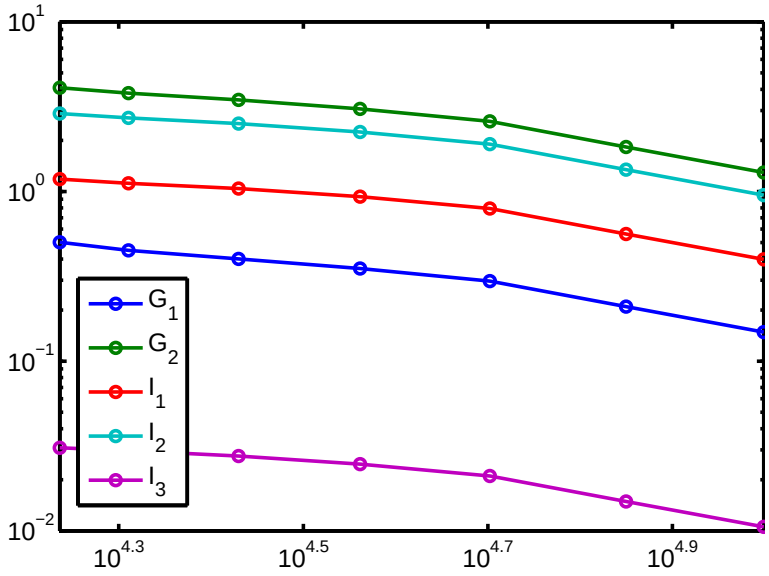


Figure 5.12: A log-log plot of the errors shown in Table 5.13.

5.4.3 A variation on the adaptive algorithm

As we see in Figure 5.10, the magnitudes of the components of the solution are quite different. Therefore, instead of using the usual estimate for the truncation error:

$$T_n = y(t_n) - y_n \approx \frac{h^2}{2} y'',$$

we compute the relative truncation error:

$$\Delta T_n = \frac{y(t_n) - y_n}{y_n} \approx \frac{h^2 y''}{2 y_n}.$$

With this change in algorithm, the errors in Table 5.14 are smaller than these in Table 5.13 even using a similar number of steps. For example, we can compare $N = 81659$ in Table 5.14 with $N = 79686$ in Table 5.13.

No. of Steps	No. of Rej	G_1	G_2	I_1	I_2	I_3
18127	234	6.04e-001	4.25e+000	1.13e+000	2.76e+000	3.03e-002
23543	140	4.33e-001	2.99e+000	7.78e-001	1.89e+000	2.08e-002
31346	213	3.06e-001	2.11e+000	5.50e-001	1.34e+000	1.47e-002
42562	133	2.17e-001	1.49e+000	3.89e-001	9.45e-001	1.05e-002
58661	152	1.53e-001	1.06e+000	2.76e-001	6.71e-001	7.48e-003
81659	72	1.09e-001	7.47e-001	1.96e-001	4.76e-001	5.36e-003

Table 5.14: Errors in the computed solution to (5.2) by a variant of the adaptive algorithm.

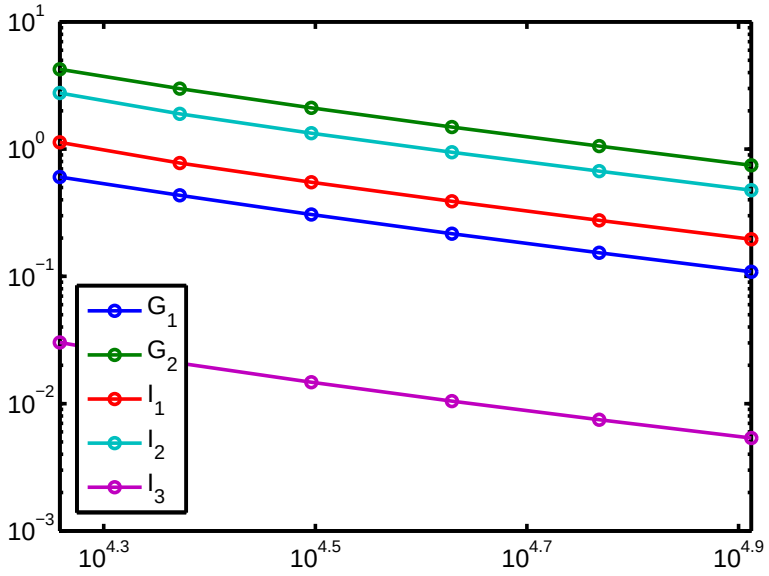


Figure 5.13: A log-log plot of the errors shown in Table 5.14.

Note that, for the example we have considered we always have, approximately, $\min_n y_n \geq 0.6$. For a different set of parameters, if $\min_n y_n \approx 0$, or if $\min_n y_n < 0$, this might be unstable.

5.5 Conclusion

In this chapter we have presented an adaptive algorithm that allows the stepsize to vary in order to capture the dynamics of systems in a highly efficient manner. The numerical result obtained by applying this algorithm to the classic van der Pol oscillator shows the significant improvement compared with Euler's method using uniform steps. Furthermore, we have investigated how to calibrate the user-chosen parameters in order to optimise the efficiency. We have also shown how to numerically solve the latest glucose insulin model, and obtained a significant improvement in terms of the efficiency and accuracy. In addition, this scheme has been successfully integrated into a DBN as shown in a recent research article [4]. Using the van der Pol oscillator (5.1) as a test example, that paper shows that the *Adaptive Time Bayesian Networks* gives more accurate results that the corresponding DBN, with 10 times fewer time steps required to obtain reasonable results. The adaptive scheme has currently being incorporated into the DBN for simulating the glucose insulin model. This is an on-going task, but initial results are promising.

Chapter 6

Conclusions

As part of a larger project, our primary goal has been to develop a suitable numerical algorithm that can be incorporated into a Dynamic Bayesian Network (DBN) that simulates glucose and insulin levels in ICU patients. The DBN is a type of expert system that encodes "expert knowledge" and automatically determines suitable values for parameters. In our case the expert knowledge is in the form of a system of differential equations. However, these differential equations themselves presents some challenges, such as discontinuities in data and stiffness. In this thesis we have studied such difficulties and suggested suitable numerical schemes to handle these issues. Furthermore, although we have not discussed it in detail here, many of our numerical results have provided important contributions to other members of the research group in achieving their goals.

The thesis has successfully achieved the key target of our role in the project by proposing numerical methods that are robust enough to generate accurate solutions to the quite difficult problems posed, but simple and efficient enough to be incorporated into the DBN. Of our suggestions, the most successful has been the adaptive time stepping algorithm of Chapter 5. This framework was suitable for differential equations with rapidly varying solutions. Moreover, it has now been incorporated into a DBN leading to the new:"Adaptive Time Bayesian Networks (ATBNs)" [4]. That study describes the new method, and has investigated its potential by applying it to the model problem (5.1). As stated in [4]:

> To evaluate the methodology, we built both a DBN and an ATBN based on a variant of the Van der Pol oscillator. It was shown that the ATBN can be run with 10 times fewer time steps, with corresponding run-time improvements, while also performing more accurate inference.

The adaptive algorithm is now a key part of the ATBN that uses the new model (5.2) to simulate insulin and glucose levels in realistic situations. Testing and refinement of the methodology are on-going, but initial results are very encouraging.

Bibliography

[1] R. N. Bergman, L. S. Phillips, and C. Cobelli. Physiologic evaluation of factors controlling glucose tolerance in man; Measurement of insulin sensitivity and β-cell glucose sensitivity from the response to intravenous glucose. *Journal of Clinical Investigation*, 68(6):1456–1467, 1981.

[2] Nicholas F. Britton. *Essential Mathematical Biology.* Springer Undergraduate Mathematics Series. Springer-Verlag London Ltd., London, 2003.

[3] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations.* John Wiley & Sons Ltd., Chichester, second edition, 2008.

[4] C. Enright, M. Madden, N. Madden, and A. T. Nhan. Adaptive time bayesian networks. *Neural Information Processing Systems*, 2011. submitted.

[5] C. G. Enright, M. G. Madden, S. Russell, N. Aleks, G. Manley, J. Laffey, B. Harte, A. Mulvey, and N. Madden. Modelling glycaemia in ICU patients: A dynamic bayesian network approach. In *BIOSIGNALS 2010 - Proceedings of the 3rd International Conference on Bio-inpsired Systems and Signal Processing, Proceedings*, pages 452–459, 2010.

[6] J. Douglas Faires and Richard Burden. *Numerical Methods.* Brooks/Cole Publishing Co., Pacific Grove, CA, second edition, 1998.

[7] Laurene Fausett. *Applied Numerical Analysis Using Matlab.* Prentice Hall, 2008.

[8] Brunkhorst F.M., Engel C., and Bloos F. et al. Intensive insulin therapy and pentastarch resuscitation in severe sepsis. *New England Journal of Medicine*, 358:125–39, 2008.

[9] B. Braun Space GlucoseControl. Integrated glucose control. Technical report, 2011. `http://www.space.bbraun.com/documents/BBraun_Space_GlucoseControl_Scientific_Folder.pdf`.

[10] N. Haverbeke, T. Van Herpe, M. Diehl, G. Van Den Berghe, and B. De Moor. Nonlinear model predictive control with moving horizon state and disturbance estimation – application to the normalization of blood glucose in the critically ill. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*, volume 17, 2008.

[11] Desmond J. Higham and Nicholas J. Higham. *MATLAB guide.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2005.

[12] Arieh Iserles. *A First Course in the Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 1996.

[13] D. S. Jones, M. J. Plank, and B. D. Sleeman. *Differential equations and mathematical biology*. Chapman & Hall/CRC Mathematical and Computational Biology Series. CRC Press, Boca Raton, FL, second edition, 2010.

[14] Liam O'Callaghan. Models of glycemic regulation. Master's thesis, National University of Ireland, Galway, 2011. Supervisor: P.T. Piiroinen.

[15] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2002.

[16] Lawrence F. Shampine and Mark W. Reichelt. The MATLAB ODE suite. *SIAM J. Sci. Comput.*, 18(1):1–22, 1997.

[17] Endre Süli and David F. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, Cambridge, 2003.

[18] Sauer T. *Numerical Analysis*. Pearson Addison Wesley, 2006.

[19] Hugo van den Berg. *Mathematical Models of Biological Systems*. Oxford University Press, 2011.

[20] T. Van Herpe, M. Espinoza, N. Haverbeke, B. De Moor, and G. Van Den Berghe. Glycemia prediction in critically ill patients using an adaptive modeling approach. *J Diabetes Sci Technol*, 1:348–356, 2007.