DISSERTATION

GLOBAL ESTIMATE AND CONTROL OF MODEL, NUMERICAL, AND PARAMETER ERROR

Submitted by

Jeffrey David Sandelin

Department of Mathematics

In partial fulfillment of the requirements

for the degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2006

COLORADO STATE UNIVERSITY

August 28, 2006

WE HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER OUR SUPERVISION BY JEFFREY DAVID SANDELIN ENTITLED "GLOBAL ESTIMATE AND CONTROL OF MODEL, NUMERICAL, AND PARAMETER ERROR" BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Dr. Dennis Ojima

Dr. Simon Tavener

Dr. James Liu

Adviser: Dr. Donald Estep

Department Head: Dr. Simon Tavener

ABSTRACT OF DISSERTATION

GLOBAL ESTIMATE AND CONTROL OF MODEL, NUMERICAL, AND PARAMETER ERROR

Ordinary differential equations (ODEs) are used extensively in the modeling of natural phenomena in nearly all scientific and engineering fields. In this thesis, we derive *a posteriori* error estimates based on adjoint analysis for global error estimation of numerical solutions of ODEs and design and implement various global error control mechanisms. Solution methods are based on continuous and discontinuous Galerkin finite element methods. We take a global approach to error estimation and control by treating the time dependent equations in a manner similar to elliptic problems and solving the equations for the entire time period, then applying *a posteriori* error estimates based on duality, the adjoint problem, the generalized Green's function, and variational analysis. We develop and explore new approaches to adaptive error control based on probability and weighted zones. These methods are implemented and compared to classic error control based on "Equidistribution of Error" using a variety of test problems. Finally, we provide analysis that shows that modeling error cannot be ignored.

Jeffrey David Sandelin
Department of Mathematics
Colorado State University
Fort Collins, Colorado 80523
Fall 2006

ACKNOWLEDGEMENTS

I would like to thank first and foremost my advisor, Dr. Donald Estep, for the direction, support, enthusiasm, and patience he has provided for the past three years. I would also like to thank Professor Simon Tavener for his support during the early years of my return to school, and Professor Ken Klopfenstein for making my first two classes in mathematics after 25 years such an enjoyable experience. In addition, I would like to thank Dr. Terry Waddle at the USGS for his flexibility and support. I would also like to thank Becky McKeown for her friendship and her insight into carbon cycle modeling. Finally, I would like to thank my wife, Terrie, for her love and support while I have pursued my education. I could not have done it without her.

This research is based upon work partially supported by the National Science Foundation under IGERT Grant No. DGE-0221595.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

ESTIMATING THE ACCURACY OF NUMERICAL SOLUTIONS AND ADAPTIVE ERROR CONTROL

Over the past several decades computing power has grown enormously and, with that growth, the ability to compute numerical solutions of increasingly more complex ordinary differential equations has also grown. Ordinary differential equations (ODEs) are used extensively in the modeling of natural phenomena in nearly all scientific and engineering fields. We are particularly interested in ecological problems, with applications ranging from the global carbon cycle and weather prediction to plankton dynamics and biological kinetics.

With the increased ability to solve hard problems, mathematics has turned to other issues than simply computing solutions. In particular, determining the accuracy of a particular computed solution has become increasingly important. This is a consequence of the increased need to rely on numerical simulations as a core scientific tool for the investigation of new phenomena. Just as physical experiments require an understanding of experimental error, it necessary to quantify the uncertainties in numerical solutions of differential equations. It is not without reason that we refer to these solutions as "approximations".

In this thesis, we derive *a posteriori* error estimates based on adjoint analysis for global error estimation of numerical solutions of ODEs and design and implement various global error control mechanisms. Our solution methods are based on continuous and discontinuous Galerkin (cG and dG) finite element methods, which, with an appropriate choice of quadrature, can be identified with certain implicit Runge-Kutta methods. We take a truly global approach to error estimation and control by treating the time dependent equations in a manner similar to elliptic problems and solving the equations for the entire time period, then applying *a posteriori* error estimates based on duality, the adjoint problem, the generalized Green's function, and variational analysis.

The *a posteriori* error estimates give an exact representation of the chosen quantity of interest in terms of residuals of the numerical solution and factors involving the generalized Green's function corresponding to the quantity of interest. The residuals measure how well the numerical solution satisfies the problem at each time, while the generalized Green's function provides a way to determine how the local production of error accumulates to affect the global quantity of interest. Using the generalized Green's function provides a way to account for the global effects of stability.

The residuals of the numerical solution are either computable or can be estimated. We approximate the adjoint factors by numerically solving the adjoint problem for the generalized Green's function. This approach was originally developed by Estep [13] following Johnson and colleagues [12], and has been used extensively since [19].

We return to this subject in order to

1. Carry out a more detailed *a posteriori* analysis that includes the effect of using quadrature to evaluate the finite element method and the effects errors in parameters, which is called modeling error

2. Develop a systematic software tool that increases the accessibility of this approach to scientists and engineers, and addresses some fundamental computational issues in this approach

3. Develop and explore new approaches to adaptive error control

4. Make a series of applications illustrating these ideas.

To accomplish item 2, the adaptive ODE solver will be combined with existing software that carries out parameter sensitivity analysis into a package called the Globally Accurate Adaptive Sensitivity Package, or GAASP.

A comment on adaptive error control is warranted. The *a posteriori* error estimates can be decomposed as the sum of time step contributions. The error control/adaptivity issue is to decide how to change the time steps in order to achieve a given error in a computed quantity of interest. The classic approach simply seeks to reduce the elements with the largest residual or "local errors". The analog in our context is to reduce the time steps with the largest contributions. This is the standard "Principle of Equidistribution" that results from viewing adaptivity as an optimization problem. However, this is not optimal in general because of cancellation between element contributions. For this reason, we seek new refinement strategies. These strategies include a probabilistic method and a weighted zone method that seeks to balance refinement between zones of positive and negative contribution. Both methods use concepts from Monte Carlo

simulation including the Fundamental Theorem of Simulation and accept-reject methology.

We provide 23 examples, many with exact solutions available to check accuracy of the computations, and show convergence and error estimation results. Twelve of the example problems were used as the test suite. Finally, we apply the theory to applications on the global carbon cycle and biokinetics. Both applications provided examples rooted in the biological and/or ecological fields. The global carbon cycle model is a complex model, even in a simplified form, and the biokinetics model provides an example with extreme change over a short time period.

Chapter 2

ORDINARY DIFFERENTIAL EQUATIONS

In this chapter, we review some of the basic results about ODEs. For a detailed look at ODE theory there are many good references including [36, 42, 43]. For purposes of this document, we restrict ourselves to an overview of existence and uniqueness results, order of convergence, and stability. These topics provide a basis for further discussion.

2.1 Definitions

An initial value problem for a system of ordinary differential equations takes the form,

$$\begin{cases} \dot{y} = f(y,t), & 0 < t \leq T, \\ y(0) = y_0, \end{cases} \tag{2.1.1}$$

where $y \in \mathbb{R}^P$, $P \geq 1$, $f : \mathbb{R}^{P+1} \to \mathbb{R}^P$, and y_0 is a given initial value. f may depend on some parameters $\lambda \in \mathbb{R}^Q$, $Q \geq 1$, and occasionally we write $f(y;\lambda)$. We may also consider y_0 as a parameter. We let $Y \approx y$ denote an approximation to y, which in our case will be produced by a finite element method. We will see in the next chapter that, with the use of quadrature to evaluate integrals defining the finite element method, we can identify the approximations with some standard Runge Kutta schemes.

An ODE is autonomous if the right hand side does not depend explicitly on time. Since we are primarily concerned with ODEs that describe natural phenomena and the ability to predict their state at some future time, we deal with non-autonomous ODEs. While there is a standard trick is to convert a non-autonomous ODE an autonomous form by introducing a new variable equal to the time variable, we do not do this because some of the applications include time dependent parameters where values are determined experimentally and given in the form of lookup tables.

We also restrict the discussion to first order ODEs, and we observe that a higher order ODE can be reduced to a system of first order differential equations through a change of variables. For example, consider the Duffing equation

$$\begin{aligned} &y'' + \delta y' - y + y^3 = 0, \\ &y(0) = a, \\ &y'(0) = b. \end{aligned} \tag{2.1.2}$$

We can redefine variables by

$$\begin{aligned} &y_0' = y_1, \quad y_1(0) = a, \\ &y_1' = y_0 - y_0^3 - \delta y_1, \quad y_1(0) = b, \end{aligned} \tag{2.1.3}$$

resulting in the system of first order equations,

$$\dot{y} = f(y, t), \tag{2.1.4}$$

where y is a vector of dimension 2, and

$$f(y) = \begin{pmatrix} y_1 \\ y_0 - y_0^3 - \delta y_1 \end{pmatrix}. \tag{2.1.5}$$

2.1.1 Existence and uniqueness

Since we are concerned with the solution of equation (2.1.1), we must consider existence and uniqueness of the solution. Computer codes used to

solve ODEs return numbers, but these are meaningless if there is no solution. The following standard result in the analysis of differential equations is presented in many of the texts on ODEs.

Theorem 2.1.1. *If $\dot{y} = f(y,t)$ is a differential equation such that $f(y,t)$ is continuous for all (y,t) in an open region $\mathcal{R} \subset \mathbb{R}^{P+1}$ containing $(y_0, 0)$, and if there exists a constant L such that,*

$$\|f(y,t) - f(y^*,t)\| \leq L \|y - y^*\|, \tag{2.1.6}$$

for all $(y,t),(y^*,t) \in \mathcal{R}$, then there exists a time $t^* > 0$ such that there is unique continuously differentiable function $y(t)$ such that

$$\dot{y}(t) = f(y(t),t),$$

and $y(0) = y_0$, for $0 \leq t \leq t^*$.

The constant L is called the Lipschitz constant and inequality (2.1.6) is called a Lipschitz condition. Proof of this theorem can be found in many ODE texts including [5]. We note that if f is globally Lipshitz, then we have global existence. We also give the following result showing that the solution depends continuously on the initial conditions.

Theorem 2.1.2. *If the differential equation $\dot{y} = f(y,t)$ satisfies the conditions of theorem 2.1.1 and has solutions $\tilde{y}_1$ and $\tilde{y}_2$ with initial conditions $\tilde{y}_{0,1}$ and $\tilde{y}_{0,2}$, respectively, then for all $\epsilon > 0$ there exists a $\delta(\epsilon, T) > 0$ such that*

$$\|\tilde{y}_{0,1} - \tilde{y}_{0,2}\| \leq \delta \Rightarrow \|\tilde{y}_1(t) - \tilde{y}_2(t)\| \leq \epsilon$$

for all $t \in [0,T], T < \infty$.

We also call this Lyapunov stability and will discuss stability later in this chapter.

2.1.2 Convergence of a numerical method

Our primary interest is computing numerical approximations of solutions. The most fundamental issue is convergence. We consider a sequence of discretizations of the time interval $[0, T]$. The order of convergence determines how rapidly the approximation converges to the exact solution as the step size decreases. We denote a particular discretization by,

$$0 = t_0 < t_1 < t_2 < \cdots < t_N = T,$$

with time steps

$$k_n = t_n - t_{n-1}, \quad 1 \leq n \leq N.$$

To avoid complicated notation, we avoid distinguishing different discretizations unless absolutely necessary. We assume, however, that the discretizations are increasingly fine in the sense that the maximum timesteps converge to zero. We abuse notation so say that $N \to \infty$, by which we mean that we consider the behavior of the limit of increasingly fine sequences of meshes. To each discretization of $[0, T]$, we associate a sequence of approximate solution values

$$\{Y_n\}_{n=1}^{N},$$

where $Y_n \approx y(t_n)$, $n = 1, \ldots, N$. We define convergence as follows.

Definition 2.1.1. *The method is said to converge if for all initial value problems satisfying theorem 2.1.1, we have*

$$\max_{0 \leq n \leq N} \|y(t_n) - Y_n\| \to 0, \text{ as } N \to \infty.$$

We also say, if $\max_n |y(t_n) - Y_n| = \mathcal{O}(k^{s+1})$, then the method is consistent with an order of accuracy $s + 1$. Convergence implies that we can reach any desired degree of accuracy by reducing step sizes. The following theorem provides conditions for the convergence of a one-step method.

Theorem 2.1.3. *Consider Φ a one-step method defined by*

$$y(t_{n+1}) = y(t_n) + k\Phi(k, t_n, y_n, y_{n+1}).$$

Suppose Φ is Lipschitz continuous in y_n and y_{n+1}, uniformly in k and t, and is consistent. Then Φ is convergent.

Proof of this theorem is provided in [38]. Figure 2.1 shows the convergence of the solution of $\dot{y} = -y$ on $[0, 2.0]$ with decreasing step sizes.

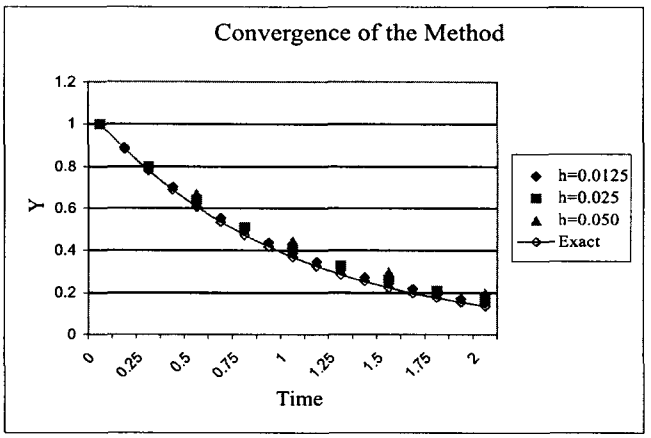


Figure 2.1: Convergence to the exact solution by using uniform step sizes of 0.5, 0.25, and 0.125.

A classic way to estimate the potential order of convergence is by comparing the Taylor expansion of the exact solution to the expansion after a single step of the chosen method. For example, consider Euler's method. The Taylor expansion gives

$$y(x_n) = y(x_{n-1}) + ky'(x_{n-1}) + \frac{k^2}{2!}y''(x_{n-1}) + \cdots, \tag{2.1.7}$$

while the approximation gives

$$Y_n = Y_{n-1} + kf(x_{n-1}). \tag{2.1.8}$$

Subtracting equation (2.1.8) from equation (2.1.7) gives $y(x_n) - Y_n = O(k^2)$, provided both started with the same value $Y(t_{n-1})$. Of course, they don't. Next we have to make a Gronwall argument in order to estimate the cumulative effect of errors over many steps [5]. This is the crudest way to account for the possible accumulation of error. A Gronwall argument costs one power of the time step, hence Euler's method converges with first order accuracy. In general, the order of convergence is one less than the order of the error term of the Taylor expansion. In using finite element methods to generate approximate solutions, the *a priori* convergence analysis takes a different form [18], but the conclusions are the same.

2.1.3 Stability

The stability of a differential equation indicates the effects on the solution of small changes in the initial conditions and parameters, which we call data. In the crudest notion of stability, stable problems have the property that small perturbations to the data yield small changes in the final solution for time in some given range. However, there are many ways to quantify "small", and this notion also depends heavily on the size of the given time range. Consider the function

$$y = y_0 e^{kt}, \tag{2.1.9}$$

where y_0 is the initial number and k is a growth ($k > 0$) or decay ($k < 0$) constant. If the true solution to a problem has $k = 1$ and $y_0 = 1.0$, then by perturbing the initial value by .5 in either direction, the error in the solution is 1.3591 at time 1.0 and 3.6945 at time 2.0, or an increase by a factor of 2.7183 over the time interval 1.0 to 2.0 (see Fig. 2.2). On the other hand, if we let $k = -1$, the distance between solutions corresponding to the same

perturbations decreases over time (see Fig. 2.3). Both of these are stable in the sense that they have a continuous dependence on the data; however, the first is dynamically unstable since the perturbations grow with time.

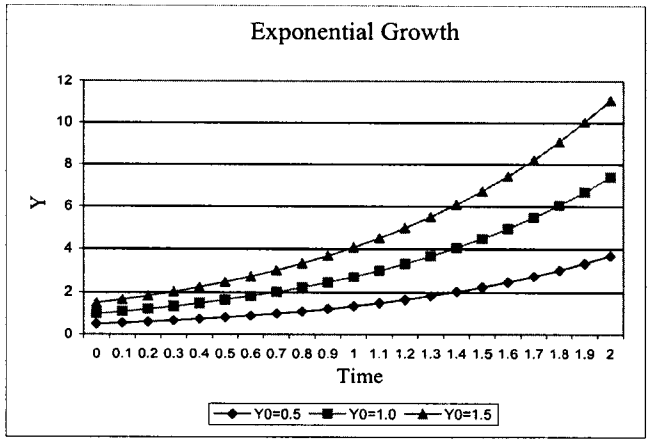


Figure 2.2: The solution $y = y_0 e^{kt}$ over $[0, 2]$ with perturbations to y_0.

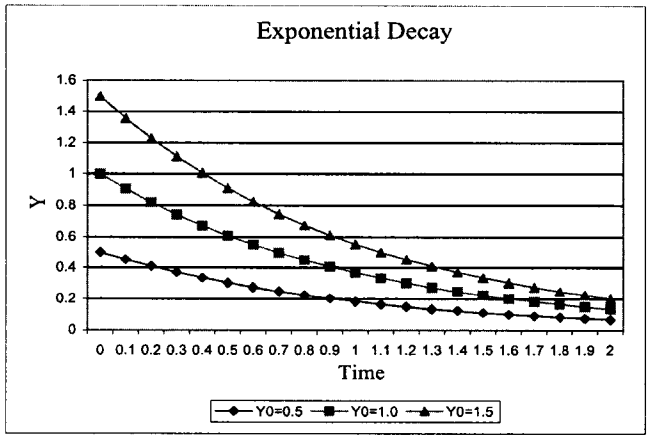


Figure 2.3: The solution $y = y_0 e^{-kt}$ over $[0, 2]$ with perturbations to y_0.

Now, most problems are more complex than a simple exponential, and in general, solutions have regions of both stability and instability. Consider the following differential equation.

$$\begin{cases} \dot{y} + (0.25 + \sin(\pi t))y^2 = 0, \\ y(0) = y_0. \end{cases} \tag{2.1.10}$$

We plot the solution and errors in Fig. 2.4. The regions of stability and instability are indicated by the accumulation and cancellation of the numerical errors. This is seen in Fig. 2.4 as growth and decay in the error graph. Typically, *a priori* bounds used for convergence analysis consider accumulation only and, consequently, tremendously overestimate the error. It is precisely this cancellation of error that motivates the new adaptive strategies we discuss later.

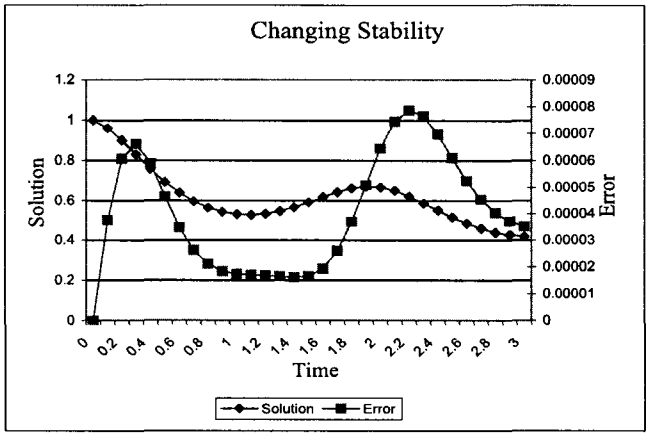


Figure 2.4: A problem with regions of stability and instability.

2.1.4 Stiffness

A classic concern in the context of adaptive error control is stiffness. The classic notion of stiffness arose in chemical kinetics, which often has solutions where components decay at dramatically different rates. If one component decays rapidly while others vary slowly, the stability of the system might affect the mesh size that can be used depending on the numerical method. For example, consider the equations,

$$\begin{aligned} \dot{y}_1 &= -0.01y_1 - 0.99y_2 + 0.99y_3, \\ \dot{y}_2 &= -y_2 - 99y_3, \\ \dot{y}_3 &= -100.0y_3, \end{aligned} \tag{2.1.11}$$

with initial conditions $y(0) = [2, 2, 1]^T$. Figure 2.5 shows the solution on the interval $[0, 1]$. While component 1 decays slowly, both components 2 and 3 decay rapidly over $[0, 0.05]$. Consequently, components 2 and 3 dictate a small step size to solve the system accurately. If we used an explicit method

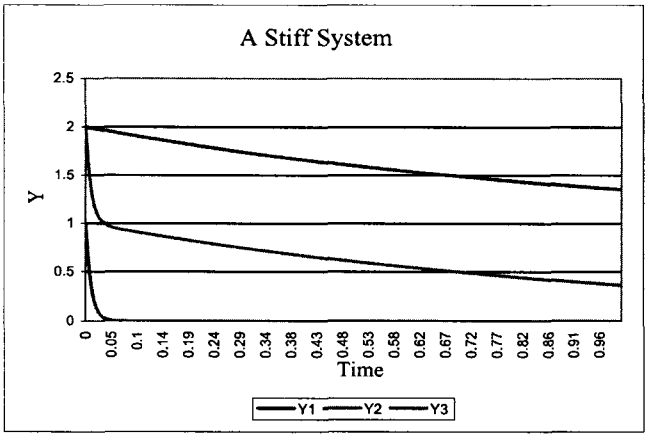


Figure 2.5: Stiff system of equations.

for such a problem, the stiffness would force the use of very small time steps, hence inefficient computations. Use of certain implicit methods could yield the benefit that large steps could be used while still maintaining accuracy over a long time interval. If we require accurate transient solutions, we might still need to use small steps during the initial transient.

In general, stiffness is often thought of as having a few components of a solution vary rapidly. The issue is capturing this transient behavior to a sufficient degree to maintain accuracy. In other words, there is an issue of reliability, in maintaining sufficient accuracy, and inefficiency, in terms of not achieving accuracy by using unnecessarily small timesteps.

Chapter 3

NUMERICAL METHODS

3.1 Galerkin Finite Element Methods

A finite element method is based on the variational formulation of the differential equation. Applying this formulation to equation (2.1.1), the problem is to find $y \in \mathcal{C}^1([0,1])$ such that,

$$\begin{cases} \int_0^T (\dot{y}, v)dt = \int_0^T (f(y,t), v)dt, \\ y(0) = y_0, \end{cases} \tag{3.1.1}$$

for all $v \in \mathcal{C}^1([0,1])$, where $(,)$ represents the $\mathbb{R}^P$ Euclidean inner product.

We use two methods based on piecewise polynomials, using $\mathcal{P}^q(t_{j-1}, t_j)$ to denote the space of polynomials of degree q and less on $I_j = (t_{j-1}, t_j)$. In each method, we define the trial space as the space where we seek to find a solution, and the test space as a set of functions used to define the approximation. As such, we discretize $[0,T]$ into intervals such that,

$$0 = t_0 < t_1 < t_2 < \cdots < t_n = T, \tag{3.1.2}$$

and compute an approximation $Y \in \mathcal{P}^q(t_{j-1}, t_j)$ successively over each interval $I_j = (t_{j-1}, t_j)$. Thus, we let

$$\mathcal{V}^{(q)} = \left\{ U : U|_{I_m} \in \mathcal{P}^{(q)}(I_m), m = 1, 2, 3, \ldots \right\},$$

and we seek $Y \in \mathcal{V}^{(q)}$.

3.1.1 The discontinuous Galerkin method (dG(q))

To solve the forward problems, we use the dG(q) method which seeks a numerical solution in a space of vector valued piecewise polynomials of degree q. We define $Y_{j-1}^{+} = \lim_{t \downarrow t_{j-1}} Y(t)$, $Y_{j-1}^{-} = \lim_{t \uparrow t_{j-1}} Y(t)$, and $[Y]_{j-1} = Y_{j-1}^{+} - Y_{j-1}^{-}$. The method is find $Y \in \mathcal{P}^q(t_{j-1}, t_j)$ such that

$$\begin{cases} \int_{t_{n-1}}^{t_n} (\dot{Y}, v)dt + ([Y]_{n-1}, v_{n-1}^{+}) = \int_{t_{n-1}}^{t_n} (f(Y,t), v)dt, \\ Y_0 = y_0, \end{cases} \tag{3.1.3}$$

for all $v \in \mathcal{P}^q(t_{n-1}, t_n)$, $n = 1, 2, \ldots, N$, where N is the total number of time steps. Note that the term $[Y]_{n-1}$ on the left hand side arises from the fact that we are integrating a discontinuous function. The derivative of Y at the node can be evaluated from left and right, thus we have the equivalent of a delta function of magnitude $[Y]$.

A system for the coefficients of Y results from the fact that we can solve for all $v \in \{p_m\}_{m=1}^{q}$ where $\{p_m\}_{m=1}^{q}$ are the basis functions for polynomials of degree q. For example, consider $q = 0$. The test space is the space of piecewise constants and has the constant function 1 as a basis. Since any $v \in \mathcal{P}^q(t_{n-1}, t_n)$ can be expressed as a linear combination of the basis functions, it suffices to test for all basis functions, v, and the method simplifies to find $Y \in \mathcal{P}^0(0, T)$ such that

$$\begin{cases} \int_{t_{n-1}}^{t_n} \dot{Y} dt + [Y]_{n-1} = \int_{t_{n-1}}^{t_n} f(Y,t)dt, \quad n = 1, 2, \ldots, N, \\ Y_0 = y_0. \end{cases} \tag{3.1.4}$$

When $Y \in \mathbb{R}^P$ we interpret this coefficient by coefficient.

We generally use quadrature to evaluate the integral on the right. For example, if we use

$$\begin{aligned} \int_{t_{n-1}}^{t_n} f(w(t), t)dt &\approx w(t_n)k_n, \\ \int_{t_{n-1}}^{t_n} f(w(t), t)dt &\approx w(t_n)k_n, \end{aligned} \tag{3.1.5}$$

to approximate the integral in (3.1.4) with $q = 0$, we get an equation for the node value $Y_n^- = Y(t_n^-)$ which is the same as the backward Euler difference scheme. The approximation for Y is given by

$$Y_n^- = Y_{n-1}^- + \int_{I_n} f(Y(t), t)dt. \tag{3.1.6}$$

Substituting $f(Y_n^-, t_n)k_n$ for $\int_{I_n} = f(Y(t), t)dt$ gives

$$Y_n^- = Y_{n-1}^- + f(Y_n^-, t_n)k_n. \tag{3.1.7}$$

The dG(q) family of methods are particularly well suited for dissipative problems, see Estep and Stuart [14].

3.1.2 The continuous Galerkin method (cG(q))

To solve the adjoint problem, we use the cG(q) methods which seek a numerical solution in a space of continuous, piecewise, vector-valued polynomials. We require the numerical solution to inherit the exact value from the previous interval at the common node, so we have one less degree of freedom to solve the ODE relative to the dG(q). The method is, for $1 \leq j \leq m$, find $Y \in \mathcal{P}^q(t_{j-1}, t_j)$ such that

$$\begin{cases} \int_{t_{n-1}}^{t_n} (\dot{Y}, v)dt = \int_{t_{n-1}}^{t_n} (f(Y, t), v)dt, \\ Y_{n-1}^+ = Y_{n-1}^- \end{cases} \tag{3.1.8}$$

for all $v \in \mathcal{P}^{q-1}(t_{n-1}, t_n)$, with $Y_0^- = y_0$. As with the dG scheme, we often use a quadrature formula. We have also implemented the popular Verlet scheme for Hamiltonian problems as a special application. This can be viewed as a cG scheme with a special quadrature. The cG(q) family of methods is well suited for problems with conserved quantites of energy, see Estep and French [18].

3.1.3 Implementing the dG(q) method

For most problems, GAASP uses the discontinuous Galerkin method for the forward solves. Recall that we compute $Y \in \mathcal{P}^q(I_n)$ such that

$$\int_{I_n} (\dot{Y}, v)dt + ([Y]_{n-1}, v_{n-1}^+) = \int_{I_n} (f(Y(t), t), v(t))dt, \tag{3.1.9}$$

for all $v \in \mathcal{P}^q(I_n)$. We set

$$Y = a_{n,1}p_1(t) + a_{n,2}p_2(t) + \cdots + a_{n,q+1}p_{q+1}(t), \tag{3.1.10}$$

where $\{p_1, \ldots, p_{q+1}\}$ is a basis for $\mathcal{P}^q(I)$. It suffices to test equation (3.1.9) for all $v \in \{p_1, \ldots, p_{q+1}\}$. Since we solve for Y in the space of discontinuous polynomials, we let $Y_{j-1}^+ = \lim_{t \downarrow t_{j-1}} Y(t)$, $Y_{j-1}^- = \lim_{t \uparrow t_{j-1}} Y(t)$, and $[Y]_{j-1} = Y_{j-1}^+ - Y_{j-1}^-$.

3.1.4 Quadrature and interpretation as Runge-Kutta schemes

When implementing equation (3.1.9) for general use, we approximate the integrals using a quadrature rule. It may not be possible to carry out the integrals in closed form, and even if it is possible, it is not desirable to force a user to compute these. Using a sufficiently high order quadrature rule does not effect the overall accuracy. As a baseline, we choose the quadrature rule to be on the same order of accuracy as the chosen numerical method. In practice, therefore, we solve

$$\int_{I_n} (\dot{Y}, v)dt + ([Y]_{n-1}, v_{n-1}^+) = \int_{I_n} (\overline{f(Y(t), t)}, v(t))dt, \tag{3.1.11}$$

where $\overline{f(Y(t), t)}$ is a polynomial constructed as

$$\overline{f(Y(t), t)} = f(Y(\tau_1), \tau_1)r_1(t) + f(Y(\tau_2), \tau_2)r_2(t) + \cdots + f(Y(\tau_M), \tau_M)r_M(t), \tag{3.1.12}$$

$\{\tau_1, \ldots, \tau_M\}$ are the quadrature points and $\{r_1(t), \ldots, r_M(t)\}$ are the basis functions for the chosen quadrature. Thus, equation (3.1.3) becomes

$$\int_{t_{n-1}}^{t_n} (\dot{Y}, v)dt + ([Y]_{n-1}, v_{n-1}^+) = \sum_{m=1}^{M_n} \omega_{m,n} f(Y(\tau_{m,n}), \tag{3.1.13}$$

where M_n indicates the number of quadrature points, $\{\tau_{n,m}\}_{m=1}^{M_n} \in (t_{n-1}, t_n)$ are the quadrature points, and $\{\omega_{n,m}\}_{m=1}^{M_n}$ are the quadrature weights at the m^{th} point. Often, the M_n are all equal, M, and $\{\tau_{n,m}\}$ are obtained by shift and scaling, in which case we write $\{\tau_{n,m}\} = \{\tau_m\}$ and $\{\omega_{n,m}\} = \{\omega_m\}$. We use the rectangle rule for the dG(0) method and two point Radau quadrature for the dG(1) method. The two point Radau quadrature leads directly to two stage, order three Runge-Kutta method for the node values. These methods were first introduced by Butcher [6] in 1964 and are discussed in detail in [5], [44], and [24]. These rules were chosen to preserve some special dissipativity properties of the dG method, see [14]. For example, consider the quadrature based on the two Radau points,

$$\tau_{m,0} = t_{m-1} + \frac{k_m}{3} \quad \text{and} \quad \tau_{m,1} = t_m,$$

with weights $\frac{3}{4}$ and $\frac{1}{4}$ respectively. This results in the following equation to step the ODE forward one time step.

$$Y_m = Y_{m-1} + \frac{3}{4} k_m f(\tilde{Y}_{m,0}, t_{m-1} + k_m/3) + \frac{1}{4} k_m f(\tilde{Y}_{m,1}, t_m). \tag{3.1.14}$$

To solve the equation, we need to evaluate the approximation at the Radau points. Applying the quadrature scaled for the location of the Radau points gives the following equations:

$$\begin{aligned} \tilde{Y}_{m,0} &= Y_{m-1} + \frac{5}{12} k_m f(\tilde{Y}_{m,0}, t_{m-1} + k_m/3) - \frac{1}{12} k_m f(\tilde{Y}_{m,1}, t_m), \\ \tilde{Y}_{m,1} &= Y_{m-1} + \frac{3}{4} k_m f(\tilde{Y}_{m,0}, t_{m-1} + k_m/3) + \frac{1}{4} k_m f(\tilde{Y}_{m,1}, t_m), \end{aligned} \tag{3.1.15}$$

where $\tilde{Y}_{m,0}$ and $\tilde{Y}_{m,1}$ are temporary intermediate values. The set of equations, (3.1.14) and (3.1.15) yield a 2-stage implicit Runge-Kutta scheme for the node values.

In each case, we solve the adjoint with a method 1 order higher than the forward solve because of the form of the adjoint weight in the error estimate, which includes the difference of the adjoint solution and a projection in the finite element space of the forward approximation. We use a cG(q+1) method for the adjoint because of the natural pairing in accuracy. The cG(q) method is actually a Petrov-Galerkin method, since the test space is one order less than the solution space.

From Estep [13], we expect convergence results for pointwise error for the dG(q) methods of

$$\|e\|_{L^\infty(0,t_n)} \leq C \max_{j \leq n} k_j^{(q+1)} \left\|y^{(q+1)}\right\|_{L^\infty(I_j)}, \quad n = 1, 2, \ldots, N, \tag{3.1.16}$$

while at nodes, there is a super convergence result under the right conditions,

$$\|e(t_n)\| \leq C \max_{j \leq n} k_j^{(2q+1)} \left\|y^{(2q+1)}\right\|_{L^\infty(I_j)}, \quad n = 1, 2, \ldots, N. \tag{3.1.17}$$

There are similar results for the cG(q) method [18], except the order of convergence at the nodes is reduced by 1,

$$\|e\|_{L^\infty(0,t_n)} \leq C \max_{j \leq n} k_j^{(q+1)} \left\|y^{(q+1)}\right\|_{L^\infty(I_j)}, \quad n = 1, 2, \ldots, N, \tag{3.1.18}$$

$$\|e(t_n)\| \leq C \max_{j \leq n} k_j^{(2q)} \left\|y^{(2q)}\right\|_{L^\infty(I_j)}, \quad n = 1, 2, \ldots, N. \tag{3.1.19}$$

Chapter 4

CLASSIC NUMERICAL ERROR ESTIMATION

In an *initial value problem*, we know the condition of the state variables at a specific point in time. Starting at this point in time, we use an appropriate solution method to solve the problem forward in time to observe how the state variables change with time. We may be interested in the state at a specific point in the future, or we may want to observe trends over time. Either way, we solve the equations by marching forward in time, solving for each time step in sequential order.

We can think of estimating the error in two different senses; *a priori* bounds and *a posteriori* estimates.

A priori bounds give a general idea of how large the error can be for a specific method applied to a general solution of a general problem. The main use of *a priori* bounds is determining general convergence properties such as order. Since an *a priori* bound does not depend on the computed solution, it can be calculated before we compute the numerical solution; however, *a priori* bounds tend to be very pessimistic. In addition, many *a priori* error bounds are based on asymptotic behavior in the limit as the step size approaches zero. *A posteriori* estimates, on the other hand, require the computed solution, but give a estimate of the error based on that particular

solution. While *a priori* are useful for choosing the numerical method to use for a given problem, *a posteriori* estimates provide the information needed for adaptive algorithms.

A priori analysis of the dG and cG schemes presented in chapter 3 provide information on the convergence properties of the schemes. In [13], Estep gives the optimal order error bound,

$$|Y(t_n) - y(t_n)| \leq \tilde{S}(t_n) \max_{m \leq n} k_m^j \max_{[t_{m-1},t_m]} \left| \frac{d^j y}{dt^j} \right|, \quad 1 \leq j \leq q+1. \tag{4.0.1}$$

$\tilde{S}(t_n)$ is a stability factor that measures the accumulation of error and can be bound by

$$\tilde{S}(t_n) \leq \frac{e^{t_n L} - 1}{L}, \tag{4.0.2}$$

where $L \neq 0$ is a Lipschitz constant. We note that the bound depends on the derivative values of the unknown solution y and that a bound on $\tilde{S}(t_n)$ must account for the worst rate in error growth resulting in an overestimate of the error. Similar results in [13] show the cG(q) method is superconvergent at time nodes for q=2.

In this thesis, we use the approach of *a posteriori* error estimates based on adjoint methods. This approach has been used extensively by Estep, Johnson, Larson, Williams, Stuart, Hansbo, French, Holst, and Mikulencak since 1994. In [13], Estep analyzes the approach for solving ordinary differential equations using the dG method. *A posteriori* analysis gives an error bound of

$$|Y(t_n) - y(t_n)| \leq S(t_n) \max_{m \leq n} k_m^j \max_{[t_{m-1},t_m]} \left| D_t^j Y(t_m) \right|, \quad 1 \leq j \leq q+1. \tag{4.0.3}$$

Comparing this to equation (4.0.1), we now have a bound that is not dependent on the derivative values of the unknown function y, but rather,

on the compute approximation, Y. The ability to compute Y allows the right-hand side to be kept at a given tolerance which also allows for robust error control. Similar findings were presented by Estep and French in [18] for the cG method. In 1998, Estep and Johnson applied this method to the Lorenz equations in [15] to compute pointwise accurate approximations over trajectories of moderate length and in the 2000 memoir [19], Estep, Larson, and Williams presented an in-depth analysis of error estimation using adjoint methods to reaction-diffusion equations. In [8], Cao and Petzold combined adjoint methods with small sample statistical methods to provide estimates and recent research by Neckels [35] and Eastman [11] provide insight into using adjoint methods for parameter sensitivity and the effects of linearization, respectively.

4.1 The Classic View of Numerical Error

To compute the solution of a differential equation numerically, we create the partition of $[0,T]$,

$$\begin{aligned} &0 = t_0 < t_1 < t_2 < \cdots < t_N = T, \\ &k_n = t_n - t_{n-1}, \end{aligned} \tag{4.1.1}$$

and compute the numerical solution, Y, on each interval successively. The classic way to understand the error is to consider that on each interval, we approximately solve the new equation for the time step,

$$\begin{cases} \dot{\tilde{y}} = f(\tilde{y},t), \quad t_{n-1} < t \leq t_n, \\ \tilde{y}(t_{n-1}^+) = Y_{n-1}^-, \end{cases} \tag{4.1.2}$$

using some numerical scheme, where

$$t_n^+ = \lim_{t \downarrow t_n} t \quad \text{and} \quad t_n^- = \lim_{t \uparrow t_n} t. \tag{4.1.3}$$

In other words, the equation on the new time step has an initial condition inherited from the approximate solution of the previous time step. The error between $\tilde{y}$ and Y,

$$Y(t_n) - \tilde{y}(t_n),$$

is called the "local error". It is not the real error, because the true solution is not likely to have the value Y_{n-1}^- at t_{n-1}. The true or global error is defined

$$Y(t_n) - y_n. \tag{4.1.4}$$

Now we consider the effect of solving each local problem with the wrong data. Assume $Y_{n-1}^- - y^-(t_{n-1}) = \delta$, so (4.1.2) is

$$\begin{cases} \dot{\tilde{y}} = f(\tilde{y}, t), \quad t_{n-1} < t \leq t_n, \\ \tilde{y}(t_{n-1}) = y^-(t_{n-1}) + \delta. \end{cases} \tag{4.1.5}$$

Recall that small perturbations can result in very large errors. Figure 4.1 illustrates the relationship between local and global error. If we consider a numerical solution to $y = 0.2e^t$ with a time step of one as being represented by the +s on the graph, we see the local error as the difference between the exact and approximate solutions of the local problem, equation (2.1.9), while the global error is the difference between the approximate solution of the local problem and the exact solution of the original ODE.

However, we also recall that most ODEs have regions of both stability and instability. Consider the logistics problem,

$$\begin{cases} \dot{y} = ay - by^2, \\ y(0) = y_0, \end{cases} \tag{4.1.6}$$

with $a = b = 2.309$ and $y_0 = 0.1$. Figure 4.2 shows both the exact and approximate solutions, as well as the global error. The region from $t = 0.0$ to $t = 1.0$ is a region of dynamic instability and we see growth in the global error, while the region after $t = 1.0$ is dynamically stable, consequently the global error tends towards 0.

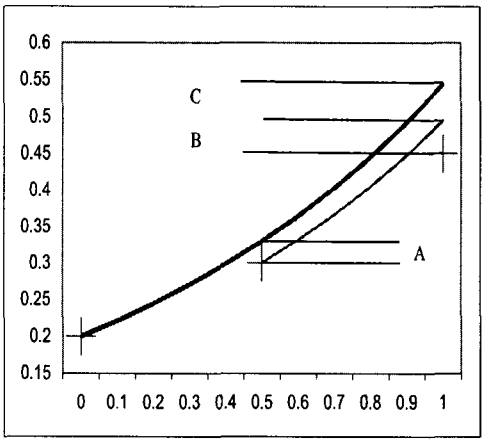


Figure 4.1: The full line shows the solution to $\dot{y} = y$ with initial condition $y_0 = 0.2$ while the partial line shows the solution to $\dot{\tilde{y}} = \tilde{y}$ with initial condition $\tilde{y}_0 = Y_1$. The points represent an approximation from a forward Euler method. The local error for the first time step is labeled A, the local error for the second time step is labeled B, and the global error is the total of B plus C.

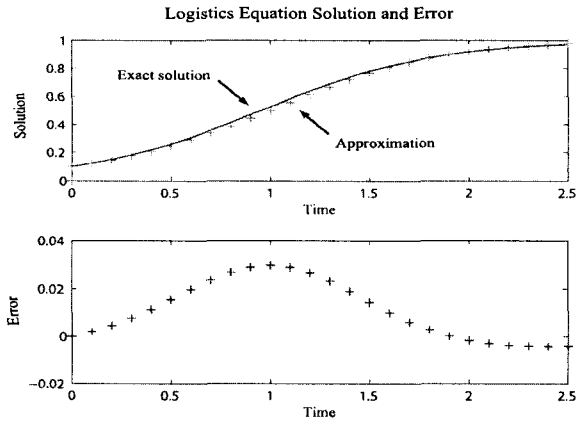


Figure 4.2: The logistics equation shows regions of dynamic stability and instability, consequently the global error can grow and shrink.

4.2 Classic Error Estimation Techniques

Many classic ODE packages implement an "error control" based on one of two approaches, Richardson extrapolation and predictor-corrector methods. In both cases, the error is estimated by comparing solutions of different accuracies; however, each uses a different approach to compute solutions of different accuracy in a fast way.

4.2.1 Producing improved accuracy using Richardson extrapolation

Richardson extrapolation is a well known technique that provides a way to modify an existing method to gain greater accuracy. However, as Skeel [40] points out, this technique is dependent on the existence of an asymtotic expansion and is also computationally expensive. Consider the following two Taylor series of $f(y,t)$ from equation (2.1.1),

$$f(y,t+k) = \sum_{m=0}^{\infty} \frac{1}{m!} k^m f^{(m)}(y,t), \tag{4.2.1}$$

and

$$f(y,t-k) = \sum_{m=0}^{\infty} \frac{1}{m!} (-1)^m k^m f^{(m)}(y,t). \tag{4.2.2}$$

By subtracting equation (4.2.2) from equation (4.2.1) and rearranging, we have

$$f'(y,t) = \frac{1}{2k}\left[f(y,t+k) - f(y,t-k)\right] - \left[\sum_{m=1}^{\infty} \frac{1}{(2m+1)!} k^{2m} f^{(2m+1)}(y,t)\right]. \tag{4.2.3}$$

We simplify the equation by writing

$$F = \varphi(k) + a_2 k^2 + a_4 k^4 + \cdots, \tag{4.2.4}$$

where

$$\begin{aligned} F &= f'(y,t), \\ \varphi(k) &= \frac{1}{2k}\left[f(y,t+k) - f(y,t-k)\right], \quad \text{and} \\ a_m &= \frac{1}{(m+1)!} f^{(m+1)}(y,t). \end{aligned} \tag{4.2.5}$$

We seek to make the error,

$$F - \varphi(k) = \sum_{m=1}^{\infty} a_{2m} k^{2m}, \tag{4.2.6}$$

smaller. Note that, for h very small, the a_2 term dominates the error. If we can eliminate this term, we can increase the accuracy of the computation. Substituting $\frac{h}{2}$ into equation (4.2.4), multiplying by 4, and subtracting from equation (4.2.4) yields

$$F = \frac{4}{3}\varphi(k/2) - \frac{1}{3}\varphi(k) - \frac{1}{4}a_4 k^4 - \frac{5}{16}a_6 k^6 - \cdots. \tag{4.2.7}$$

We see that, by combining function evaluations with k and $k/2$, we now have an estimate of F that is order $\mathcal{O}(k^4)$.

We can apply the process again to gain even more accuracy at the cost of more function evaluations.

With the improved solution, the question is how to estimate the error. In practice, the error is estimated using a difference between the higher and lower order solutions. Let F be the approximation and $\tilde{F}$ be the approximation using the extrapolant. The error estimate, $e = F - \tilde{F}$, is

$$\begin{aligned} e &= \varphi(k) + a_2 k^2 + a_4 k^4 + \cdots \\ &\quad - \frac{4}{3}\varphi(k/2) + \frac{1}{3}\varphi(k) + \frac{1}{4}a_4 k^4 + \cdots, \\ &= \frac{4}{3}\left[\varphi(k) - \varphi(k/2)\right] + (O)(k^2). \end{aligned} \tag{4.2.8}$$

In classic methods for error control, the solution is computed for a time step and the error is estimated using the technique described above. If

the estimate is greater than the global tolerance divided by the number of intervals, that interval is refined and a new solution computed. This procedure is repeated until the tolerance is reached. Subsequent steps are then computed in the same way.

4.2.2 Producing improved accuracy using predictor-corrector methods

Another approach to estimating error is the use of predictor-corrector methods. These methods use the idea of multistep methods. Single step methods are distinguished by the fact that they compute the value at the new position based only on the value at the previous position. Taylor series methods and Runge-Kutta methods both fall into the category of single step methods. In order to take advantage of earlier information, we can also use what are known as multistep methods. Multistep methods, in general, take the following equation

$$a_m Y_n + a_{m-1} Y_{n-1} + \cdots + a_0 Y_{n-m} = k(b_m f_n + b_{m-1} f_{n-1} + \cdots + b_0 f_{n-m}) \quad (4.2.9)$$

where $a_0, a_1, \cdots, a_m$ and $b_0, b_1, \cdots, b_m$ are constants, and f_i is the function f evaluated at y_i, t_i. $b_m = 0$ yields an explicit equation and $b_m \neq 0$ gives an implicit equation. A detailed look at multistep methods can be found in most texts on numerical analysis, including [2] and [28]. Her we look at how multistep are use in predictor-corrector methods.

In predictor-corrector methods, an explicit method, the "predictor", is used to compute the next solution value. An implicit method, which is both more stable and accurate, is then used to improve on the prediction. Since the predictor is providing the value for the new step, the computational expense of the implicit method is avoided. These methods commonly use

multistep methods, but can also be implemented using single step methods. For example, consider the trapezoidal method

$$Y_{n+1} = Y_n + \frac{k}{2}\left[f(Y_n, t_n) + f(Y_{n+1}, t_{n+1})\right], \quad n \geq 0. \tag{4.2.10}$$

This is an example of a one step method and, since Y_{n+1} occurs on both sides of the equation, is an implicit method. The value for Y_{n+1} on the right hand side is provided by solving an explicit method, for example, Euler's method,

$$Y_{n+1} = Y_n + kf(Y_n, t_n). \tag{4.2.11}$$

Thus, equation (4.2.11) predicts the value for Y_{n+1}, and equation (4.2.10) corrects the value. The local error is estimated using the difference between the two equations. Note that the above example is used for it's simplicity in explanation, but is generally not used in practice because of it's low order of convergence.

As with Richardson's extrapolation, this method relies on the asymptotic behavior of the solution. Since *a priori* we do not have a test to determine an adequately small time step, the reliability of the solution may be uncertain.

Again, we estimate the error of the method by taking the difference between the two solutions. Let $\tilde{Y}$ be the approximation from the predictor, and Y be the approximation from the corrector. Since we compute both the predictor and corrector on each time step, the local truncation error, $e = \tilde{Y}_{n+1} - Y_{n+1}$, is given by

$$\begin{aligned} e &= kf(Y_n, t_n) - \frac{k}{2}\left[f(Y_n, t_n) + f(\tilde{Y}_{n+1}, t_{n+1})\right], \\ &= \frac{k}{2}f(Y_n, t_n) - \frac{k}{2}f(\tilde{Y}_{n+1}, t_{n+1}), \\ &= \frac{k}{2}\left[f(Y_n, t_n) - f(\tilde{Y}_{n+1}, t_{n+1})\right]. \end{aligned} \tag{4.2.12}$$

Chapter 5

VARIATIONAL ANALYSIS USING DUALITY AND THE LINEARIZED ADJOINT

5.1 Background

Over the past decade, advances have been made in using the adjoint problem as a means of computing *a posteriori* error estimates for differential equations [8, 12, 13, 16, 33]. Since the adjoint plays a large role in the computation of the error estimates, it is important to understand the details involved. In this chapter we discuss the foundations of the adjoint. We begin with definitions, explain the error analysis in the context of linear algebra, and then discuss the application to ordinary differential equations.

5.2 Definitions

The following definitions provide a basis for the discussion of variational analysis using duality and the adjoint.

Definition 5.2.1. *A sequence* $\{x_n\}$ *in* X *is a* ***Cauchy Sequence*** *if for every* $\epsilon > 0$ *there exists an* N *such that* $\|x_n - x_m\| < \epsilon$ *for all* $n, m > N$.

Definition 5.2.2. *A* ***Banach Space*** *is a normed vector space where every Cauchy sequence in the space converges to a limit also in the space.*

Definition 5.2.3. *$\mathcal{L}(X, Y)$ is defined as the set of all linear transformations from X to Y.*

Definition 5.2.4. *If X is a normed vector space, the space $X^* = \mathcal{L}(X, \mathbb{R})$ is the* ***dual space*** *of X. We may also say that X^* is the space of all bounded linear functionals with norm*

$$\|y\|_{X^*} = \sup_{x \in X, \|x\|=1} |y(x)| = \sup_{x \in X, x \neq 0} \frac{|y(x)|}{\|x\|_X}, \quad y \in X^*.$$

Definition 5.2.5. *An* inner product *on a vector space V, denoted by $<,>$, is a fuction $f : (u, v) \rightarrow \mathbf{F}$ and has the properties:*

- $< v, v > \geq 0$ *for all* $v \in V$,
- $< v, v >= 0$ *if and only if* $v = 0$,
- $< u + v, w >=< u, w > + < v, w >$ *for all* $u, v, w \in V$,
- $< av, w >= a < v, w >$ *for all* $a \in \mathbf{F}$ *and all* $v, w \in V$,
- $< v, w >= \overline{< w, v >}$ *for all* $v, w \in V$.

Definition 5.2.6. *A* ***linear map*** *from X to Y is a function $T : X \rightarrow Y$ with the following properties:*

$$T(x_1 + x_2) = Tx_1 + Tx_2, \quad \forall x_1, x_2 \in X,$$

$$T(\alpha x) = \alpha T(x), \quad \forall \alpha \in \boldsymbol{F}, \quad \forall x \in X,$$

where X and Y are normed vector spaces.

While $\mathbf{F}$ can be any field, we restrict the discussion to the reals.

Definition 5.2.7. *A* ***linear functional*** *is a linear map from a vector space X to $\mathbb{R}$. If the linear functional is bounded, it is a* ***continuous linear functional.***

Definition 5.2.8. *For $T \in \mathcal{L}(X,Y)$, the adjoint T^* is the function from Y to X with the property that, for a fixed $y \in Y$,*

$$< Tx, y >=< x, T^*y > \tag{5.2.1}$$

for all $x \in X$.

Note that we can define the linear map T by it's matrix A and that, for a real-valued matrix A, the adjoint is the transpose of A, or A^T.

Definition 5.2.9. *Let X and Y be normed vector spaces and $L \in \mathcal{L}(X,Y)$. $L^* \in \mathcal{L}(Y^*, X^*)$ is the unique linear transformation satisfying*

$$y^*(L(x)) = (L^*y^*)(x), \quad \forall x \in X.$$

L^* is called the adjoint or dual of L.

We can also express (5.2.9) using bracket notation by

$$< L(x), y^* >=< x, L^*(y^*) >, \quad x \in X, \quad y^* \in Y^*, \tag{5.2.2}$$

and we call this the bilinear identity.

5.3 Error Estimates Using the Adjoint

The approach to *a posteriori* error analysis is easily explained in the context of solving linear systems of equations. Consider a system of linear algebraic equations

$$f(x) = b, \tag{5.3.1}$$

where f, x, and b are all in $\mathbb{R}^N$. We wish to estimate the error in the numerical approximation, $X \approx x$, to the system of equations represented by

$$Ax = b. \tag{5.3.2}$$

We can compute the residual R as

$$R = AX - b \tag{5.3.3}$$

which indicates how well the approximation satisfies the original equation. We would like to know the actual error, $e = x - X$. Since we don't know x, we estimate the projection of e in the direction of the dual data, ψ, provided by the dual problem

$$A^{\mathsf{T}}\varphi = \psi \tag{5.3.4}$$

with a variational argument. Then we have

$$|(e,\psi)| = \left|(e, A^{\mathsf{T}}\varphi)\right| = |(Ae,\varphi)| = |(R,\varphi)|\,. \tag{5.3.5}$$

The Cauchy-Schwarz Inequality gives

$$|(R,\varphi)| \leq \|R\|\,\|\varphi\|$$

consequently,

$$|(e,\psi)| = \left|(e, A^{\mathsf{T}}\varphi)\right| = |(Ae,\varphi)| \leq \|R\|\,\|\varphi\|\,. \tag{5.3.6}$$

The choice of ψ is determined by the quantity of interest. An estimate on the first component is obtained by choosing $\psi = [100\cdots0]^{\mathsf{T}}$, while an average error is obtained when $\psi = [111\cdots1]^{\mathsf{T}}\frac{1}{N}$. The bound for the error estimate is now a computable value based on the residual and the solution of the dual problem. We call $\|\varphi\|$ the stability factor, which is related to the condition number of the matrix A. The stability factor determines the potential for large error.

We now consider (5.3.1) where f is nonlinear. The residual error is now

$$R = f(X) - b, \tag{5.3.7}$$

which gives

$$f(x) - f(X) = -R. \tag{5.3.8}$$

Using the integral mean value theorem to derive a linear form gives

$$f(x) - f(X) = \int_0^1 f'(sx + (1-s)X)(x - X)ds, \tag{5.3.9}$$

where f' is the Jacobian matrix of f. Thus,

$$Ae = -R, \tag{5.3.10}$$

where $\int_0^1 f'(sx + (1-s)X)(x - X)ds$. By linearizing around the average of x and X, we now have a linear problem. In practice, we have to linearize around X, which introduces an error in the estimates due to linearization, see [11].

For differential equations, we consider the underlying Hilbert space to be a subspace of $L^2(0,T]$ with the usual L^2 inner product and norm. We identify $<,>$ with the L^2 inner product using the Riesz Representation theorem.

Thus, for a differential operator L, equation (5.2.2) becomes

$$\int_0^T Lu(t)v(t)dt = \int_0^T u(t)L^*v(t)dt, \tag{5.3.11}$$

and we call this Green's Identity for homogeneous boundary conditions. We require that Green's Identity hold for all sufficiently smooth functions that vanish on the boundary conditions. Note that if we do not impose boundary conditions on $u(t)$ or $v(t)$ we have

$$\int \left[Lu(t)v(t) - u(t)L^*v(t)\right] dt = \text{boundary terms at 0 and } T, \tag{5.3.12}$$

where the boundary terms depend on $u(t)$ and $v(t)$ and their derivatives on the boundary. We call equation (5.3.12) the extended Green's Identity. This is the identity that we generally use.

Recall that a system of ODEs takes the form

$$\begin{cases} \dot{y} = f(y,t), & 0 < t \leq T, \\ y(0) = y_0. \end{cases} \tag{5.3.13}$$

Since we deal primarily with nonlinear problems, the residual relation for the approximation $Y \approx y$, $f(y) - f(Y) = -R$, is nonlinear. The integral mean value theorem allows the relation to be written as

$$\hat{f}'e = \int_0^1 f'(sy + (1-s)Y)ds\, e = f(y) - f(Y) = -R, \tag{5.3.14}$$

where f' is the Jacobian matrix of f. Since this requires knowledge of the exact solution, we simply use the linearization around the approximate solution which gives

$$\hat{A}e = \int_0^1 f'(Y)ds\, e \approx -R. \tag{5.3.15}$$

We use two adjoint problems

$$\begin{cases} -\dot{\varphi} = \hat{f}'(Y(t),t)^\top \varphi, & T > t > 0, \\ \varphi(T) = \psi_2, \end{cases} \tag{5.3.16}$$

and

$$\begin{cases} -\dot{\varphi} = \hat{f}'(Y(t),t)^\top \varphi + \psi_1(t), & T > t > 0, \\ \varphi(T) = 0. \end{cases} \tag{5.3.17}$$

We will see in sections 6.1 and 6.2 that these correspond to the quantites of interest $(e(T), \psi_2)$ and $\int_0^T (e(t), \psi_1(t))dt$, respectively.

Chapter 6

A POSTERIORI ERROR ANALYSIS

6.1 Estimating the Error in a Quantity of Interest Valued at the Endpoint

The goal is to arrive at an error estimate based as much as possible on computable values, using the concepts put forth in chapter 5. We begin with equation eqrefdual1 and take the inner product of both sides with $e = y - Y$ to get

$$-(\dot{\varphi}, e) - (\hat{f}'^{\mathsf{T}}\varphi, e) = 0. \tag{6.1.1}$$

Integrating from 0 to T, we obtain

$$-\int_0^T (\dot{\varphi}, e)dt - \int_0^T (\hat{f}'^{\mathsf{T}}\varphi, e)dt = 0. \tag{6.1.2}$$

Since we solve the problem by breaking the time period into N intervals, $0 = t_0 < t_1 < t_2 < \cdots < t_N = T$, the error computation is also computed interval by interval, thus

$$-\sum_{n=1}^{N}\int_{t_{n-1}}^{t_n} (\dot{\varphi}, e)dt - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n} (\hat{f}'^{\mathsf{T}}\varphi, e)dt = 0. \tag{6.1.3}$$

Now e is continuous on each subinterval but possibly discontinuous at nodes, so we can use integration by parts on the first integral to get

$$-\sum_{n=1}^{N}(e,\varphi)|_{t_{n-1}}^{t_n}+\sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\varphi,\dot{e})dt-\sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\hat{f}'^{\mathsf{T}}\varphi,e)dt=0. \tag{6.1.4}$$

Looking specifically at the first sum and substituting $e=y-Y$, we see

$$\sum_{n=1}^{N}(e,\varphi)|_{t_{n-1}}^{t_n}=\sum_{n=1}^{N}(y-Y,\varphi)|_{t_{n-1}}^{t_n}, \tag{6.1.5}$$

which, when expanded gives

$$\begin{aligned}\sum_{n=1}^{N}(e,\varphi)|_{t_{n-1}}^{t_n}=&\left[\left(y_{t_1}^{-}-Y_{t_1}^{-},\varphi(t_1)\right)-\left(y_{t_0}^{+}-Y_{t_0}^{+},\varphi(t_0)\right)\right.\\&+\left(y_{t_2}^{-}-Y_{t_2}^{-},\varphi(t_2)\right)-\left(y_{t_1}^{+}-Y_{t_1}^{+},\varphi(t_1)\right)+\cdots\\&\left.+\left(y_{t_N}^{-}-Y_{t_N}^{-},\varphi(t_N)\right)-\left(y_{t_{N-1}}^{+}-Y_{t_{N-1}}^{+},\varphi(t_{N-1})\right)\right].\end{aligned} \tag{6.1.6}$$

Isolating the inner products involving the t_0 terms and the t_N terms, and regrouping gives

$$\begin{aligned}\sum_{n=1}^{N}(e,\varphi)|_{t_{n-1}}^{t_n}=&-\left[-(e(0),\varphi(0))\right.\\&+\left(y_{t_1}-Y_{t_1}^{-},\varphi(t_1)\right)-\left(y_{t_1}-Y_{t_1}^{+},\varphi(t_1)\right)\\&+\left(y_{t_2}-Y_{t_2}^{-},\varphi(t_2)\right)-\left(y_{t_2}-Y_{t_2}^{+},\varphi(t_2)\right)+\cdots\\&+\left(y_{t_{N-1}}-Y_{t_{N-1}}^{-},\varphi(t_{N-1})\right)-\left(y_{t_{N-1}}-Y_{t_{N-1}}^{+},\varphi(t_{N-1})\right)\\&\left.+(e(T),\varphi(T))\right].\end{aligned} \tag{6.1.7}$$

We let $\varphi_n=\varphi(t_n)$. We see that equation (6.1.7) simplifies to

$$\sum_{n=1}^{N}(e,\varphi)|_{t_{n-1}}^{t_n}=-(e(0),\varphi(0))+\sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1})+(e(T),\varphi(T)). \tag{6.1.8}$$

Substituting this into equation (6.1.4) gives

$$\begin{aligned}0=&-\left[-(e(0),\varphi(0))-\sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1})+(e(T),\varphi(T))\right]\\&+\sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\varphi,\dot{e})dt-\sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\hat{f}'^{\mathsf{T}}\varphi,e)dt,\end{aligned} \tag{6.1.9}$$

or

$$(e(T),\varphi(T)) =(e(0),\varphi(0)) + \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) + \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\varphi,\dot{e})dt - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\hat{f}'^{\mathsf{T}}\varphi,e)dt. \tag{6.1.10}$$

Using the fact that $(\hat{f}'^{\mathsf{T}}\varphi,e) = (\hat{f}'e,\varphi)$, we have

$$(e(T),\varphi(T)) =(e(0),\varphi(0)) + \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) + \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\varphi,\dot{e})dt - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\hat{f}'e,\varphi)dt. \tag{6.1.11}$$

We again substitute $e = y - Y$ into the terms involving integrals in the equation to get

$$(e(T),\varphi(T)) =(e(0),\varphi(0)) + \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) + \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\varphi,\dot{y}-\dot{Y})dt - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\hat{f}'(y-Y),\varphi)dt. \tag{6.1.12}$$

Since $\hat{f}'(y-Y) = f(y) - f(Y)$, we then have

$$\begin{aligned}(e(T),\varphi(T)) =&(e(0),\varphi(0)) + \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) \\ &+ \sum_{n=1}^{N}\left[\int_{t_{n-1}}^{t_n}(\dot{y},\varphi)dt - \int_{t_{n-1}}^{t_n}(\dot{Y},\varphi)dt\right] \\ &- \sum_{n=1}^{N}\left[\int_{t_{n-1}}^{t_n}(f(y),\varphi)dt + \int_{t_{n-1}}^{t_n}(f(Y),\varphi)dt\right].\end{aligned} \tag{6.1.13}$$

Since $\dot{y} - f(y) = 0$, this becomes

$$(e(T),\varphi(T)) = (e(0),\varphi(0)) + \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\dot{Y} - f(Y),\varphi)dt. \tag{6.1.14}$$

On each interval, we also have, by Galerkin orthogonality, that

$$\int_{t_{n-1}}^{t_n} (\dot{Y} - f(Y,t), v)dt - ([Y]_{n-1}, v_{n-1}^+) = 0 \tag{6.1.15}$$

for all v in the space of test functions. We let $v = \pi_k\varphi$ where $\pi_k\varphi$ is a projection or interpolant of φ in the space of test functions and add equation (6.1.15) to equation (6.1.10). This results in the error representation formula

$$\begin{aligned}(e(T), \varphi(T)) = (e(0), \varphi(0)) + \sum_{n=1}^{N} \int_{t_{n-1}}^{t_n} ((\dot{Y} - f(Y,t), \varphi)dt + \sum_{n=1}^{N}([Y]_{n-1}, \varphi_{n-1}) \\ + \sum_{n=1}^{N} \int_{t_{n-1}}^{t_n} ((\dot{Y} - f(Y,t), \pi_k\varphi)dt + \sum_{n=1}^{N}([Y]_{n-1}, \pi_k\varphi_{n-1})\end{aligned} \tag{6.1.16}$$

and finally,

$$\begin{aligned}(e(T), \varphi(T)) = (e(0), \varphi(0)) + \sum_{n=1}^{N} \left(- \int_{t_{n-1}}^{t_n} (\dot{Y} - f(Y,t), (\varphi - \pi_k\varphi))dt \right) \\ + \sum_{n=1}^{N}([Y]_{n-1}, (\varphi - \pi_k\varphi)_{n-1}^+).\end{aligned} \tag{6.1.17}$$

Equation (6.1.17) gives a representation of the error generated by the discretization of the domain.

However, the integrals are evaluated using some quadrature scheme, so we also need to account for quadrature error. Since we solve the differential equations by evaluating the integral on the right hand side of the weak form of equation (2.1.1) using some quadrature scheme, Galerkin orthogonality actually reads

$$\sum_{n=1}^{N} \int_{I_n} \left(\dot{Y} - \overline{f(Y)}, \pi_k\varphi\right) dt + \sum_{n=1}^{N} ([Y]_{n-1}, \pi_k\varphi_n) = 0. \tag{6.1.18}$$

Recall from chapter 3 that $\overline{f(Y(t),t)}$ is an approximating polynomial.

To add in the quadrature error, we subtract equation (6.1.18) from equation (6.1.14) which gives

$$\begin{aligned}(e(T),\varphi(T)) = (e(0),\varphi(0)) &+ \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}((\dot{Y},\varphi)dt) + \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}((-f(Y,t),\varphi)dt) \\ &+ \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) \\ &- \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}((\dot{Y},\pi_k\varphi)dt - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}((-\overline{f(Y,t)},\pi_k\varphi)dt \\ &- \sum_{n=1}^{N}([Y]_{n-1},\pi_k\varphi_{n-1}^{+}).\end{aligned} \tag{6.1.19}$$

We then add

$$\sum_{n=1}^{N}\left(\int_{t_{n-1}}^{t_n}(\overline{f(Y,t)},\varphi)dt - \int_{t_{n-1}}^{t_n}(\overline{f(Y,t)},\varphi)dt\right) \tag{6.1.20}$$

to equation (6.1.19), resulting in

$$\begin{aligned}(e(T),\varphi(T)) = (e(0),\varphi(0)) &+ \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}((\dot{Y},\varphi)dt) + \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}((-f(Y,t),\varphi)dt) \\ &+ \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) \\ &- \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}((\dot{Y},\pi_k\varphi)dt - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}((-\overline{f(Y,t)},\pi_k\varphi)dt \\ &- \sum_{n=1}^{N}([Y]_{n-1},\pi_k\varphi_{n-1}^{+}) \\ &+ \sum_{n=1}^{N}\left(\int_{t_{n-1}}^{t_n}(\overline{f(Y,t)},\varphi)dt - \int_{t_{n-1}}^{t_n}(\overline{f(Y,t)},\varphi)dt\right).\end{aligned} \tag{6.1.21}$$

Rearranging and combining terms results in the final error representation formula:

$$(e(T),\varphi(T)) = (e(0),\varphi(0)) - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\dot{Y} - \overline{f(Y,t)})(\varphi - \pi_k\varphi)dt \quad (6.1.22a)$$

$$- \sum_{n=1}^{N}\left([Y_{n-1}],(\varphi - \pi_k\varphi)^+_{n-1}\right) \quad (6.1.22b)$$

$$- \sum_{n=1}^{n}\left(\int_{t_{n-1}}^{t_n}(f(Y,t),\varphi)dt - \int_{t_{n-1}}^{t_n}(\overline{f(Y,t)},\varphi)dt\right). \quad (6.1.22c)$$

where (6.1.22a) and (6.1.22b) describe the discretization error and (6.1.22c) describes the error for the choice of quadrature.

As an alternative, we can add and subtract

$$\sum_{n=1}^{N}\left(\int_{t_{n-1}}^{t_n}(f(Y,t),\pi_k\varphi)dt - \int_{t_{n-1}}^{t_n}(f(Y,t),\pi_k\varphi)dt\right). \quad (6.1.23)$$

This leads to

$$(e(T),\varphi(T)) = (e(0),\varphi(0)) - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\dot{Y} - f(Y,t))(\varphi - \pi_k\varphi)dt \quad (6.1.24a)$$

$$- \sum_{n=1}^{N}\left([Y_{n-1}],(\varphi - \pi_k\varphi)^+_{n-1}\right) \quad (6.1.24b)$$

$$- \sum_{n=1}^{N}\left(\int_{t_{n-1}}^{t_n}f(Y,t)\pi_k\varphi dt - \int_{t_{n-1}}^{t_n}(\overline{f(Y,t)},\pi_k\varphi)dt\right). \quad (6.1.24c)$$

In practice, we use the polynomial approximations represented by equations (3.1.10) and (3.1.12). Substituting these into equations (6.1.22a) and

(6.1.22c), we then compute the error estimate by

$$\begin{aligned}(e(T),\varphi(T)) &= (e(0),\varphi(0)) \\ &- \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}\left(\left(\dot{Y} - f\left(\sum_{m=1}^{M+1} a_m p_m(t), t\right)\right), (\varphi - \pi_k\varphi)\right) dt \\ &- \sum_{n=1}^{N}\left([Y_{n-1}], (\varphi - \pi_k\varphi)^+_{n-1}\right) \\ &- \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}\left(f\left(\sum_{m=1}^{M+1} a_m p_m(t), t\right), \pi_k\varphi\right) dt \\ &+ \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}\left(\sum_{s=1}^{S} f\left(\sum_{m=1}^{M+1} a_m p_m(\tau_i), \tau_i\right) r_s(t), \pi_k\varphi\right) dt.\end{aligned} \tag{6.1.25}$$

6.2 Average Error

If the quantity of interest is valued on $(0,T)$, we start with equation (5.3.17). Again, we take the inner product with $e = y - Y$ and integrate to get

$$\int_0^T (e,\psi_1)dt = -\int_0^T (\dot{\varphi},e)dt - \int_0^T (\hat{f}'^{\mathsf{T}}\varphi, e)dt. \tag{6.2.1}$$

As above, we decompose the integrals over the sequence of intervals and use integration by parts on the first integral. The gives

$$\begin{aligned}\int_0^T (e,\psi_1)dt &= -(e(T),\varphi(T)) + (e(0),\varphi(0)) + \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) \\ &+ \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\varphi,\dot{e})dt - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\hat{f}'^{\mathsf{T}}\varphi, e)dt.\end{aligned} \tag{6.2.2}$$

We assume no error in the initial condition of the adjoint problem. Thus $e(T) = 0$ and the first term drops out leaving

$$\begin{aligned}\int_0^T (e,\psi_1)dt &= (e(0),\varphi(0)) + \sum_{n=1}^{N}([Y]_{n-1},\varphi_{n-1}) \\ &+ \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\varphi,\dot{e})dt - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n}(\hat{f}'^{\mathsf{T}}\varphi, e)dt.\end{aligned} \tag{6.2.3}$$

We note that the right hand side of equations (6.2.3) and (6.1.10) are identical. We argue as above to get the error representation formula for the average error

$$\int_0^T (e,\psi_1)dt = (e(0),\varphi(0)) - \sum_{n=1}^{N}\int_{t_{n-1}}^{t_n} (\dot{Y} - \overline{f(Y,t)})(\varphi - \pi_k\varphi)dt \quad (6.2.4a)$$

$$- \sum_{n=1}^{N} \left([Y_{n-1}], (\varphi - \pi_k\varphi)^+_{n-1}\right) \quad (6.2.4b)$$

$$- \sum_{n=1}^{N} \left(\int_{t_{n-1}}^{t_n} (f(Y,t),\varphi)dt - \int_{t_{n-1}}^{t_n} (\overline{f(Y,t)},\varphi)dt \right). \quad (6.2.4c)$$

Chapter 7

IMPLEMENTATION

The above concepts were coded into a general ODE solver. The solver is written in C, and while it is compiled and linked using the Microsoft Visual C++ compiler, it is written in ANSI C so it can be easily ported to other environments (i.e., Unix Gnu C). All code is new with the exception of the linear algebra routines which are taken from the CVODE code written by Scott Cohen and Alan Hindmarsh at the Lawrence Livermore National Laboratory, Livermore, California [9]. The program is currently comprised of over 40 functions totaling several thousand lines of code.

7.1 Solving the Forward Problem

For the forward solver, we manipulate the equations to form an algebraic system. For example, consider the 2 stage, order 3 Runge-Kutta method based on two Radau points. We start with equation (3.1.15) and move all terms involving the $\tilde{Y}$ values to the left side of the equation. This gives

$$\begin{aligned} \tilde{Y}_{m,0} - \frac{5}{12}k_m f(\tilde{Y}_{m,0}, t_{m-1} + \frac{k_m}{3}) + \frac{1}{12}k_m f(\tilde{Y}_{m,1}, t_m) = Y_{m-1}, \\ \tilde{Y}_{m,1} - \frac{3}{4}k_m f(\tilde{Y}_{m,0}, t_{m-1} + \frac{k_m}{3}) - \frac{1}{4}k_m f(\tilde{Y}_{m,1}, t_m) = Y_{m-1}, \end{aligned} \tag{7.1.1}$$

which can be written as

$$\begin{pmatrix} I - \frac{5}{12}k_m f(\tau_1) & \frac{1}{12}k_m f(\tau_2) \\ -\frac{3}{4}k_m f(\tau_1) & I - \frac{1}{4}k_m f(\tau_2) \end{pmatrix} \begin{pmatrix} \tilde{Y}_1 \\ \tilde{Y}_2 \end{pmatrix} = \begin{pmatrix} Y_{m-1} \\ Y_{m-1} \end{pmatrix} \tag{7.1.2}$$

where $\tau_1 = t_{m-1} + \frac{k_m}{3}$ and $\tau_2 = t_m$. Note that, for a system of n equations, $\tilde{Y}_1$, $\tilde{Y}_2$, and Y_{m-1} are all vectors in $\mathbb{R}^n$. The system of equations is solved using a quasi-Newton method where the Jacobian of the matrix is approximated using finite differences. Thus, for the Jacobian, J, we have

$$J \approx \begin{pmatrix} \overline{\frac{\partial f_1}{\partial x_1}} & \cdots & \overline{\frac{\partial f_1}{\partial x_n}} \\ \vdots & \ddots & \vdots \\ \overline{\frac{\partial f_m}{\partial x_1}} & \cdots & \overline{\frac{\partial f_m}{\partial x_n}} \end{pmatrix}, \tag{7.1.3}$$

where

$$\overline{\frac{\partial f_i}{\partial x_j}} = \frac{f_i(x_j + \delta) - f_i(x_j)}{\delta}. \tag{7.1.4}$$

We use $\delta = 1.01\sqrt{\text{machine epsilon}}$ in order to get the best approximation while avoiding issues of machine roundoff errors. By computing the finite difference approximation of the Jacobian, we avoid requiring the user to supply an exact Jacobian.

To begin the Newton iteration we make an initial guess. The code provides two options for the initial guess; either the approximation at the previous time step, or an approximation computed using forward Euler's method. For either guess, we use a hybrid algorithm to try to insure convergence without requiring a decrease in the step size selected by the adaptive error control. We compute the original Newton step and check to see if this leads to a decrease in the Newton function value. If not, we halve the step taken in the Newton direction, repeating until there is a decrease.

In order to compute the *a posteriori* error estimate, we numerically solve the adjoint problem corresponding to the forward problem linearized

around the approximation Y. However, the discontinuities in Y are awkward in this context as it forces the adjoint solution to use the same nodes. As a way around this difficulty, with a side benefit of reducing memory, we compute a cubic spline $\overline{Y}$ of the approximation Y based on the algorithm in Kincaid and Cheney [28] and linearize around the spline. This decouples the numerical solutions of the forward and adjoint problems. Use of the spline is discussed in the next section.

7.2 Solving the Adjoint Problem

We note from chapter 6 that the error computation requires the term, $\varphi - \pi_k\varphi$. Since $\pi_k\varphi$ is in the same space as the approximation from the dG(q) method, we compute an approximate solution $\Phi \approx \varphi$ that is higher order than the dG(q) method used for the forward solve. For the finite element approximation of the adjoint problem,

$$\begin{cases} -\dot{\varphi} = \overline{f'}(Y(t))^\top \varphi + \psi_1, \quad T > t > 0, \\ \varphi(T) = \psi_2, \end{cases} \tag{7.2.1}$$

we use the cG(q) method based on a discretization $T = s_N > s_{N-1} > \cdots > s_0 = 0$. We use this discretization since the adjoint problem runs backwards in time. In the case of $\psi_2 = 0$, we compute $\Phi \in \mathcal{P}^q(s_{j-1}, s_j)$ such that

$$\begin{cases} -\int_{s_{n-1}}^{s_n} (\dot{\Phi}, v)dt = \int_{s_{n-1}}^{s_n} (\hat{f}'(\overline{Y}(t), t)\Phi, v)dt + \int_{s_{n-1}}^{s_n} (\psi_1, v)dt, \\ \Phi_{j-1}^+ = \Phi_{j-1}^- \end{cases} \tag{7.2.2}$$

for all $v \in \mathcal{P}^{q-1}(s_{n-1}, s_n)$, $n = \tilde{N}, \tilde{N}-1, \ldots, 1$. Again, we have the issue of choosing a quadrature. For a low order method, we use dG(0) (eg. backward Euler with quadrature) for the forward solve and cG(1) (eg. trapezoidal rule with quadrature) for the adjoint solve. For the high order method, we use dG(1) (eg. 2 stage, order 3 implicit Runge-Kutta method based on the

2 point Radau quadrature) for the forward solve, and cG(2) with 2 point Gauss quadrature (eg. a 2 point, order 4 implicit Runge-Kutta method) for the adjoint solve [5, 24, 44].

Next, we note that this problem runs backwards in time. In the implementation, we perform the change of variables,

$$s = T - t, \tag{7.2.3}$$

so,

$$\frac{d\varphi}{dt} = -\frac{d\varphi}{ds} \tag{7.2.4}$$

and with the obvious change in notation, the problem becomes

$$\begin{cases} \int_{s_{n-1}}^{s_n} (\dot{\Phi}, v)dt = \int_{s_{n-1}}^{s_n} (\hat{f}'(\overline{Y}(T-t))^\top \Phi, v)dt + \int_{s_{n-1}}^{s_n} (\psi(T-t), v)dt, \\ \Phi_{n-1}^+ = \Phi_{n-1}^-, \quad n = 1, 2, \ldots, \tilde{N}. \end{cases} \tag{7.2.5}$$

The case of $\psi_1 = 0$ is handled similarly. The dual solver computes a system of equations derived in a similar manner as the forward solver. For example, using the trapezoidal rule to approximate the integrals for cG(1) gives the equation

$$\left(I - (f'(\overline{Y}(s_n), s_n))^\top \frac{k_n}{2}\right) \varphi_n = \left(I + f'(\overline{Y}(s_{n-1}), s_{n-1})^\top \frac{k_n}{s}\right) \varphi_{n-1} + (\psi(s_n) + \psi(s_{n-1})) \frac{k_n}{2} \tag{7.2.6}$$

As mentioned, we use a spline of the forward solution for Y. We build $f'(\overline{Y})^\top$ at each point required for the adjoint solver by building a numerical Jacobian using finite differences, á la quasi-Newton, at each point where the matrix has to be evaluated and computing an explicit transpose. Solving the adjoint problem this way uses roughly the same number of function evaluations of f as the forward solve, which is nonlinear, typically takes.

Finally, we compute a spline, $\overline{\Phi}$ of Φ for use in computing the *a posteriori* error estimates. In this way we can easily evaluate Φ on the mesh used for Y. The conclusion is that we have made a series of choices in implementation that allows a controlled approximation of the quantities in the theoretical framework that gives access to the techniques without requiring the user to do anything other than provide data, the model f, a quantity of interest, and a tolerance.

7.3 Error Computations

Using equation (6.1.10) through equation (6.1.14), we estimate the error associated with the approximation. Since the error term is composed of three parts, the discretization error, the jump term, and the quadrature error, each piece is computed separately. This also allows identification of the major sources of the error, whether from the discretization error or the quadrature error. This information can be used to determine whether a finer mesh or higher quadrature should be used in the solution method. For both the discretization error and the quadrature error, it is necessary to evaluate integrals accurately. We use a 6 point Gauss rule to capture integrals in the estimate. Note, we do not use this in creating the approximation.

7.3.1 Discretization error

On each interval, discretization error is estimated by

$$\int_{t_{n-1}}^{t_n} ((\dot{Y} - f(Y,t)), (\overline{\Phi} - \pi_k\overline{\Phi}))dt + ([Y_{n-1}], (\overline{\Phi} - \pi_k\overline{\Phi})^+_{n-1}), \qquad (7.3.1)$$

where $\dot{Y}$ is computed from the approximation, $f(Y,t)$ is an evaluation of the function at the quadrature nodes, $\overline{\Phi}$ is retrieved from the spline at the

quadrature nodes, and $\pi_k\overline{\Phi}$ is a projection of $\overline{\Phi}$ into the appropriate space based on the method used. The jump term

$$([Y_{n-1}], (\overline{\Phi} - \pi_k\overline{\Phi})^+_{n-1}), \tag{7.3.2}$$

is computed directly from the solution. For the dG(0) method, $[Y_{n-1}] = Y_n - Y_{n-1}$ since the solution space is piecewise constant. For the dG(1) method, the solution is piecewise linear and satisfies the solution at the quadrature nodes. Thus,

$$[Y_{n-1}] = Y_n - \frac{dY}{dt}k_n - Y_{n-1} \tag{7.3.3}$$

where $\frac{dY}{dt}$ is the derivative of the solution over the interval. Figure 7.1 shows the computation of the jump term.

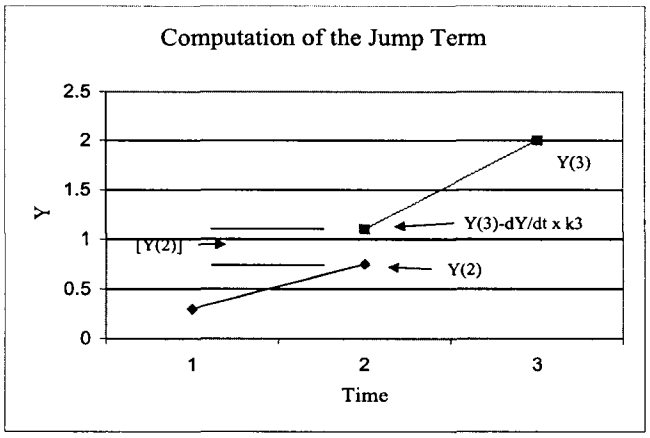


Figure 7.1: Computation of the jump term. Y is given at $t = 2$ and $t = 3$ with lines showing the piecewise linear approximations. The jump term is computed at $t = 2$.

7.3.2 Quadrature error

On each interval, quadrature error is estimated by

$$\int_{t_{n-1}}^{t_n} ((f(Y,t) - \overline{f(Y,t)}), \pi_k\overline{\Phi})dt. \tag{7.3.4}$$

We compute both f and $\overline{f}$, the projection of f used to build the quadrature rule, at the nodes for the 6 point Gauss rule. For dG(1), $\overline{f}$ is the interpolant associated with the quadrature rule used as shown in equation (3.1.12).

Chapter 8

ADAPTIVE STRATEGIES

The goal of adaptive error control is to generate a computational mesh that produces a result with an acceptable level of accuracy using an approximately minimal amount of computational work. This means, in general, that we wish to create a computational mesh with as few nodes as possible, while still achieving the desired level of accuracy.

Adaptivity makes it possible to solve models using time steps appropriate to computing the desired information from the solution. The classic consideration adjusts the time step to account for variations in the size of the residual or the so called "local" error. In our approach, the adjoint solution is used to determine the effects of stability. Areas of instability, as indicated by increases in the dual weights, may need a finer mesh than stable areas. Consider, for example, the logistics equation given by

$$\dot{y} = ay - by^2, \tag{8.0.1}$$

with $a = b = 2.309$ and $p(0) = 0.1$.

Figure 8.1 shows the approximation and exact solution for the logistics equation before any refinement. We see the solution exhibits relatively large derivative values until the solution curve begins to level out at approximately $t = 2.0$. In Fig. 8.2, we see the difference between the approximation

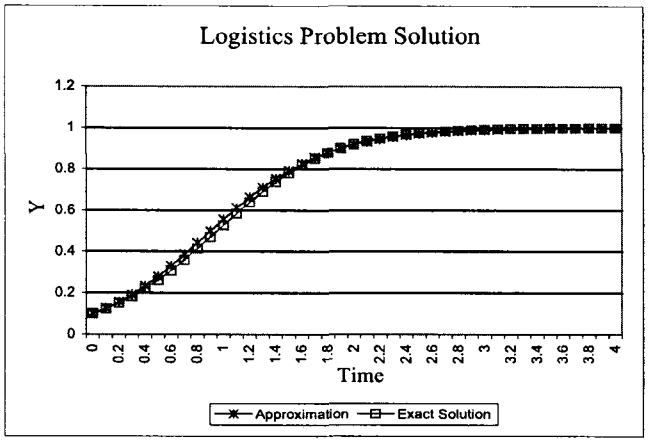


Figure 8.1: Exact solution and approximation for the logistics equation.

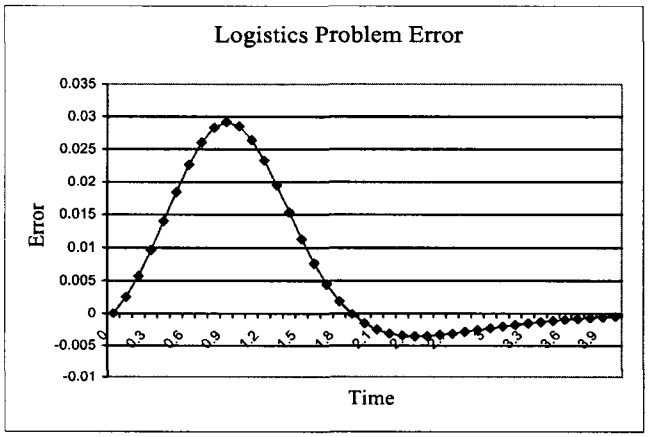


Figure 8.2: Error for the logistics equation measured as the difference between the exact solution and the approximation.

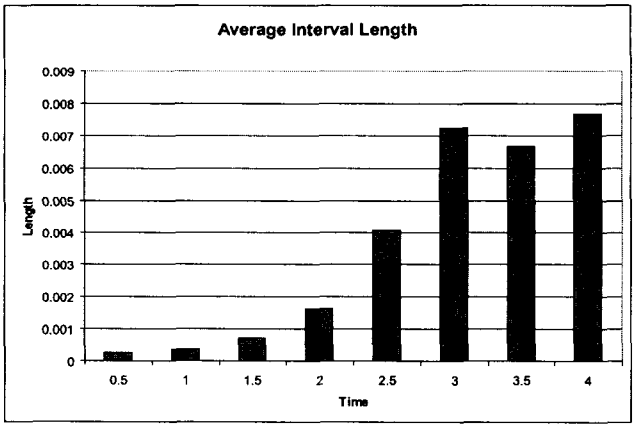


Figure 8.3: Average interval lengths for the logistics equation after reaching tolerance.

and the exact solution, which indicates that the largest errors occur when the derivative values are largest. This illustrates both the stability aspects of the logistics equation and the effects of cancellation of error. The growth in error through $t = 1.0$ indicates this is a region of dynamic instability, in which errors accumulate, while the corresponding decrease in error from $t = 1.0$ to $t = 2.0$ indicates this is a dynamically stable region, and clearly illustrates the cancellation of error. By $t = 2.0$, we see the negative contributions have eliminated the error for the positive contributions. Figure 8.3 shows the average interval length over regions of 0.5 on the time axis after refinement based on *a posteriori* error estimates. Clearly, the adaptivity refined most in the regions where the derivative values are the largest and the solution is unstable. Without adaptive refinement, the choice would be to solve the equation using the smallest time step, and without *a posteriori* error estimates, that time step is unknown. This illustrates the usefulness of adaptive techniques.

Formally, the error representation (equations (6.1.22a), (6.1.22b), and (6.1.22c)) can be written

$$\text{error} \approx \sum_{n=1}^{N} \text{error contribution}_n. \tag{8.0.2}$$

If there were no cancellation of error, so this is a sum of quantities with the same sign, we could frame the problem as an optimization problem that has a well defined solution in the Principle of Equidistribution. This states that the optimal choice balances the internal contributions so they are equal. Thus, we refine in order to achieve equal contributions from each interval. For this reason, standard approaches to adaptive error control replace (8.0.2) by

$$\text{error} \leq \sum_{n=1}^{N} \text{error contribution}_n. \tag{8.0.3}$$

and then optimize a bound and not an accurate estimate. We also note that in the classic approach, the global effects of stability are ignored, and the bound (8.0.3) is further analyzed to

$$|\text{error}| \leq C \sum_{n=1}^{N} |\text{internal residuals}_n|, \tag{8.0.4}$$

or

$$|\text{error}| \leq C \sum_{n=1}^{N} |\text{“local” errors}_n|. \tag{8.0.5}$$

In this thesis, we consider four mesh refinement strategies. (Note that the code can be run without adaptivity.) All four strategies require a user defined global error tolerance (TOL) and include:

1. the closest analog of the classic approach, refinement of all intervals with adjoint weighted error contribution E_n that satisfy

$$E_n > TOL/N, \quad \text{where } N \text{ is the current number of intervals,} \tag{8.0.6}$$

2. a probabilistic version in which intervals are chosen for refinement with probability based on their contribution,

3. a probabilistic selection based on both error contribution and interval length,

4. a "weighted zonal strategy" which seeks to accomodate cancellation of error. The weighted zonal strategy also employs 3.

All of these options are different than the classic approach described in chapter 4 since they are global and treat the time dependent problem like an elliptic problem in the sense that time is considered as a whole and refinement is then determined on an interval by interval basis. After solving the problem for the entire time period, we then solve the adjoint problem and compute the dual weights, and finally, use the dual weighted residual to determine interval contribution. Figure 14.1 in chapter 14 outlines this procedure.

The first adaptivity option is the deterministic adaptive mesh approach that uses equidistribution of error adapted to use *a posteriori* error estimates, while the rest use novel approaches described below.

Given the error representation formula, we can approach the result in two ways. The interval contribution on I_n is

$$\int_{I_n} \left((R, W) + (Q, \overline{\Phi})\right) dt - \left([Y_{n-1}], (\overline{\Phi} - \pi_k \overline{\Phi})^+_{n-1}\right), \tag{8.0.7}$$

where $R = \dot{Y} - f(Y,t)$, $W = \overline{\Phi} - \pi_h \overline{\Phi}$, and $Q = -f(Y,t) + \overline{f(Y,t)}$ with $\overline{f(Y,t)}$ defined as in chapter 3. Since the equation involves vector inner products, we can view the resulting error representation (for P dimensions)

as

$$\int_{I_n} \left((R,W) + (Q,\overline{\Phi})\right) dt - \left([Y_{n-1}], (\overline{\Phi} - \pi_k\overline{\Phi})^+_{n-1}\right) =$$
$$\sum_{m=1}^{P} \int_{I_n} \left((R_m,W_m) + (Q_m,\overline{\Phi}_m)\right) dt - \left([Y_{n-1}]_m, (\overline{\Phi}_m - \pi_k\overline{\Phi}_m)^+_{n-1}\right), \tag{8.0.8}$$

which allows error contributions to be determined for each component of the systems of equations. See Logg [33] and [34] for a discussion of multi-adaptive methods where the mesh is refined differently for each component. Here we consider the error for all components as a whole.

8.1 Equidistribution of Error

The first strategy is based on the traditional approach to error control. As mentioned, equidistribution of error yields the optimal choice in the limited circumstances in which there is no cancellation of errors. In this approach, we seek to satisfy the following equation,

$$E_n \leq \frac{\text{TOL}}{N} \tag{8.1.1}$$

where TOL is the global error tolerance, N is the number of intervals, and E_n is the interval contribution. Taking the absolute value and factoring out k_n^q where k_n is the step size on the n^{th} interval, and q is the order of the method, we have

$$k_n^q \left|\frac{E_n}{k_n^q}\right| \leq \frac{\text{TOL}}{N}. \tag{8.1.2}$$

$|E_n|/k_n^q$ is relatively independent of the step. Hence, we can project the best value of k_n;

$$k_{new}^q \approx \frac{k_n^q \times \text{TOL}}{N \times |E_n|}. \tag{8.1.3}$$

Thus, the new time step is defined by

$$\begin{aligned} k_{new} &= \left[\frac{k_n^q \times \text{TOL}}{N \times \text{Interval Contribution}}\right]^{\frac{1}{q}} \\ &= k_n \left[\frac{\text{TOL}}{N \times E_n}\right]^{\frac{1}{q}} \end{aligned} \tag{8.1.4}$$

and the interval is divided accordingly.

In evolution problems, there is generally accumulation of error over time, and the effect the error contributed by a given time step has on subsequent steps is unknown. This motivates a search for new approaches.

8.2 Probabilistic Selection

We consider the absolute values of the interval contributions normalized by the total error estimate as a probability distribution of the error. Since the error contributions are based on both the residual and the stability factor, we do not expect the overestimation of traditional methods. In addition, since the effects of any given time step's error contribution on subsequent time steps is unknown, the probabilistic approach allows for refinement of time steps with small error contribution. Figure 8.4 shows the normalized absolute values of the interval contributions for a specific simulation. By using absolute values, we have removed consideration of the cancellation of error. In the probabilistic approaches, we choose intervals to refine randomly according to the element distribution function. It can be difficult to choose random numbers according to an arbitrary distribution, however, the Fundamental Theorem of Simulation provides a method to do this. For a given distribution $g(x)$, we can write

$$g(x) = \int_0^{g(x)} du.$$

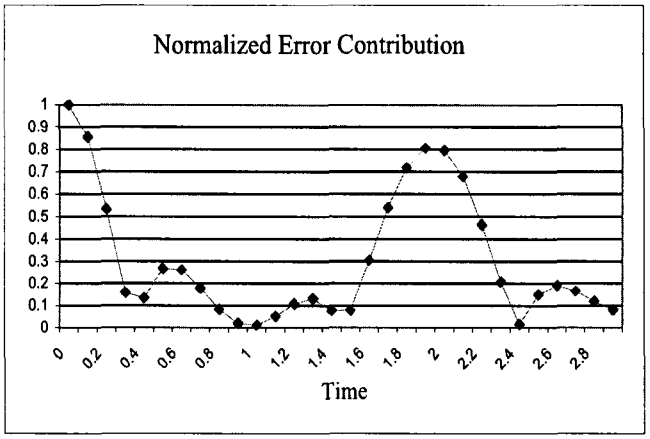


Figure 8.4: Absolute values of the interval contributions.

Note that g is the marginal density of the joint distribution

$$(X, U) \sim \mathcal{U}\{(x, u) : 0 < u < g(x)\}.$$

Thus, we can generate from the joint distribution by generating uniform random variables on the constrained set. This information gives rise to the Fundamental Theorem of Simulation.

Theorem 8.2.1. *Simulating*

$$X \sim g(x)$$

is equivalent to simulating

$$(X, U) \sim \mathcal{U}\{(x, u) : 0 < u < g(x)\}.$$

Simply put, if $A \subset B$, we can generate a uniform sample on A by generating a uniform sample on B and keeping only those elements that are in A. Theorem 8.2.1 is expressed in general terms in the following practical corollary.

Corollary 8.2.1. *Let $X \sim g(x)$ and $h(x)$ be a density function that satisfies $g(x) \leq Mh(x)$ for some constant $M \geq 1$. Then, to simulate $X \sim g$, it is sufficient to generate $Y \sim h$ and $U|Y = h \sim \mathcal{U}(0, Mh(y))$, until $0 < u < g(y)$.*

$h(x)$ is chosen so that simulating the density is easy, e.g. $h \equiv 1$ is common. The implementation of this corollary is called the Accept-Reject Method.

To simulate a distribution by the Accept-Reject Method, we generate a uniform distribution on $[0, T] \times [0, 1]$. This distribution is then constrained by the probability distribution defined by the normalized interval contributions. Thus, all points of the uniform distribution that fall in the shaded region shown on Fig. 8.5 are accepted, while those that fall outside the shaded region are rejected. We note that the probability distribution is, in

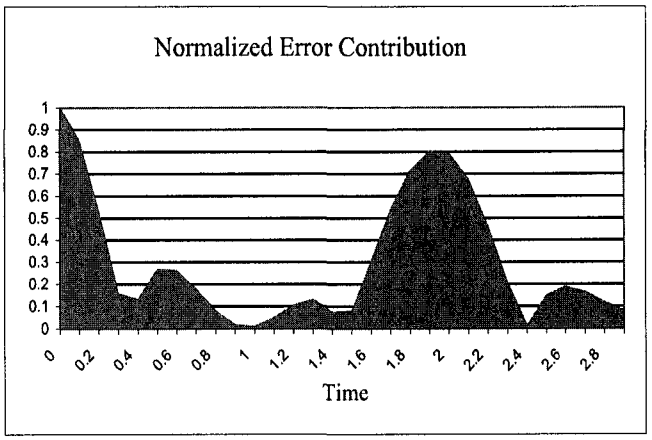


Figure 8.5: Accept and reject regions of the normalized distribution.

effect, a product of both the interval contribution distribution and the time mesh, since the mesh is not uniform. This is discussed in further detail below.

The probabilistic options choose intervals to refine based on random number generation, giving weight to those intervals contributing the most to the overall error, but allowing for selection of intervals with a small contribution. Options 2 and 3 differ in that option 2 makes the selection based solely on the contribution to the error, while option 3 also uses the length of the time step in the selection process.

For option 2, we define a cumulative density function on the error, $E(l)$, where

$$E(l) = \sum_{n=1}^{l} |E_n| , \tag{8.2.1}$$

where E_n is the contribution of the n^{th} interval. A random number is generated between 0 and $E(N)$ and compared to the cumulative density function. If the random number falls between $E(n-1)$ and $E(n)$, then the n^{th} interval if flagged for refinement. Intervals are selected up to the total percent as specified by the user. Thus the probability of an interval being chosen is

$$Pr = \left(\frac{E_n - E_{n-1}}{E_N} \right) . \tag{8.2.2}$$

For option 3, we determine the maximum error contribution, $E_{max} = \max_{n \leq N}(E_n)$, and normalize all error contributions by $\hat{E}_n = E_n / E_{max}$. Consequently, $\hat{E}_n \in (0, 1)$ for all n. The choice of time interval is computed similarly to the method used in option 3 for the error contribution, only the cumulative density function is based on the interval lengths. Thus, the probability of a time interval being selected is

$$Pr = \left(\frac{t_n - t_{n-1}}{t_N} \right) . \tag{8.2.3}$$

After a time interval is determined, a random number, $r \in [0, 1]$, is computed and compared to the normalized error, E_n. If $r < E_n$, the n^{th} interval

is marked for refinement. Thus, the overall probability that the n^{th} interval is refined is

$$Pr = \left(\hat{E}_n \frac{t_n - t_{n-1}}{t_N} \right). \tag{8.2.4}$$

Both of these options allow for an interval to be selected multiple times.

An integral part of this procedure is determining the number of intervals to flag for refinement. Recall, the error estimate is given by $\sum_{n=1}^{N} E_n$. After refinement, we have $\sum_{n=1}^{N} \sum_{m=1}^{N_n} \tilde{E}_{n,m}$ where N_n is the number of subdivisions of the original n^{th} interval, and $\tilde{E}_{n,m}$ is the contribution of the m^{th} subinterval of the n^{th} interval. Our goal is

$$\sum_{n=1}^{N} \sum_{m=1}^{N_n} \tilde{E}_{n,m} = TOL. \tag{8.2.5}$$

For a method of order q, we reduce the error contribution by $(k_{n,m}/k_n)^q$, where $k_{n,m} = k_n/N_n$. Thus, we have

$$\sum_{m=1}^{N_n} \tilde{E}_{n,m} = E_n \left(\frac{k_{n,m}}{k_n} \right)^q. \tag{8.2.6}$$

Since the number of subintervals on any given interval is $N_n = k_n/k_{n,m}$, we solve for $k_n = N_n k_{n,m}$. Also, from equation (8.2.5), we have that $\sum_{m=1}^{N_n} \tilde{E}_{n,m} = TOL/N$. Substituting these into equation (8.2.6), we have

$$\begin{aligned} \frac{TOL}{N} &= E_n \left(\frac{k_{n,m}}{M_n k_{n,m}} \right)^q \\ &= E_n \left(\frac{1}{N_n^q} \right). \end{aligned} \tag{8.2.7}$$

Solving for N_n we have

$$N_n = \left(\frac{E_n N}{TOL} \right)^{\frac{1}{q}}. \tag{8.2.8}$$

We then sum the total subdivisions for each interval.

8.3 Weighted Zonal Strategy

In this approach, the global strategy is to balance the positive and negative contributions in order to promote maximal cancellation. Since the error estimate takes cancellation of error into account, by balancing those areas of positive and negative contribution, we allow cancellation of errors to effect step size selection.

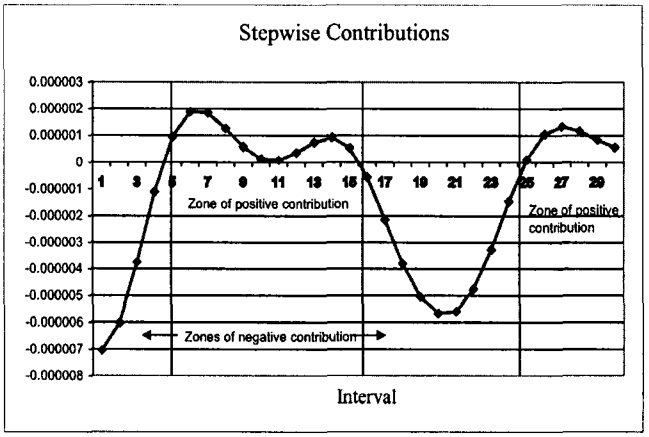


Figure 8.6: Stepwise Contributions

Figure 8.6 shows a graph of stepwise contributions. The graph is divided into zones of positive and negative contributions. For purposes of discussion, we number the zones consecutively with the first zone on the left being zone 1. The following table lists the total contributions for each zone. We see the total positive contribution is 1.43e-05 while the total negative contribution is -5.02e-05.

Zone	Contribution
1	-1.79e-05
2	9.28e-06
3	-3.23e-05
4	5.04e-06

In order to define the strategy, consider the following definitions:

N = number of intervals to refine

N_p = number of intervals with positive contribution

N_n = number of intervals with negative contribution

E_i = contribution of the i^{th} interval

E_{pos} = sum of contributions of all intervals with positive contribution

E_{neg} = sum of contributions of all intervals with negative contribution

First, we apportion the number of interval marked for refinement to the various zones. Thus,

$$N_i = N\frac{|E_i|}{E_{pos} + |E_{neg}|}, \tag{8.3.1}$$

where N_i is the number of intervals in the i^{th} zone that should be marked for refinement. For example, given the numbers in the first table, we would add intervals to each zone up to the numbers in the following table.

Zone	Number of intervals to refine
1	6
2	3
3	10
4	1

Next, within each zone, we select intervals for refinement using a probablistic method. The first step is to normalize the contributions within a zone. Within each zone, the normalized contribution for an interval is

$$Enorm_n = \frac{|E_n|}{\sum_{m=1}^{M} |z_m|} \tag{8.3.2}$$

where M is the number of intervals in the zone. The following table shows the contributions and normalized contributions for zone 1 of the example.

Interval	Contribution	Normalized
1	-7.04e-06	0.393
2	-6.02e-06	0.336
3	-3.75e-06	0.209
4	-1.11e-06	0.062

The following algorithm determines an interval to mark for refinement.

1. Generate a random number between 0 and $T = t_{end} - t_{begin}$ where t_{begin} is the beginning time for the zone and t_{end} is the ending time for the zone. This is compared to the sequence $(0, t_2 - t_{begin}, t_3 - t_{begin}, \ldots, t_m - t_{begin})$ where t_i is beginning time for the i^{th} interval in the zone, and determines the interval to check.

2. Generate a random number in $[0, 1]$ and compare to the normalized contribution of the interval. If the random number is less than the normalized contribution, the interval is marked for refinement.

3. Repeat 1 and 2 until the required number of intervals are marked.

Note that some intervals may be marked for refinement more than once.

Multiple refinements for a single interval differ based on the solution method. The low order method requires more refinement to reduce the contribution, thus an interval marked j times is bisected j times, resulting in 2^j intervals. For the high order method, the interval is simply divided into $j + 1$ uniform intervals.

8.4 Choice of Strategy

With multiple strategies available, the natural question is determining circumstances under which each strategy is efficient. This depends heavily on the targeted quantity of interest. One distinguishing feature of a quantity of interest is whether it is valued over $[0, T]$ or just at T.

For the endpoint error, we estimate $(e(T), \psi)$ (see equation (6.1.10)), and observe that this appears to depend heavily on cancellation of error. We can have large positive and negative contributions, but if the two are nearly equal then we still have a final error that is small. This suggests that the weighted zonal strategy is the most efficient for the endpoint error. Since our concern is only the accuracy at the final time, we can leave intervals with higher contribution unrefined as long as we balance the positive and negative contributions.

On the other hand, for the average error, we estimate $\int_0^T (e, \psi)dt$ (see equation (6.2.3)). By computing on an interval by interval basis, we have

$$\begin{aligned} \int_0^T (e, \psi)dt &= \sum_{n=1}^{N} \int_{I_n} (e, \psi)dt \\ &\approx \sum_{n=1}^{N} (e(t_n), \psi(t_n))\Delta t, \end{aligned} \tag{8.4.1}$$

which is a weighted average of the endpoint errors. Our observation is that intervals that yield a large error contribution to an endpoint error, $(e(t_n), \psi(t_n))$, also have a large error and thus raise the average significantly. Consequently, the probabilistic strategy appears to lower the average error by refining those intervals where the error is large.

Figure 8.7 graphs the endpoint error for a sequence of simulations with ending times ranging from 0.1 to 4.0 in increments of 0.1 compared to the actual error computed as the difference between the exact solution and the approximation. Figure 8.8 shows the interval contributions to the average error. By comparing these graphs, we see that the contribution to the average error is high in the same regions where the endpoint error is high. Thus, by choosing intervals where the dual weighted interval contribution

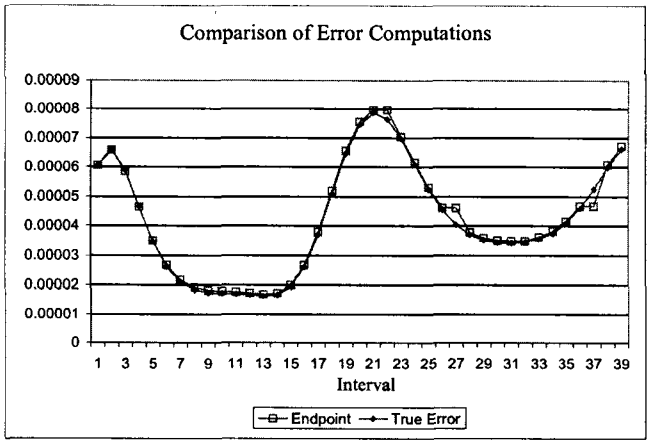


Figure 8.7: Comparison of Average Error Contributions and Endpoint Errors

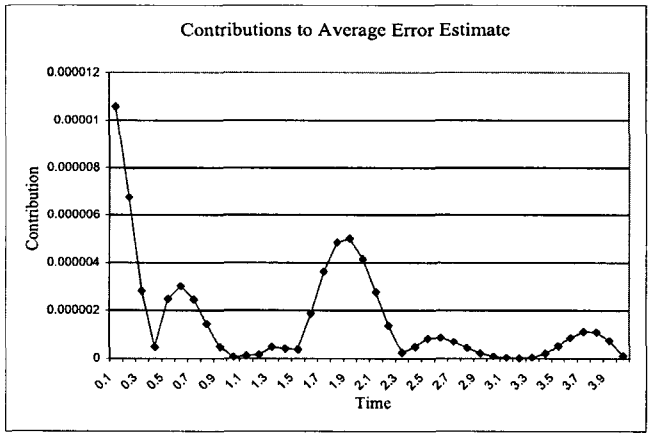


Figure 8.8: Comparison of Average Error Contributions and Endpoint Errors

Option	Endpoint	Average
Deterministic	887	694
Probabilistic	321	160
Weighted Zonal	321	383

Table 8.1: Interval requirements for each method to reach a specified tolerance.

is high, those intervals where the actual error is high are refined, lowering the average.

As a test problem we used example problem 10 with the initial step size set to 0.2 and the completion tolerance set to 1e-05. We see in the following table that the probabilistic and weighted zonal strategies both performed better than the deterministic method, and that the probabilistic method performed better than the weighted zonal strategy for the average error and worse for the endpoint error.

Chapter 9

TEST SUITE

In this chapter we list the suite of test problems and their purpose. The test suite was chosen based on varying degrees of complexity and the desire to test for specific results. For example, we chose simple problems where the computations could be made in other software packages as a means to validate the code, and we chose problems with changing stability to verify the adaptivity. Many problems used in testing the software and methods were also chosen for having known exact solutions. In addition, several test problems without exact solutions were chosen for their prominence in various other ODE test suites.

9.1 Test Suite

9.1.1 Problem 1: Exponential decay

The exponential decay problem is given by

$$\dot{y} = -\lambda y \tag{9.1.1}$$

where λ is a parameter, usually set to 1.0, and initial condition $y(0) = 1$. The exact solution is given by

$$y = e^{-\lambda t}. \tag{9.1.2}$$

This problem was chosen since the forward and dual problems have known solutions. In addition, this problem was used to verify convergence rates for the low and high order methods, and the computation of the discretization error. Since we compute $\int(f(y),v)dt = \int(-\lambda y, v)dt$, the problem has no quadrature error. Thus, all error is from the discretization.

9.1.2 Problem 2: Exponential decay with forcing term

The second test problem includes an oscillation forcing term that provides a way to test quadrature error. It is

$$\dot{y} + y = \sin(t), \tag{9.1.3}$$

with initial condition $y(0) = 1$, and exact solution

$$y = 1.5e^{-t} + 0.5(\sin(t) - \cos(t)). \tag{9.1.4}$$

The forcing term adds significant quadrature error to the problem.

9.1.3 Problem 3: A linear system

Problem 3 is given by

$$\begin{aligned} \dot{y}_1 &= y_1 + 4y_2 - e^t, \\ \dot{y}_2 &= y_1 + y_2 + 2e^t, \end{aligned} \tag{9.1.5}$$

with initial conditions $y(0) = [4.0, 1.25]^\top$ and exact solution

$$\begin{aligned} y_1 &= 4e^{3t} + 2e^{-t} - 2e^t, \\ y_2 &= 2e^{3t} - e^{-t} + 0.25e^t. \end{aligned} \tag{9.1.6}$$

Problem 3 was used to verify the proper functioning of the code with systems of equations.

9.1.4 Problem 4: The logistics problem

The logistics equation is a first order, nonlinear, autonomous differential equation used to model population dynamics, and is given by

$$\begin{cases} \dot{y} = ay - by^2, \\ y(0) = y_0, \end{cases} \tag{9.1.7}$$

with exact solution

$$y(t) = \frac{ay_0 e^{at}}{a - by_0 + by_0 e^{at}} \tag{9.1.8}$$

where a and b are positive parameters and $y_0 > 0$ is the initial condition. The ratio of the parameters, a/b, is called the carrying capacity and $\lim_{t\to\infty} p(t) = a/b$. While the logistics equation is easily solved, it provides an interesting example when discussing accuracy, quantity of interest, and the role of adaptive solvers.

9.1.5 Problem 5: A nonlinear problem with changing stability

This test problem is given by

$$\dot{y} + (0.25 + \sin(\pi t))y^2 = 0 \tag{9.1.9}$$

with initial condition $y(0) = 1.0$, with exact solution

$$y = \frac{\pi}{\pi + 1 + 0.25\pi t - \cos(\pi t)}. \tag{9.1.10}$$

Problem 5 was chosen for two reasons. First, it allowed testing of the nonlinear solver. Second, the changing stability property provided an excellent test case for the adaptive strategies.

9.1.6 Problem 6: A stable, nonlinear system

Problem 6 is given by

$$\begin{aligned} \dot{y}_1 &= -y_3 y_1 + y_2, \\ \dot{y}_2 &= -y_1 - y_2 y_3, \\ \dot{y}_3 &= y_4, \\ \dot{y}_4 &= -y_3, \end{aligned} \tag{9.1.11}$$

with initial conditions $y(0) = [1, 1, 1, 1]^\top$ and exact solution

$$\begin{aligned} y_1 &= (\cos(t) + \sin(t))e^{-1+\cos(t)-\sin(t)}, \\ y_2 &= (\cos(t) - \sin(t))e^{-1+\cos(t)-\sin(t)}, \\ y_3 &= \cos(t) + \sin(t), \\ y_4 &= \cos(t) - \sin(t). \end{aligned} \tag{9.1.12}$$

This problem was used to test the nonlinear solver on a system of equations.

9.1.7 Problem 7: An unstable system

This test problem is given by the equations

$$\begin{aligned} \dot{y}_1 &= \tfrac{y_1}{2(1+t)} - 2ty_2^2, \\ \dot{y}_2 &= \tfrac{y_2}{2(1+t)} + 2ty_1^2, \end{aligned} \tag{9.1.13}$$

with initial conditions $y(0) = [1.0, 0.0]^\top$ with exact solution given by

$$\begin{aligned} y_1 &= \sqrt{1+t}\cos(t^2), \\ y_2 &= \sqrt{1+t}\sin(t^2). \end{aligned} \tag{9.1.14}$$

Problem 7 was chosen to test both nonlinear systems and general adaptivity results.

9.1.8 Problem 8: The Vinograd problem

The Vinograd equations are given by

$$\dot{y} + A(t)y = 0, \tag{9.1.15}$$

where $A(t)$ is the matrix

$$\begin{pmatrix} 1 + 9\cos^2(6t) - 6\sin(12t) & -12\cos^2(6t) - \frac{9}{2}\sin(12t) \\ 12\sin^2(6t) - \frac{9}{2}\sin(12t) & 1 + 9\sin^2(6t) + 6\sin(12t) \end{pmatrix}, \tag{9.1.16}$$

and the initial conditions are $[-1, 3]^\top$. We note that the eigenvalues for A are 1 and 10, independent of time. We might conclude that, as for constant coefficients systems, the solution decays to zero with time. However, the exact solution is given by

$$u(t) = C_1 e^{2t} \begin{pmatrix} \cos(6t) + 2\sin(6t) \\ 2\cos(6t) - \sin(6t) \end{pmatrix} + C_2 e^{-13t} \begin{pmatrix} \sin(6t) - 2\cos(6t) \\ 2\sin(6t) + \cos(6t) \end{pmatrix}, \tag{9.1.17}$$

where C_1 and C_2 are constants. Since the second term decreases exponentially with time, the first term dominates the solution which, in fact, grows exponentially.

9.1.9 Problem 9: The two body problem

The Two-Body problem simulates the motion of a very light body orbiting a very heavy stationary body under the influence of the gravity. The Two Body problem is a Hamiltonian system given by

$$\begin{aligned} \dot{y}_1 &= y_3, \\ \dot{y}_2 &= y_4, \\ \dot{y}_3 &= \frac{-y_1}{\left(y_1^2+y_2^2\right)^{3/2}}, \\ \dot{y}_4 &= \frac{-y_2}{\left(y_1^2+y_2^2\right)^{3/2}}, \end{aligned} \tag{9.1.18}$$

with initial conditions $y(0) = [0.4, 0, 0, 2.0]^\top$. The exact solution for the two body problem is periodic and given by

$$\begin{aligned} y_1 &= \cos(\tau) - 0.6, \\ y_2 &= 0.8\sin(\tau), \\ y_3 &= \frac{-\sin(\tau)}{1-0.6\cos(\tau)}, \\ y_4 &= \frac{0.8\cos(\tau)}{1-0.6\cos(\tau)}, \end{aligned} \tag{9.1.19}$$

with $\tau : 0.6\sin(\tau) = t$. The solution is generally unstable, but weakly so, and there are regions of significant dynamic stability and instability in each period, while the overall accumulation is milder. The dG method is not well

suited for this problem. Since it has a dissipative character, it introduces a systematically biased numerical error. Energy conserving or symplectic schemes can work better in the sense of preserving accuracy [14], and we have implemented the standard Verlet scheme for Hamiltonian systems as an option. We will explore this in future work.

9.1.10 Problem 10: The Lorenz equations

The Lorenz equations are a well known example of chaos, see [15]. There is not steady exponential growth of error however, but periods of relative dynamix stability and instability that lead to relatively large periods of sub-exponential growth. There is a region near a separatix, however, which causes exponential growth. The solutions pass this region frequently. This is a good test of error estimation as there is no way to determine if a solution is in this region by looking at residuals or local error, see [15]. These equations are given by

$$\begin{aligned} \dot{y}_1 &= -\sigma y_1 + \sigma y_2, \\ \dot{y}_2 &= r y_1 - y_2 - y_1 y_3, \\ \dot{y}_3 &= -b y_3 + y_1 y_2, \end{aligned} \tag{9.1.20}$$

where σ, r, and b are parameters. We use $\sigma = 10.1$, $r = 28.0$, and $b = 8/3$. While the Lorenz equations do not have an exact solution, we chose this problem to compare against other known results. In particular, the equations were run and verified against an accurate Matlab computed solution.

9.1.11 Problem 11: The Hires problem

The hires problem is a chemical reaction problem involving eight reactands. It is used to describe high irradiance responses of growth and plant tissue differentiation. This problem is presented as a test problem in both

[44] and [25]. The Hires problem is defined by

$$\begin{aligned}
\dot{y}_1 &= -1.71y_1 + 0.43y_2 + 8.32y_3 + 0.0007,\\
\dot{y}_2 &= 1.71y_1 - 8.75y_2,\\
\dot{y}_3 &= -10.03y_3 + 0.43y_4 + 0.035y_5,\\
\dot{y}_4 &= 8.32y_2 + 1.71y_3 - 1.12y_4,\\
\dot{y}_5 &= -1.745y_5 + 0.43y_6 + 0.43y_7,\\
\dot{y}_6 &= -280y_6y_8 + 0.69y_4 + 1.71y_5 - 0.43y_6 + 0.69y_7,\\
\dot{y}_7 &= 280y_6y_8 - 1.81y_7,\\
\dot{y}_8 &= -280y_6y_8 + 1.81y_7,
\end{aligned} \tag{9.1.21}$$

with initial conditions $[1, 0, 0, 0, 0, 0, 0, 0.0057]^\top$.

9.1.12 Problem 12: The Oregonator equations

The Oregonator equations are a stiff system of nonlinear ODEs. These equations present very different time scales over which changes in the solution occur, both within a single component, and between components. They describe certain chemical reactions where oscillations in structure and color occur after an initial period of inactivity. The equations are given by

$$\begin{aligned}
\dot{y}_1 &= s(y_2 - y_1y_2 + y_1 - qy_1^2),\\
\dot{y}_2 &= \tfrac{1}{s}(-y_2 - y_1y_2 + y_3),\\
\dot{y}_3 &= w(y_1 - y_3),
\end{aligned} \tag{9.1.22}$$

where $s = 77.27$, $w = 0.161$, $q = 8.375 \times 1{-}^{-6}$, and the initial conditions are $[1, 0, 0]^\top$.

9.2 Additional Example Problems

Additional test problems were included in the code, but were not specifically used as test cases. They are provided as standard test problems included in other test suites or are of general interest. Some of these problems are the van der Pol equations, the Mathieu equations, a plankton/algae model and a molecular dynamics model.

Chapter 10

NUMERICAL EXAMPLES

We consider numerical examples for two purposes; verifying that the mathematics is correctly implemented and the software is built properly, and testing the ideas set forth in this thesis. Verification of the software includes checking that the computations are correct, verifying the convergence rates, and verifying the computation of the discretization and quadrature error. Once we are certain the software is correct, we look at experimental results concerned with the behavior of error and adaptive error control.

10.1 Implementation Verification

10.1.1 Solution methods

Verification of the forward and adjoint solvers was accomplished by choosing problems with known solutions and comparing the approximations with the exact solutions. Several problems were selected to cover all aspects of the forward and adjoint solvers, in particular, we use linear and nonlinear, scalar and multiple dimension problems with various stability properties. As an example, Fig. 10.1 shows the approximation plotted over the exact solution. The visual comparison between the true solution and the approximation was the first step in verifying the solution methods.

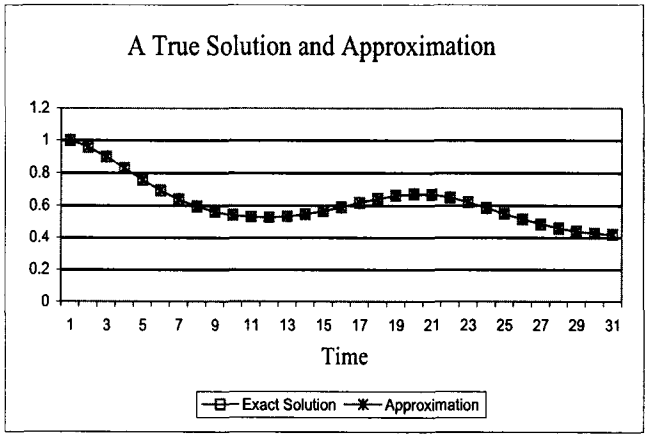


Figure 10.1: The approximation and exact solution for problem 5.

Low and High Order Method Error Results

h	Low Order	High Order
0.1	0.007522	2.02e-06
0.05	0.003749	2.56e-07
0.025	0.001871	3.21e-08
0.0125	0.000935	3.94e-09
0.00625	0.000467	4.07e-10

Table 10.1: Error results for the low and high order numerical methods.

10.1.2 Convergence rates

Problem 1 was used to verify convergence rates for the low and high order methods. Table 10.1 shows error results for a sequence of decreasing step sizes. Error values are given at the ending time, $T = 3.0$, for both the low order method and the high order method.

By plotting this data on a log-log graph with log(stepsize) on the x-axis and log(error) on the y-axis, we see slopes for the low order and high order lines are 1.00 and 3.07, respectively, which match our expectations for the backward Euler's method and the superconvergence results at time nodes

for the dG(1) method, equivalently the implicit Runge-Kutta based on two Radau points.

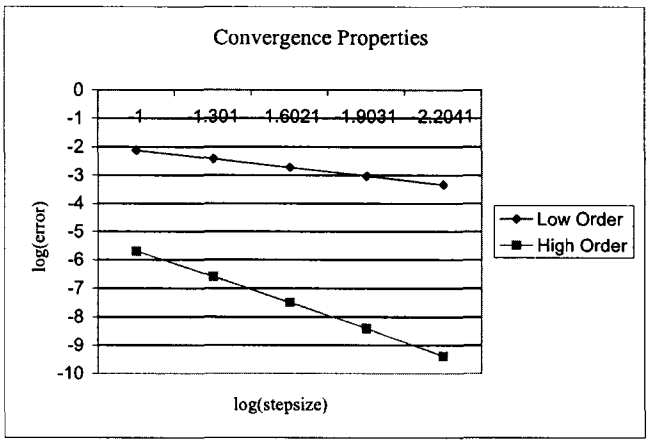


Figure 10.2: Convergence rates for both the low and high order methods.

10.1.3 Error computations

A visible way to test accuracy of error estimates are plots of the ratio of the error estimate to the true error. If the error estimate is accurate, we should see the ratio error estimate/true error close to 1.0.

In general, the error estimation includes both discretization and quadrature error. In order to isolate discretization error, we chose problem 1 as the first test as this problem has no quadrature error. Accurate results in the error computation verify the accuracy of the discretization error computation (see Table 10.2).

To verify the estimate for the quadrature error, we use problem 2, which illustrates the need for the quadrature error term. The problem was run with a constant step size of 0.1 for a sequence of ending times. As a test, we did a computation that neglected the contribution of the quadrature, which results in erratic ratios. The quadrature error estimate was then

Low and High Order Method Error Ratios

h	Low Order	High Order
0.2	1.201	1.019
0.1	1.100	1.009
0.05	1.050	1.004
0.025	1.025	1.004
0.0125	1.0125	1.018

Table 10.2: Error ratios for the low and high order numerical methods.

added to the code and the sequence rerun, yielding acceptable accuracy in the estimates. The following graph shows the estimate/error ratios with and without the quadrature estimates included. This clearly shows the need to estimate the quadrature error in order to achieve accurate estimates.

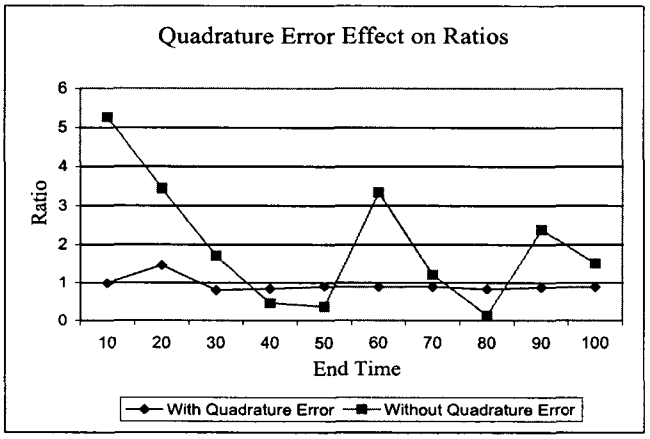


Figure 10.3: Estimate/error ratios for a variety of end times with and without quadrature error included.

We also used the Vinograd equations (problem 8) as a test for both solution and error computation. Recall that the Vinograd equations are given by

$$\dot{y} + A(t)y = 0, \tag{10.1.1}$$

where $A(t)$ is the matrix

$$\begin{pmatrix} 1+9\cos^2(6t)-6\sin(12t) & -12\cos^2(6t)-\frac{9}{2}\sin(12t) \\ 12\sin^2(6t)-\frac{9}{2}\sin(12t) & 1+9\sin^2(6t)+6\sin(12t) \end{pmatrix}. \quad (10.1.2)$$

We see in Fig. 10.4 that the solution for both components is oscillatory and exponentially increasing. Consequently, the endpoint error increases dramatically as the final time increases.

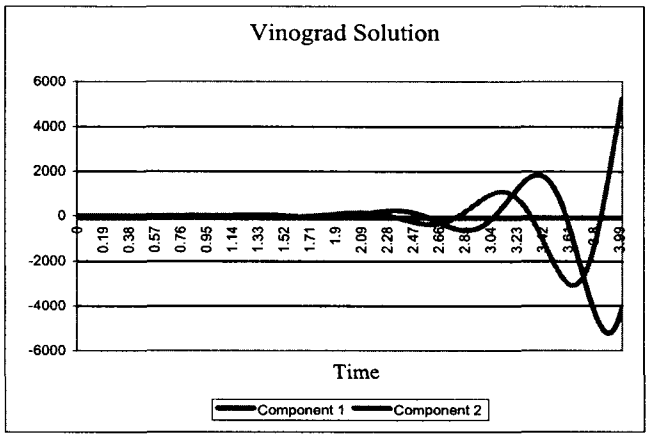


Figure 10.4: Solutions for the Vinograd equations.

We first use the dG(0)/cG(1) pair with a fixed ending time of 4.0 using and increasing numbers of time steps, i.e., a decreasing step size. Table 10.3 and Fig. 10.5 shows that the error at the end time decreases as the step size decreases and the estimate/error ratio also approaches 1. Note, however, that while the ratio approaches 1, a ratio of 1.2 when the component errors are in the thousands indicates a reasonably accurate estimate of the error.

10.2 Adaptivity Results

In this section, we consider stability analysis and adaptivity.

Error Results and Ratios

Steps	c1 Ratio	c1 Est.	c2 Ratio	c2 Est.
80	1.124	4525	1.217	-6190
160	1.109	3815	1.132	-4891
320	1.061	2576	1.067	-3233
640	1.031	1520	1.034	-1891
1280	1.016	828.9	1.017	-1028
2560	1.008	433.4	1.008	-536.2
5120	1.004	221.7	1.004	-274.0
10240	1.002	112.0	1.002	-138.5

Table 10.3: Ratios and error estimates for each component in the Vinograd equations at $T = 4$.

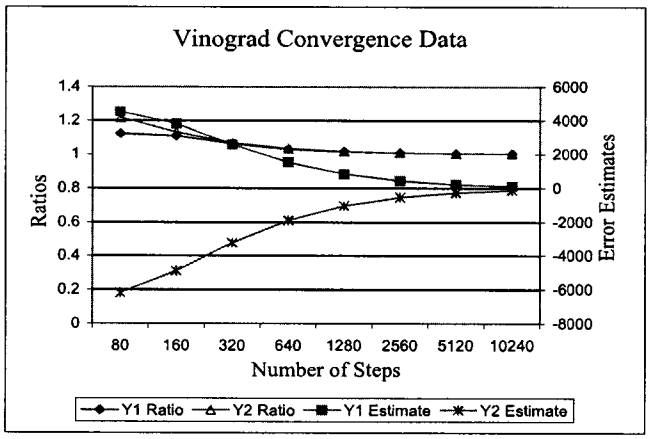


Figure 10.5: Convergence of ratios and solutions for the Vinograd equations.

10.2.1 The Logistics equation

For the logistics equation, we discuss the behavior of different adaptive strategies for the final time error and average error. We consider the logistics equation ($\dot{y} = ay - by^2$) with parameters $a = b = 2.309$ and $y_0 = 0.1$. The simulation was executed to an ending time of 3.0 with a stepsize of 0.1 for both the endpoint error and the average error. Results of simulations for endpoint and average error show values of -3.01e-06 and 2.2e-06, respectively. The interval contributions for both the endpoint and average error values are shown in Figs. 10.6 and 10.7 and the absolute values are shown in Figs. 10.8 and 10.9. For both quantities of interest, we see positive and negative contributions that lead to cancellation of error in the final value. Figures 10.10 10.11 show the accumulation of the contribution. For the average error, we see cancellation of the contributions over the first half of the simulation, with only a slight rise in the accumulation over the second half. The endpoint error contributions also show cancellation in the first half of the simulation, but here we see a more drastic increase in the accumulation over the second half.

Figures 10.12 and 10.13 show the solutions to the adjoint problems for both the endpoint and average error. Since the discretization error is factored by the adjoint weight $\overline{\Phi} - \pi_k \overline{\Phi}$ and the quadrature error by $\overline{\Phi}$, we see higher weights in the first quarter for the average error and in the last quarter for the endpoint error.

Traditional techniques for adaptivity have the user set a tolerance, compute a local tolerance by dividing the user tolerance by the number of steps, and refine all intervals where the absolute value of the contribution is greater than the local tolerance. This strategy is based entirely on the data

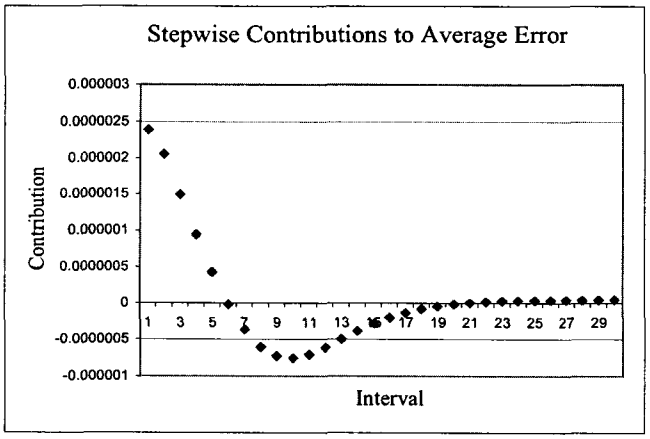


Figure 10.6: Stepwise Error Contributions for the Average Error Problem

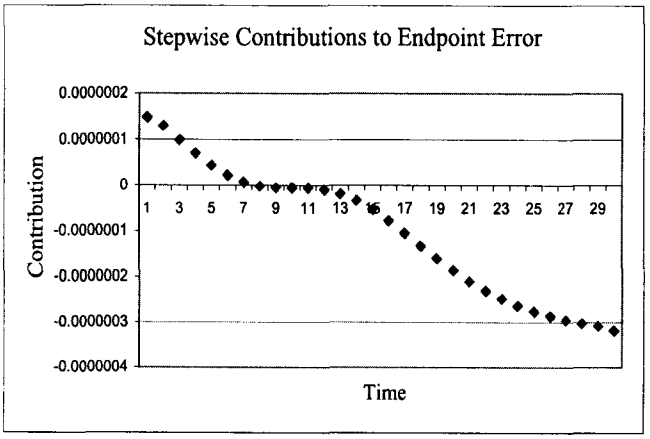


Figure 10.7: Stepwise Error Contributions for the Endpoint Error Problem

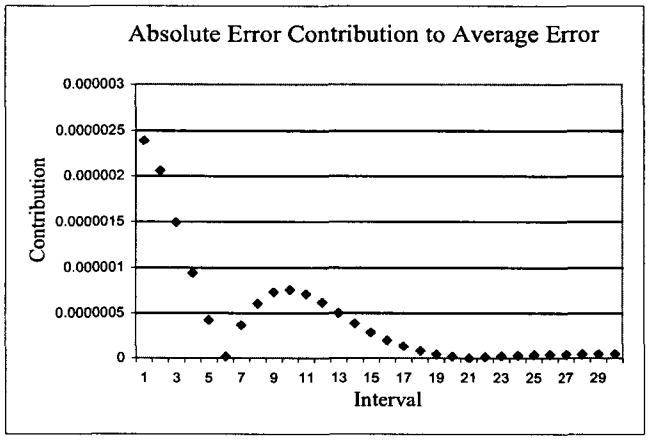


Figure 10.8: Absolute Value of the Error Contributions for the Average Error Problem

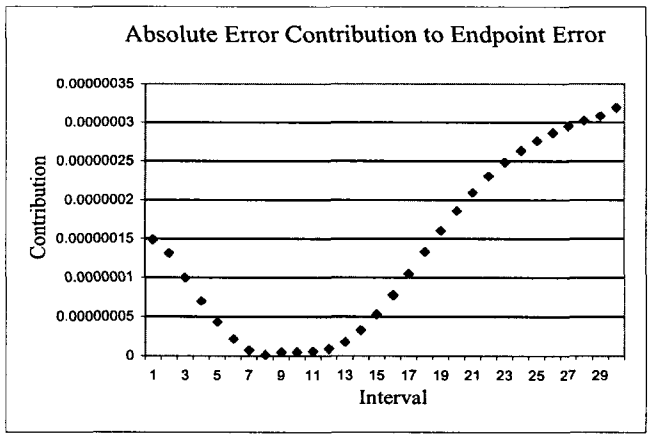


Figure 10.9: Absolute Value of the Error Contributions for the Endpoint Error Problem

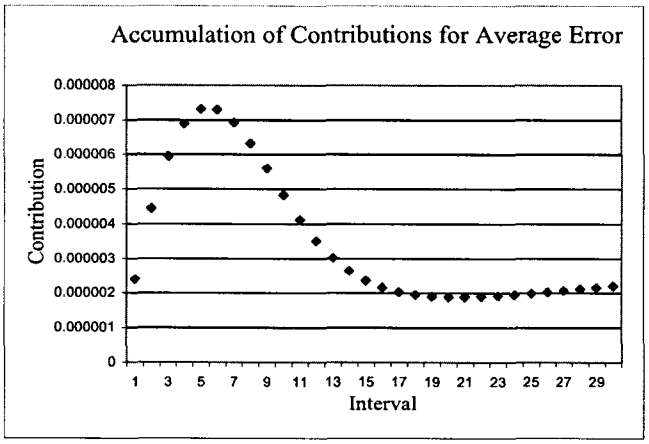


Figure 10.10: Accumulation of the Error Contributions for the Average Error Problem

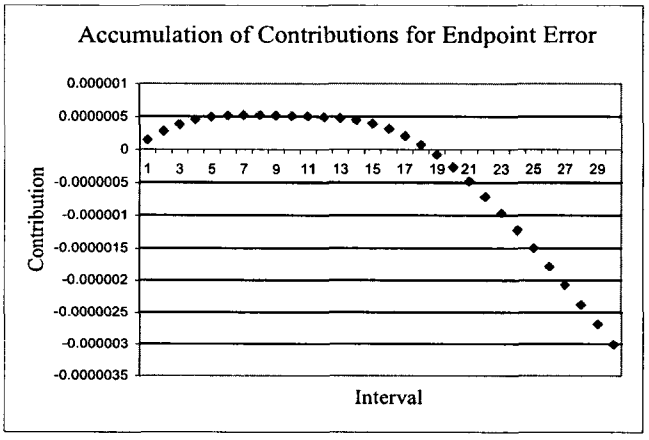


Figure 10.11: Accumulation of the Error Contributions for the Endpoint Error Problem

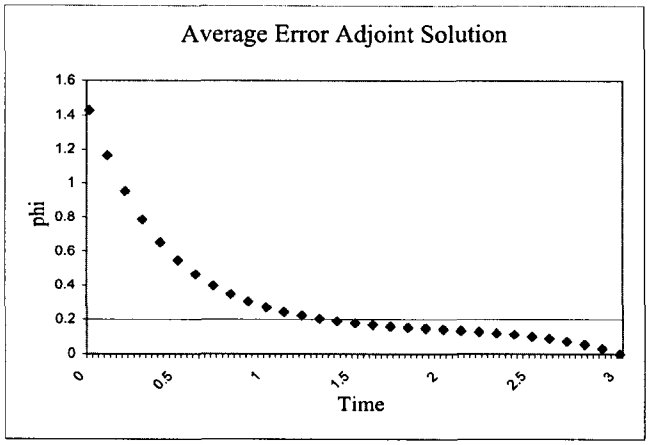


Figure 10.12: Solution to the Adjoint Problem for the Average Error Problem

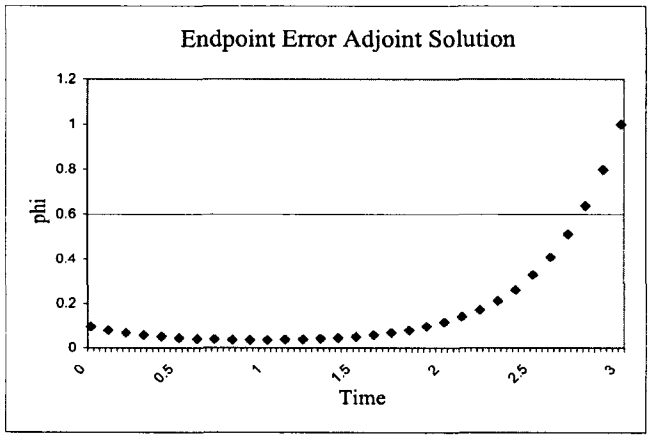


Figure 10.13: Solution to the Adjoint Problem for the Endpoint Error Problem

seen in Figs. 10.8 and 10.9. Since this method overestimates the error, it also overrefines the mesh, as seen in Fig. 10.14. The endpoint error data shows refinement would occur in the first quarter and last half of the simulation, while the average error data shows refinement would occur primarily in the first half.

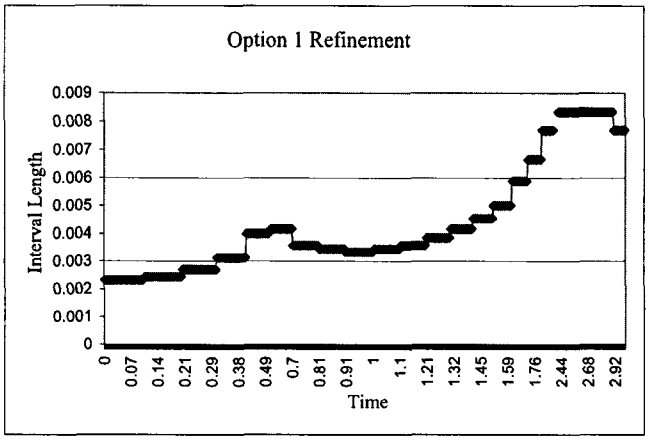


Figure 10.14: Resulting interval lengths after one refinement cycle using option 1.

In probabilistic adaptivity, we specify a number of the intervals to be refined and choose intervals to refine based on interval length and relative contribution of the interval. Thus, while a large interval has a higher probability of being chosen, if the relative contribution to the error is small, it may not be marked for refinement. However, since all intervals accumulate error, we do not exclude any intervals from the possibility of refinement.

Rather than simply refining intervals with contribution above some tolerance, it may be beneficial to refine while weighting areas based on their relative contribution. The accumulation of error in the average error simulation shows an increase in accumulation over the first five intervals with a subsequent cancellation over the following 11 intervals. We wish to

balance the refinement between these two zones so that the cancellation remains balanced. In this example, since the intervals start uniform, the probabilistic approach favors those with the largest contribution, thus an interval in the zone of positive contribution is more likely to be refined than an interval in the zone of negative contribution. Also note that from 1.6 to the end of the simulation there is very little accumulation. Since the relative contribution of each interval in this period is low, the probability of refinement is low, but not excluded entirely. In Figs. 10.15 and 10.16 we see the results of one refinement cycle for options 4 and 5, respectively. In both cases, the method uses just over 10% the number of intervals used by option 1.

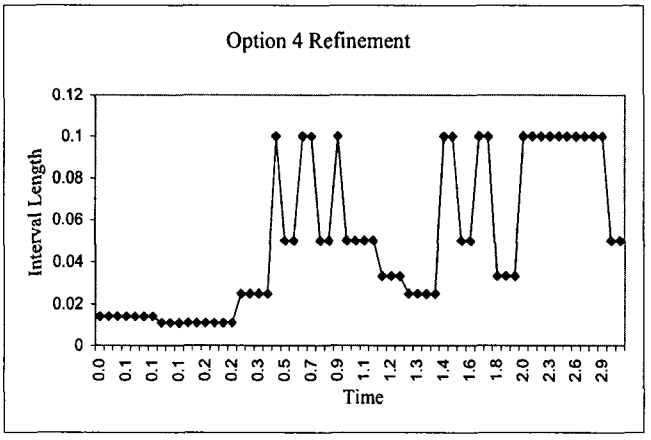


Figure 10.15: Resulting interval lengths after one refinement cycle using option 4.

10.2.2 Application to modeling biokinetic enzymes

Biology is the study of life, including the study of microscopic cells and their chemical reactions. Enzymes are specialized protein molecules that increase the rate of a chemical reaction by lowering the activation energy

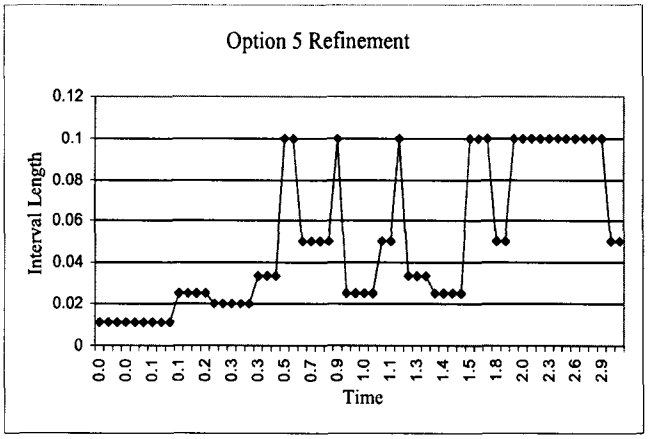


Figure 10.16: Resulting interval lengths after one refinement cycle using option 5.

of the reaction they effect. This increase in the rate of reaction causes a dramatic change in the chemical process. An in-depth look at enzymes can be found in many texts on biology and biochemistry including [31] and [22]. In this thesis, we simulate this change by using the logistics with changes in the parameters and starting conditions. Figure 10.17 shows the solution to the logistics equation ($\dot{y} = ay - by^2$) with the parameters changed to $a = 20$ and $b = 2$, and the initial condition changed to $y_0 = 0.00001$. We see a dramatic change in the solution from $t = 0.5$ to $t = 0.8$. This is similar in scope to the change expected in a catalytic reaction.

In Fig. 10.18, we see the solution for an initial uniform mesh of $k = 0.1$ compared to refined meshes resulting from two different tolerances. While the coarse mesh solution captures the final time solution and the trend of the increase, it has a lag when the increase occurs and, consequently, fails to capture the correct behavior through the region of increase. Using average errors and final tolerances of 0.01 and 0.001 show convergence to the exact solution. Thus, the refined solutions capture the behavior of the increase.

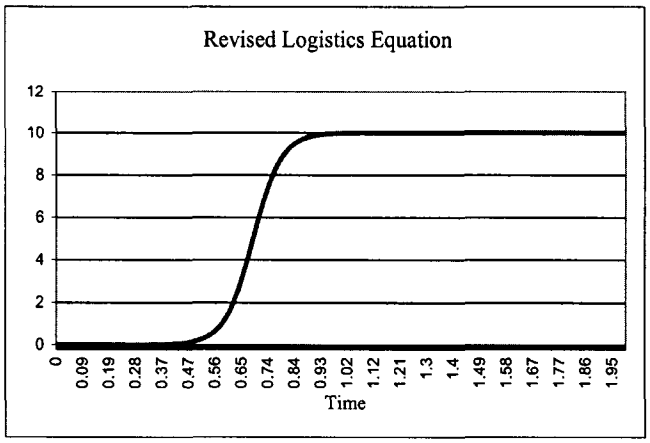


Figure 10.17: The logistics equation with the parameters changed to simulate biokinetic enzymes.

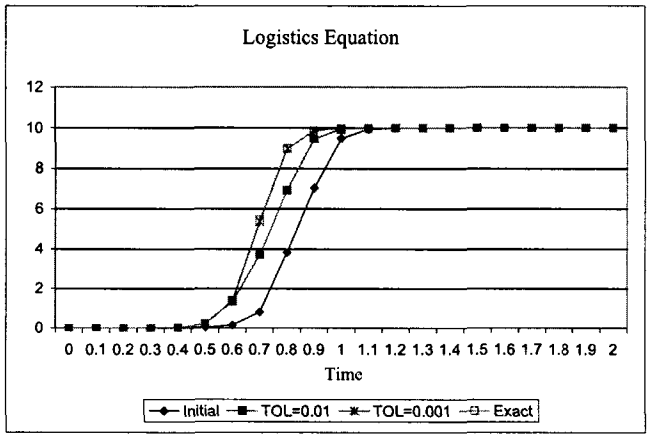


Figure 10.18: Refined solutions compared to the initial and exact solutions.

The solution to the adjoint problem for the average error, Fig. 10.19, indicates that refinement should occur in the first half of the simulation. The algorithm refines to keep the solution accurate approaching the unstable region around $t = 1.0$. We see in Fig. 10.20 that the refinement occurs exclusively in the first half. By $t = 0.09$ we see no refinement. Thus, early refinement keeps the solution accurate through the period of rapid increase, while no refinement is necessary once the solution approaches the final value.

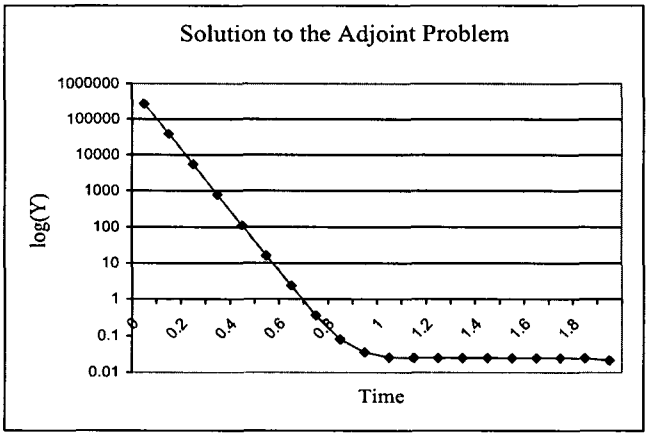


Figure 10.19: The adjoint solution for the average error simulation.

10.2.3 A nonlinear problem with changing stability

Problem 5 is a nonlinear problem with changing stability. Consequently, this problem exhibits both regions of accumulation and cancellation of error. In Fig. 10.21, we see the results of perturbing the initial conditions for this problem. Figure 10.22 shows the width of the envelope containing these perturbed solutions. As we move through the regions of stability and instability, we see the envelope shrink and grow accordingly.

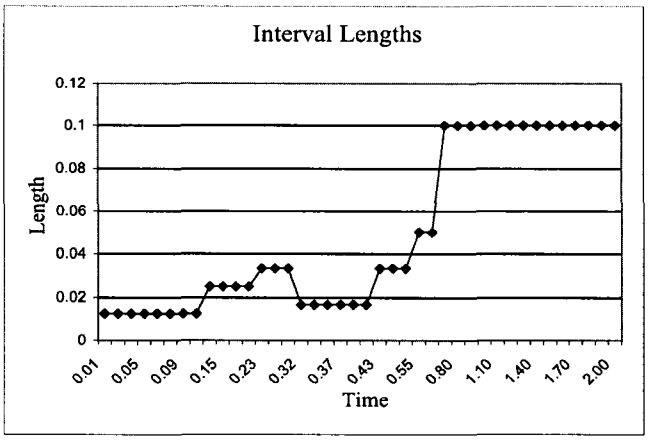


Figure 10.20: Interval lengths for the revised logistics problem after adaptive refinement.

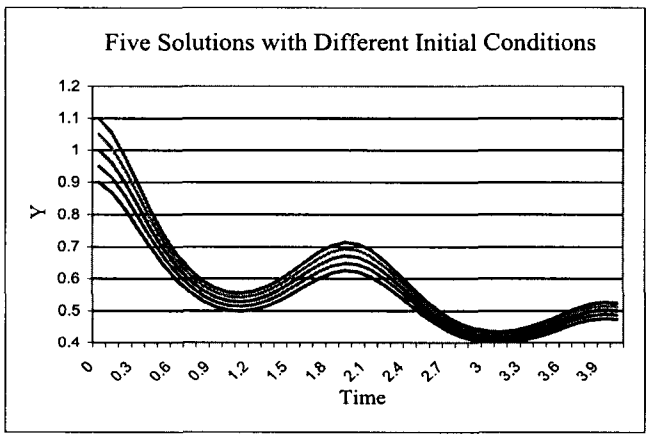


Figure 10.21: Solutions to problem 5 with perturbed initial conditions.

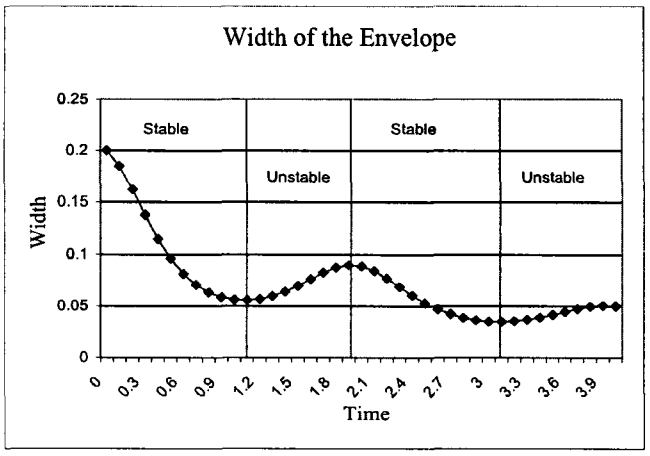


Figure 10.22: Width of the envelope containing the perturbed solutions.

The first test ran the problem over a short time span encompassing only two zones and represents the first detailed look at the weighted zonal strategy. We used an ending time of 1.0 with a time step of 0.05. The problem was run using three of the refinement options: equidistribution of error, probabilistic based on both contribution and step size, and the weighted zonal strategy. The refinement tolerance was set to 10^{-7}. Figure 10.23 shows the no refinement solution and interval contributions. We see a zone of large negative contribution followed by a zone of smaller positive contribution resulting in a net estimate of -2.41×10^{-7}. Results from the simulations show all three refinement options completing in three iterations.

Refinement option 1 required 76 timesteps when the simulation reached the final tolerance and showed almost uniform refinement with small error contributions at the start and finish cancelling out. Since the refinement option looks only at the absolute value of the contribution, all intervals are weighted equally, and since the overall tolerance is set small, most intervals fall into the refinement bin.

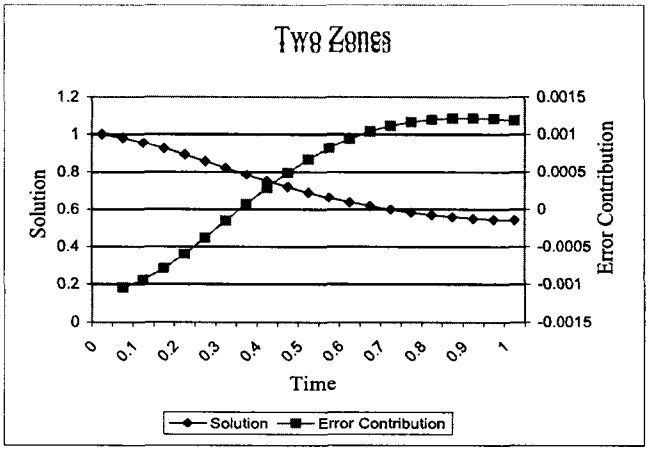


Figure 10.23: Solution and interval contributions for problem 5.

Figure 10.24 through 10.27 shows the results from refinement options 4 and 5. For option 4, since we compare the absolute value of the error contribution against the tolerance, we still expect to see a more uniform refinement. However, since the choice of intervals is a weighted random choice, we see less refinement than option 1. In fact, while option 1 refined almost all intervals a second time, option 4 resulted in only 12 intervals of length 0.0125. Both options refined all, or all but 1 in the case of option 4, intervals after the first iteration. While option 1 then refined all but 5 intervals after the second iteration, we see option 4 refined only 6. Option 5 tries to balance refinement between zones of positive and zones of negative contribution. As shown in Fig. 10.23, we see the negative zone followed by the positive zone. The total negative contribution of -3.76E-06 is almost three times the total positive contribution of 1.34E-06, and occurs in 8 versus 12 intervals, respectively. As such, during the first iteration, the algorithm splits refinement intervals 60/40 in favor of the negative zone. Consequently, we see more refinement in the negative zone as the algorithm tries to balance the positive and negative contributions.

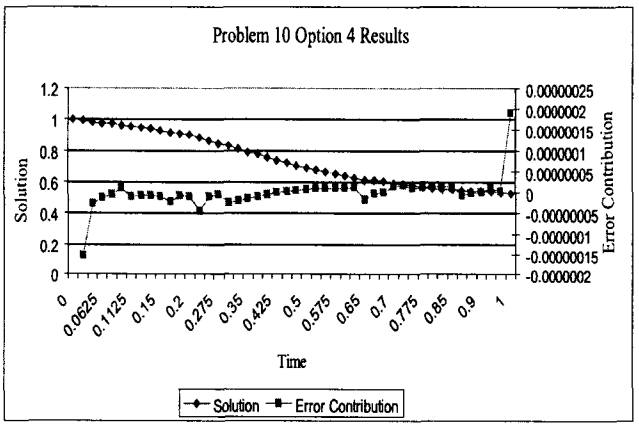


Figure 10.24: Interval contributions after reaching tolerance for option 4.

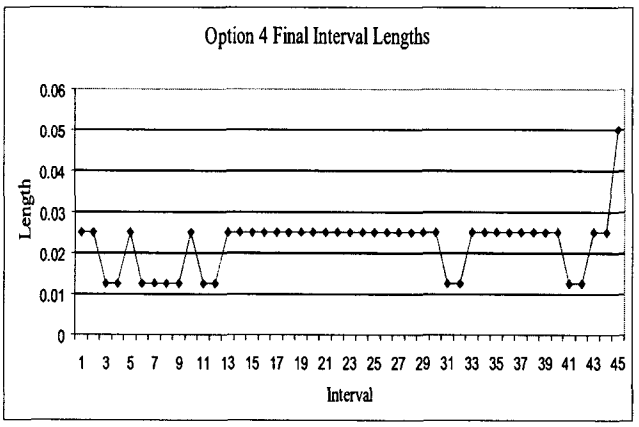


Figure 10.25: Interval lengths after reaching tolerance for option 4.

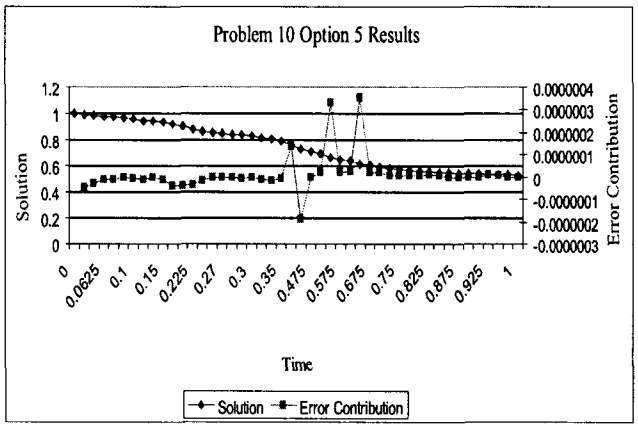


Figure 10.26: Interval contributions after reaching tolerance for option 5.

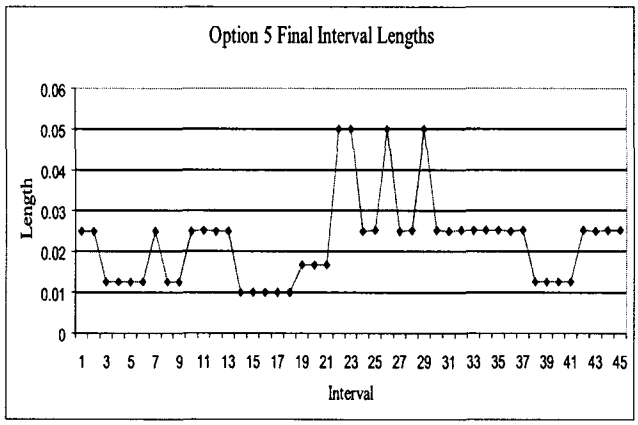


Figure 10.27: Interval lengths after reaching tolerance for option 5.

While this simple example shows little difference between options 4 and 5, it clearly shows more efficiency in acheiving the desired tolerance for both options 4 and 5 over option 1.

If we extend the time to $T = 4.0$, the changing stability introduces additional zones into the problem. Figures 10.28 and 10.29 show periods of positive and negative error contribution and, clearly, we see the negative zones contribute more to the error than the positive zones. This is verified by Figs. 10.30 and 10.31, which show an end result of a negative error value. Consequently, we refine more heavily in the zones of negative contribution, even though the zones of positive contribution comprise a larger part of the domain.

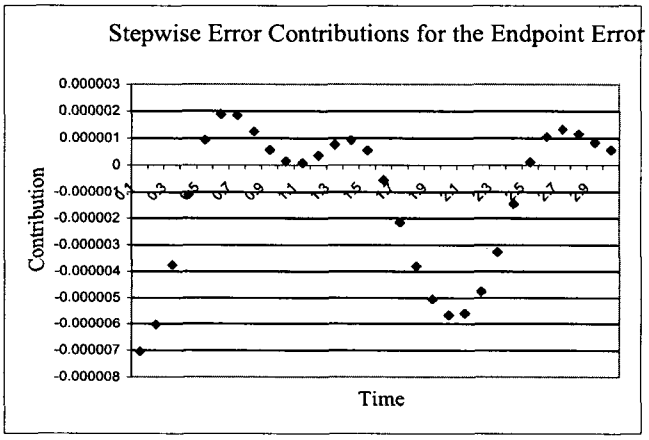


Figure 10.28: Endpoint error stepwise contributions for problem 5.

We then moved to a longer time span. The problem was run to $T = 4$ and results for the three options were compared. Table 10.4 shows the final interval counts for each of the refinement options. Since option 1 does not consider the cancellation of error while both options 4 and 5 do, we see a significant improvement in efficiency with option 4 requiring only 24% and option 5 20% of the number of intervals used by option 1.

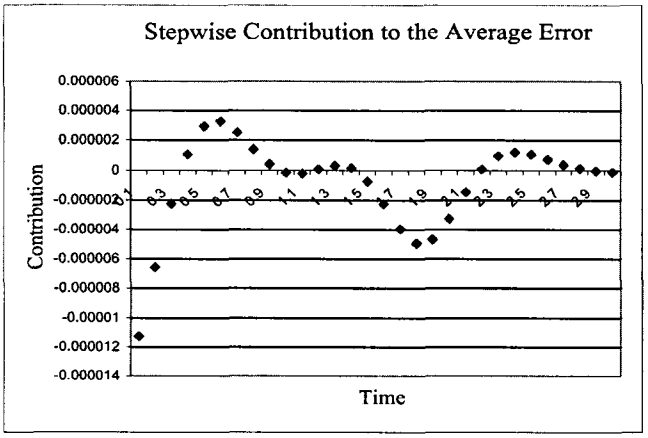


Figure 10.29: Average error stepwise contributions for problem 5.

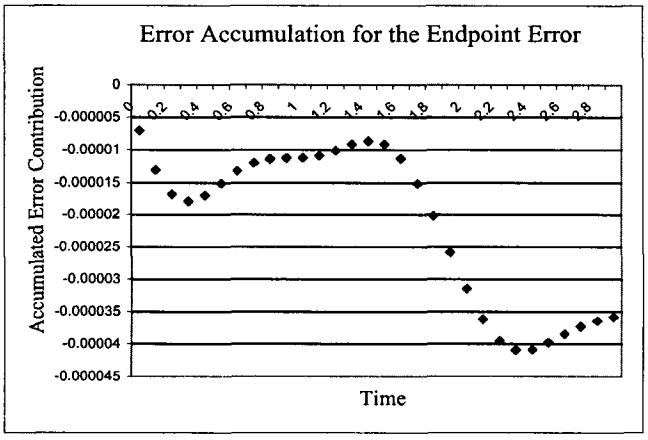


Figure 10.30: Endpoint error accumulation for problem 5.

Method	Final Interval Count
Option 1	886
Option 4	214
Option 5	180

Table 10.4: Final interval count after refinement using each of three options.

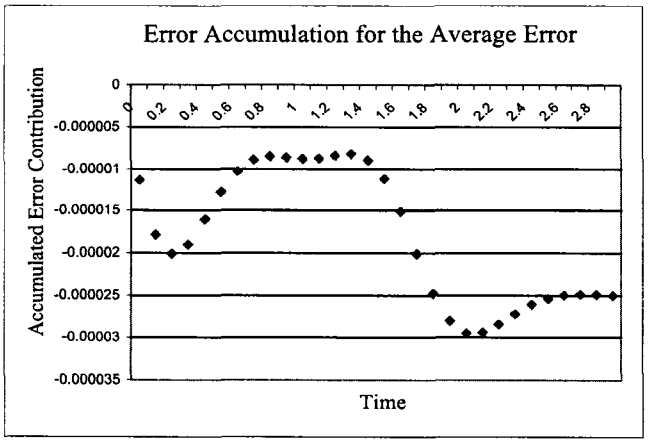


Figure 10.31: Average error accumulation for problem 5.

10.2.4 An unstable system

Problem 7 provides an unstable system. The solution exhibits oscillatory behavior with slow growth in both components. In Figs. 10.32 and 10.33, we plot numerical approximations for two different step sizes. It is clear that the wrong step size leads to an incorrect solution to the problem. In particular, Fig. 10.32 shows that, with a coarse time step, the solution damps to zero in both components. By reducing the stepsize, we plot a better approximation to the solution in Fig. 10.33.

The behavior of the stepwise error contributions, Fig. 10.34 show oscillations along with exponential growth. There are a small number of intervals that contribute a negative value, consequently, we expect the average error results to be similar for both options.

We ran problem 7 to $T = 7.0$ with equivalent tolerances for options 4 and 5. In fact, we see for this test the average error results are identical while the endpoint error results show a significant increase in efficiency in option 5, using only 45% the number of intervals as option 4.

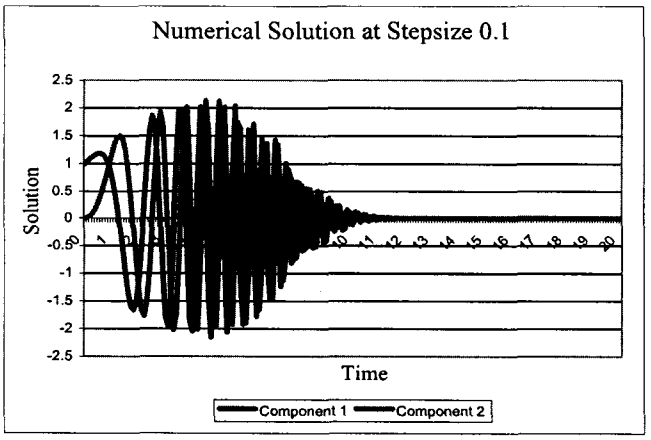


Figure 10.32: Numerical solution to problem 7 at step size of 0.01.

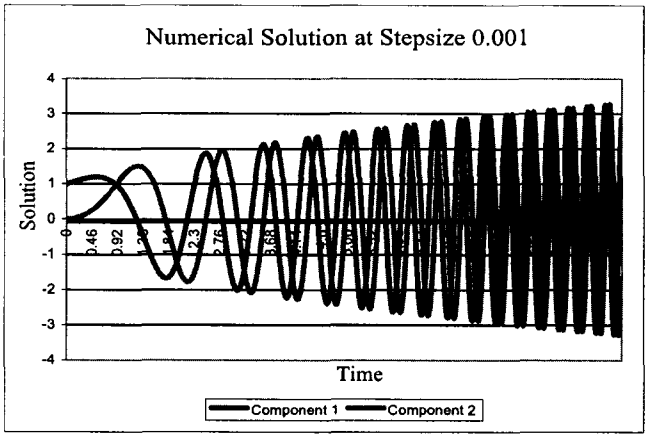


Figure 10.33: Numerical solution to problem 7 at step size of 0.001.

Method	Endpoint	Average
Option 4	56002	7002
Option 5	25206	7002

Table 10.5: Final interval count after refinement for each quantity of interest using options 4 and 5.

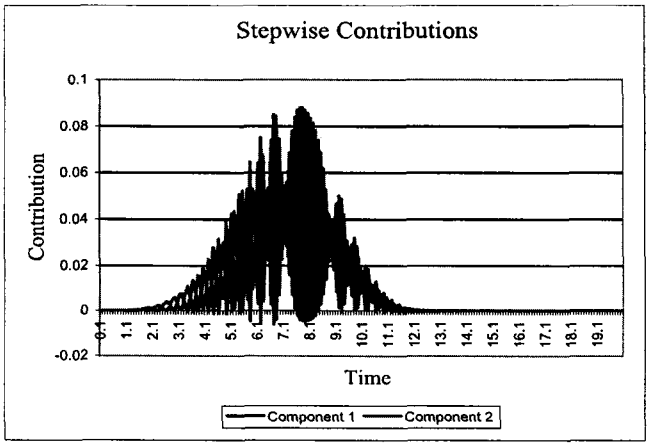


Figure 10.34: Stepwise interval contributions for problem 7 with initial step size of 0.001.

10.2.5 The two body problem

The Two-Body problem simulates the motion of two orbiting bodies and the influence of the gravitational fields. This problem presents a unique challenge when using a dissipative scheme like the dG method because the method induces a bias that results in systematic decrease in accuracy. Figure(10.35) shows the approximation of the first and second components (the positional components) of the low order solution run to an ending time of 26. The true solution is periodic while the numerical solution shows the gradual deterioration of the solution.

In Fig. 10.36 we take a different look at the solution (component 1 only). We see the deterioration as both a shortening in period and dampening of the amplitude. Note, also, that the solution to the dual problem is synchronized with the approximation.

Figure 10.37 shows the stepwise contributions for component 1. Clearly, the major contribution to the error occurs in a neightborhood of the 2π point. We see further in Figs. (10.38) and (10.39) that, not only does the

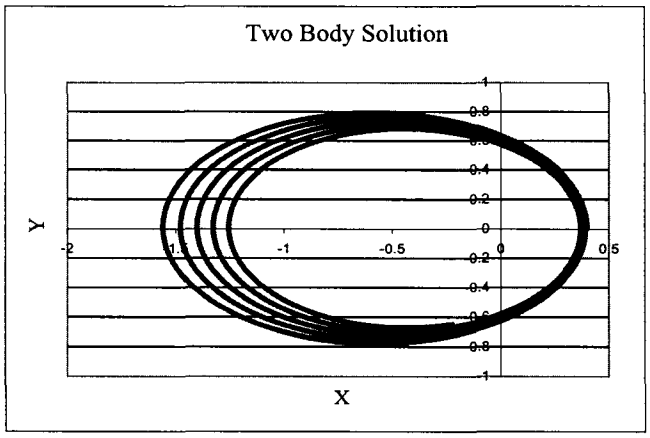


Figure 10.35: The numerical solution of the Two Body problem showing deterioration.

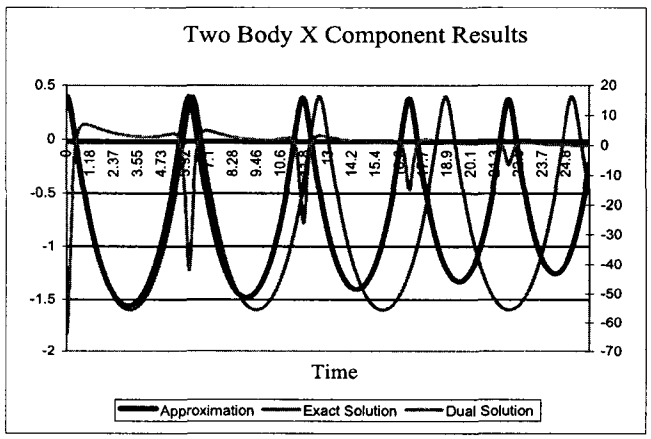


Figure 10.36: Comparison of the numerical solution and the exact solution for the X component.

error accumulate on the 2π cycle, but the accumulation decreases over time. Clearly, Fig. 10.39 shows that the error approaches some limit. This indicates that refinement will occur within localized regions, but local regions in the early parts of the simulation will be refined more than the later parts.

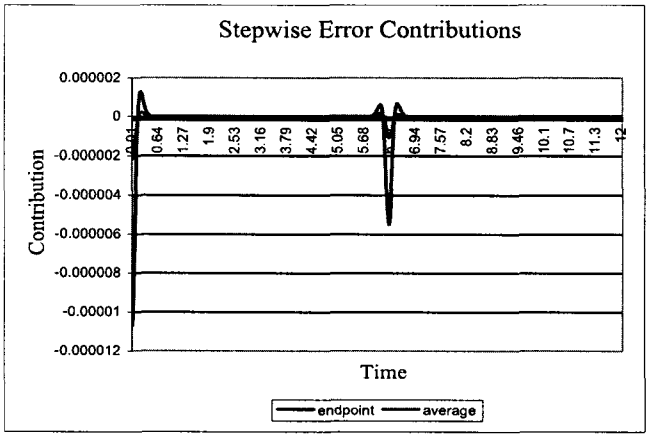


Figure 10.37: Stepwise interval contributions for the X component.

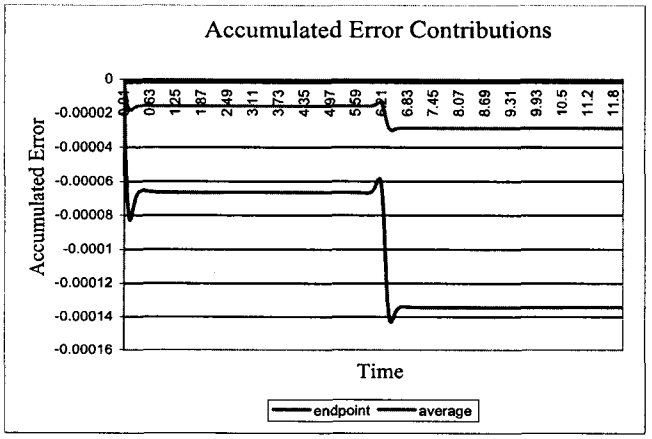


Figure 10.38: Accumulated interval contributions for the X component.

We note that similar results are seen for the Y component and the velocity components.

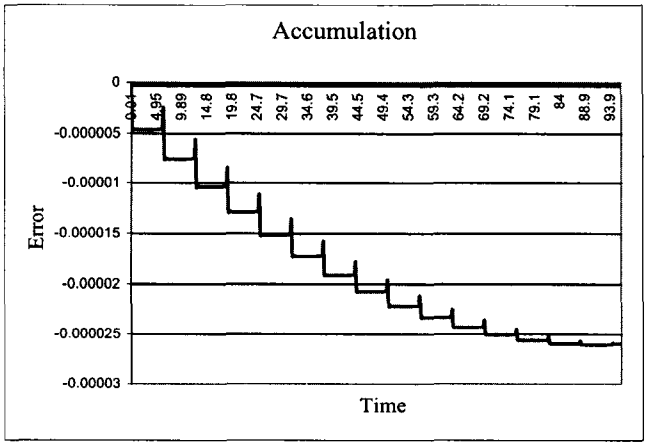


Figure 10.39: Accumulated interval contributions for the X component over a long time period.

Another interesting aspect of the two body problem is the effects of ending time on the stability factors and, consequently, the interval contributions for the problem. When running simulations for different ending times, we noted that the interval accumulation was sometimes positive and sometime negative. Figure 10.40 shows a sequence of accumulated error plots for simulations with ending times from 16.5 to 17.2 in increments of 0.1. This is caused by changes in the dual solution as the ending time of the simulation is changed. Since the dual solver runs backwards in time, the initial condition of 1.0 for the dual solve changes it's location and affects the dual solution.

As mentioned above, we expect refinement to occur within a neighborhood of integer multiples of 2π. We ran the two body preoblem to an ending time of $T = 21.0$ with stepsize 0.01 and completion tolerance of 0.0001. Both options completed in one refinement cycle with endpoint error values of 1.59E-05 for option 4 and 5.88E-06 for option 5. We also see a similar pattern for refinement with both options. Option 4 shows

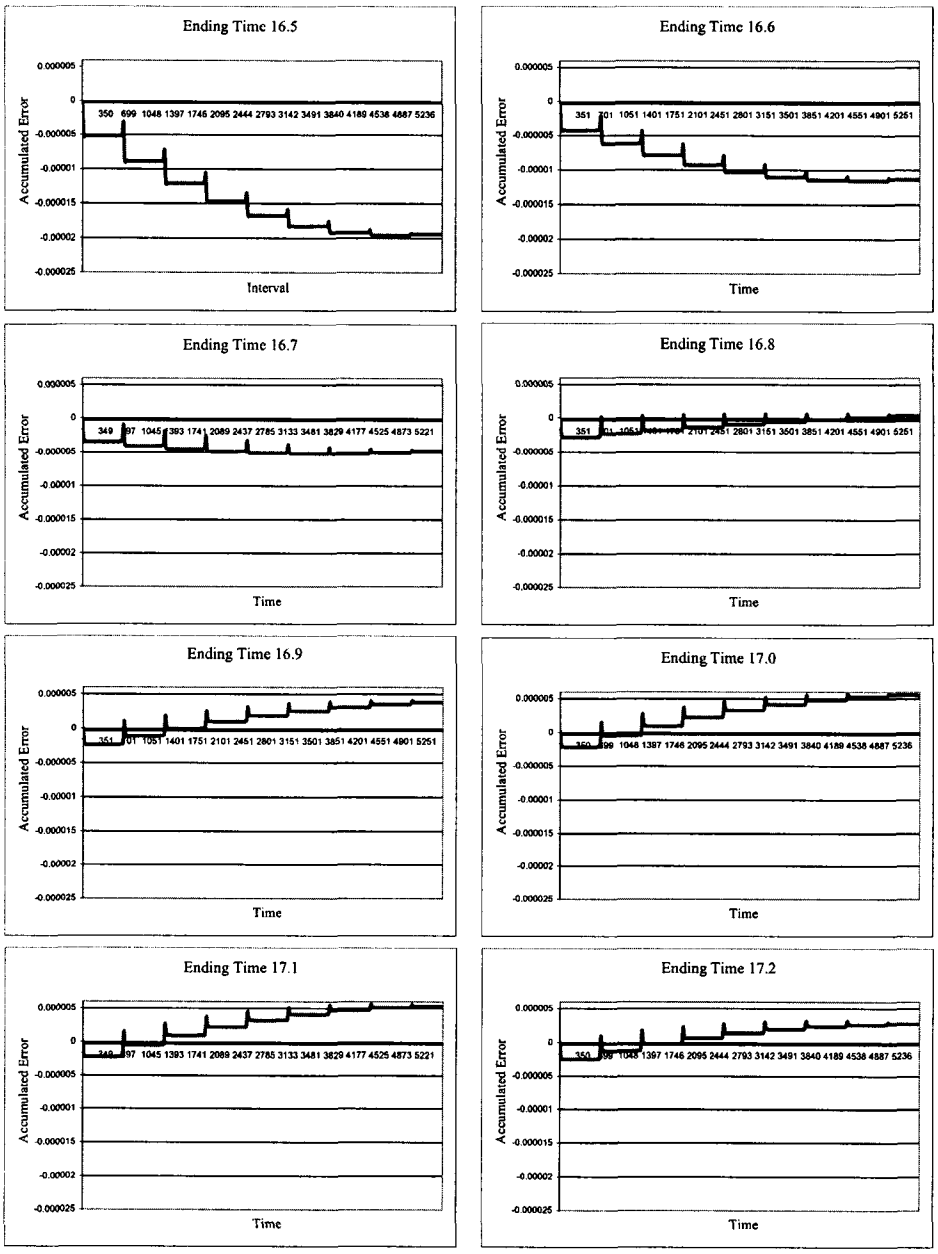


Figure 10.40: Sequence of accumulated error plots for simulations with ending times 16.5 through 17.2.

slightly more refinement away from 2π, but both show heavy refinement in the expected neighborhoods. Both options completed using 4202 intervals.

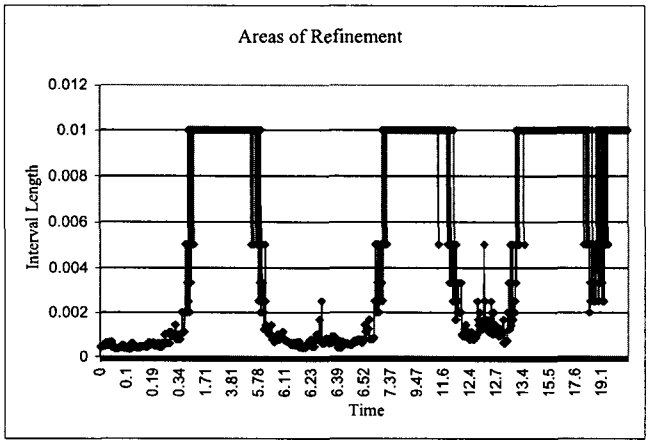


Figure 10.41: Regions of refinement for the Two Body problem using option 5.

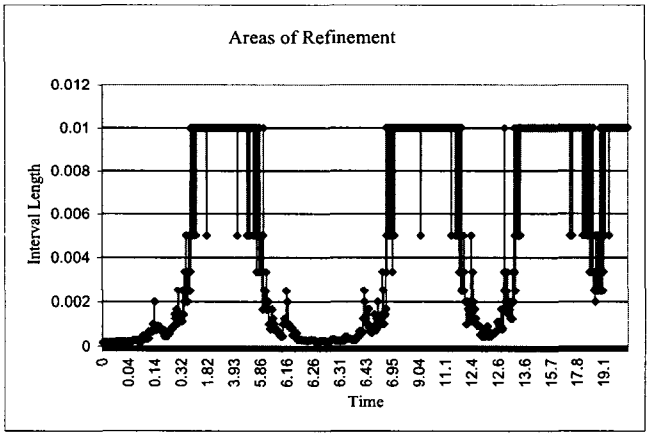


Figure 10.42: Regions of refinement for the Two Body problem using option 4.

10.2.6 The Lorenz equations

Since we don't have a true solution of the Lorenz equations, we verify the approximations using a numerical solution obtained using Matlab

with a very large number of time steps. Figure 10.43 shows the solution of our simulation and Fig. 10.44 shows the difference between the simulation and the reference approximation run in Matlab. Reduction of the tolerances given to Matlab brought the Matlab solution closer to our solution, indicating a more accurate solution from our computer code.

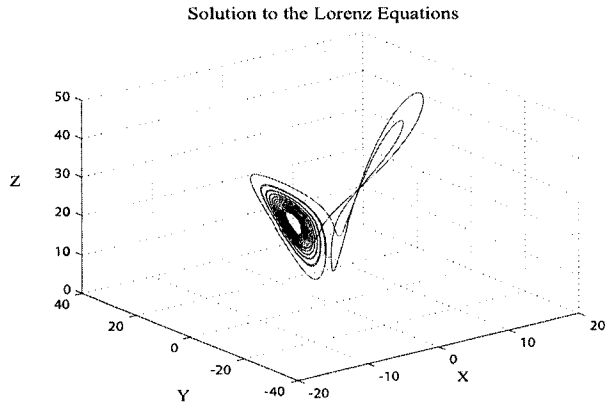


Figure 10.43: Numerical solution to the Lorenz equations.

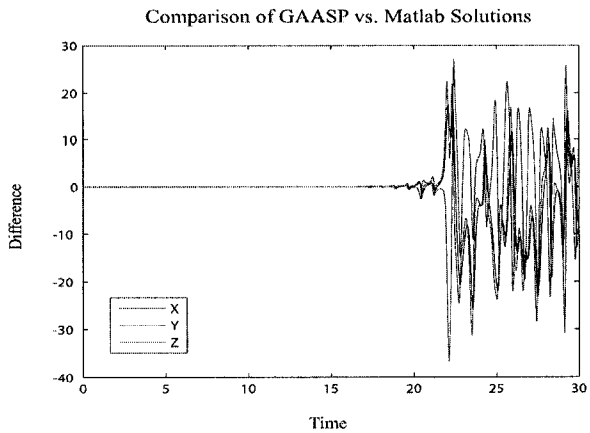
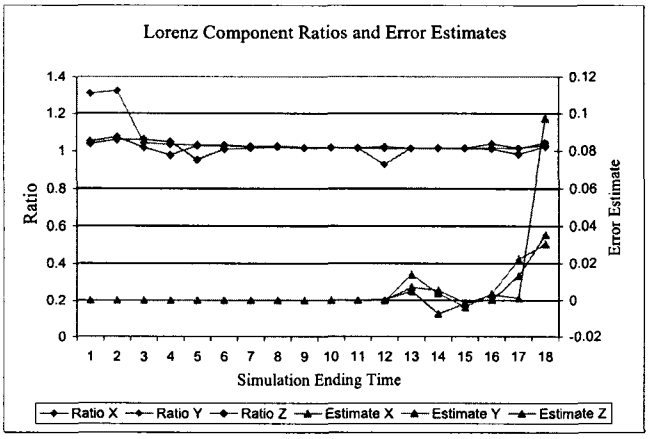


Figure 10.44: Difference between the numerical solutions produced by GAASP and Matlab.

Figure 10.2.6 shows ratios of the error estimates and actual error where the actual error is computed by running the simulation at a time step

of 0.001 and 0.0001. The endpoint error estimate was compared to the difference in the coarse and fine mesh solutions. We see ratios consistently near 1, showing accurate estimates. We also see a dramatic increase in the actual errors at time $t = 13$. We note that Estep et.al. in [16] shows an accurate solution until $t \approx 17.8$ followed by 100% error for $t \geq 17.8$. For our simulation, an initial run was made and new initial values extracted at $t = 5$. This became the new starting point for our simulation. We see the increase in error at $t = 13$ matches the results in [16].



Ending Time	X Estimate	X Ratio	Y Estimate	Y Ratio	Z Estimate	Z Ratio
1	-2.156E-07	1.052	-3.052E-08	1.309	4.577E-07	1.038
2	3.933E-07	1.074	2.420E-08	1.324	-1.157E-06	1.060
3	3.074E-07	1.019	1.308E-06	1.042	8.935E-07	1.060
4	-2.032E-07	0.976	-2.281E-06	1.034	-2.522E-06	1.048
5	2.313E-06	1.027	3.703E-06	1.032	-4.220E-07	0.953
6	-4.633E-06	1.027	-7.986E-06	1.032	2.289E-06	1.009
7	7.942E-06	1.023	1.105E-05	1.025	-5.595E-06	1.015
8	-1.674E-05	1.023	-1.246E-05	1.027	2.202E-05	1.021
9	2.697E-05	1.019	4.024E-05	1.019	-1.798E-05	1.015
10	-7.818E-05	1.018	-8.348E-05	1.018	8.067E-05	1.017
11	4.277E-05	1.014	8.560E-05	1.014	9.109E-05	1.016
12	3.604E-06	0.929	-5.439E-05	1.024	2.174E-04	1.012
13	7.230E-03	1.014	1.393E-02	1.014	5.122E-03	1.013
14	5.721E-03	1.015	3.797E-03	1.015	-7.348E-03	1.014
15	-1.526E-03	1.016	-3.751E-03	1.015	-1.883E-03	1.013
16	3.118E-04	1.039	3.765E-03	1.014	3.238E-03	1.010
17	1.333E-02	1.014	2.245E-02	1.014	1.088E-03	0.982
18	3.545E-02	1.030	3.043E-02	1.044	9.748E-02	1.023

Table 10.6: Error estimates, actual errors, and ratios for the Lorenz equations at various ending times.

Stepwise error contributions indicate the refinement occurs in the time mesh prior to the jump in solution error. However, Fig. 10.46 shows the solution trajectories in the XY plane. We see a manifold where small errors in the trajectory can lead the solution to the wrong fixed point. Tests with adaptivity have shown no improvement. If the trajectory is extremely close to the manifold, it may not be possible to refine the early part of the simulation enough to prevent the jump to the wrong fixed point.

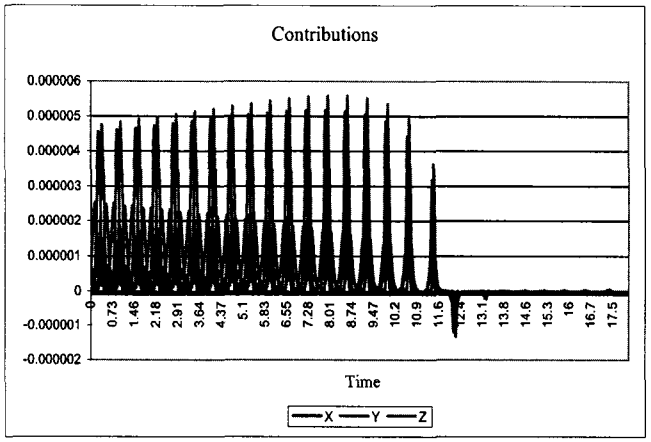


Figure 10.45: Stepwise contributions to the estimate.

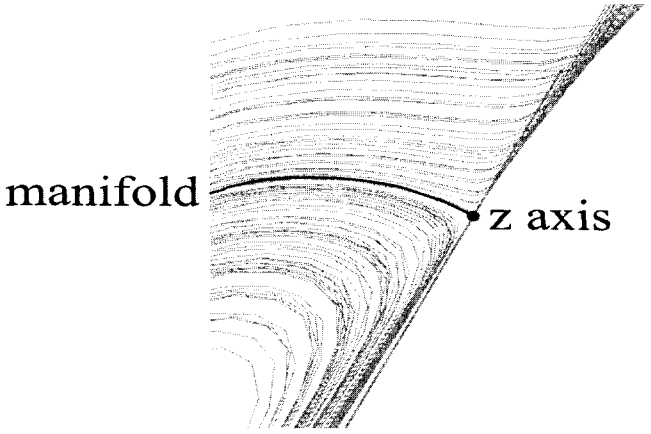


Figure 10.46: Region of extreme sensitivity.

10.3 Some Standard ODE Test Suite Problems

This section looks at the results of test problems 11 and 12. These problems are common in ODE test suites and descriptions can be found in [44] and [25].

10.3.1 Hires

The Hires problem was run to an ending time of 5 with an initial step size of 0.05. The solution is shown in Fig. 10.47 for all eight components. The stiffness of the problem is illustrated specifically in the differences between the sets of components 2, 7, and 8, and 5 and 6. Components 5 and 6 show slow growth over the given time period, while 2, 7, and 8 all show an extreme change over a short time span.

The Hires problem was run to an ending time of 5 with an initial stepsize of 0.2 for both endpoint and average error. Ending tolerance was set to 10^{-8}. Figures 10.48 and 10.49 show contributions to the average and endpoint error by component and total.

Contributions to the average error are primarily from $t = 0$ to 1, while for the endpoint error we see contributions throughout the time domain. As such, we expect less refinement for the average error. Table 10.7 shows the number of refinement cycles required to reach tolerance and the final interval counts. Results are as expected with options 4 and 5 performing better than option 1 by factors ranging from .36 to as little as .13.

10.3.2 Oregonator

Problem 13 was run to an ending time $T = 50.0$ with an initial step size of 0.005. Figure 10.50 shows the solution for the three components

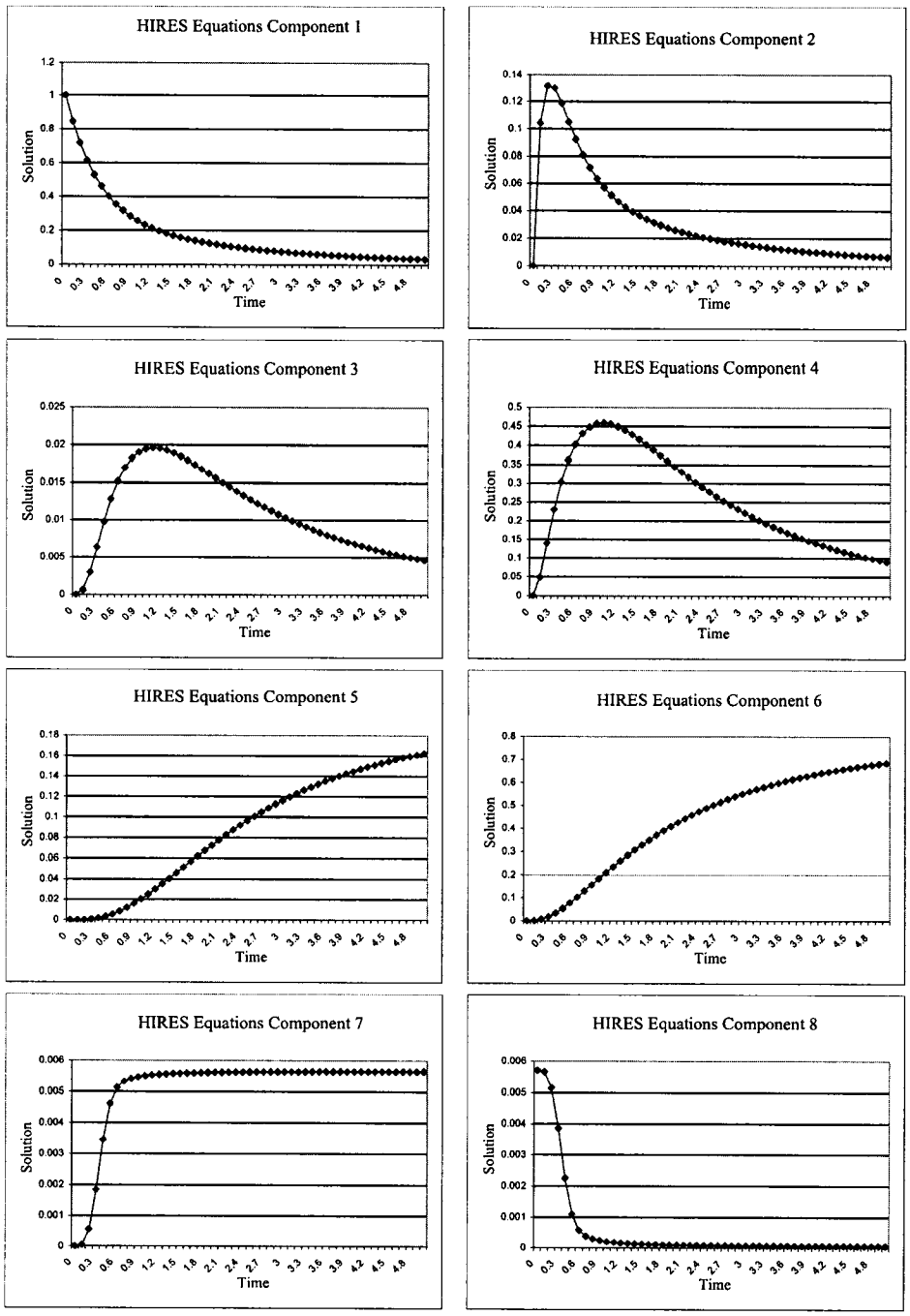


Figure 10.47: Approximation for the components of the Hires equations.

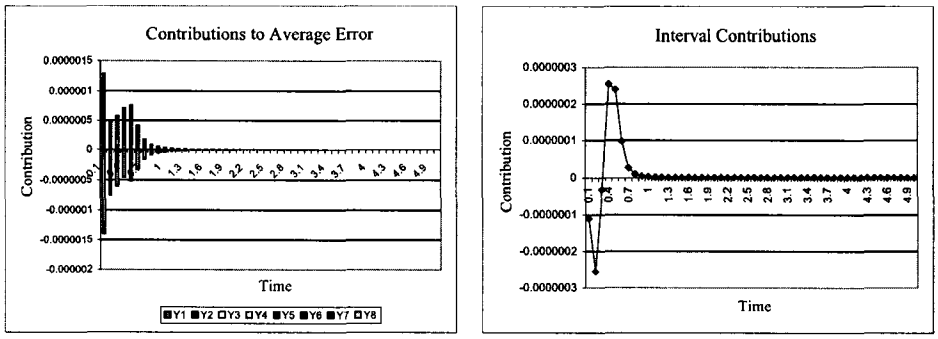


Figure 10.48: Component and total interval contributions for the average error.

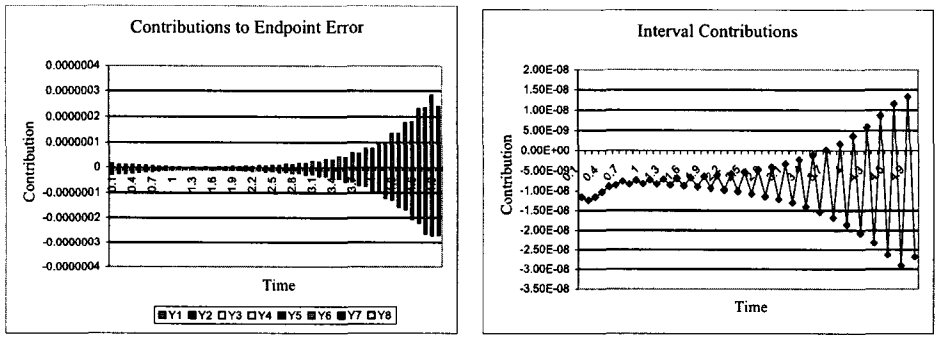


Figure 10.49: Component and total interval contributions for the endpoint error.

Method	Iterations (Endpt)	Intervals (Endpt)	Iterations (Avg)	Intervals (Avg)
Option 1	2	985	2	1142
Option 4	4	200	4	200
Option 5	5	352	3	148

Table 10.7: Final interval count for the Hires problem after refinement using each of three options.

of the equations. Of particular interest is the behavior at $t = 20.4$. This dramatic change in the solution is periodic, but the period is long and the run does not encompass the next period.

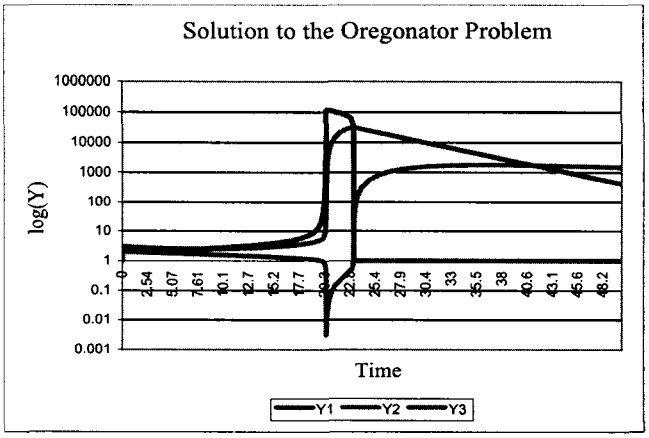


Figure 10.50: Solution of the Oregonator equations with an initial stepsize of 0.005.

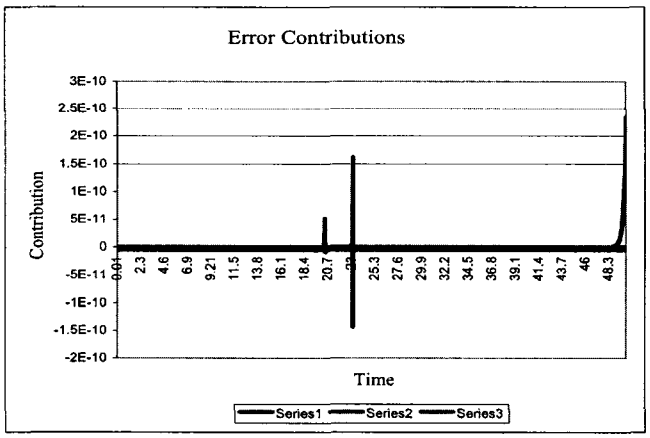


Figure 10.51: Interval contributions for the Oregonator problem for endpoint error.

For Fig. 10.52, we plot only from $t = 19$ to $t = 24$. Outside this range, the contributions are negligible. Of particular interest are the contributions

near the points $t = 20$ and $t = 23$. This corresponds to the period of chemical reaction and is captured well by the error contributions.

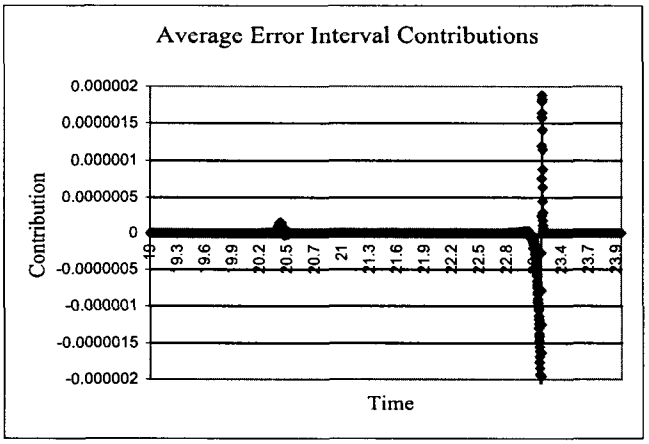


Figure 10.52: Interval contributions for the Oregonator problem for average error.

The problem was run to an ending time of 50 with an initial step size of 0.005 and a completion tolerance of 10^{-8} for refinement options 1, 4, and 5. Option 1 required 188279 intervals to reach tolerance, while option 4 required 20001 and option 5 required 20108. Figures 10.53 and 10.54 show the interval lengths after reaching tolerance for options 4 and 5. Both options show extensive refinement through the period of reaction, specifically at the beginning of the reaction and the latter half of the reaction. Option 4 concentrates additional refinement over the first third of the run, while option 5 concentrates additional refinement over the last two-thirds of the run. The option 5 results would seem to indicate a steady level of error contribution relative to the first third, but the numbers shown in the results are too small ($< 10^{-}12$) to be conclusive.

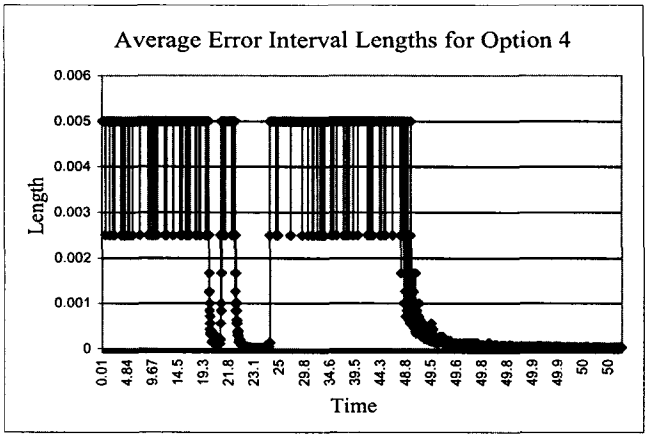


Figure 10.53: Interval lengths after reaching tolerance for the Oregonator problem, average error, option 4.

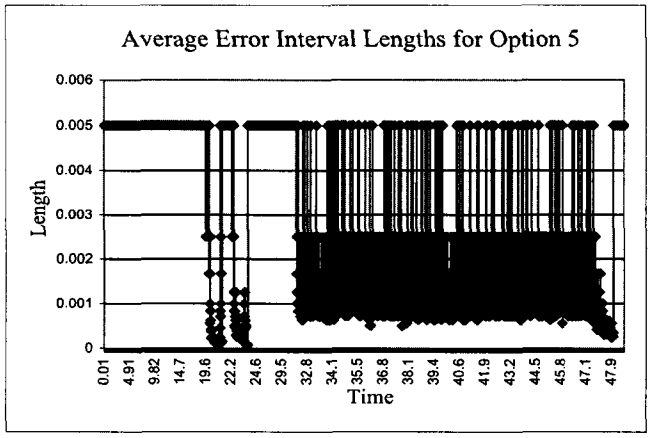


Figure 10.54: Interval lengths after reaching tolerance for the Oregonator problem, average error, option 5.

Chapter 11

AN APPLICATION: THE CARBON CYCLE

11.1 Background

Atmospheric carbon dioxide (CO_2) is a primary factor in the change in heat entering and leaving the earth's climate system. Consequently, the ability to quantify this is important in the science of climate change. Anthropogenic CO_2, seasonal uptake by photosynthesis, and dissolution of CO_2 into oceans dominate the long term changes in atmospheric content. However, analyses have indicated that changes in the terrestrial carbon are a leading cause in interannual variability. There are three components in the terrestrial carbon exchange;

- CO_2 fertilization, or the idea that rising concentrations of CO_2 in the atmosphere may increase or decrease plant production,
- land use changes such as deforestation, which change CO_2 emissions,
- and the processes involved in the exchange.

The third consideration is directly affected by the first two, and combines the process of photosynthetic production and respiration. Since interannual

and decadal variability in climate can create imbalances in photosynthesis and respiration and, consequently, interannual variations in terrestrial carbon, it is important to know the numerical models representing these processes are accurate and consistent with observations.

11.2 The K-Model Terrestial Carbon Cycle Model

A carbon cycle model is a set of processes that track the movement of carbon between the atmosphere, oceans, and terrestrial ecosystems. While there are many carbon cycle models of varying degrees of complexity, in this thesis we use the K-Model. This is a aggregated, terrestrial carbon cycle model where K refers to the turnover time coefficients for the various carbon pools. Details of the model and studies performed using this model can be found in Vukicevic, et.al. [46]. In general, CO_2 is emitted by "source" processes and is absorbed by "sink" processes. For the K-Model, sources include carbon entering the atmosphere via respiration and heterotrophic respiration associated with decomposition, while the sink is CO_2 taken from the atmosphere by the process of photosynthesis. Heterotrophic respiration is represented in the K-Model by three pools; active soil organic matter (SOM), passive SOM, and litter. Two additional pools monitor vegetation and atmospheric carbon.

The K-Model is defined by five equations and 16 parameters which describe the transfer of carbon between pools. The equations are,

$$\begin{aligned}
\dot{S}_1 &= R - P \\
\dot{S}_2 &= P - k_2 \times S_2 \\
\dot{S}_3 &= k_2 \times S_2 - k_3 \times S_3 \\
\dot{S}_4 &= k_3 \times t_{34} \times S_3 - k_4 \times S_4 - Q_r \times T \\
\dot{S}_5 &= k_3 \times t_{35} \times S_3 - k_5 \times S_5,
\end{aligned} \tag{11.2.1}$$

where S_1 through S_5 represents the pools for the atmosphere, terrestrial vegetation, litter and detritus, active soil organic matter, and slow and passive soil organic matter, respectively; T is the temperature anomaly; and Q_r and Q_p are linear temperature response rates for respiration and production, respectively. Auxiliary equations compute respiration R, production P, and the nutrient mineralization rate β. These equations are given by,

$$R = Q_r T + k_3 D S_3 + k_4 S_4 + k_5 S_5, \tag{11.2.2}$$

$$P = Q_p T + cn_2 \beta F, \tag{11.2.3}$$

and

$$\beta = \frac{k_3 S_3}{cn_3} + \frac{k_4 S_4}{cn_4} + \frac{k_5 S_5}{cn_5} - \frac{k_3 f_{34} S_3}{cn_4} - \frac{k_3 f_{35} S_3}{cn_5} + \frac{Q_r T}{cn_4}. \tag{11.2.4}$$

Parameters used in this model, along with their values, are shown in table 11.1. Values for the parameters were determined through use of optimization experiments of the K-model with a lagged N-feedback mechanism. Details of the optimization procedures are in [46]. Note that equations (11.2.2) through (11.2.4) are dependent on the temperature anomaly data, T. Since data measurements are on a monthly time scale, any simulation using time steps other than monthly must use some approximation for temperature anomaly data not on measurement nodes. This is especially important when using adaptivity. In this case, a spline of the anomaly data is computed and the spline provides the data for all nodes. Since this is an approximation of the data, additional error is created and, as such, we need to account for this error. Modeling error can also occur from data measurement errors. Measured data is used to determine parameter values,

K-Model Parameters

Name	Description	Default Value
K2	Vegetation turnover rate	0.17
K3	Litter turnover rate	0.042
K4	Active SOM turnover rate	0.014
K5	Passive SOM turnover rate	0.00083
CN2	C:N vegetation	66.3
CN3	C:N litter	50.0
CN4	C:N active SOM	37.5
CN5	C:N passive SOM	22.5
DCMPT3	Slope of K3 vs temp	1.0
DCMPT4	Slope of K4 vs temp	1.0
DCMPT5	Slope of K5 vs temp	0.43
QPROD	Slope of NPP vs temp	1.66
QRESP	Slope of resp vs temp	2.00
NUE	Strength of NUE feedback	0.59
RSP	Resp fraction of decompisition	0.45
FSLOW	Slow vs Passive partitioning	0.80

Table 11.1: K-model parameters and the default values.

consequently errors in the data measurement results in inaccuracies in parameter values. In chapter 12, we will look at extending the error analysis to account for possible modeling error.

The pools represent the cumulative difference in carbon storage as a result of temperature variability and can thus have positive or negative effects as shown in Fig. 11.1.

11.3 Model Results

The model was run for the full range of temperature anomaly data, 1979 through 1994, and was run for both the endpoint error of the atmospheric carbon pool and average error for the whole system. Average error showed large amounts of fluctuation (Fig. 11.2) in the contributions. In addition, there is cancellation between the components of the system. Error

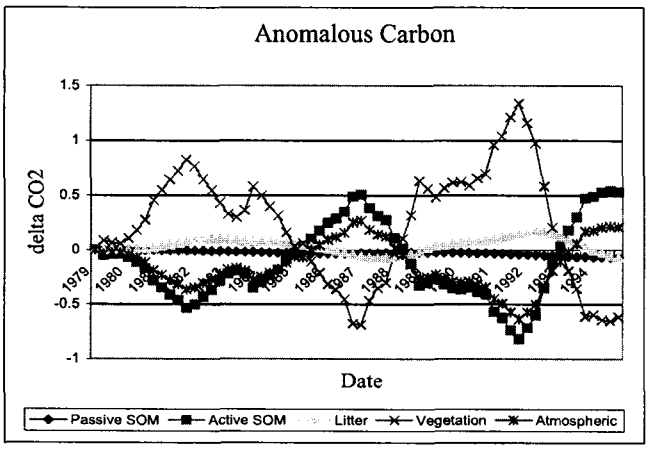


Figure 11.1: Solution curves showing the cumulative difference in carbon storage for the 5 pools.

contributions for the active soil organic matter (SOM) and the atmospheric CO_2 cancel the error contributions from vegetation. Consequently, there is no refinement for average error. Accumulation of error (Fig. 11.3) also shows the cancellation between components.

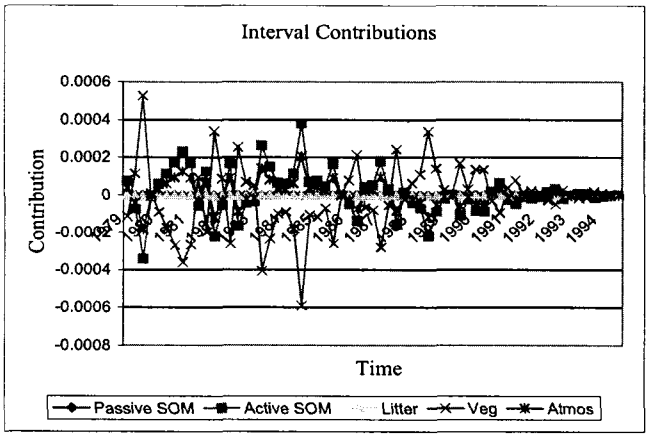


Figure 11.2: Interval contributions for the average error simulation.

Since atmospheric CO_2 is a major contributor to the greenhouse effect, we now consider the endpoint error of the atmospheric carbon pool as our quantity of interest. Figure 11.4 graphs the interval contributions

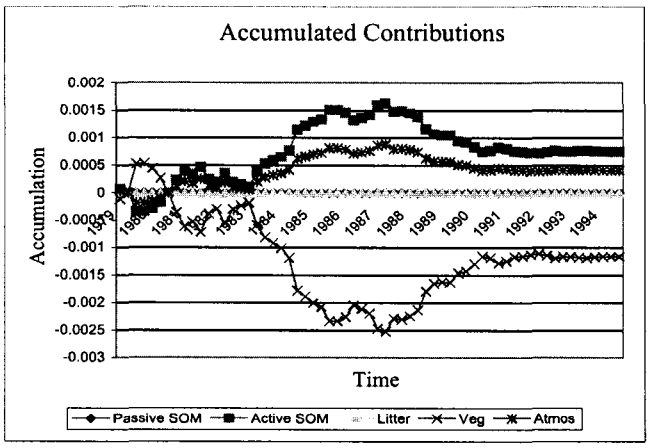


Figure 11.3: Accumulated contributions for the average error simulation.

for the quantity of interest along with the trendline for the contributions. The trendline is echoed in the accumulated contributions (Fig. 11.5) as the trendline crossing the X axis near 1987 matches the downward trend in the accumulation beginning at the same time.

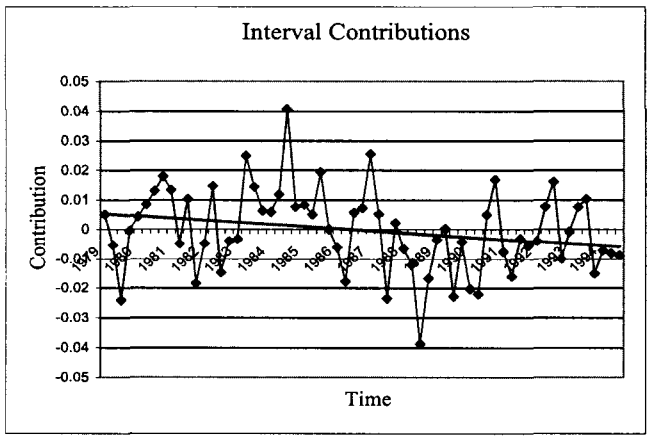


Figure 11.4: Interval contributions to the endpoint error for the atmospheric carbon pool.

Refinement options 1, 4, and 5 were run for the endpoint error on atmospheric carbon with a tolerance of 0.001. Option 1 reached tolerance after one refinement cycle while options 4 and 5 both reached tolerance

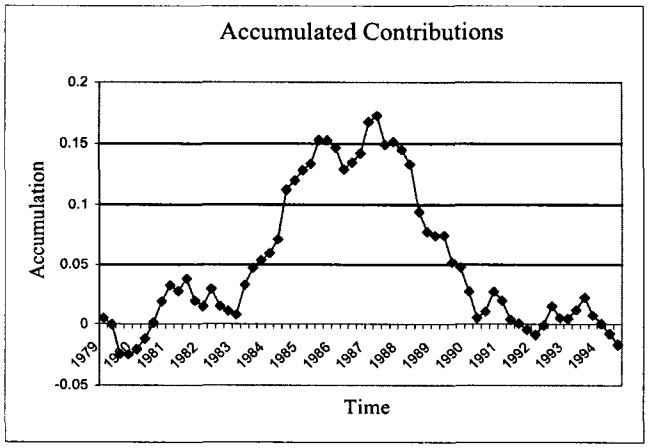


Figure 11.5: Accumulated contributions for the endpoint error in the atmospheric carbon pool.

after three. However, option 1 required 2173 intervals, while options 4 and 5 used 504 and 672, respectively. Figures 11.6 through 11.8 show final interval lengths for each option after reaching tolerance. Option 1 shows

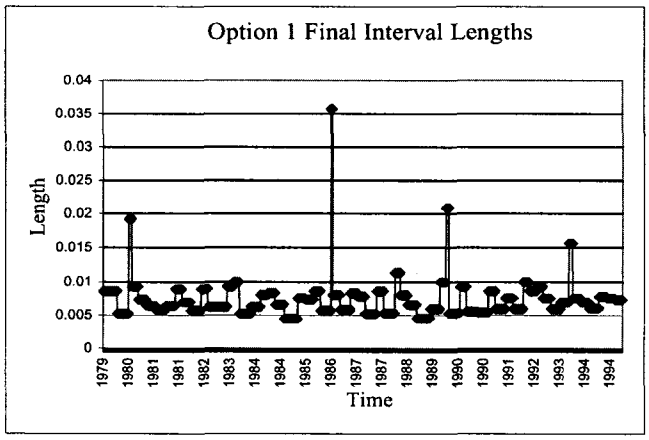


Figure 11.6: Final interval lengths after reaching tolerance for option 1. This option shows almost uniform refinement over time.

almost uniform refinement. This is a result of contributions ranging from -0.048 to 0.051 with 49 of the 63 intervals falling in the range -0.02 to 0.02.

Results show little difference between option 4 and 5. This is expected because of the high variability between positive and negative values of the contributions. Option 4 shows two regions of consistently high refinement. The region from interval 182 to interval 203, and the region from interval 245 to interval 300. These correspond to the largest positive and negative contributions at 1984.79 and 1988.79. Likewise, option 5 shows two regions with high refinement. The region from interval 65 to 96 corresponds to the positive zone centered at 1981.04, while the region from interval 197 to 271 corresponds to the positive zone centered at 1984.79.

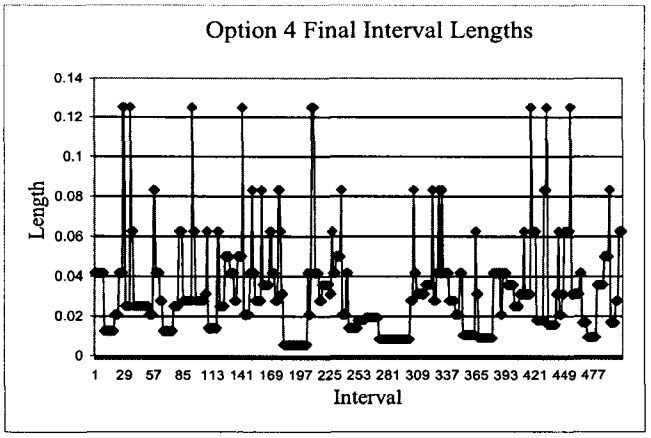


Figure 11.7: Final interval lengths after reaching tolerance for option 4.

The probabilistic and weighted zonal strategies both show marked improvement in efficiency over the traditional refinement method. The weighted zonal strategy uses only 31% of the intervals used by the traditional method, and the probabilistic method uses on 23%. In addition, adjoint based *a posteriori* error estimates show the region centered around 1987 and 1988 to be the region of greatest inaccuracy. We note, however, that the cancellation of error between components of the system reduce the usefulness of the average error.

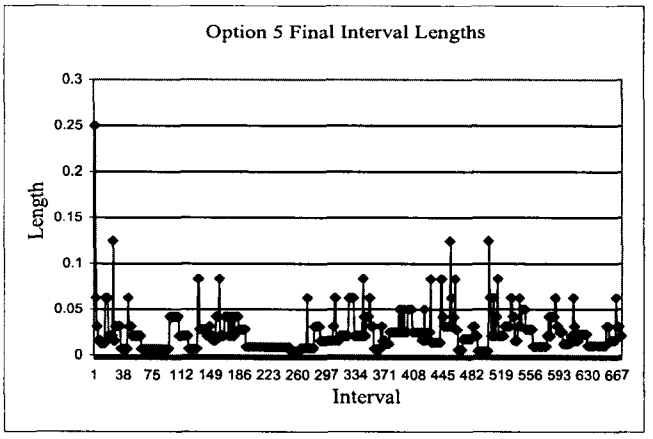


Figure 11.8: Final interval lengths after reaching tolerance for option 5.

Chapter 12

EXTENDING THE ANALYSIS TO INCLUDE THE EFFECT OF VARIATION IN PARAMETERS

We now consider the effect of variations in the parameters. This is important in the practical use of ODEs for modeling when parameters are obtained by measurements, and hence, subject to measurement error, or are intermittent or poorly resolved. Indeed, sensitivity to variation in parameters might be a primary reason to investigate a new terrestrial carbon cycle model.

To account for modeling error, we seek to add a modeling term to the error representation. Consider a differential equation

$$\begin{cases} \dot{y} = f(y(t); \lambda), & 0 < t \leq T, \\ y(0) = y_0, \end{cases} \tag{12.0.1}$$

where λ is a parameter in $\mathbb{R}^Q$. We consider a particular solution, $\tilde{y}$, satisfying

$$\begin{cases} \dot{\tilde{y}} = f(\tilde{y}(t); \tilde{\lambda}), & 0 < t \leq T, \\ \tilde{y}(0) = y_0, \end{cases} \tag{12.0.2}$$

where $\tilde{\lambda}$ is a reference value for the parameter λ. In recent work, Neckels [35] provides the following expression for modeling error:

$$\int_0^T (e, \psi)dt = (y(0) - \tilde{y}(0), \bar{\varphi}(0)) + \int_0^T (f_\lambda(\tilde{y}; \tilde{\lambda})(\lambda - \tilde{\lambda}), \bar{\varphi})dt + \int_0^T (R, \bar{\varphi})dt \tag{12.0.3}$$

where $e = y - \tilde{y}$, $\tilde{\varphi}$ is the adjoint solution solving

$$\begin{cases} -\dot{\tilde{\varphi}} = f'(\tilde{y}(t); \tilde{\lambda})^* \tilde{\varphi} + \psi, & T \geq t \geq 0, \\ \tilde{\varphi}(T) = 0, \end{cases} \tag{12.0.4}$$

f_λ denotes the matrix,

$$\begin{pmatrix} \frac{\partial f_1}{\partial \lambda_1} & \cdots & \frac{\partial f_1}{\partial \lambda_q} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_p}{\partial \lambda_1} & \cdots & \frac{\partial f_p}{\partial \lambda_q} \end{pmatrix}, \tag{12.0.5}$$

and R is a higher order expression in $\lambda - \tilde{\lambda}$. It is assumed f is Lipschitz continuous and, consequently, continuous in the parameter, λ. In Neckels work [35], the effect of numerical error is neglected. Here, we consider the effect of using numerical approximations $Y \approx y$, $\tilde{Y} \approx \tilde{y}$, and $\Phi \approx \tilde{\varphi}$. We assume here for simplicity that $y(0) = \tilde{y}(0)$, so consider

$$\int_0^T (e, \psi) dt \approx \int_0^T (f_\lambda(\tilde{y}; \tilde{\lambda})(\lambda - \tilde{\lambda}), \tilde{\varphi}) dt. \tag{12.0.6}$$

We add and subtract $\int_0^T (f_\lambda(\tilde{Y}; \tilde{\lambda})(\lambda - \tilde{\lambda}), \tilde{\varphi}) dt$ to get

$$\begin{aligned} \int_0^T (e, \psi) dt \approx & \int (f_\lambda(\tilde{y}; \tilde{\lambda})(\lambda - \tilde{\lambda}), \tilde{\varphi}) dt \\ & + \int_0^T (f_\lambda(\tilde{Y}; \tilde{\lambda})(\lambda - \tilde{\lambda}), \tilde{\varphi}) dt \\ & - \int_0^T (f_\lambda(\tilde{Y}; \tilde{\lambda})(\lambda - \tilde{\lambda}), \tilde{\varphi}) dt. \end{aligned} \tag{12.0.7}$$

Rearranging and combining terms gives

$$\begin{aligned} \int_0^T (e, \psi) dt \approx & \int_0^T (f_\lambda(\tilde{Y}; \tilde{\lambda})(\lambda - \tilde{\lambda}), \tilde{\varphi}) dt \\ & + \int_0^T ([f_\lambda(\tilde{y}; \tilde{\lambda}) - f_\lambda(\tilde{Y}; \tilde{\lambda})](\lambda - \tilde{\lambda}), \tilde{\varphi}) dt. \end{aligned} \tag{12.0.8}$$

We then substitute $\tilde{\Phi}$ to get

$$\begin{aligned} \int_0^T (e, \psi) dt \approx & \int_0^T (f_\lambda(\tilde{Y}; \tilde{\lambda})(\lambda - \tilde{\lambda}), \tilde{\Phi}) dt \\ & + \int_0^T ([f_\lambda(\tilde{y}; \tilde{\lambda}) - f_\lambda(\tilde{Y}; \tilde{\lambda})](\lambda - \tilde{\lambda}), \tilde{\Phi}) dt. \end{aligned} \tag{12.0.9}$$

Looking specifically at the second integral on the right hand side, consider the i^{th} component

$$\left([f_\lambda(\tilde{y};\tilde{\lambda}) - f_\lambda(\tilde{Y};\tilde{\lambda})](\lambda - \tilde{\lambda}), P\tilde{h}i\right)_i = \sum_{j=1}^{Q}\left[\frac{\partial f_i}{\partial \lambda_j}(\tilde{y};\tilde{\lambda}) - \frac{\partial f_i}{\partial \lambda_j}(\tilde{Y};\tilde{\lambda})\right](\lambda_j - \tilde{\lambda}_j). \quad (12.0.10)$$

Now,

$$\frac{\partial f_i}{\partial \lambda_j}(\tilde{y};\tilde{\lambda}) - \frac{\partial f_i}{\partial \lambda_j}(\tilde{Y};\tilde{\lambda}) = \sum_{l=1}^{P}\frac{\partial^2 f_i}{\partial \lambda_j \partial y_l}(\tilde{Y};\tilde{\lambda})(\tilde{y}_l - \tilde{Y}_l) + \sum_{l=1}^{P} R_{i,j,l}, \quad (12.0.11)$$

where $R_{i,j,l}$ is higher order in $\tilde{y} - \tilde{Y}$. Neglecting this in our approximation,

$$\left([f_\lambda(\tilde{y};\tilde{\lambda}) - f_\lambda(\tilde{Y};\tilde{\lambda})](\lambda - \tilde{\lambda}), P\tilde{h}i\right)_i \approx \sum_{j=1}^{Q}\sum_{l=1}^{P}\frac{\partial^2 f_i}{\partial \lambda_j \partial y_l}(\tilde{Y};\tilde{\lambda})(\tilde{y}_l - \tilde{Y}_l)(\lambda_j - \tilde{\lambda}_j). \quad (12.0.12)$$

Hence,

$$\left([f_\lambda(\tilde{y};\tilde{\lambda}) - f_\lambda(\tilde{Y};\tilde{\lambda})](\lambda - \tilde{\lambda}), P\tilde{h}i\right)_i \approx \sum_{l=1}^{P}\left(\frac{\partial}{\partial \lambda_j}f'(\tilde{Y};\tilde{\lambda})(\tilde{y}_l - \tilde{Y}_l), \Phi\right)(\lambda_j - \tilde{\lambda}_j). \quad (12.0.13)$$

where $f'(\tilde{Y};\tilde{\lambda})$ is the Jacobian of f with respect to Y and $\frac{\partial}{\partial \lambda_j}f'$ is computed componentwise. Taking adjoints, we have

$$\left([f_\lambda(\tilde{y};\tilde{\lambda}) - f_\lambda(\tilde{Y};\tilde{\lambda})](\lambda - \tilde{\lambda}), \tilde{\Phi}\right)_i \approx \sum_{l=1}^{P}\left((\tilde{y}_l - \tilde{Y}_l), \frac{\partial}{\partial \lambda_j}f'(\tilde{Y};\tilde{\lambda})^*\tilde{\Phi}\right)(\lambda_j\tilde{\lambda}_j), \quad (12.0.14)$$

and moving the sum gives

$$\left([f_\lambda(\tilde{y};\tilde{\lambda}) - f_\lambda(\tilde{Y};\tilde{\lambda})](\lambda - \tilde{\lambda}), \tilde{\Phi}\right)_i \approx \left((\tilde{y}_l - \tilde{Y}_l), \sum_{l=1}^{P}\frac{\partial}{\partial \lambda_j}f'(\tilde{Y};\tilde{\lambda})^*\tilde{\Phi}\right)(\lambda_j\tilde{\lambda}_j). \quad (12.0.15)$$

Interpreting the componentwise, we abuse notation to write

$$\begin{aligned}\int_0^T (e,\psi)dt \approx & \int_0^T (f_\lambda(\tilde{Y};\tilde{\lambda})(\lambda-\tilde{\lambda}),\tilde{\Phi})dt \\ & + \int_0^T ((\tilde{y}-\tilde{Y}),(f'_\lambda(\tilde{Y};\tilde{\lambda})(\lambda-\tilde{\lambda}))^*\tilde{\Phi})dt.\end{aligned} \tag{12.0.16}$$

Note, we can define $\tilde{\psi_p} = (f'_\lambda(\tilde{Y};\tilde{\lambda})(\lambda-\tilde{\lambda}))^*\tilde{\Phi})$ and the second integral on the right is a linear functional of the numerical error,

$$\int_0^T (\tilde{y}-\tilde{Y},\tilde{\psi_p})dt. \tag{12.0.17}$$

We can estimate this by solving a second adjoint problem,

$$\begin{cases} -\dot{\tilde{\varphi}}_p = f'(\tilde{y}(t);\tilde{\lambda})^*\tilde{\varphi}_p + \psi_p, & T \geq t \geq 0, \\ \tilde{\varphi}_p(T) = 0. \end{cases} \tag{12.0.18}$$

Hence, we can estimate the effect of using a numerical approximation to evaluate the sensitivity estimate, (12.0.3), rather than the true solution, by computing a special *a posteriori* estimate.

Recall problem 4, the logistics problem. This problem has two parameters, a and b. As an experiment, the problem was run with the parameter b fixed and varying values for both the initial condition and the parameter a. We see in Fig. 12.1 that the error estimate can have an order of magnitude difference based on the initial conditions and parameter value.

We can conclude from this that the effects of numerical error can be significant, and particularly in the case that $\lambda - \tilde{\lambda}$ is large. Recall from equation (12.0.16) that the error estimate includes a term involving the inner product of the numerical error and $\tilde{\psi_p}$ which includes the modeling error. Consequently, if the numerical error is large, we increase the effects of the modeling error. This appears to go against the conventional wisdom that says numerical error can be ignored when the data error is very large. Implementing this concept into GAASP is a goal for our future work.

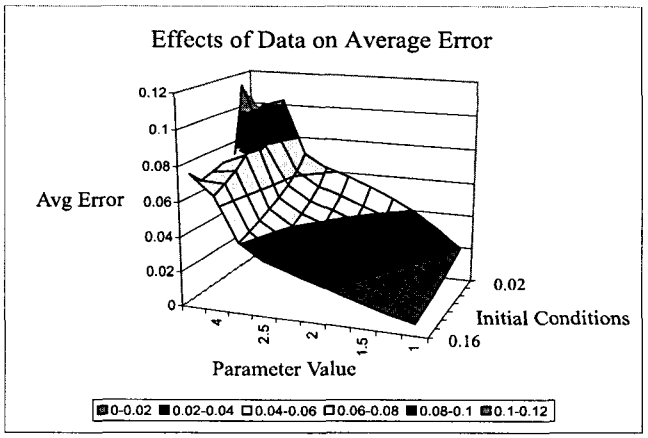


Figure 12.1: Sensitivity of the error estimate for the logistics equation to changes in the parameter a and the initial conditions.

Chapter 13

CONCLUSIONS

Estimating and improving the accuracy of solutions to differential equations is important in all scientific investigations. While adaptive methods have been in use for some time, an optimization framework based on using the Principle of Equidistribution of Error applied to an upper bound, which is the guiding principle used in classic adaptive refinement, is not optimal since estimates are based on residuals and do not account for possible cancellation of error. By solving a time dependent ODE using a global approach analogous to elliptic problems, computing the solution to the adjoint problem, and applying variational principles, we have demonstrated that, depending on the changing stability of the problem, cancellation of errors can occur and traditional methods overestimate the amount of refinement necessary to bring the quantity of interest to the desired accuracy.

We have derived the error representation formula, including quadrature error, for ordinary differential equations and have developed a software tool that provides adaptive error control. This tool provides the groundwork for a larger, comprehensive software package for solving ODEs and performing parameter sensitivity analysis. This tool is especially useful for scientists in the ecology and natural resource fields in that sophisticated methods can be used without technically sophisticated understanding of functional

analysis on the part of the scientist. Specifically, our software tool performs adaptive error control using adjoint based *a posteriori* error estimates without requiring the scientist to provide an adjoint problem in addition to the differential equation being solved.

With large model, efficiency in the solution methods is critical. We have developed new adaptive error control methods based on probability and the Fundamental Theorem of Simulation, as well as making efficient use of the cancellation properties in error. Since there is accumulation of error at every time step, and it is unknown what effect the accumulation of error at any given time step has, allowing refinement at intervals other than just those with the highest error contribution assists in the control of error without using the inefficient number of intervals required by traditional refinement techniques. Results show the probabilistic method and the weighted zonal strategy presented in this thesis increase efficiency dramatically. Test problems show achievement of tolerance with as few as 25% of the intervals required by the equidistribution of error. In addition, we have shown that quantity of interest plays a role in the method used. Endpoint error values are more easily controlled by weighting the refinement properly between regions of positive and negative contribution. Consequently, the weighted zonal strategy proves better for endpoint error than the probabilistic method specifically when we have a problem with changing stability properties. While these properties may not be know in advance, the weighted zonal strategy performed equal to the probabilistic method in these cases, and both performed significantly better than equidistribution of error.

As an example problem, we implemented the K-Model into the code. The model was provided by an outside source, and implementation into the

software proved easy. This is an important consideration in any software tool intended for use by the scientific community. Use of the K-Model also brought to the forefront that problem of modeling error. While modeling error has been studied before, we have shown that modeling error and numerical error are integral parts of the error control process.

We see as a further direction for this research, an in depth look at coupling the modeling and numerical error. This research combines work done in this thesis with work previously completed on modeling error. In addition, the concept of parallel integration requires some knowledge of where splits in the time line can be placed. The information provided by this tool can be useful in the description of stability properties of a time dependent ODE as well as where error is low. Coarse solves can provide the information necessary to determine the proper location for splits for the fine resolution parallel solves.

Chapter 14

SOFTWARE DESCRIPTION AND USER'S GUIDE

In this chapter we discuss the organization of the software package and its use. We include a brief description of the major function calls in the adaptive loop, and describe how to add a user defined problem to the code.

14.1 General Description

The ODE solver component of GAASP is a general purpose ODE solver with adaptivity using error estimates based on the generalized Green's function. The user can choose one of 23 example problems or add his/her own application through the testproblem.h include file plus some additional coding. Note that, while the error computation requires the adjoint problem, the user is not required to supply these equations.

14.2 Code Outline

Figure 14.1 shows the general outline of code flow for the adaptive loop. The first step in the code is to determine the total number of time steps in the solution, generate a mesh, and allocate memory. The time mesh is then passed to the forward problem solver. The forward solve performs a quasi-Newton's method for each time step in the mesh. A prediction is made

(currently the approximation from the previous time step) and passed to the Newton solver where a finite difference approximation of the Jacobian is computed and the Newton iteration is performed. While the forward solve is performed by a quasi-Newton's method, the dual solve, since we linearize around the forward approximation, is a direct computation.

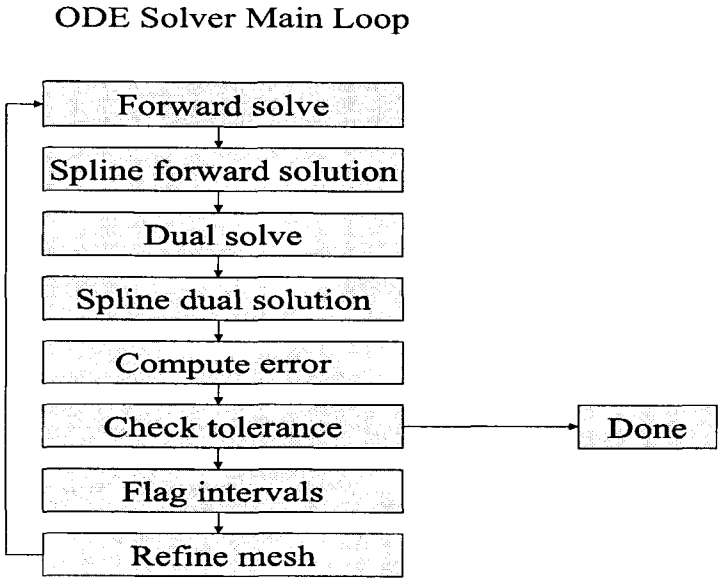


Figure 14.1: General code flow.

14.3 Testproblem.h

The **testproblem.h** header file determines which problem is being solved, which options are used, and what output is produced. These selections are made through **define** compiler directives. Options are available for which ODE to solve, be it an example problem or user defined problem, initial and ending times, initial time step, and various options for the adaptive error control. These are listed in table 14.3.

14.4 Functions

A list of the functions used, inputs, and outputs are found in the appendix.

14.5 Adding a Differential Equation

When adding a differential equation to the code, there are several files that need modification. The file **f.cpp** contains the actual differential equation, the file **initConditions.cpp** holds the initial conditions, and the file **testproblem.h** defines the various options used by the code. In addition, if the problem being added is a test problem and has a true solution, the definition of the true solution is added to the file **ftrue.cpp**. Consider example 11, the unstable system. The equations defining the problem are

$$\begin{aligned} \dot{y}_1 &= \frac{y_1}{2(1+t)} - 2ty_2^2, \\ \dot{y}_2 &= \frac{y_2}{2(1+t)} + 2ty_1^2, \end{aligned} \tag{14.5.1}$$

with initial conditions $y(0) = [1.0, 0.0]^\top$. The equations are entered into **f.cpp** as

$$\text{yout}[0] = \text{yin}[0]/(2 * (1 + t)) - 2 * t * \text{yin}[1];$$

and

$$\text{yout}[1] = \text{yin}[1]/(2 * (1 + t)) + 2 * t * \text{yin}[0];$$

where the variables yout[0] and yout[1] store the values for $\dot{y}_1$ and $\dot{y}_2$, respectively. Likewise, the variables yin[0] and yin[1] store the values for y at the previous time step. Next, the initial conditions must be entered into **initConditions.cpp**. The initial conditions are entered into a two dimensional variable named soln[i][j], where i is the index for the equation in the

system and j is the index for the position. Enter the following lines in **initConditions.cpp**.

$$\text{soln}[0][0] = 1.0;$$

and

$$\text{soln}[1][0] = 0.0;$$

If the true solution is known and required, the equations are entered into **ftrue.cpp**.

14.6 Simulation Results

Output from the program is determined by the switches listed in table 14.3. Output is written to the files listed below.

1. **fsoln.txt** is the default output file. This space delimited file is always created and contains a column for time and a column with the solution for each component in the system of differential equations.

2. **endpoint_error.txt** and **average_error.txt** are optional files that contain error control information. These space delimited files show the interval contribution to the error (if **PRINTCONTRIB** is set to 1) and the final error estimate and ratio. Note that if the user defined problem does not have a true solution, the ratio is meaningless.

3. **endpoint_dual.txt** and **average_dual.txt** are optional files that store the solution to the adjoint problem as computed for the chosen quantity of interest.

4. **errordetails.txt** is an optional file that contains details of the error computation. For each time step there is a value for both the discretization error and quadrature error for each component.

KEY	DESCRIPTION	ARG TYPE
EXAMPLE⟨*arg*⟩	the example problem	INT
SYSDIM ⟨*arg*⟩	sets the number of equations in the system	INT
TRUESOLUTION	defined if there is a true solution	
LAMBDA ⟨*arg*⟩	used in EXAMPLE8	REAL
NPARAM ⟨*arg*⟩	number of parameters for systems required parameter files to be read	INT
TSTART ⟨*arg*⟩	starting time	REAL
TEND ⟨*arg*⟩	ending time	REAL
STEPSIZE ⟨*arg*⟩	size of the numerical method time increment	REAL
NODEDENSITY ⟨*arg*⟩	1 = dual mesh same as forward mesh 2 = dual mesh twice the forward mesh	INT
INTERPTYPE ⟨*arg*⟩	0 = piecewise constant 1 = piecewise linear	INT
INTERPOPTION ⟨*arg*⟩	0 = beginning point 1 = midpoint 2 = ending point	INT
NTOL ⟨*arg*⟩	Newton's method tolerance	REAL
METHOD ⟨*arg*⟩	0 = dG(0)/cG(1) 1 = dG(1)/cG(2)	INT
PSITYPE ⟨*arg*⟩	0 = endpoint error, 1 = integral error	INT
COMPONENT ⟨*arg*⟩	defines component for error output (between 1 and SYSDIM)	INT
TOL ⟨*arg*⟩	Completion tolerance	REAL
REFINETYPE ⟨*arg*⟩	0 = bisection 1 = equidistribution of error	INT
REFINEOPT ⟨*arg*⟩	0 = no refinement, one solve only 1 = refine all intervals with contribution > (TOL/intervals) 2 = refine a user defined percentage of intervals based on contribution 3 = probabilistic selection based on contribution 4 = probabilistic selection based on contribution and interval length	INT
REFPCT ⟨*arg*⟩	Used by REFINEOPT 3,4 as percent number of intervals to refine each refinement cycle.	REAL

KEY	DESCRIPTION	ARG TYPE
MAXCYCLES ⟨*arg*⟩	maximum number of refinement cycles	INT
SEED ⟨*arg*⟩	0 = Fixed seed 1 = Time generated seed	INT
PRINTCONTRIB ⟨*arg*⟩	0 = don't print 1 = print contributions to error.txt	INT
PRINTTRUE ⟨*arg*⟩	0 = don't print 1 = print true solution to tsoln.txt	INT
PRINTDUAL ⟨*arg*⟩	0 = don't print 1 = print dual solution to dsoln.txt	INT
PRINTSPLN ⟨*arg*⟩	0 = don't print 1 = print forward spline 2 = print dual spline	INT
USETRUE ⟨*arg*⟩	Linearize around true (test purposes only)	INT
ERRORMIN ⟨*arg*⟩	Do not report ratio if error is less than ERRORMIN	REAL

Bibliography

[1] Rene Aid and Laurent Levacher, Numerical investigations on global error estimation for ordinary differential equations, *Journal of Computational and Applied Mathematics*, 82:21-39, 1997.

[2] Kendall E. Atkinson, *An Introduction to Numerical Analysis.* John Wiley and Sons, New York, New York, 1989.

[3] Sheldon Axler, *Linear Algebra Done Right*, Springer-Verlag, New York, New York, 1997.

[4] Paul Blanchard, R. Devaney, and G. Hall, *Differential Equations*, Brooks/Cole Publishing, Pacific Grove, California, 1998.

[5] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*, John Wiley and Sons, West Sussex, England, 2003.

[6] J. C. Butcher, Implicit Runge-Kutta Processes, *Mathematics of Computation*, 18(85):50-64, 1964.

[7] J. C. Butcher, Integration processes Based on Radau Quadrature Formulas, *Mathematics of Computation*, 18(86):233-244, 1964.

[8] Y. Cao and L. Petzold, A Posteriori Error Estimation and Global Error Control for Ordinary Differential Equations by the Adjoint Method, *SIAM J. Sci. Comput.*, 26:359-374, 2004.

[9] S. Cohen and A. Hindmarsh, CVODE, a Stiff/Nonstiff ODE Solver in C, *Computers in Physics*, 10(2):138-143, March-April 1996.

[10] J. E. Dennis and Robert B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM Classics in Applied Mathematics, Philadelphia, PA, 1996.

[11] Sean Eastman, Analysis and Application of the Nonlinear Power Method, Doctoral Dissertation, Colorado State University, 2005.

[12] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson, *Computational Differential Equations*, Press Syndicate of the University of Cambridge, New York, New York, 1996.

[13] Donald Estep, A Posteriori Error Bounds and Global Error Control for Approximations of Ordinary Differential Equations, *SIAM J. Numer. Anal.*, 32:1-48, 1995.

[14] Donald Estep and A. M. Stuart, The Dynamic Behavior of the Discontinuous Galerkin Method and Related Difference Schemes, *Math. of Comp.*, 71:1075-1103, 2001.

[15] Donald Estep and Claes Johnson, The Pointwise Computability of the Lorenz System, *Mathematical Models and Methods in Applied Science*, 8:1277-1305, 1998.

[16] Donald Estep, Michael Holst, and Duane Mikulencak, Accounting for stability: a posteriori error estimates based on residuals and variational analysis, *Communications in Numerical Methods Engineering*, 18:15-30, 2002.

[17] Donald Estep, A Short Course on Duality, Adjoint Operators, Green's Functions, and A Posteriori Error Analysis, Unpublished, 2004

[18] Donald Estep and Donald French, Global Error Control for the Continuous Galerkin Finite Element Method for Ordinary Differential Equations, *Mathematical Modelling and Numerical Analysis*, 28(7):815-852, 1994.

[19] Donald Estep, M. Larson, and R. Williams, Estimating the Error of Numerical Solutions of Systems of Reaction-Diffusion Equations, *Memoirs of the American Mathematical Society*, 146(696):viii+109, 2000.

[20] Laurence Fausett, *Numerical Methods Using MathCad*, Prentice Hall, Upper Saddle River, Jew Jersey, 2002.

[21] C. William Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, Englewood Cliffs, New Jersey, 1971.

[22] Clyde F. Herreid, II, *Biology*, Macmillan Publishing Co. Inc., New York, New York, 1977.

[23] A. R. Humphries and A. M. Stuart, Runge-Kutta Methods for Dissipative and Gradient Dynamical Systems, *SIAM J. Numer. Anal.*, 13(5):1452-1485, 1994

[24] Arieh Iserles, *A First Course in the Numerical Analysis of Differential Equations*, Cambridge University Press, Cambridge, United Kingdom, 1996.

[25] Test Set for IVP Solvers, Universita degli Studi di Bari, Dipartimento di Matematica, http://pitagora.dm.uniba.it/ testset/index.htm

[26] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, SIAM Press, Philadelphia, Pennsylvania, 1995.

[27] C. T. Kelley, *Solving Nonlinear Equations with Newton's Method*, SIAM Press, Philadelphia, Pennsylvania, 1995.

[28] David Kincaid and W. Cheney, *Numerical Analysis*, Brooks/Cole Publishing, Pacific Grove, California, 1996.

[29] J. D. Lambert, *Numerical Methods for Ordinary Differential Equations*, John Wiley and Sons, West Sussex, England, 1995.

[30] Cornelius Lanczos, *Linear Differential Operators*, Dover Publications, Inc., Mineola, New York, 1997.

[31] Albert L. Lehninger, *Bioenergetics*, W. A. Benjamin, Inc., Menlo Park, California, 1973.

[32] C. C. Lin and L. A. Segel, *Mathematics Applied to Deterministic Problems in thte Natural Sciences*, SIAM Classics in Applied Mathematics, Philadelphia, PA, 1988.

[33] Anders Logg, Multi-Adaptive Galerkin Methods for ODES I, *SIAM J. Sci. Comput.*, 24:1879-1902, 2003.

[34] Anders Logg, Multi-Adaptive Galerkin Methods for ODES II: Implementation and Applications, *SIAM J. Sci. Comput.*, 25:1119-1141, 2003.

[35] David Neckels, Variational Methods for Uncertainty Quantification, Doctoral Dissertation, Colorado State University, 2005.

[36] Lawrence Perko, *Differential Equations and Dynamical Systems*, Springer-Verlag, New York, New York, 2001

[37] Christian P. Robert and George Casella *Monte Carlo Statistical Methods*, Springer Science+Business Media, Inc., New York, New York, 2004.

[38] Olof Runborg, *Consistency and Convergence of One-Step Methods*, http://www.nada.kth.se/kurser/kth/2d1250/onestep.pdf

[39] L. F. Shampine, R. C. Allen, Jr., and S. Pruess, *Fundamentals of Numerical Computing*, Wiley Press, New York, New York, 1997.

[40] Robert D. Skeel, Thirteen Ways to Estimate Global Error, *Numer. Math.*, 48:1-20, 1986.

[41] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York, New York, 1993.

[42] Steven H. Strogatz, *Nonlinear Dynamics and Chaos*, Perseus Books Publishing, LLC, Cambridge, MA, 1994.

[43] A.M. Stuart and A. R. Humphries, *Dynamical Systems and Numerical Analysis*, Cambridge University Press, Cambridge, United Kingdom, 1998.

[44] Ernst Hairer and G. Wanner, *Solving Ordinary Differentail Equations II*, Springer-Verlag, New York, New York, 1991.

[45] Ferdinand Verhulst, *Nonlinear Differential Equations and Dynamical Systems*, Springer, New York, New York, 2006.

[46] T. Vukecevic, B. Braswell, and D. Schimel, A Diagnostic Study of Temperature Controls on Global Terrestrial Carbon Exchange, *Tellus Series B-Chemical and Physical Meteorology*, 53(2):150-170, 2001.

Appendix A

FUNCTION DESCRIPTIONS

Here we list all functions used in the ODE solver component of GAASP. In addition, there are three source code files, dense.c, llnlmath.c, and nvector.c, that are taken from CVODE [9]. Note that many of these functions use global variables that are defined in the header of odesolver.cpp.

This appendix contains a listing of functions used in the ODE solver component of GAASP. In addition, there are three source code files, dense.c, llnlmath.c, and nvector.c, that are taken from CVODE [9]. Note that many of these functions use global variables that are defined in the header of odesolver.cpp.

Name:	**computeError**
Description:	Loops through the time steps and calls the appropriate error computation function.
Input:	Number of steps (int) Dimension of the system (int) Number of time steps in the adjoint problem (int)
Outputs:	Array of interval error contributions (double **)
Functions used:	gauss, intervalError

Name:	**computeTrueError**
Description:	Loops through time steps and computes the true error.
Input:	Number of steps (int)
	Dimension of the system (int)
	Number of time steps in the adjoint problem (int)
Outputs:	Array of interval error values (double **)
Functions used:	trueIntervalError

Name:	**dualSolve**
Description:	This is the main solver loop for the dual problem.
Input:	Array of time nodes (double *)
	Number of steps for the dual solution (int)
	Number of equations in the system (int)
	ID for component QOI (int)
Outputs:	Solution to the dual problem in a global variable.
Functions used:	fdmethod

Name:	**f**
Description:	Evaluates the function f.
Input:	Current values (double *)
	Current time (double)
Outputs:	New values (double *)
Functions used:	None

Name:	**fdjac**
Description:	Computes the finite difference approximation of the Jacobian.
Input:	Number of equations in the system (int)
	Point of approximation (archaic)
	Predicted values (double *)
	New time (double)
	Old time (double)
	Forward/dual flag (int)
	Method flag
Outputs:	Approximation of the Jacobian (double **)
Functions used:	macheps, f, fMethod

Name:	**fdmethod**
Description:	Evaluates integrals for the adjoint problem using the requested order.
Input:	Array of values from previous time step (double *) Point for new computation (double *) Time at the new node (double) Time at the old node (double) Ending time (double) Number of steps in the adjoint problem (int) Dimension of the system (int)
Outputs:	Array of solution values (double **)
Functions used:	f, fdjac, transpose, predict, psi, matrixSolve, splint

Name:	**flagIntervals**
Description:	Flags intervals for refinement based on the chosen method.
Input:	Array of time nodes (double *) Array of interval contributions (double **) Global tolerance (double) Number of time intervals (int) Dimension of the system (int) Chosen refinement option (int)
Outputs:	Array of interval flags (double *)
Functions used:	None

Name:	**fmethod**
Description:	Evaluates integrals for the forward problem using the requested order.
Input:	Solution at previous time step (double *) Predicted solution at new time step (double *) New time (double) Previous time (double) Dimension of the system (int)
Outputs:	New time step solution values (double *)
Functions used:	

Name:	**forwardSolve**
Description:	This is the main solver loop for the forward problem.
Input:	Array of time nodes (double *) Number of intervals (int)
Outputs:	Solution to the forward problem in a global variable.
Functions used:	f, l2Norm, newton, predict, initConditions

Name:	**ftrue**
Description:	Computes the exact solution of the differential equation.
Input:	Specified time (double)
	Dimension of the system (int)
Outputs:	Exact solution (double *)
Functions used:	None

Name:	**gauss**
Description:	Computes the error estimate for the low order method using a 5 point Gauss rule to evaluate integrals.
Input:	Dimension of the system (int)
	Gauss rule (int)
	Beginning time node of the interval (int)
	Number of time steps (int)
Outputs:	Interval contribution to the estimate (double *)
Functions used:	interpolant, splint, f, uPrime

Name:	**gausstrueerror**
Description:	Computes the error on the interval using the exact solution.
Input:	Node index (int)
	Number of time steps (int)
	Dimension of the system (int)
Outputs:	Error on the interval (double *)
Functions used:	psi, fTrue, splint, projection

Name:	**initConditions**
Description:	Sets the initial conditions for the differential equation.
Input:	None
Outputs:	None
Functions used:	None

Name:	**interpolant**
Description:	Computes the piecewise constant interpolant.
Input:	Type (not used)
	Option flag (int)
	Node index (int)
	Dimension of the system (int)
Outputs:	Interpolant value (double)
Functions used:	None
Note:	Option flag determines beginning, middle, or end point interpolant.

Name:	**intervalError**
Description:	Computes the error estimate for the high order method using a 5 point Gauss rule to evaluate integrals.
Input:	Dimension of the system (int) Gauss rule (int) Beginning time node of the interval (int) Number of time steps (int)
Outputs:	Interval contribution to the estimate (double *)
Functions used:	interpolant, splint, f, uPrime

Name:	**l2norm**
Description:	Computes the L2 norm of a vector.
Input:	Vector (double *) Dimension (int)
Outputs:	Value of the norm (double *)
Functions used:	None

Appendix B

FUNCTION DESCRIPTIONS (CONT.)

Name:	**macheps**
Description:	Computes machine epsilon.
Input:	None
Outputs:	Computed value.
Functions used:	None

Name:	**matrixSolve**
Description:	Solves Ax=b for x.
Input:	Dimension of the matrix (int)
	Double array containing the matrix (double **)
	Array containing the load vector (double *)
Outputs:	Solution vector (double *)
Functions used:	DenseAllocMat, DenseAllocPiv, NVNew
Note:	This is a modification of a CVODE [9] function.

Name:	**newton**
Description:	Performs a quasi-Newton iteration on the current time step.
Input:	Array of current solution values (double *)
	Array of predicted solution values (double *)
	Array of new solution values (double *)
	Time step ending time (double)
	Time step beginning time (double)
	Number of equations in the system (int)
Outputs:	Returns 0 if no convergence or 1 otherwise.
Functions used:	f, fMethod, fdjac, newtonStep, l2Norm

Name:	**newtonstep**
Description:	Solve $F'(X)H = -F(X)$ for H.
Input:	Number of equations (int)
	Finite difference approximation of the Jacobian (double **)
	Array for function F evaluated at current point (double *)
Outputs:	The Newton step (double *)
Functions used:	CVode functions in dense.c

Name:	**odeSolver**
Description:	This is the main driver for the ODESolver software.
Input:	See testproblem.h
Output:	See chapter 14.
	Other outputs based on options selected in testproblem.h.
Functions used:	forwardSolve, dualSolve, spline, splint, computeError, computeTrueError, fTrue, flagIntervals, refineMesh, printSoln, printTrueSoln, printError, printDual, printSpln, readData

Name:	**predict**
Description:	Generates a prediction for the Newton step.
Input:	Previous values (double *)
	Old time (double)
	New time (double)
	Number of equations in system (int)
	Predictor flag (int)
Outputs:	Predicted values (double *)
Functions used:	

Name:	**printDual**
Description:	Writes the solution of the adjoint problem to file.
Input:	Dimension of the system (int)
	Number of steps (int)
	Array of times (double *)
	Array of solution values (double **)
Outputs:	None
Functions used:	None

Name:	**printError**
Description:	Writes error information to file.
Input:	Number of time steps (int)
	Total error (double *)
	Total true error (double *)
	Endpoint error value (double *)
	Ratio value (double *)
	Number of refinement cycles (int)
	Array of time nodes (double *)
	Array of interval contributions (double *)
	Dimension of the system (int)
Outputs:	None
Functions used:	None

Name:	**printSoln**
Description:	Writes the computed solution to file.
Input:	Dimension of the system (int)
	Number of time steps (int)
	Array of time nodes (double *)
	Array of solution values (double **)
Outputs:	None
Functions used:	None

Name:	**printSpln**
Description:	Writes the values of the spline to file.
Input:	Dimension of the system (int)
	Number of time steps (int)
	Array of time nodes (double *)
	Array of spline values (double **)
Outputs:	None
Functions used:	None

Name:	**printTrueSoln**
Description:	Writes the exact solution values to file.
Input:	Dimension of the system (int)
	Number of time steps (int)
	Array of time nodes (double *)
	Array of exact solution values (double **)
Outputs:	None
Functions used:	None

Name:	**projections**
Description:	Computes the projection onto a piecewise linear interpolant.
Input:	Specified time (double)
	X value at beginning interval point (double)
	X value at ending interval point
	Interval beginning time (double)
	Interval ending time (double)
Outputs:	Projection (double)
Functions used:	None

Name:	**psi**
Description:	Computes the value for $\psi()$ at a given node.
Input:	Dimension of the system (int)
	Requested time (double)
Outputs:	Value of $\psi()$ (double *)
Functions used:	None

Name:	**refineMesh**
Description:	Refines the time mesh at nodes flagged by flagInterval().
Input:	Refinement flags (int *)
	Number of old nodes (int)
	Old mesh (double *)
	Interval error contributions (double **)
	Dimension of the system (int)
	Refinement option (int)
Outputs:	Refined mesh (double *)
Functions used:	None

Name:	**spline**
Description:	Computes a cubic spline of the given data.
Input:	X values (double *)
	Y values (double *)
	Number of points (int)
	Derivative at first point (double)
	Derivative at last point (double)
Outputs:	Array of second derivatives (double *)
Functions used:	None

Name:	**transpose**
Description:	Computes the transpose of a matrix.
Input:	Initial matrix (double **)
	Number of rows (int)
	number of columns (int)
Outputs:	Transposed matrix (double **)
Functions used:	None

Name:	**uPrime**
Description:	Computes the derivative of the interpolant of the solution at a given node.
Input:	Dimension of the system (int)
	Interpolant flag (int)
	Specified node index (int)
Outputs:	Derivative value (double *)
Functions used:	None