NUMERICAL STABILITY & NUMERICAL SMOOTHNESS OF ORDINARY DIFFERENTIAL EQUATIONS

Kaitlin Reddinger

A Thesis

Submitted to the Graduate College of Bowling Green State University in partial fulfillment of the requirements for the degree of

MASTER OF ARTS

August 2015

Committee:

Tong Sun, Advisor

So-Hsiang Chou

Kimberly Rogers
ABSTRACT

Tong Sun, Advisor

Although numerically stable algorithms can be traced back to the Babylonian period, it is believed that the study of numerical methods for ordinary differential equations was not rigorously developed until the 1700s. Since then the field has expanded - first with Leonhard Euler’s method and then with the works of Augustin Cauchy, Carl Runge and Germund Dahlquist. Now applications involving numerical methods can be found in a myriad of subjects. With several centuries worth of diligent study devoted to the crafting of well-conditioned problems, it is surprising that one issue in particular - numerical stability - continues to cloud the analysis and implementation of numerical approximation.

According to professor Paul Glendinning from the University of Cambridge, “The stability of solutions of differential equations can be a very difficult property to pin down. Rigorous mathematical definitions are often too prescriptive and it is not always clear which properties of solutions or equations are most important in the context of any particular problem. In practice, different definitions are used (or defined) according to the problem being considered. The effect of this confusion is that there are more than 57 varieties of stability to choose from” [10]. Although this quote is primarily in reference to nonlinear problems, it can most certainly be applied to the topic of numerical stability in general.

We will investigate three of the main issues surrounding numerical stability in the error analysis and show that numerical smoothing should have been the right concept in delivering better error estimations. Therefore, the materials on numerical stability in textbooks and classrooms should be replaced by numerical smoothness.
ACKNOWLEDGMENTS

I would like to thank the following people for their unwavering support and encouragement during the writing of this thesis. A great deal of gratitude goes first and foremost to my advisor, Dr. Tong Sun, for his mentorship, time and patience throughout this past year. Secondly, I would like to thank my committee members as well as the Bowling Green State University Mathematics Department and staff for answering endless questions, funding my education and providing me with a very generous teaching assistantship. It has been an enjoyable experience. Last, but certainly not least, I want to thank my parents, David and Sue Reddinger. Your prayers and late night pep talks were invaluable elements to the completion of this paper.
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CHAPTER 1</strong> INTRODUCTION</td>
</tr>
<tr>
<td>1.1 Terminology</td>
</tr>
<tr>
<td>1.2 A Brief Historical Overview of Numerical Methods</td>
</tr>
<tr>
<td>1.3 Traditional Numerical Analysis</td>
</tr>
<tr>
<td>1.3.1 Global Error and Convergence</td>
</tr>
<tr>
<td>1.3.2 Consistency, Zero Stability and Global Error Bounds</td>
</tr>
<tr>
<td>1.3.3 Additional Stability Issues</td>
</tr>
<tr>
<td><strong>CHAPTER 2</strong> THE INSTRUCTIONAL OVERSIGHT</td>
</tr>
<tr>
<td>2.1 Indecipherable Context</td>
</tr>
<tr>
<td>2.2 Inescapable Ambiguity</td>
</tr>
<tr>
<td><strong>CHAPTER 3</strong> NUMERICAL SMOOTHING</td>
</tr>
<tr>
<td>3.1 Error Splitting</td>
</tr>
<tr>
<td>3.2 Revised Error Splitting</td>
</tr>
<tr>
<td>3.3 Smoothing Behavior</td>
</tr>
<tr>
<td>3.4 Example</td>
</tr>
<tr>
<td>3.5 Numerical Smoothness for Error Estimation</td>
</tr>
<tr>
<td><strong>BIBLIOGRAPHY</strong></td>
</tr>
<tr>
<td><strong>APPENDIX A: MATLAB SCRIPTS</strong></td>
</tr>
<tr>
<td>Figure</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>1.1</td>
</tr>
<tr>
<td>3.1</td>
</tr>
<tr>
<td>3.2</td>
</tr>
<tr>
<td>3.3</td>
</tr>
</tbody>
</table>
LIST OF TABLES

1.1 IVP Exact Solution and Approximation Values (Euler Method) . . . . . . . . . . . . . . . . . . . . . . 6
1.2 Global Error at time $t_n$, $e_n = |y(t_n) - y_n|$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.1 Differences in Error Splitting . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
CHAPTER 1 INTRODUCTION

We should not be so naive to think that the invention of the computer is what prompted the initial study of numerical methods. Indeed, the usefulness and practicality in being able to approximate solutions dates as far back as the third century A.D., when Heron of Alexandria prescribed a method for calculating the square root of a non-square number \[14\]. Although Heron did not wield modern analysis terms like “convergence” and “stability” to describe his work, he nonetheless succeeded in setting a precedent for investigating the accuracy of a numerical scheme which we still follow today. That precedent being the study of error propagation.

Throughout this paper, we mean to examine the traditional forms of error analysis within the context of numerically solving initial value problems for ordinary differential equations. In particular, we will eventually come to focus on the pitfalls undergraduate students face when dealing with the issue of numerical stability, and we will offer an instructional remedy to this issue in the form of numerical smoothing. What follows should therefore aid in the essential understandings of numerical analysis for both instructors and students alike.

1.1 Terminology

Before any further discussion, we may wish to outline a few basic ideas necessary to understand the contents of this paper. Recall that an ordinary differential equation (ODE) is an equation containing derivatives of one or more dependent variables with respect to only one independent variable. The order of an ODE is determined by the highest derivative appearing in the function. Symbolically, we may write the equation by solving uniquely for the highest order derivative. That is, we can rewrite the general form of an nth-order ordinary differential equation given by $F(t, y, y', \ldots, y^{(n)}) = 0$, where $F$ is a real-valued function of $n + 2$ variables, in the following manner:

$$\frac{d^n y}{dt^n} = f(t, y, y', \ldots, y^{(n-1)}),$$
where \( f \) is a real-valued continuous function \([27]\). This is the common notation displayed in most introductory ODE textbooks. Of course, we say the ODE is linear whenever \( F \) is linear.

Quite simply, ODEs appear naturally within settings that analyze change. We may readily find examples in such fields as science and technology, business, medicine, astronomy, data analysis and physics. And although many analytical techniques for deriving a solution to a particular ODE are widely known, the vast majority of ordinary differential equations cannot be solved exactly. Instead we construct an approximation to the solution. Here in lies the need for a numerical method (i.e. an algorithm that leads to an approximation). What follows is a short sketch of the type of problem and the kinds of methods we will be exploiting in our study of numerical stability.

1.2 A Brief Historical Overview of Numerical Methods

The first and perhaps simplest numerical method to greatly impact the study of numerical analysis was published in 1768 by Leonhard Euler in his work, *Institutionum calculi integralis* \([2]\). Often referred to as the forward Euler method, this algorithm is primarily used to solve an initial value problem by means of interpolation. That is, each iteration produces an approximated value of the dependent variable by moving a short distance along the line tangent to the previous value. Using our notation from above, consider the following example of a generic initial value problem for \( t \in [t_0, t_f] \):

\[
\frac{dy}{dt} = f(t, y(t))
\]

\[
y(t_0) = y_0.
\]

Given a step size \( h \), the interval \([t_0, t_f]\) is broken into a sequence of times at which the approximation curve will be constructed: \( t_0, t_1 = t_0 + h, t_2 = t_0 + 2h, ..., t_n = t_0 + nh, ..., t_f = t_0 + Nh \) where \( n = 0, 1, 2, ..., N \). For the sake of simplicity, let \( y_0 \doteq y(t_0), y_1 \doteq y(t_1), ..., y_n \doteq y(t_n), ..., y_N \doteq y(t_f) \) denote the interpolation points of the approximation. We find these points by the following time-stepping scheme:

\[
y_{n+1} = y_n + hf(t_n, y_n)
\]
for \( n = 0, 1, 2, \ldots, N - 1 \). The forward Euler method is also termed *explicit*, meaning that later values are calculated solely from current values. Notice that the simple elegance of this scheme lies in merely replacing the derivative in the initial value problem by a first order finite difference approximation [13].

In short, Euler’s work is now considered the genesis of present-day methods for solving initial value problems. Many years later in 1824, Augustin Cauchy introduced an *implicit* version of the forward Euler method. Here implicit means that the method utilizes both current and future values to form the solution. Cauchy’s innovation is also referred to as the backward Euler method [11]. This time-stepping scheme, written

\[
y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}),
\]

highlights an interesting difference from the explicit Euler method in that \( y_{n+1} \) appears on both sides of the scheme’s equation. This particular attribute influences the overall smoothness of the solution, which is of great importance when analyzing the stability of the approximation.

Not long after Cauchy’s use of an implicit Euler method, a letter penned by Francis Bashforth in 1855 brought to light the usefulness of a multi-step method credited to John Couch Adams [26]. Known today as the Adams-Bashforth methods, they are considered to be fundamental algorithms in the family of linear multi-step methods (LMMs). These methods, which we will study in further detail in the next section, combine values of the dependent variable and its derivative (computed in any number of previous steps) to approximate successive iterations [13].

After the multi-step model approach came the 1895 publication of Carl Runge and Martin Wilhelm Kutta’s method, which produced an approximation by taking weighted averages of \( y' \) at nearby values. The classical Runge-Kutta method, sometimes called the modified Euler method, is essentially a one-step method in which calculations from several time stages are utilized [11]. Germund Dahlquist and Åke Björck give a more intuitive notion for this method, “The idea behind the Runge-Kutta methods is to compute the value of \( f(t, y(t)) \) at several strategically chosen points
near the solution curve in the interval \((t_n, t_n + h)\), and to combine these values in such a way as to get a good accuracy in the computed increment \(y_{n+1} - y_n\) \([5]\). This “weighted point” idea can easily be seen in the simplest Runge-Kutta method known as Heun’s method \([5]\):

\[
\begin{align*}
k_1 &= hf(t_n, y_n), \\
k_2 &= hf(t_n + h, y_n + k_1), \\
y_{n+1} &= y_n + \frac{1}{2}(k_1 + k_2).
\end{align*}
\] (1.4)

As \((1.4)\) suggests, these methods are similar to the explicit and implicit Euler methods in that we wish to approximate \(y(t_n + h)\) by \(y_{n+1}\). It should be obvious then that these methods assume \(y(t_n)\) is known. In general, Runge-Kutta methods take on the following form when used to solve IVPs like \((1.1)\):

\[
y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i
\] (1.5)

where \(k_i = hf(t_n + c_i h, y_n + \sum_{j=1}^{s} a_{i,j} k_j), \quad i = 1, \ldots, s.\)

That is,

\[
\begin{align*}
k_1 &= hf(t_n + c_1 h, y_n + (a_{1,1} k_1 + a_{1,2} k_2 + \cdots + a_{1,s} k_s)) \\
k_2 &= hf(t_n + c_2 h, y_n + (a_{2,1} k_1 + a_{2,2} k_2 + \cdots + a_{2,s} k_s)) \\
&\vdots \\
k_s &= hf(t_n + c_s h, y_n + (a_{s,1} k_1 + a_{s,2} k_2 + \cdots + a_{s,s} k_s)).
\end{align*}
\] (1.6)

So, to define a certain method one must supply \(s\), and the coefficients \(a_{i,j}\) (for \(1 \leq i, j \leq s\)), the weights \(b_i\) (for \(i = 1, \ldots, s\)), and the nodes \(c_i\) (for \(i = 1, \ldots, s\)). This information is typically organized in what is called a Butcher array \([17]\):
Of course, within this brief outline of numerical methods used for solving initial value problems, countless other generalizations have surfaced including the Gaussian quadrature scheme, Moulton’s method, the Trapezoidal method, and Gear’s method, to name but a few. Derivation of the schemes mentioned thus far is not particularly important to the purpose of this paper. What needs to be addressed, though, is how we can best describe the approximations resulting from these methods. Qualities like convergence, consistency, and stability need to have ubiquitously concrete meanings if we mean to compare solutions from one algorithm to another. This is the heart of the matter at hand, particularly the issue of numerical stability.

1.3 Traditional Numerical Analysis

1.3.1 Global Error and Convergence

Our primary focus will be initial value problems for ordinary differential equations as outlined in the previous two sections. Traditionally, the way to analyze the methods used to solve such problems is to focus on the error propagation exhibited by the approximation. As an example, let’s consider the following initial value problem

\[
y'(t) = [\cos(y(t))]^2, \quad 0 \leq t \leq 10
\]

\[
y(0) = 0,
\]

which appears as exercise 8.2.1 in [1]. We will solve this problem using Euler’s method (1.2) with three different step sizes. Since the exact solution to this IVP is known to be \( y(t) = \arctan(t) \), we
can also compare selected values of the exact solution to the approximation values (see Table 1.1). The step sizes used to obtain the approximation values are specified in the third, fourth and fifth column headings of Table 1.1. Also, the time index \( n \) is provided for each approximation value. This serves mainly as a reminder to the reader that one should always be comparing values at a fixed time. That is, we cannot assume that by matching indices we are in fact comparing the same moment across various solutions [13]. For example, when \( h = 0.2 \), \( t_{10} = 2 \), but when \( h = 0.1 \), \( t_{10} = 1 \). Thus, in order to make sensible comparisons we need to first fix a time, \( T \), and then calculate the appropriate index, \( n = (T - t_0)/h \), at which to sample the approximation. In this example \( t_0 = 0 \).

<table>
<thead>
<tr>
<th>( t_n )</th>
<th>( y(t_n) )</th>
<th>( y_n, h = 0.5 )</th>
<th>( y_n, h = 0.2 )</th>
<th>( y_n, h = 0.05 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.10714872</td>
<td>1.19433632 (n=4)</td>
<td>1.14030595 (n=10)</td>
<td>1.11525376 (n=40)</td>
</tr>
<tr>
<td>4</td>
<td>1.32581766</td>
<td>1.36758989 (n=8)</td>
<td>1.34249312 (n=20)</td>
<td>1.32998470 (n=80)</td>
</tr>
<tr>
<td>6</td>
<td>1.40564765</td>
<td>1.42974153 (n=12)</td>
<td>1.41534958 (n=30)</td>
<td>1.40808374 (n=120)</td>
</tr>
<tr>
<td>8</td>
<td>1.44644133</td>
<td>1.46225706 (n=16)</td>
<td>1.45281769 (n=40)</td>
<td>1.44804386 (n=160)</td>
</tr>
<tr>
<td>10</td>
<td>1.47112767</td>
<td>1.48239031 (n=20)</td>
<td>1.47566539 (n=50)</td>
<td>1.47226792 (n=200)</td>
</tr>
</tbody>
</table>

It is easy to see from Table 1.1 that the approximation values computed by (1.2) will lie closer to the true solution values whenever \( h \to 0 \). This observation is further supported by Figure 1.1 where each numerical solution (in blue) is plotted individually with the analytical solution (in red).

![Figure 1.1: True Solution vs. Forward Euler Approximation](image-url)
Since values of the exact solution are known for this example, we can calculate the global error at a particular time. In other words, at time $T = t_n$ ($n$ being the number of steps needed to reach the time in question, i.e. $t_n = t_0 + nh$) we can measure the error in the approximation by finding the difference between the value of the true solution at time $t_n$ and the approximation value $y_n$,

$$e_n = |y(t_n) - y_n|.$$  

(1.8)

Table 1.2: Global Error at time $t_n$, $e_n = |y(t_n) - y_n|$

<table>
<thead>
<tr>
<th>$t_n$</th>
<th>$e_n, h = 0.5$</th>
<th>$e_n, h = 0.2$</th>
<th>$e_n, h = 0.05$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.08718760 (n=4)</td>
<td>0.03315723 (n=10)</td>
<td>0.00810504 (n=40)</td>
</tr>
<tr>
<td>4</td>
<td>0.04177223 (n=8)</td>
<td>0.01667546 (n=20)</td>
<td>0.00416704 (n=80)</td>
</tr>
<tr>
<td>6</td>
<td>0.02409388 (n=12)</td>
<td>0.00970193 (n=30)</td>
<td>0.00243609 (n=120)</td>
</tr>
<tr>
<td>8</td>
<td>0.01581573 (n=16)</td>
<td>0.00637636 (n=40)</td>
<td>0.00160253 (n=160)</td>
</tr>
<tr>
<td>10</td>
<td>0.01126264 (n=20)</td>
<td>0.00453772 (n=50)</td>
<td>0.00114025 (n=200)</td>
</tr>
</tbody>
</table>

By inserting the values from Table 1.1 into (1.8) we can estimate the global error for each approximation at $t_n = 2, 4, 6, 8, 10$ (these errors are displayed in Table 1.2). It appears that a decrease in step size mirrors a decrease in the global error at time $T = t_n$. Again we come to the conclusion that as $h \to 0$, the numerical solution seems to approach the true solution. This numerical investigation prompts our first definition of convergence.

**Definition 1.1.** [13] For all $t \in [t_0, t_f]$ if the global error $e_n = |y(t_n) - y_n| \to 0$ at time $t_n = t_0 + nh = T$ then the numerical method is said to converge to the solution $y(t)$ of the initial value problem at $t = T$.

Obviously only methods which are convergent are of any use and, intuitively, one might conjecture that any desired level of accuracy may be reached simply by taking smaller and smaller step sizes [13]. This is true when using simple schemes like the forward or the backward Euler method, but we need to generalize this definition for more complex schemes.
1.3.2 Consistency, Zero Stability and Global Error Bounds

As it turns out, Definition 1.1 is a bit too elementary. Technically speaking, the forward and backward Euler methods detailed in the previous section are considered “one-step methods,” meaning each value in the approximation is determined by the value preceding it. The implicit and explicit Euler methods are actually special cases of a larger family of methods known as the linear multistep methods (LMM), which involve additional previous points and derivative values.

For the IVP (1.1) with step size $h$, a linear combination of $y_n$ and $f_n = f(t_n, y_n)$ is used to form the general structure of a $k$-step LMM [13]:

$$y_{n+k} + a_{k-1}y_{n+(k-1)} + \ldots + a_0 y_n = h(b_k f_{n+k} + b_{k-1} f_{n+(k-1)} + \ldots + b_0 f_n), \quad (1.9)$$

with starting values $y_0 = \omega_0, y_1 = \omega_1, \ldots, y_{k-1} = \omega_{k-1}$ such that

$$\lim_{h \to 0} \frac{nh}{h} = t^* - t_0 \quad y_n = y(t^*), \quad (1.12)$$

holds for all $t^* \in [t_0, t_f]$.

Of course, we can simplify (1.9) by writing

$$\sum_{j=0}^k a_j y_{n+j} = h \sum_{j=0}^k b_j f(t_{n+j}, y_{n+j}) \quad (1.11)$$

and assuming $a_k = 1$. Notice (1.11) is an explicit scheme when $b_k = 0$. Now, Definition 1.1 needs to be adjusted in order to account for these additional starting values. We will adopt Griffiths and Higham’s definition of convergence for LMMs.

**Definition 1.2. [13]** A LMM (1.9) with starting values satisfying (1.10) is said to be convergent if for all IVPs (1.1) that possess a unique solution $y(t)$ for $t \in [t_0, t_f]$,

$$\lim_{h \to 0, nh = t^* - t_0} y_n = y(t^*), \quad (1.12)$$

holds for all $t^* \in [t_0, t_f]$. 
The only issue then lies in properly choosing coefficients for the LMM (1.9) so that convergence is guaranteed. In other words, to create the LMM we seek coefficients \(a_0, \ldots, a_{k-1}\) and \(b_0, \ldots, b_k\) so that for an arbitrary continuously differentiable function \(z(t)\),

\[
z(t_{n+k}) + a_{k-1}z(t_{n+(k-1)}) + \ldots + a_0z(t_n) = h(b_kz'(t_{n+k}) + b_{k-1}z'(t_{n+(k-1)}) + \ldots + b_0z'(t_n)) + \tau_n.
\]

(1.13)

Letting \(z = y\), where \(y' = f(t, y)\) and dropping the \(\tau_n = O(h^{p+1})\) term is how we achieve the general \(k\)-step LMM (1.11) [13]. To be precise, \(\tau_n\) denotes the local truncation error, whose general formula is given in [17] by

\[
\tau(t_{n+k}) = \frac{1}{h} \left[ \sum_{j=0}^{k} a_j y(t_{n+j}) - h \sum_{j=0}^{k} b_j f(y(t_{n+j})) \right].
\]

(1.14)

By Taylor series expansion of \(y(t_{n+j})\) and \(y'(t_{n+j})\), this gives us

\[
\tau(t_{n+k}) = \frac{1}{h} \left[ \sum_{j=0}^{k} a_j y(t_n) + \sum_{j=0}^{k} (ja_j - b_j) y'(t_n) + h \sum_{j=0}^{k} \left( \frac{1}{2}j^2a_j - jb_j \right) y''(t_n) + \ldots \right].
\]

(1.15)

Details of the above LTE expansion as well as the following definition can be found in [17].

**Definition 1.3.** [17] We say that an LMM is consistent if \(\tau \to 0\) as \(h \to 0\).

As LeVeque points out, this definition requires

\[
\sum_{j=0}^{k} a_j = 0 \quad \text{and} \quad \sum_{j=0}^{k} ja_j = \sum_{j=0}^{k} b_j.
\]

(1.16)

Thus, a method’s consistency depends only on the coefficients \(a_0, \ldots, a_{k-1}\) and \(b_0, \ldots, b_k\), not the IVP. In comparison to the global error defined in (1.8), it is important to understand that the global error is a propagation of the local error, \(\tau_n\), not a simple accumulation. So we must keep consistency solely in terms of the LTE.

The other condition surrounding the coefficients \(a_0, \ldots, a_{k-1}\) and \(b_0, \ldots, b_k\) is called zero-stability. Our definition for this term stems from Definition 1.3. Notice that the conditions in (1.16) can be
written much more compactly when we define the following polynomials \[ \frac{17}{17} \] :

\[
\begin{align*}
\rho(\xi) &= \sum_{j=0}^{k} a_j \xi^j, \\
\sigma(\xi) &= \sum_{j=0}^{k} b_j \xi^j.
\end{align*}
\tag{1.17}
\]

That is, the first condition in \( 1.16 \) is simply \( \rho(1) = 0 \) and the second condition of \( 1.16 \) can be re-written as \( \sigma(1) = \rho'(1) \). We call the polynomials in \( 1.17 \) the characteristic (or generating) polynomials of the LMM, and although they make short work of writing our consistency conditions, they are more commonly used to define zero-stability \[ \frac{17}{17} \].

**Definition 1.4.** \[ \frac{13}{13} \] We say that an LMM is **zero-stable** if the roots of \( \rho(\xi) \) lie on or within the unit circle.

In other words, the consistency and zero stability conditions surrounding our choice of coefficients for the LMM ensure that we not only have control over the local error, but we also have the added confidence that a small change in the initial value will result in a small change in the solution. We now have the tools required to utilize Dahlquist’s famous theorem, the proof of which can be found in \[ 5 \].

**Theorem 1.1.** \[ \frac{13}{13} \] A linear multi-step method is convergent if, and only if, it is both consistent and zero-stable.

Recall that we originally defined convergence by means of the global error. The next question one would logically ask is, what kind of error bounds can be computed? This question concerns error propagation. A slight adaptation of one such theorem is provided below from William C. Gear’s book, *Numerical Initial Value Problems in Ordinary Differential Equations* \[ 8 \].

Before stating the theorem, we must first clarify some of Gear’s notation. Let

\[
y_{n+1} = y_n + h\phi(y_n, t_n, h)
\tag{1.18}
\]

denote a general one-step method where \( \phi \) is defined using \( f(y(t), t) = y'(t) \) which is given in the statement of the initial value problem. And let \( \tau_n \) denote the local truncation error of the general
one-step method given by

\[ y(t_{n+1}) = y(t_n) + h\phi(y(t_n), t_n, h) - \tau_n. \]  \hspace{1cm} (1.19)

We will also make use of the following definition.

**Definition 1.5.** \[8]\ We say that \( \phi \) satisfies a Lipschitz condition on \( y \) if there exists a constant \( L \) such that

\[ |\phi(y, t, h) - \phi(\hat{y}, t, h)| \leq L|y - \hat{y}| \]

for all \( t_0 = 0 \leq t \leq t_f \) and all \( y, \hat{y} \).

Now observe that the difference between (1.18) and (1.19) yields:

\[ y_{n+1} - y(t_{n+1}) = y_n - y(t_n) + h(\phi(y_n, t_n, h) - \phi(y(t_n), t_n, h)) + \tau_n. \]  \hspace{1cm} (1.20)

By (1.8) notice that \([y_{n+1} - y(t_{n+1})]\) is the global error at time \( t_{n+1} \) and \([y_n - y(t_n)]\) is the global error at time \( t_n \). If we let \( \phi(y_n, t_n, h) - \phi(y(t_n), t_n, h) \leq e_n L_f \) then from (1.20) we have

\[
e_{n+1} \leq e_n + h(e_n L_f) + \tau_n
= e_n (1 + hL_f) + \tau_n.
\]

Taking absolute values,

\[
|e_{n+1}| \leq |e_n (1 + hL_f) + \tau_n|
\leq |e_n(1 + hL_f)| + |\tau_n|
\leq |e_n|(1 + h|L_f|) + |\tau_n|.
\]  \hspace{1cm} (1.21)

When \( \phi \) satisfies a Lipschitz condition then we have \(|L_\phi| \leq L_f = L\), and by making use of Taylor
series expansion we can bound the local truncation error for a method of order \(r\),

\[ |\tau_n| \leq Dh^{r+1} \]

where \(D\) is determined by \(y''\) and computable constants. And so we rewrite (1.21) as

\[ |e_{n+1}| \leq |e_n|(1 + hL) + Dh^{r+1}. \tag{1.22} \]

With this we prove the following lemma.

**Lemma 1.1.** \([8]\) If the total error at time \(t_n\) satisfies (1.22) for \(t_0 \leq t \leq t_f\), then

\[ |e_n| \leq Dh^{r+1}(1 + hL)^n - \frac{1}{hL}(1 + hL)^n|e_0| \]

\[ \leq \frac{Dh^{r+1}}{hL}((e^{Lt_f} - 1) + e^{Lt_f}|e_0|). \]

**Proof.**

We must show that the first inequality holds for all \(n \geq 0\), so we proceed by induction.

When \(n = 0\),

\[ |e_0| \leq Dh^r\left(\frac{(1 + hL)^0 - 1}{L}\right) + (1 + hL)^0|e_0| \]

\[ = Dh^r\left(\frac{1 - 1}{L}\right) + |e_0| \]

\[ = |e_0|. \]

Now we assume the first inequality is true for all \(n\) and show that the inequality holds for \(n + 1\).

\[ |e_{n+1}| \leq |e_n|(1 + hL) + Dh^{r+1} \]

\[ \leq \left[ Dh^{r+1}\left(\frac{(1 + hL)^n - 1}{hL}\right) + (1 + hL)^n|e_0| \right] (1 + hL) + Dh^{r+1} \]

\[ = \left[ Dh^{r+1}\left(\frac{(1 + hL)^n - 1}{hL}\right) (1 + hL) + (1 + hL)^{n+1}|e_0| + Dh^{r+1} \right]. \]
Hence,

\[
|e_{n+1}| \leq Dh^{r+1} \left( \frac{[(1 + hL)^n - 1](1 + hL)}{hL} + 1 \right) \left( 1 + hL \right)^{n+1} |e_0| \\
= Dh^{r+1} \left( \frac{1 + hL}{hL} \right)^{n+1} + (1 + hL)h^{n+1} |e_0| \\
= Dh^{r+1} \left( \frac{(1 + hL)^{n+1} - 1}{hL} \right) + (1 + hL)h^{n+1} |e_0|.
\]

Since \( t_0 = 0 \leq t = hn \leq t_f \) and for \( hL \geq 0, 1 + hL \leq e^{Lh} \), we have \( (1 + hL)^n \leq e^{Lh} \leq e^{Ll_f} \).

This shows the second inequality, which proves the lemma \[8\]. \( \square \)

Now we’re ready to prove Gear’s theorem for total error.

**Theorem 1.2.** \[8\] Let \( \phi(y, t, h) \) be continuous in \( y, t \) and \( h \) for \( t_0 = 0 \leq t \leq t_f, 0 \leq h \leq h_0 \) and all \( y \). If \( \phi \) satisfies a Lipschitz condition for all \( t \) and \( |\tau_n| \leq Dh^{r+1} \), then the global error is bounded by

\[
|y_n - y(t_n)| \leq Dh^r \frac{e^{Lt_f} - 1}{L} + e^{Lt_f} |y_0 - y(t_0)|. \tag{1.23}
\]

**Proof.**

We consider the global error at time \( t_{n+1} \) as defined in \(1.20\). Taking absolute values we have

\[
|e_{n+1}| = |e_n + h(\phi(y_n, t_n, h) - \phi(y(t_n), t_n, h)) + \tau_n| \\
\leq |e_n| + h|\phi(y_n, t_n, h) - \phi(y(t_n), t_n, h)| + |\tau_n| \\
\leq |e_n| + hL|e_n| + Dh^{r+1} \\
= |e_n|(1 + hL) + Dh^{r+1}.
\]

From here we apply the lemma and obtain the result of the theorem \[8\]. \( \square \)

The reader should take note here that the bound on the global error is exponential, which is troubling. We will return to this result in Chapter 3 in a further discussion on error propagation.

1.3.3 Additional Stability Issues

In reference to Theorem 1.1 it should be explicitly stated that consistent methods are not always convergent. That is, consistency is a necessary condition for convergence but insufficient without the presence of stability, as Theorem 1.1 points out. Unfortunately, there is still a prob-
lem. If consistency and zero-stability imply convergence as in Definition (1.2), then we are still dealing with stability in terms of a limit depending upon a refined step size for LMMs. As stated earlier in the discussion on global error, a decrease in step size will cause the numerical solution to approach the true solution. However, an arbitrarily small step size could potentially incur tremendous computational expenses. Thus, we need to know more than the simple fact that a method will converge in the “limiting sense.” Ideally, we need a stronger form of stability, one that is applicable for a fixed step size [17].

This leads us to the widely used notion of absolute stability, traditionally introduced in numerical analysis textbooks by examining the behavior of LMMs on the following IVP test problem with initial data \( y(t_0) = y_0 \) and a given time step \( h \):

\[
y'(t) = \lambda y(t) + g(t).
\] (1.24)

In fact, most texts consider the simplest case of (1.24) by assuming \( g(t) = 0 \) (see [1]). A small convention is enforced here by reasoning that \( \lambda \) may in fact be complex, and if so, we thereby assume its real part to be negative. The reason for this is simple. We can readily observe that the general solution to (1.24) when \( g(t) = 0 \) is of the form \( y(t) = ce^{\lambda t} \) for some constant \( c \), which will tend to zero through indefinite time when the real part of \( \lambda \) is negative. Thus, a more useful definition of stability for a fixed step size should describe LMMs that produce solutions exhibiting behavior similar to the general solution when applied to the given test problem. We will again appeal to the Griffiths and Higham text for our next definition.

**Definition 1.6.** [13] A linear multi-step method with a given fixed step size is absolutely stable if, when applied to the test problem \( y'(t) = \lambda y(t) \) where the real part of the complex number \( \lambda \) is negative, its solutions tend to zero as \( t_n \to \infty \) for any choice of starting values.

Notice that by defining absolute stability in this way we are still ensuring that the global error decreases over time. However, as Griffiths and Higham point out, absolute stability is not guaranteed for every choice of step size. Depending on the LMM, there are in fact different regions
of stability \[1\]. For example, let’s apply the “one step” explicit Euler method (1.2) to our test problem. This yields the following:

\[
y_{n+1} = y_n + h(\lambda y_n) \\
= (1 + h\lambda)y_n
\]  \hfill (1.25)

where \((1 + h\lambda)\) is called the growth factor \[1\]. By Definition (1.6) we see that the explicit Euler method is absolutely stable when \(|1 + h\lambda| < 1\). Hence, for a positive real-valued step size \(h\) and a complex-valued \(\lambda\) with negative real part, \(|1 + h\lambda| < 1\) maps the open disk of radius 1 centered at -1. Notice how this restricts our choice of step size. When \(\lambda\) is simply a negative real number, we have \(h < -2/\lambda\). In other words, to achieve absolute stability using the explicit Euler method, our step size needs to be comparatively small, in which case the method is often said to be conditionally stable.

For another example, let’s find the absolute stability region for the implicit Euler method. This time we apply (1.3) to the test problem, and with a few steps of elementary algebra we arrive at

\[
y_{n+1} = \frac{y_n}{(1 - h\lambda)}
\]

where \((1 - h\lambda)\) is the growth factor \[1\]. Now notice that for negative real \(\lambda\), we are guaranteed \((1 - h\lambda) < 1\) for any positive real step size. The ability to choose \(h\) “arbitrarily large” is why the implicit Euler method is called unconditionally stable. Of course, in practice the step size is still chosen to be rather small, but in terms of absolute stability we needn’t worry when using this implicit method \[17\].

The concept of absolute stability is especially important when analysing numerical solutions to stiff problems, which exhibit a somewhat counterintuitive behavior. For some problems when the solution curve oscillates wildly, one would assume that an extremely small step size would be necessary to construct an approximation. Similarly, one might assume that a larger step size can be chosen for smooth solution curves. This however is not necessarily the case with stiff problems.
Sometimes (even when the solution approaches a line with almost zero slope) the numerical system forces the step size to be exceedingly small. For this reason, the absolute stability region for stiff problems must be the entire left half of the complex plane. Such methods are called $A$-stable \cite{17}.

By now the problem should be obvious. The trouble is that our definition of stability tends to fluctuate depending on the problem, and sometimes (as with stiff equations) stability depends on the system itself. We will explore in further detail several of these issues in Chapter 2 from an academic standpoint, while Chapter 3 will attempt to remedy the current analysis of error propagation.
CHAPTER 2 THE INSTRUCTIONAL OVERSIGHT

The biggest problem underlying the known literature on numerical stability is not the clarity of any one definition but the lack of similitude from one author’s interpretation to another. Some textbooks define particular types of stability within the context of specific examples while others simply avoid giving a definition altogether. A quick survey of numerical analysis textbooks may best demonstrate this point.

2.1 Indecipherable Context

Let’s first tackle the contextual confusion surrounding stability. What follows is several segments taken from three different textbooks on numerical analysis, all of which deal with numerically solving initial value problems. Our first passage comes from the third edition of Atkinson and Han’s text, Elementary Numerical Analysis:

When numerically solving the initial value problem \( y' = f(t, y(t)) \) given \( y(t_0) = y_0 \), we will generally assume that the solution \( y(t) \) is being sought on a given interval \( t_0 \leq t \leq t_f \). In that case, it is possible to obtain the following result on stability. Make a small change in the initial value for the initial value problem, changing \( y_0 \) to \( y_0 + \epsilon \). Call the resulting solution \( y_\epsilon(t) \) [such that] \( y_\epsilon'(t) = f(t, y_\epsilon(t)) \) and \( y_\epsilon(t_0) = y_0 + \epsilon \), where \( f \) is a continuous function. Then it can be shown that for some \( c > 0 \) and for all small values of \( \epsilon \),

\[
\max_{t_0 \leq t \leq t_f} |y_\epsilon(t) - y(t)| \leq c\epsilon
\]

Thus, small changes in the initial value \( y_0 \) will lead to small changes in the solution \( y(t) \) of the initial value problem. This is a desirable property for a variety of very practical reasons. Virtually all initial value problems are stable in the sense specified in (2.1); but
is only a partial picture of the effect of small perturbations of the initial value \( y_0 \). If the max error in (2.1) is much larger than \( \epsilon \), then the initial value problem is usually considered to be \textit{ill-conditioned}.

(Atkinson and Han 373)

The main idea provided by this definition, which is perhaps the most commonly cited among undergraduate textbooks, is that the numerical solution must have a continuous reliance on the initial value. The magnification of error at later stages depends on the error occurring in the beginning stages, which according to Atkinson and Han has less to do with the method and more to do with the problem itself. Now consider the following excerpt from Melvin J. Maron’s \textit{Numerical Analysis A Practical Approach} [18]:

A formula of the form \( y_{j+1} = y_j + h\phi \) can generally be viewed as the result of replacing \( y' \) in an IVP by a finite difference approximation. Although it is not immediately apparent, such replacements can introduce \textit{parasitic terms} of the form \( C e^{vt_j} \) and/or \((-1)^j C e^{vt_j}\) into the calculated solution that have no relation to the exact solution. If \( v > 0 \), they grow exponentially in magnitude as \( t_j \) increases, in which case the method is called \textbf{unstable} for the IVP. The \textbf{stability} of a method can be determined by examining the step sizes \( h \) for which the method gives accurate solutions to three test problems:

(I) \( y' \equiv 0, y(0) = 1 \)

(II) \( y' \equiv 1, y(0) = 0 \)

(III) \( y' \equiv \lambda y, y(0) = 1 \).

Methods that exhibit instability for Test Problems I or II for some values of \( h \) are called \textit{strongly unstable}; those that exhibit instability for some values of \( \lambda \) in Test Problem III are called \textit{weakly unstable}.

(Maron 359)
In Maron’s definition there is a slight duality. The stability depends on whether or not the method produces exponential terms in the solution as well as the step size for the test problems. However it is not clear which is more favorable - methods that produce solutions devoid of parasitic terms or methods that “work” independent of the step size (even though we have no indication of what is meant by the phrase “the method gives accurate solutions”). In fact, the last sentence of of the above excerpt leads us to believe that systematically checking different step sizes is more of diagnostic step for categorizing the type of instability. Finally, let us examine the observation given in Stephen Kellison’s *Fundamentals of Numerical Analysis* [16]:

Several methods are available in analyzing stability. The first is to perform the solution algorithm twice carrying more decimal places the second time than the first, that is, in computer terminology to do the solution both in single precision and double precision. This is one way of controlling the accumulation of the round-off error component of total error. (Kellison 241)

In contrast to the previous two definitions, Kellison interprets error propagation as an argument of accuracy versus precision, as though stability were influenced solely by the inexactness of finite precision arithmetic. We will see in the next section that this is not how we want to think about error propagation.

Given the above passages, it would appear that if a numerical analyst were asked to give a presentation on stability, as a general tool used within the practice of error analysis, he or she may actually discuss one of three things: the stability of a given initial value problem, the stability of the numerical method used to solve the problem, or the stability of the approximation itself. It is not entirely clear which scenario is preferred. Do we frame the problem in a particular way before even making an attempt at the solution? Do we compile a list of conditions under which only certain numerical methods are acceptable? Or should the analysis be focused on the characteristics exhibited by the approximation alone? In fact, Kellison states that stability is actually dependent on all three factors quoted above [16]:
1. The method used

2. The differential equation being solved

3. The interval $h$ chosen to interpolate the values in the approximation.

These excerpts are just a few of the many interpretations of numerical stability. In the eyes of a student, the lack of similitude makes it very difficult to cross-reference definitions from different texts. Even more problematic is the instructor’s duty to not only be able to explain the differences but to know which one is the correct interpretation.

2.2 Inescapable Ambiguity

Another issue the average undergraduate in mathematics faces is the ever growing list of the different kinds of stability. For example, scrolling through the index of John Dormand’s Numerical Methods for Differential Equations: A Computational Approach one would not only find the word stability as one of the entries, but also a sub-list of the following types: A-stability, absolute stability, extended stability, relative stability, strong stability, weak stability and zero stability [6]. Recall that only two of these terms (absolute and zero stability) appeared in the general analysis of LMM’s as discussed in Section 1.3. Alternatively, Celia and Gray’s book, Numerical Methods for Differential Equations: Fundamental Concepts for Scientific and Engineering Applications, discusses conditional stability, neutral stability and unconditional stability [3].

In light of the terms listed above, which is by no means exhaustive, the obvious question awaiting any instructor is - can the types of stability be qualitatively ranked? According to John R. Rice, author of Numerical Methods, Software, and Analysis, “the word stability appears throughout numerical computations and, in general, refers to continuous dependence of a solution on the data of the problem or method. Stability also has precise technical meanings (not all the same) in different areas as well as this general one” [20]. In other words, without some definitive means of comparing one type of stability to another, there is little to no utility in establishing categories at all. Especially if there is no singular definition of stability that is itself ubiquitous to all contexts.

For example, Kellison specifically typifies a method as relatively stable “if an error introduced
into the solution process behaves similarly to the true solution, that is, increases or decreases in proportion to the true solution so that relative error remains constant or even decreases” [16]. But this is essentially the definition given by Atkinson and Han which says that a method is considered stable when a small change does not result in unbounded absolute error [1]. Kellison’s use of the word “relatively” insinuates a lower quality when in reality, these two authors are expressing the same definition.

A similar problem occurs when two authors appear to be giving the same definition but with slightly different wording. Compare Atkinson and Han’s definition from the previous section to the following definition from Greenbaum and Chartier’s book, *Numerical Methods: Design, Analysis and Computer Implementation of Algorithms* [12]:

The general, explicit one-step method can be written in the form

\[ y_{k+1} = y_k + h\phi(t_k, y_k, h). \] (2.2)

The one-step method (2.2) is stable if there is a constant \( K \) and a step size \( h_0 \) such that the difference between two solutions \( y_n \) and \( \tilde{y}_n \) with initial values \( y_0 \) and \( \tilde{y}_0 \), respectively, satisfies

\[ |y_n - \tilde{y}_n| \leq K|y_0 - \tilde{y}_0|, \] (2.3)

whenever \( h \leq h_0 \) and \( t_0 + nh \leq t_f \). (Greenbaum and Chartier 270)

Both definitions might be interpreted to say that stability is a kind of boundedness. That is, the difference between two solutions is proportional to the difference in their initial values. However, these two definitions do not exactly match since Atkinson and Han are defining stability in terms of the initial value problem, while Greenbaum and Chartier are defining stability in terms of the numerical method.

Oddly enough, some textbooks side-step the details altogether. It is interesting that Maron prompts the detailed investigation of test problems to analyze a method’s stability, when Rice
condenses his definition into one sentence: a numerical method is stable “if the error in the final computed value is proportional to the sum of the local errors made in the steps during the calculation” [20]. Similarly, the third edition to Devi Prasad’s textbook, An Introduction to Numerical Analysis, covers a myriad of algorithms with examples (Euler’s method, Runge-Kutta methods of various orders, and several multi-step methods like Milne’s method and the Adams-Bashforth methods) yet never once mentions stability throughout the entire chapter on ODEs [19].

Clearly over the years stability has not only become a homonym for a variety of settings, but so many variations of the definition have surfaced that it has become impossible to distinguish an optimal description for the approximation. If the goal of mathematics education is to enlighten and train the next generation of analysts, then glossing over the details is no longer an option.
CHAPTER 3  NUMERICAL SMOOTHING

3.1 Error Splitting

In order to shift away from a dependence on the vague numerical stability concepts we must return to our earlier discussion on error analysis, more specifically the understanding of global error and local error. Recall that while we stated that global error was not a simple accumulation of local error, we can define global error in terms of the local error and propagation error. This is called error splitting.

Again we consider the initial value problem \[ \text{(1.1)} \] and assume there exists a smooth solution, \( y(t) \). In practice, we do not know \( y(t) \), so we must apply some numerical scheme to approximate it. Let \( y_N(t) \) denote the numerical approximation (for a visual interpretation see Figure 3.1 [left] where a hypothetical \( y(t) \) and \( y_N(t) \) are plotted over some arbitrary interval, \( [t_n, t_{n+1}] \)). By our formula for global error \[ \text{(1.8)} \], the total error at time \( t_{n+1} \) is given by

\[
e_{n+1} = y(t_{n+1}) - y_N(t_{n+1}). \tag{3.1}
\]

Going one step further, we can actually “split” \[ \text{(3.1)} \] by adding and subtracting a theoretical value, \( \tilde{y}_N(t_{n+1}) = y(t_n) + h\phi(y(t_n), t_n, h) \):

\[
y(t_{n+1}) - y_N(t_{n+1}) = \left[ y(t_{n+1}) - \tilde{y}_N(t_{n+1}) \right] + \left[ \tilde{y}_N(t_{n+1}) - y_N(t_{n+1}) \right]. \tag{3.2}
\]

This new term represents an approximation value at \( t_{n+1} \), which is found by applying the numerical scheme to the unknown value, \( y(t_n) \), at time \( t_n \). In other words, \( \tilde{y}_N(t_{n+1}) \) is not computable in practice, but it gives us a way of dividing the total error at \( t_{n+1} \) into the local error and the error propagated by the numerical scheme \[ \text{(22)} \] (see Figure 3.1 [right]). It is beneficial to note here the connection between our current notation and the notation we used in Gear’s theorem (Theorem 1.2) from Chapter 1. That is, \( y_{n+1} = y_N(t_{n+1}) = y_N(t_n) + h\phi(y_N(t_n), t_n, h) \).
Now to be clear, the first difference in the right hand side of (3.2), \( [y(t_{n+1}) - \tilde{y}_N(t_{n+1})] \), represents the local error. Defining local error in this way is akin to finding the residual after plugging \( y(t) \) into the numerical scheme. The second difference, \( [\tilde{y}_N(t_{n+1}) - y_N(t_{n+1})] \), represents the error propagated by the numerical scheme \([22]\). Figure 3.1 is not meant to be misleading. The size of the numerical propagation error at \( t_{n+1} \) may be smaller than, equal to or larger than the size of the total error (between the true solution and the approximation) at the previous time step, \( t_n \). Ideally, we would want the propagated error to be smaller. That is, for step size \( h \) and constant \( \delta \) depending on the numerical scheme,

\[
\|\tilde{y}_N(t_{n+1}) - y_N(t_{n+1})\| \leq (1 + \delta h)\|y(t_n) - y_N(t_n)\| \tag{3.3}
\]

in an appropriate norm \([22]\). Equation (3.3) is called the Von Neumann stability condition in the literature. Notice that (3.3) is only problematic when \( \delta > 0 \). When \( \delta < 0 \) the propagated error behaves as desired, and when \( \delta = 0 \) we in fact have a simple accumulation (or “error transportation”) of local error into the global error.

Several different theorems have come about over the years in an attempt to justify this method of error splitting. The proofs of which conclude that an appropriate bound can be found for the global error at any time step. We have already seen a common misconception concerning this total error bound within the simplest context of one-step methods:

**Theorem 3.1.** \([8]\) Let \( y_{n+1} = y_n + h\phi(y_n, t_n, h) \) be a one-step method of order \( r \) for approximating
the solution of a differential equation where $\phi$ is determined by $f(y(t), t) = y'(t)$ as stated in the initial value problem. If $\phi(y, t, h)$ is continuous and satisfies a Lipschitz condition and if the local truncation error is bounded by $Dh^{r+1}$, then

$$|y_N - y(t_N)| \leq Dh^r \frac{e^{Lt_f} - 1}{L} + e^{Lt_f} |y_0 - y(t_0)| \quad (3.4)$$

where $t_f = Nh + t_0$.

The problem with this bound on the total error is of course the presence of the exponential function. $L$ depends on $\phi(y, t, h)$, namely the scheme. It tends to be very large for complex equations and schemes. Even for a modest value, say $L = 10$, and a short time interval where perhaps $t_f = 10$, then already we have $e^{Lt_f} = e^{100} > 10^{43}$, which is unbearably large. One simply cannot afford to make the step size $h$ small enough to counteract such an effect. A seemingly boundless global error estimate does little to justify the error splitting demonstrated in Figure 3.1 except for theoretical convergence. Therefore, we need to move away from error propagation that depends on the scheme.

3.2 Revised Error Splitting

Essentially, proving (3.3) is equivalent to proving numerical stability, which is known to be a sufficient condition for convergence of a consistent scheme. However, it is also well known that proving this version of numerical stability can be incredibly difficult depending on the context of the IVP and the numerical scheme used. We need a way around this task, which we will accomplish in the following way.

Consider replacing $\tilde{y}_N(t_{n+1})$ in (3.2) with $\hat{y}(t_{n+1})$:

$$y(t_{n+1}) - y_N(t_{n+1}) = [y(t_{n+1}) - \hat{y}(t_{n+1})] + [\hat{y}(t_{n+1}) - y_N(t_{n+1})]. \quad (3.5)$$

The reader should interpret $\hat{y}(t_{n+1})$ as a true solution value for the same ODE but with $y_N(t_n)$ acting as the initial condition at $t_n$. That is, $\hat{y}(t)$ satisfies (1.1) with $\hat{y}(t_n) = y_n = y_N(t_n)$ (see
Figure 3.2: (Left) Traditional Error Splitting (Right) Revised Error Splitting

Figure 3.2). For convenience we have provided the error splitting found in the current literature again. Compare and contrast the traditional method of error splitting to the revised error splitting in Figure 3.2.

Now we can think of the global error as the sum of the the error propagated by the ODE, \([y(t_{n+1}) - \hat{y}(t_{n+1})]\), and the local error, \([\hat{y}(t_{n+1}) - y_N(t_{n+1})]\), as in (3.5). By splitting the global error so that numerical error propagation is eliminated from the analysis, we are able to direct our attention away from the worrisome task of establishing numerical stability. Instead, we focus on something called numerical smoothness \([22]\). Notice that our ability to estimate \([\hat{y}(t_{n+1}) - y_N(t_{n+1})]\), the local error, depends on the smoothness of \(\hat{y}(t)\). But \(\hat{y}(t)\) is a solution to the ODE with a numerically computed initial value, \(y_N(t_n)\). Thus, the behavior of \(\hat{y}(t)\) really depends on how the values preceding \(y_N(t_n)\) were calculated by the numerical scheme \([22]\).

In other words, the smoothness of \(\hat{y}(t)\) depends on whether or not the numerical scheme used to reach \(y_N(t_n)\) has “smoothing properties” - meaning the scheme can prohibit the solution \(\hat{y}(t)\) from changing wildly throughout \([t_n, t_{n+1}]\) in the presence of stiff components. Table 3.1 may be helpful in understanding the difference between these two types of error splitting. The splitting shown in (3.2) tells us that the ODE controls the smoothness of our solution, while the numerical scheme is responsible for the consistency displayed in the local error as well as the propagation error. On the other hand, the splitting in (3.5) gives the ODE control over the propagation error and leaves the numerical scheme responsible for controlling the local error. This way, consistency becomes the sole job of the numerical scheme and we can ensure numerical smoothness at any time step as long
as the scheme exhibited smoothing in the previous time step. Hence, we can guarantee control over

<table>
<thead>
<tr>
<th>ODE Scheme</th>
<th>Splitting (3.2) smoothness error propagation &amp; consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Splitting (3.5) error propagation smoothness &amp; consistency</td>
<td></td>
</tr>
</tbody>
</table>

the local error at each step by deliberately using numerical schemes that exhibit smoothing, thereby eliminating the burden of proving numerical stability. Also, unlike numerical stability, establishing numerical smoothness is not such a costly process nor is it dependent upon a particular context. An even greater advantage is that we can still use consistency along with numerical smoothness to prove convergence. The only question is, how do we know whether or not a scheme possesses smoothing properties?

### 3.3 Smoothing Behavior

Conveniently enough, we may return to our model problem (1.24) for ODEs to answer this question. We will assume $g(t) = 0$ and $\lambda$ is defined as in Definition 1.6 for the non-stiff case with $|\lambda| \leq M < \infty$ for some finite real number $M$. Recall that in Section 1.3, we deduced the general solution to be $y(t) = ce^{\lambda t}$ for some constant $c$. If we fix a positive time $T$, it is easy to see that for any $\lambda$ this solution and its derivatives are uniformly bounded [22]:

\[
|y^{(k)}(T)| = |\lambda^k y(T)| \\
= |\lambda^k ce^{\lambda T}| \\
= |\lambda^k||c||e^{\lambda T}| \\
\leq |\lambda|^k|c||e^{\lambda T}| \\
\leq |\lambda|^k|c||e^{\lambda T}| \\
\leq M^k e^{MT} |c|.
\]

The term uniformly bounded is used since we are considering a function that depends on two parameters, $\lambda$ and $k$. In fact, we may consider this a smoothing property of the model equation
Now, just like in our study of numerical stability, we want our scheme to exhibit the same behavior as the solution to the test problem.

We begin by showing a criterion for a Runge-Kutta scheme to have smoothing properties. Recall from Section 1.2 the general form of a Runge-Kutta method:

\[ y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \]  \hspace{1cm} (3.7)

where

\[ k_i = f(t_n + c_i h, y_n + h \sum_{j=1}^{s} a_{i,j} k_j) \], \hspace{0.5cm} i = 1, ..., s

and \( y'(t) = f(t, y(t)) \). Applying this method to the test problem means

\[ k_i = \lambda (y_n + h \sum_{j=1}^{s} a_{i,j} k_j) \] \hspace{0.5cm} \text{for } i = 1, ..., s.

Letting \( \mu = h \lambda \) we have

\[ Y^i = y_n + \mu \sum_{j=1}^{s} a_{i,j} Y^j \], \hspace{0.5cm} i = 1, ..., s \]  \hspace{1cm} (3.8)

where the superscripts denote the time step index for approximations \( Y^1, Y^2, ... \) satisfying

\[ Y^n \approx y(t_n). \]

We can manipulate (3.8) a bit further by letting \( Y = (Y^1, Y^2, ..., Y^s)^T \) and \( A \) be the matrix with entries \( a_{i,j} \):

\[ Y = y_n e + \mu A Y \]
where \( e \in \mathbb{R}^n \) denotes a vector of ones. Thus,

\[
Y - \mu AY = y_n e \\
Y(I - \mu A) = y_n e \\
Y = y_n (I - \mu A)^{-1} e.
\]

This tells us that (3.7) applied to the test problem can be written as follows [7]:

\[
y_{n+1} = y_n + \mu \sum_{j=1}^{n} b_j Y^j \\
= y_n + \mu b^T Y \\
= y_n + \mu b^T (y_n (I - \mu A)^{-1} e) \\
= [1 + \mu b^T (I - \mu A)^{-1} e] y_n \\
= R(\mu) y_n.
\] (3.9)

We call \( R(h\lambda) = 1 + \mu b^T (I - \mu A)^{-1} e \) the stability function. In fact, another way to represent (3.9) is to write \( y_n = [R(\mu)]^n y_0 \) for \( y_0 = y(0) \). What we need to do now is to examine the solution given by

\[
\hat{y}(t) = y_n e^{\lambda(t-t_n)}
\] (3.10)

since the local error in (3.5) depends on the smoothness of \( \hat{y}(t) \) in \( (t_n, t_{n+1}) \). We proceed as before, working with the k-th derivative of (3.10):

\[
\hat{y}^{(k)}(t) = y_n \lambda^k e^{\lambda(t-t_n)} \\
= [R(\mu)]^n \lambda^k y_0 e^{\lambda(t-t_n)} \\
= S(\lambda, h, T) y_0 e^{\lambda(t-t_n)}
\] (3.11)

where \( S(\lambda, h, T) = \lambda^k [R(\mu)]^n \) and \( n = \frac{T}{h} \) represents the number of steps for a fixed positive time \( T \) and step size \( h \). Compare (3.11) to (3.6). We showed \( y^{(k)}(T) = \lambda^k ce^{\lambda T} \) was uniformly
bounded for a fixed positive time \( T \) and concluded that our test problem must have smoothing properties. A similar understanding applies here. The boundedness of \( S(\lambda, h, T) \) will prove that \( \hat{y}^{(k)}(t) \) will be bounded in each time step. Hence, the general Runge-Kutta scheme (3.9) will have a smoothing effect. We call \( S(\lambda, h, T) \) the smoothing function and \( y_n \lambda^k \) the smoothing indicator of the model problem [22]. We summarize this notion in the following definition, in which the conditions imposed upon the parameter \( \lambda \) ensure that our definition holds in the presence of stiff ODEs and certain PDE problems.

**Definition 3.1.** [22] Let \( \mathcal{D} \) be a subset of \( \{(h, \lambda) : h > 0, \lambda \in \mathbb{C}, |\lambda| \leq M < \infty, |Im(\lambda)| \leq -\tan(\alpha)Re(\lambda) \} \) for some nonnegative real number \( M \) and \( 0 < \alpha < \frac{\pi}{2} \). If there exists a positive continuous function \( K(T) \) for \( T > 0 \) such that \( |S(\lambda, h, T)| \leq K(T) \) for all \( (h, \lambda) \in \mathcal{D} \), then the scheme has a smoothing effect of order \( k \) in \( \mathcal{D} \).

Now, we use Definition 3.1 to finish our discussion on the smoothing properties of general Runge-Kutta schemes as originally discovered and written in [22]. According to (3.9), since the stability function \( R(\mu) \) is a rational function of \( \mu \), we let \( P(\mu) \) and \( Q(\mu) \) be polynomials for \( \mu = h\lambda \) so that

\[
R(\mu) = \frac{P(\mu)}{Q(\mu)}.
\]

Therefore, the boundedness of \( S(h, \lambda, T) \) - and hence the smoothing effect of (3.9) - depends on the degree of \( P(\mu) \) and the degree of \( Q(\mu) \). Let’s first consider the case when the degree of \( P(\mu) \) is greater than the degree of \( Q(\mu) \). When this happens, as \( |\mu| \to \infty \), then \( |R(\mu)| > 1 \), and so the stability region will be bounded. Now assume the stability region is contained in the disk \( \{\mu : |\mu| \leq K < \infty\} \) (i.e. \( |h\lambda| \leq K \) ). Then for any \( \mu = h\lambda \) in a compact subset of the stability region where \( |R(\mu)| \neq 1 \), there exists a real number \( \beta \in (0, 1) \) such that \( |R(\mu)| \leq \beta \). Hence, \( |S(h, \lambda, T)| \leq |\lambda|^k \beta^n \leq |\lambda|^k \beta^{\frac{T}{h} |\lambda|} \) where \( n = \frac{T}{h} \) is uniformly bounded. Notice here that taking a very small step size \( (h) \) will make \( n \) (the number of steps) increase, which causes \( \beta \) to be very small. In other words, shrinking the step size lowers the upper bound on \( S(\lambda, h, T) \) and magnifies the smoothing effect. This case also leads us to conclude that explicit schemes, like the forward
Euler method, have smoothing properties wherever their stability regions are defined [22].

Secondly, we consider the case when the degree of $P(\mu)$ is less than the degree of $Q(\mu)$. Without loss of generality assume $T = kmh$ for some $m \in \mathbb{Z}$. If $-\text{Re}(\mu)$ is large enough, then there exists $\epsilon > 0$ so that $|R(\mu)| \leq \frac{1}{1-\epsilon \text{Re}(\mu)}$. Consequently,

$$
|S(h, \lambda, T)| = |\lambda|^k \left| \frac{1}{1-\epsilon \text{Re}(\mu)} \right|^{\frac{T}{\pi}} \\
= |\lambda|^k \left| \frac{1}{1-\epsilon \text{Re}(\mu)} \right|^{km} \\
= \left| \frac{\lambda}{|1-\epsilon \text{Re}(\mu)|^m} \right|^k \\
\leq \left| \frac{k \sqrt{1+\epsilon^2}}{T} \right|^k.
$$

(The last inequality follows from a simple Calculus argument.)

Lastly, we consider the case when the degree of $P(\mu)$ and $Q(\mu)$ are in fact equal. This requires us to examine when $|R(\mu)| < 1, |R(\mu)| > 1$ and when $|R(\mu)| = 1$, which can be found in [22].

3.4 Example

We conclude our discussion on the advantages of numerical smoothing over numerical stability with an illustrative example. Consider the initial value problem

$$
y'(t) = \lambda (\sin(\omega t) - y(t)) \\
y(0) = 1 \\
t \in [0, 10]
$$

(3.12)

where $\lambda > 0$, $\omega > 0$ are parameters. Let’s first solve the problem analytically by the method of integrating factors. We begin by writing the equation in standard form,

$$
y'(t) + \lambda y(t) = \lambda \sin(\omega t).
$$

(3.13)
Thus, our integrating factor is
\[ e^{\int \lambda dt} = e^\lambda. \] (3.14)

Now we solve:
\[ \frac{d}{dt} [e^\lambda y] = \lambda \sin(\omega t)e^\lambda. \] (3.15)

Integrating both sides of (3.15) yields
\[ e^\lambda y = \int \lambda \sin(\omega t)e^\lambda dt. \] (3.16)

We solve the integral on the right hand side of the above equation by parts. Let \( u = \sin(\omega t), \)
\( du = \omega \cos(\omega t)dt, \) \( v = e^\lambda, \) \( dv = \lambda e^\lambda dt, \) then
\[ \int \lambda \sin(\omega t)e^\lambda dt = e^\lambda \sin(\omega t) - \int \omega e^\lambda \cos(\omega t)dt. \] (3.17)

Integrating by parts one more time, let \( u' = \omega \cos(\omega t), \)
\( du' = -\omega^2 \sin(\omega t)dt, \) \( v' = \frac{e^\lambda}{\lambda}, \) \( dv' = e^\lambda dt. \) Then,
\[ \int \lambda \sin(\omega t)e^\lambda dt = e^\lambda \sin(\omega t) - \left[ \frac{\omega}{\lambda} e^\lambda \cos(\omega t) + \frac{\omega^2}{\lambda} \int e^\lambda \sin(\omega t)dt \right] \]
\[ = e^\lambda \sin(\omega t) - \frac{\omega}{\lambda} e^\lambda \cos(\omega t) - \frac{\omega^2}{\lambda} \int e^\lambda \sin(\omega t)dt. \] (3.18)

A few more steps of basic algebra leads us to the following:
\[ \int e^\lambda \sin(\omega t)dt = \frac{e^\lambda \left[ \sin(\omega t) - \frac{\omega}{\lambda} \cos(\omega t) \right]}{\lambda + \frac{\omega^2}{\lambda}} + c. \] (3.19)

Substituting (3.19) into (3.16) gives
\[ e^\lambda y = \lambda \int \sin(\omega t)e^\lambda dt \]
\[ = \lambda \left[ \frac{e^\lambda (\sin(\omega t) - \frac{\omega}{\lambda} \cos(\omega t))}{\lambda + \frac{\omega^2}{\lambda}} \right] + c. \] (3.20)
Finally, we solve (3.20) for $y$,

$$y(t) = \left[ \frac{\lambda \sin(\omega t) - \omega \cos(\omega t)}{\lambda + \frac{\omega^2}{\lambda}} \right] + \frac{c}{e^{\lambda t}}$$  \hspace{1cm} (3.21)$$

where the given initial condition may be used to uniquely solve for $c$. Since $y(0) = 1$, we have $c = 1 + \frac{\lambda \omega}{\lambda^2 + \omega^2}$. That is,

$$y(t) = \frac{\lambda^2 \sin(\omega t) - \omega \lambda \cos(\omega t)}{\lambda^2 + \omega^2} + \left[ 1 + \frac{\lambda \omega}{\lambda^2 + \omega^2} \right] e^{-\lambda t}$$

$$= \frac{\lambda}{\sqrt{\lambda^2 + \omega^2}} \left[ \frac{\lambda \sin(\omega t)}{\sqrt{\lambda^2 + \omega^2}} - \frac{\omega \cos(\omega t)}{\sqrt{\lambda^2 + \omega^2}} \right] + \left[ 1 + \frac{\lambda \omega}{\lambda^2 + \omega^2} \right] e^{-\lambda t} \hspace{1cm} (3.22)$$

where $\cos(\phi) = \frac{\lambda}{\sqrt{\lambda^2 + \omega^2}}$ and $\sin(\phi) = \frac{\omega}{\sqrt{\lambda^2 + \omega^2}}$. Let $s = \sqrt{\lambda^2 + \omega^2}$ and we can rewrite our solution in the following condensed form:

$$y(t) = c e^{-\lambda t} + \frac{\lambda}{s} \sin(\omega t - \phi)$$  \hspace{1cm} (3.23)$$

with $c = 1 + \frac{\lambda}{s} \sin(\phi)$ and $\phi = \sin^{-1}\left(\frac{\omega}{\sqrt{\lambda^2 + \omega^2}}\right)$.

Now we consider numerically solving (3.12) using the backward Euler method and the Crank-Nicolson method with $\lambda = 2000$ and $\omega = 1$ as parameter values. The Crank-Nicolson method, which is more commonly called the Trapezoidal Method within the context of ODEs, is a second order implicit Runge-Kutta method given by the following time-stepping scheme:

$$y_{n+1} = y_n + \frac{1}{2} h(f(t_n, y_n) + f(t_{n+1}, y_{n+1})).$$  \hspace{1cm} (3.24)$$

Also referred to as the modified Euler method, this A-stable, convergent scheme can be viewed as an average of the backward and forward Euler methods. More importantly, the Trapezoidal Method is considered to be “the most accurate among the A-stable linear multistep methods” according to [21] (a proof of convergence can be found in [15]).
Utilizing the MATLAB script provided in the Appendix, we produce the following plots comprising of both the analytical and numerical solutions over the interval $[0, 1]$ with a step size of 0.02.

Figure 3.3(a) depicts the true solution (3.23) in red for the given parameter values, the solution given by the backward Euler method in blue and the numerical solution arising from the Crank-Nicolson (or Trapezoidal) method in green. These results are quite surprising. While both numerical solutions are obviously bounded over the interval, the backward Euler method out-performs the Trapezoidal method in minimizing error propagation (see figures for error estimates).

Figure 3.3(a) alone best demonstrates the dangers in relying on numerical stability throughout the error analysis. We can easily see that, although the Crank-Nicolson method for ODEs is convergent and perfectly stable, its lack of smoothness elicits a large computational expense. The second plot, Figure 3.3(b) should help the reader visualize the smoothing behavior of the Trapezoidal method. The pink lines depict the local ODE solutions, which represent true solution curves for the ODE with the iteration values produced by the numerical scheme acting as initial conditions within each interval.

3.5 Numerical Smoothness for Error Estimation

The previous example has shown that it is necessary for a numerical scheme to have the numerical smoothing and/or numerical smoothness maintenance property. Otherwise even an absolutely stable scheme may have extremely poor convergence. In fact, what is more important than the necessity is the sufficiency of numerical smoothness in delivering error estimation. In Sun and Ewing [23], a framework is given for error estimation, which is based on the second kind of error splitting presented earlier. In the framework, local error is estimated according to the smoothness of the numerical solution, while global error is propagated according to the solutions of the differential equations. In this framework, one can achieve not only optimal convergence with the order of local error, but also optimal error propagation. Namely, the error estimates are always written in the form of

$$cP(f)h^k F(S)$$
where $c$ is a benign constant; $P(f)$ is the error propagation factor which only depends on the differential equation determined by $f$; $h^k$ is the designed optimal order of the scheme; $F(S)$ is always a computable bounded function of the smoothness indicator $S$. The error propagation is optimal since it only depends on the differential equation. Comparing to the traditional textbook error estimates, here numerical smoothness is presented by $F(S)$, which plays the role of the high order derivatives. The role and advantages of numerical smoothness in error analysis can be explained as below.

For local error, instead of an error indicator, we try to establish a certain smoothness measurement on the solution $\hat{y}(t)$ and use it to estimate the local error $\hat{y}(t_{n+1}) - y_{n+1}$. For error propagation, we try to find out how a differential equation propagates error. The smoothness of $\hat{y}(t)$ in $[t_n, t_{n+1}]$ depends on the smoothness of its initial value $y_n$, therefore this is indeed about numerical smoothness.

It is worthwhile to point out that the numerical smoothness approach is returning to the most basic idea of the numerical approximation of an initial value problem. That is, once $y_n$ has been computed, since $y(t_n)$ is unknown, the best thing to do in the computation of $y_{n+1}$ is to follow the solution $\hat{y}(t)$ of the initial value problem satisfying $\hat{y}(t_n^+) = y_n$. To estimate the error of the scheme in this local approximation, it is natural to use the smoothness of the “curve” given by $\hat{y}(t)$. However, for stiff ODE systems, the smoothness of the “curve” has to be studied. Of course, what is also to be studied is how to use the smoothness for the local error estimation. This has been a blind point in the literature. The direct reason for the existence of such a blind point for PDEs is that most spatial approximations of solutions are technically not smooth functions.

A deeper reason for the existence of the blind point is probably the dominance of the numerical stability concept in textbooks. However, the numerical stability concept in the literature has been a mixture of two concepts: error propagation and smoothness maintenance. In the sense of the von Neumann stability condition and the sufficiency part of the Lax Equivalence Theorem, numerical stability is about numerical error propagation. In the sense of some nonlinear stability theory and the necessity part of the Lax Equivalence Theorem, numerical stability is about the smoothness
maintenance of a numerical scheme as an operator. In the literature of parabolic problems, the concept of numerical dissipation has been used to cover the insufficiency of numerical stability in enforcing the smoothing effects of schemes. Due to the famous Lax Equivalence Theorem, numerical stability is known to be necessary and perhaps sufficient for convergence. Most of the research has been on that route. For more on the necessity of numerical smoothness see [4].

The alternative is simply to replace the concept of numerical stability by the concept of numerical smoothness. Here a numerical scheme is not responsible for error propagation at all. A scheme is responsible for two things. First, it needs to make the numerical solution \( y_n \) resemble the smoothness of \( y(t) \). Second, as in the Lax Equivalence Theorem, a numerical scheme needs to be consistent, so that the local error \( \tilde{y}(t_{n+1}) - y_{n+1} \) is small. If the smoothness of a numerical solution can be proven, one will be able to get a priori global error estimates. This is certainly possible for relatively simple problems. If the smoothness of a numerical solution cannot be proven, which is often the case for complex nonlinear problems, one can always compute a set of smoothness indicators out of \( y_n \), and perform a posteriori error analysis accordingly.

It is extremely important to notice that the computation of the smoothness indicators is the key in bypassing the well-known difficulty of nonlinearity. Smoothness maintenance of numerical computation usually comes from the inherited or artificial diffusion of a scheme, which is a global property because diffusion makes solutions smoother over time. For nonlinear problems, it is hard to do proofs on global properties. High performance schemes are oftentimes very complex. The complexity makes it even harder to do proofs on numerical smoothness. Even when a proof is possible, it may involve the applications of recursive inequalities and the Gronwall Lemma, which tends to over-estimate the smoothness parameters. The computation of smoothness indicators avoids all these overwhelming difficulties, because it delivers the currently available smoothness of \( y_n \) and \( \tilde{y}(t) \) in each time step.

The papers [22], [23], [24], and [25] show that, based on the numerical smoothness indicators, one can derive sharp local error estimates at each time step. Moreover, because a smoothness indicator is computed at the beginning of a time step and used for the local error estimation in the
time step, it naturally serves for adaptive error control. If a strategy for adaptive error control is based on an error indicator, it must be locally \textit{a posteriori}. If the error indicator determines that the error has been too large, the computation of the step may need to be redone. In contrary, a numerical smoothness indicator is computed at $t_n$, thus enabling a locally \textit{a priori} error control strategy in $[t_n, t_{n+1}]$, hence nothing will need to be redone.

Better error analysis, clearer criteria for scheme design, optimal error propagation, optimal local error, ability to handle complex equations, and natural adaptive refinement are obviously major advantages in using the concept of numerical smoothing. In future textbooks, the hope is that numerical smoothing will replace the concept of numerical stability.
Figure 3.3: Analytical vs. Numerical ODE Solutions for Stiff Equations
BIBLIOGRAPHY


APPENDIX A: MATLAB SCRIPTS

Crank Nicolson Method for ODEs (Trapezoidal Method)

%OUTPUT: x and y (coordinate pairs for plotting the approximation)
%INPUT: lambda = a real-valued parameter
% omega = a real-valued parameter
% xstart = where to begin the first iteration
% yinitial = given initial value of problem
% xfinal = where to stop the iterative method
% n = number of iterations to be completed between xstart and xfinal

function [x, y] = crank_nic(lambda, omega, xstart, yinitial, xfinal, n)

h = (xfinal - xstart)/n;

%Initialization of x and y as two arrays:

x = [xstart zeros(1, n)];
y = [yinitial zeros(1, n)];

for i = 1 : n

x(i + 1) = x(i) + h;
y(i + 1) = (y(i) * (1 - ((h * lambda)/2)) + (((h * lambda)/2) * (sin(omega * x(i)) + ...
    sin(omega * x(i + 1)))))/(1 + ((h * lambda)/2));
end

%True Solution:

% g = @(x)(1/sqrt(2)) * sin(x - asin(1/sqrt(2))) + (1 + (1/sqrt(2)) * (1/sqrt(2)) * exp(-x);
%Create coordinate pairs for the true solution:

xexact = [0 : 0.001 : 10];
yexact = g(xexact);

%Plot the approximation (in green) and the true solution (in red) together:

plot(x, y, 'g', xexact, yexact, 'r')
Backward Euler Method for ODEs

%OUTPUT: x and y (coordinate pairs for plotting the approximation)
%INPUTS: lambda = a real-valued parameter
% omega = a real-valued parameter
% xstart = where to begin the first iteration
% initial = given initial value of problem
% xfinal = where to stop the iterative method
% n = number of iterations to be completed between xstart and xfinal

function [x, y] = backward_euler(lambda, omega, xstart, yinitial, xfinal, n)

h = (xfinal - xstart)/n;

%Initialization of x and y as two arrays:

x = [xstart zeros(1,n)];
y = [yinitial zeros(1,n)];

for i = 1 : n
    x(i + 1) = x(i) + h;
    y(i + 1) = (y(i) + lambda * h * sin(omega * x(i + 1)))/(1 + lambda * h);
end

%True Solution:

g = @(x)(1/sqrt(2))*sin(x - asin(1/sqrt(2))) + (1 + (1/sqrt(2))*(1/sqrt(2)))*exp(-x);

%Create coordinate pairs for the true solution:
\( x_{exact} = [0 : 0.001 : 10]; \)

\( y_{exact} = g(x_{exact}); \)

\% Plot the approximation (in green) and the true solution (in red) together:

\( plot(x, y, 'g', x_{exact}, y_{exact}, 'r') \)

\( xlabel('x') \)

\( ylabel('y') \)

\( legend('Backward Euler', 'Analytical') \)

\( repline(0, 0) \)

\( axis([0 x_{final} -1 1]) \)

end

**Script file: Figure 3.3(a)**

\( T = 1.0; \)

\( n = 50; \)

\( lmd = 2000.0; \)

\( omg = 1.0; \)

\( s = sqrt(lmd * lmd + omg * omg); \)

\( f = @(x, y) lmd * (sin(omg * x) - y); \)

\% Calculate exact solution:

\( fe = asin(omg/s); \)

\( g = @(x)(1.0 + sin(fe) * lmd/s) * exp(-lmd * x) + sin(omg * x - fe) * lmd/s; \)

\( xe = [0 : 0.001 : T]; \)

\( ye = g(xe); \)

\% Call functions:

\[ x3, y3 = backward_euler(lmd, omg, 0, 1, T, n); \]

\[ x4, y4 = crank_nic(lmd, omg, 0, 1, T, n); \]

\% Plot:
plot(xe, ye,'r', x3, y3,'b', x4, y4,'g')
xlabel('x')
ylabel('y')
legend('Analytical','BackwardEuler','Crank – Nicolson')
axis([0 T -1.1 1.1])

% Estimate errors:
error3 = ['Backwarderror : ' num2str(-100 * (ye(end) - y3(end))/ye(end))
''];
error4 = ['Crank – Nicolsonerror : ' num2str(-100 * (ye(end) - y4(end))/ye(end))
''];
error = error3; error4; text(0.5, -0.83, error)

Script file: Figure 3.3(b)

T = 1.0;
n = 50;
h = T/n;
lmd = 2000.0;
omeg = 1.0;
s = sqrt(lmd * lmd + omg * omg);
f = @(x, y) lmd * (sin(omg * x) - y);

% Calculate exact solution:
fe = asin(omg/s);
g = @(x) (1.0 + sin(fe) * lmd/s) * exp(-lmd * x) + sin(omg * x - fe) * lmd/s;

gg = @(j, c, x) c * exp(-lmd * (x - j * h)) + sin(omg * x - fe) * lmd/s;
x = linspace(0, T, 1000);

ye = g(x);

% Call functions:
[x3, y3] = backward_euler(lmd, omg, 0, 1, T, n);
[x4, y4] = crank_nic(lmd, omg, 0, 1, T, n);
%Plot:

plot(xe, ye,'r', x3, y3,'b', x4, y4,'g')

xlabel('x')
ylabel('y')

axis([0 T -1.1 1.1])

%Estimate errors:

error3 = ['Backwarderror :' num2str(-100 * (ye(end) - y3(end))/ye(end))'%'];
error4 = ['Crank - Nicolsonerror :' num2str(-100 * (ye(end) - y4(end))/ye(end))'%'];
error = error3; error4;
text(0.5, -0.83, error)

hold on;

for i = 1 : 50
    xee = [i * h : h/20 : i * h + h];
    C = y4(i + 1) - sin(omg * i * h - fe) * lmd/s;
    yee = gg(i, C, xee);
    plot(xee, yee,'m')
end

legend('Analytical','BackwardEuler','Crank - Nicolson','LocalODEsolution')