INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 4, Number 2, Pages 212–242

VARIABLE STEP-SIZE SELECTION METHODS FOR IMPLICIT INTEGRATION SCHEMES FOR ODES

RAYMOND HOLSAPPLE, RAM IYER, AND DAVID DOMAN

Abstract. Implicit integration schemes for ODEs, such as Runge-Kutta and Runge-Kutta-Nyström methods, are widely used in mathematics and engineering to numerically solve ordinary differential equations. Every integration method requires one to choose a step-size, h, for the integration. If h is too large or too small the efficiency of an implicit scheme is relatively low. As every implicit integration scheme has a global error inherent to the scheme, we choose the total number of computations in order to achieve a prescribed global error as a measure of efficiency of the integration scheme. In this paper, we propose the idea of choosing h by minimizing an efficiency function for general Runge-Kutta and Runge-Kutta-Nyström integration routines. This efficiency function is the critical component in making these methods variable step-size methods. We also investigate solving the intermediate stage values of these routines using both Newton's method and Picard iteration. We then show the efficacy of this approach on some standard problems found in the literature, including a well-known stiff system.

Key Words. Runge-Kutta, implicit integration methods, variable step-size methods, solving stiff systems

1. Introduction

Recently, there has been interest in the literature concerning the use of geometric integration methods, which are numerical methods that preserve some geometric quantities. For example, the symplectic area of a Hamiltonian system is one such concern in recent literature [1, 2, 3, 4]. Tan [5] explores this concept using implicit Runge-Kutta integrators. Hamiltonian systems are of particular interest in applied mathematics, and in fact we test our variable step-size selection method on a well-known Hamiltonian system in Section 4.2. Furthermore, Hairer and Wanner [6, 7] showed that although implicit Runge-Kutta methods can be difficult to implement, they possess the strongest stability properties. These properties include A-stability and A-contractivity (algebraic stability). These are the main reasons we choose to investigate variable integration step-size selection using Runge-Kutta methods.

First order ordinary differential equations are solved numerically using many different integration routines. Among the most popular methods are Runge-Kutta methods, multistep methods and extrapolation methods. Hull, Enright, Fellen and Sedgwick [8] have written an excellent comparison of these types of methods. They test a number of Runge-Kutta methods against multistep methods based on Adams formulas and an extrapolation method due to Bulirsch and Stoer [9]. A goal of that paper was to compare these different types of methods as to how

Received by the editors February 20, 2006 and, in revised form, May 4, 2006.
2000 *Mathematics Subject Classification.* 65L05, 65L06, 34A09.

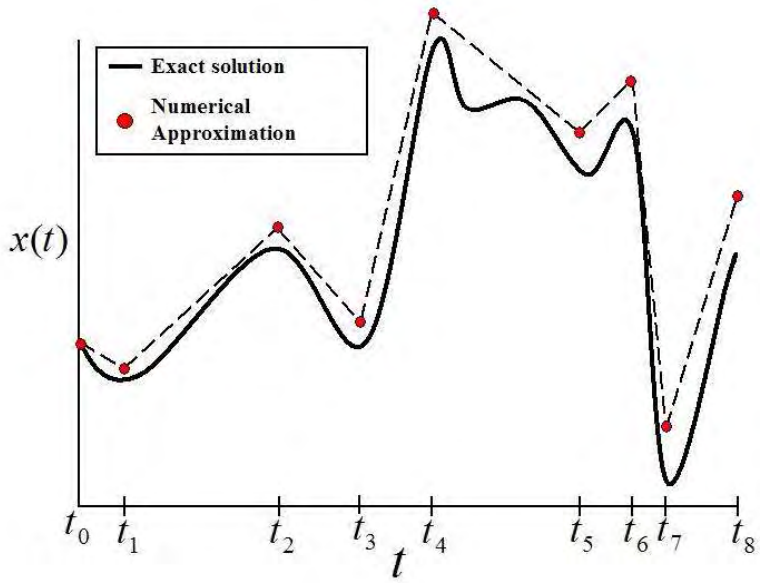

FIGURE 1.0.1. Illustration of variable step-sizes and error propagation in numerical integration

they handle routine integration steps under a variety of accuracy requirements. Implicit or explicit integration methods require one to choose a step-size, h, for the integration. One of the questions Bulirsch and Stoer investigate is a strategy for deciding what step-size h to use as the methods progress from one step to another. Others have investigated this very same problem in the past [8, 10, 11, 12].

In this paper, we propose the idea of choosing variable step-sizes by minimizing an efficiency function for general Runge-Kutta and Runge-Kutta-Nyström integration routines. As every implicit integration scheme has a global error inherent to the scheme, we choose the total number of computations in order to achieve a prescribed global error as a measure of efficiency of the integration scheme. For illustration purposes, consider Figure 1.0.1, referring to the solution of (2). Let $\tilde{x}(t_k)$ be our approximation to $x(t_k)$. We determine the variable step-sizes $h_1, h_2, \ldots, h_8$, where $h_k = t_k - t_{k-1}$, so that we minimize an efficiency function that minimizes the sum of the total number of computations to compute $\tilde{x}(t_k)$ for $k = 1, 2, \ldots, 8$ and the global error that propagates from the local truncation errors at each step of integration. To the best of our knowledge, our proposed method is novel.

The paper that most closely parallels the spirit of our optimization is that of Gustafsson and Söderlind [13]. They arrive at a function very similar to (31) using approximations while optimizing convergence rates, α, for a fixed-point iteration. They conclude that $\alpha_{\text{opt}} = e^{-1} = h_{\text{opt}}\overline{L}\|A\|$. They do not carry the argument further and include the global error in calculating the step-size, h, as we have done here.

One of the most important benefits of using a variable step-size numerical method is it's effectiveness at solving stiff initial value problems when combined with an implicit integration routine. Stiff systems are found in the description of atmospheric phenomena, chemical reactions occurring in living species, chemical kinetics (e.g. explosions), engineering control systems, electronic circuits, lasers, mechanics, and

molecular dynamics (Aiken [14]). A prototypical stiff system is the Van der Pol oscillator with the appropriate choice of parameters. We study this system using our method in Section 4.4.

In the rest of this section, we briefly describe variable step-size approaches found in the literature. Hull, Enright, Fellen and Sedgwick [8] approach this topic as follows. First, they determine $h_{\max}$, which is a measure of the "scale" of a problem. This helps to allow them from not stepping past any interesting fluctuations in the solution. Then, for their Runge-Kutta methods they compute τ, an estimate on the local truncation error, which must be bounded by the tolerance, ε. They then compute

$$h_{\text{new}} = \min\left\{h_{\max}, 0.9h_{\text{old}}\left(\varepsilon/\tau\right)^{1/p}\right\}. \tag{1}$$

where p is the order of the Runge-Kutta routine being used.

Stoer and Bulirsch [10] arrive at a very similar solution to this problem. To describe what they do, we first note that throughout this paper, we will consider solving the following first order ordinary differential equation:

$$\frac{dx}{dt} = f(t,x), \quad x(0) = x_0 \in \mathbb{R}^n, \tag{2}$$

where $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz continuous in the second argument, i.e. for any $t \in \mathbb{R}$ and any vectors $x \in \mathbb{R}^n, y \in \mathbb{R}^n$, we have

$$\|f(t,x) - f(t,y)\| \leq L(t)\|x - y\|, \tag{3}$$

where $L(\cdot) \in L^\infty[0,T]$. Here we mention that the the norms in (3) can be any $p-$norm $(1 \leq p \leq \infty)$. The same should be assumed for all norms in the remainder of the paper. When implementing the examples, the authors use the familiar 2-norm, $\|\cdot\|_2$. Stoer and Bulirsch consider two discretization methods, Φ_1 and Φ_2, of Runge-Kutta type to solve (2). The first method, Φ_1, is of order p and the second method, Φ_2, is of order $p+1$. In other words, they first compute

$$\bar{x}_{k+1} = \bar{x}_k + h_{\text{old}}\Phi_1(t_k, \bar{x}_k; h_{\text{old}}) \tag{4}$$

$$\hat{x}_{k+1} = \bar{x}_k + h_{\text{old}}\Phi_2(t_k, \bar{x}_k; h_{\text{old}}). \tag{5}$$

For a more detailed description of Φ_1 and Φ_2, please consult [10]. Then, denoting the tolerance by ε, and given a current step-size, h_{old}, they obtain:

$$h_{\text{new}} = h_{\text{old}}\left|\frac{\varepsilon}{\bar{x}_{k+1} - \hat{x}_{k+1}}\right|^{1/(p+1)}. \tag{6}$$

Stoer and Bulirsch go on to recommend after extensive numerical experimentation, that equation (6) be altered to

$$h_{\text{new}} = 0.9h_{\text{old}}\left|\frac{\varepsilon h_{\text{old}}}{\bar{x}_{k+1} - \hat{x}_{k+1}}\right|^{1/p}. \tag{7}$$

In Section 4.2, we compare the proposed variable step-size selection method with this method described by Stoer and Bulirsch.

Gustafsson [15, 16] uses an equation similar to (7) as a basis for designing a variable step-size method that uses control theoretical techniques to choose the step-sizes. Gustafsson even applies his methods to both explicit [15] and implicit [16] Runge-Kutta integration methods. Although Gustafsson explores step-size control for implicit integration routines, his methods are in no way similar to the method proposed in this paper.

As one can see, formulas (1),(6) and (7) depend on some measure of the local error at the $(k+1)$-st step of integration. Stoer and Bulirsch [10] also point out that there is another way to determine h_{new}, but it requires one to estimate higher order derivatives of f. For example, a fourth order Runge-Kutta method would require one to estimate derivatives of f of the fourth order. Not only is this very costly, but this other method uses the local truncation error at the k-th step of integration.

Houwen [11] took a similar approach to adjusting step-sizes. Again let ε be the tolerance. Houwen then forms a discrepance function $d(t_k, x_k; h_{\text{old}})$ at the point (t_k, x_k). Then the new step-size is determined to be the solution of the equation

$$\|d(t_k, x_k; h_{\text{old}})\| = \varepsilon. \tag{8}$$

Houwen considers three types of discrepance functions:

(1) an approximation to the local discretization error
(2) the residual term left when the local difference solution is submitted into the differential equation
(3) the discrepance of linearity of the differential equation

The first two clearly are functions that are some measure of local error of the difference scheme being used. The discrepance of linearity method is merely a way to choose h_{new} such that the Jacobian matrix for non-linear systems does not change very much,(i.e. within some tolerance ε), over the interval $[t_k, t_k + h_{\text{new}}]$. This method also deals with some measure of local stability of the differential equation.

Cano and Duran [12] investigate variable step-size selection using linear multistep methods. Again consider equation (2). Given a tolerance ε, they let

$$h_n = \varepsilon s(x(t_n), \varepsilon) + \mathcal{O}(\varepsilon^p), \tag{9}$$

where p is the order of the method and s is function satisfying the following:

(1) $s_{\min} \leq s(x, \varepsilon) \leq s_{\max}$, with $s_{\min}, s_{\max} > 0$,
(2) s is C^∞ in both arguments and all the derivatives of s are bounded.

Iserles [17] uses a Milne device as a variable step-size controller for embedded Runge-Kutta methods and for multi-step methods. We will explore this idea more thoroughly when we present a similar approach at controlling another parameter that arises in our analysis. We will investigate this idea in Section 3.5.

2. Implicit Integration Methods

Numerical methods for solving initial value problems such as (2) may be either explicit or implicit. The focus of this paper is concentrated on using implicit methods. In this section, we describe two classes of implicit numerical integration schemes and how one might use the methods to solve (2). We assume the solution exists for $t \in [0, T]$, with $T > 0$.

2.1. Runge-Kutta Methods. A detailed description of a general s-stage Runge-Kutta method for the solution of (2) can be found in many publications [1, 3, 18]. The general method for a fixed step-size, h, is described below:

$$y_{i_k} = x_k + h \sum_{j=1}^{s} a_{ij} f(t_k + c_j h, y_{j_k}), \quad i = 1, ..., s, \tag{10}$$

$$x_{k+1} = x_k + h \sum_{i=1}^{s} b_i f(t_k + c_i h, y_{i_k}), \quad x_0 = x(0). \tag{11}$$

In the above equations, the y_{i_k} are stage-values that must be computed at every step of the integration, and x_k approximates the exact solution $x(t_k)$ at the point $t_k = kh$, where h is the fixed step-size of integration. The a_{ij} and b_i are unique to any particular Runge-Kutta scheme and the c_i satisfy

$$c_i = \sum_{j=1}^{s} a_{ij} \ , \quad i = 1, \ldots, s \tag{12}$$

For notational purposes, define the following:

(13)

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1s} \\ a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{s1} & a_{s2} & \cdots & a_{ss} \end{bmatrix}, \ Y_k = \begin{bmatrix} y_{1_k} \\ y_{2_k} \\ \vdots \\ y_{s_k} \end{bmatrix}, \ c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_s \end{bmatrix}, \ X_k = \begin{bmatrix} x_k \\ x_k \\ \vdots \\ x_k \end{bmatrix}, \ \bar{A} = A \otimes I$$

where I is the $n \times n$ identity matrix and $\otimes$ is the Kronecker product. In other words, $\bar{A}$ is the $ns \times ns$ matrix direct product of A and I. Furthermore, consider the function $\tilde{f} : \mathbb{R} \times \mathbb{R}^{ns} \to \mathbb{R}^{ns}$ defined by

$$\tilde{f}(t_k, Y_k) = \begin{bmatrix} f(t_k + c_1 h, y_{1_k}) \\ f(t_k + c_2 h, y_{2_k}) \\ \vdots \\ f(t_k + c_s h, y_{s_k}) \end{bmatrix}. \tag{14}$$

Now we can write the system of ns equations given in equation (10) as

$$Y_k = X_k + h\bar{A}\tilde{f}(t_k, Y_k). \tag{15}$$

For each k this is an implicit equation involving the vectors $\{y_{i_k}\}_{i=1}^{s}$. Equation (15) can be solved using Newton's method or fixed point iteration (Picard iteration). Let's consider Picard iteration. We solve (15) for each k using the following iterative scheme:

$$Y_k^{j+1} = X_k + h\bar{A}\tilde{f}\left(t_k, Y_k^j\right) = F\left(t_k, Y_k^j\right). \tag{16}$$

For any fixed k, the iterative scheme given in (16) will converge to the solution of (15) provided that F satisfies a favorable condition. The following theorem addresses this convergence.

Theorem 2.1. *Consider the iterative scheme given by (16). Let $L(t)$ be the function from (3), and let A be the $s \times s$ matrix from (13). If $hL(t_k)\|A\| < 1$ then there exists a unique vector $\overline{Y} \in \mathbb{R}^{ns}$ such that $F(t_k, \overline{Y}) = \overline{Y}$ for any point $t_k \in [0, T]$ that is fixed. Furthermore, the sequence $Y_k^{j+1} = F(t_k, Y_k^j)$ converges linearly to $\overline{Y}$.*

Proof. The proof of this theorem involves showing that the Lipschitz constant of F is $hL(t_k)\|A\|$. This process is not difficult and is actually a well-known result; see Hairer, Nørsett, and Wanner [19]. □

Theorem 2.1 suggests how one might implement equations (11) and (16) to solve (2) on $[0, T]$. The starting vector $x_0 \in \mathbb{R}^n$ is known. In general, assume x_k is known. Use the following procedure to compute x_{k+1}.

(1) Choose a tolerance $\varepsilon > 0$ as small as you wish.
(2) Choose a starting guess for the s stage-values, and denote this guess as Y_k^0.
(3) For $j = 0, 1, 2, ...$, compute the following:
    (a) $Y_k^{j+1} = F\left(t_k, Y_k^j\right)$

(b) $\delta = \left\| Y_k^{j+1} - Y_k^j \right\|$.

(4) If $\delta \leq \varepsilon$, let $Y_k = Y_k^{j+1}$.

(5) Use the s $n \times 1$ stage-value vectors determined in step four to explicitly compute x_{k+1}.

The method described above is known as Picard iteration; Newton's method might also be used to solve for the stage-values. A theorem on the convergence of Newton's method is more complicated than Theorem 2.1; it is not sufficient to assume $hL(t_k)\|A\| < 1$ in order to guarantee that Newton's method converges. The Newton-Kantorovich Theorem [10, 20, 21] provides sufficient conditions for existence of a solution to the iteration and the uniqueness of that solution. To solve for the stage-values using Newton's method, step three should be replaced by the following:

(3) Define $G(t_k, Y_k) = F(t_k, Y_k) - Y_k$, and for $j = 0, 1, 2, ...$, compute the following:

(a) $Y_k^{j+1} = Y_k^j - \left(DG\left(t_k, Y_k^j\right)^{-1} \right) G\left(t_k, Y_k^j\right)$

(b) $\delta = \left\| G\left(t_k, Y_k^{j+1}\right) \right\|$

where DG represents the Jacobian matrix of G.

2.2. Runge-Kutta-Nyström Methods. Runge-Kutta-Nyström (RKN) methods are similar to Runge-Kutta methods, but are designed to solve second-order systems. Consider the following system:

$$\frac{d^2x}{dt^2} = f(t,x), \quad x(0) = x_0 \in \mathbb{R}^n, \quad \dot{x}(0) = v_0 \in \mathbb{R}^n, \tag{17}$$

which can be written as

$$\frac{dv}{dt} = f(t,x); \quad \frac{dx}{dt} = v, \quad x(0) = x_0, \quad v(0) = v_0. \tag{18}$$

We assume a solution exists on $[0,T]$ for $T > 0$. A general s-stage RKN method may be used to solve (18) on $[0,T]$, and is described by J.M. Sanz-Serna and M.P. Calvo [3] as follows:

$$y_{i_k} = x_k + h\gamma_i v_k + h^2 \sum_{j=1}^{s} a_{ij} f(t_k + \gamma_j h, y_{j_k}), \quad i = 1, ..., s \tag{19}$$

$$v_{k+1} = v_k + h \sum_{i=1}^{s} b_i f(t_k + \gamma_i h, y_{i_k}) \tag{20}$$

$$x_{k+1} = x_k + hv_k + h^2 \sum_{i=1}^{s} \beta_i f(t_k + \gamma_i h, y_{i_k}). \tag{21}$$

Exactly as with the Runge-Kutta methods, the y_{i_k} are the stage-values and must be solved for implicitly. In addition to the definitions in (13), define

$$\Gamma = \begin{bmatrix} \gamma_1 & 0 & \cdots & 0 \\ 0 & \gamma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \gamma_s \end{bmatrix}, V_k = \begin{bmatrix} v_k \\ v_k \\ \vdots \\ v_k \end{bmatrix}, \overline{\Gamma} = \Gamma \otimes I, \tag{22}$$

where I is the $n \times n$ identity matrix. Now, the ns equations in (19) may be written as

$$Y_k = X_k + h\overline{\Gamma}V_k + h^2\bar{A}\tilde{f}(t_k, Y_k). \tag{23}$$

Again, we have an implicit equation for the s stage-value vectors $\{y_{i_k}\}_{i=1}^{s} \subset \mathbb{R}^n$.

Just as in Section 2.1, we may solve (23) using Newton's method or by using Picard iteration. If we use Picard iteration, we have the following iterative scheme:

$$Y_k^{j+1} = X_k + h\overline{\Gamma}V_k + h^2\bar{A}\tilde{f}(t_k, Y_k^j) = H(t_k, Y_k^j). \tag{24}$$

Using a similar approach as in the proof of Theorem 2.1, one can easily prove the following theorem.

Theorem 2.2. *Consider the iterative scheme given by (24). Let $L(t)$ be the function from (3), and let A be the $s \times s$ matrix from (13). If $h^2 L(t_k)\|A\| < 1$ then there exists a unique vector $\overline{Y} \in \mathbb{R}^{ns}$ such that $H(t_k, \overline{Y}) = \overline{Y}$ for any point $t_k \in [0, T]$ that is fixed. Furthermore, the sequence $Y_k^{j+1} = H(t_k, Y_k^j)$ converges linearly to $\overline{Y}$.*

If we choose to use Newton's method to solve (23) for the stage-values, it is not sufficient to assume $h^2 L(t_k)\|A\| < 1$. Once again, we may refer to Stoer and Bulirsch [10] for conditions that guarantee convergence of Newton's method.

3. Step-Size Selection

When implementing a Runge-Kutta (or Runge-Kutta-Nyström) numerical integration routine, we have shown it is sufficient to assume that $hL(t_k)\|A\| < 1$ (or $h^2 L(t_k)\|A\| < 1$) to guarantee convergence of the implicit scheme when using a Picard iteration. A very similar result is also found in the literature. Shampine [14] determines that a sufficient condition for convergence of the function iteration used to determine the stage-values is $h\gamma L < 1$, where h is the step-size, L is the Lipschitz constant of f, and γ is a constant depending only on the Runge-Kutta formula. Shampine does not go on to describe γ in any grater detail, but for our derivation this γ is replaced by $\|A\|$. The same result (as Shampine) is also found in Gustafsson and Söderlind [13]. One might wonder though, is there an optimal choice, in the sense of computational efficiency, for h? If so, how might it be found?

3.1. Optimization Using Picard Iteration.

Consider solving (2) numerically on the interval $[0, T]$ which is partitioned by the following sequence of points: $\{kh\}_{k=0}^{K}$. In the k-th sub-interval the convergence of the Picard iteration is linear, so the number of computations required for convergence, to within ε, to the fixed point of (16) can be written as a function of the Lipschitz constant of the function F : $N_k = \phi\left(hL(t_k)\|A\|\right)$. Then the total number of computations over the interval $[0, T]$ can be written as $N(h) = \sum_{k=1}^{K} N_k$. In the following, we find an explicit expression for $\phi(\cdot)$ for Runge-Kutta methods.

Consider the following inequalities:

$$\left\|Y_k^{j+1} - Y_k^j\right\| = \left\|F(t_k, Y_k^j) - F(t_k, Y_k^{j-1})\right\| \tag{25}$$

$$\leq hL(t_k)\|A\| \left\|Y_k^j - Y_k^{j-1}\right\| \tag{26}$$

$$\leq (hL(t_k)\|A\|)^2 \left\|Y_k^{j-1} - Y_k^{j-2}\right\| \tag{27}$$

$$\vdots$$

$$\leq (hL(t_k)\|A\|)^j \left\|Y_k^1 - Y_k^0\right\| \tag{28}$$

$$= C_k \left(hL(t_k)\|A\|\right)^j, \tag{29}$$

where $C_k = \left\|Y_k^1 - Y_k^0\right\|$ is fixed for each k and depends on the guess Y_k^0. Since $hL(t_k)\|A\| < 1$, we must have $\left\|Y_k^{j+1} - Y_k^j\right\| \to 0$ as $j \to \infty$. Suppose we want $\delta = \left\|Y_k^{j+1} - Y_k^j\right\| \leq \varepsilon$; then, it is sufficient to have $C_k \left(hL(t_k)\|A\|\right)^j \leq \varepsilon$. As a stopping criterion in the k-th step of integration, we choose to stop the fixed point iteration at the smallest natural number N_k greater than or equal to M where M satisfies $C_k \left(hL(t_k)\|A\|\right)^M = \varepsilon$. Then we have

$$M = \frac{\ln\left(\varepsilon/C_k\right)}{\ln\left(hL(t_k)\|A\|\right)} \qquad \text{and} \qquad N_k = \lceil M \rceil. \tag{30}$$

Now let $C = \max_k C_k$ and $\overline{L} = \sup_{t\in[0,T]} L(t)$. In (30), ε and C_k depend on the user. Once these are chosen, the choice of h depends on the differential equation being solved, through the Lipschitz constant $L(t_k)$, and on the integration method being implemented, through $\|A\|$. We will try to minimize M by adjusting the choice of h to the problem parameters $L(t_k)$ and $\|A\|$. Notice that $C(h\overline{L}\|A\|)^M = \varepsilon$ implies that $C_k(hL(t_k)\|A\|)^M \leq \varepsilon$ for each k. Thus, we minimize the cost function

$$J_1(h) = K\frac{\ln\left(\varepsilon/C\right)}{\ln\left(h\overline{L}\|A\|\right)} = \frac{T\ln\left(\varepsilon/C\right)}{h\ln\left(h\overline{L}\|A\|\right)}, \tag{31}$$

which is equivalent to maximizing

$$J_2(h) = \frac{h\ln\left(h\overline{L}\|A\|\right)}{T\ln\left(\varepsilon/C\right)}. \tag{32}$$

By computing $\arg\min J_2(h)$, one finds the step-size h that minimizes the number of computations for the iterative scheme to converge. If this were the only measure of optimality of concern, it is easily shown, through a calculus argument, that the cost function $J_2(h)$ is maximized when $h = \left(e\overline{L}\|A\|\right)^{-1}$.

However, one might also want the global error of the numerical solution to be as small as possible. Global error in any numerical integration scheme depends on the scheme being used. In this paper, we are concentrating on Runge-Kutta schemes. The global error for Runge-Kutta schemes also varies depending on the scheme one chooses to implement. For the purpose of explanation, let us consider the implicit midpoint rule. We will also use the implicit midpoint rule on most of the examples. At the conclusion of this subsection, we describe how one would implement this optimization technique for an arbitrary Runge-Kutta method. The implicit midpoint rule is a one-stage Runge-Kutta method where $a_{11} = \frac{1}{2}$ and $b_1 = 1$ in (10) and (11). The implicit midpoint method has global error $\mathcal{O}(Th^2)$.

Then to find h, we alter the cost function J_2 and maximize the following:

$$J_3(h) = \frac{h \ln(h\overline{L}\|A\|)}{T \ln(\varepsilon/C)} - \kappa T h^2, \tag{33}$$

where κ is a number to be chosen. The solution is determined by solving

$$\ln(h\overline{L}\|A\|) + 1 + \lambda^2 h\overline{L}\|A\| = 0 \tag{34}$$

for h given an arbitrary value for λ. For details see Appendix 6.2.

In practice, we actually make the substitution $x = h\overline{L}\|A\|$ and solve

$$\ln x + 1 + \lambda^2 x = 0 \tag{35}$$

for x and then compute $h = \dfrac{x}{\overline{L}\|A\|}$. The solution to this equation exists and is unique. This is because the h that solves (34) is the unique global maximum of the function J_3, which exists because of the concavity of J_3. Furthermore, (34) must be solved numerically for $\lambda \neq 0$; for example, Newton's method or Picard iteration may be used. For $\lambda = 0$, notice that the solution is $h = \left(e\overline{L}\|A\|\right)^{-1}$ which was discussed earlier.

If one is interested in finding an equation similar to (34) for a Runge-Kutta method other than the implicit midpoint method, two things change in (33). First, $\|A\|$ will certainly change when the method changes. It will be necessary to change the second term as well. Suppose the method chosen has global error $\mathcal{O}(Th^r)$. Then, (33) becomes

$$\tilde{J}_3(h) = \frac{h \ln(h\overline{L}\|A\|)}{T \ln(\varepsilon/C)} - \kappa T h^r. \tag{36}$$

If we define

$$\kappa = -\lambda^2 \frac{(\overline{L}\|A\|)^{r-1}}{2T^2 \ln(\varepsilon/C)}\,, \tag{37}$$

we discover that we must now solve

$$\ln x + 1 + \lambda^2 x^{r-1} = 0 \tag{38}$$

for x after again making the substitution $x = h\overline{L}\|A\|$. This is the only difference in implementation of the proposed method when one is not using the implicit midpoint rule.

3.2. Local Optimization Using Picard Iteration. If one considers the analysis presented in the previous section, an obvious question might arise. Is it possible to repeat the process while minimizing local error and local computations instead of global error and total computations? It turns out that the answer is no.

Instead of minimizing (31), i.e. maximizing (32), if we want to minimize the total number of computations locally we minimize the cost function

$$\hat{J}_1(h) = \frac{\ln(\varepsilon/C)}{\ln\left(h\overline{L}\|A\|\right)}\,. \tag{39}$$

Equivalently, we can maximize

$$\hat{J}_2(h) = \frac{\ln\left(h\overline{L}\|A\|\right)}{\ln(\varepsilon/C)}\,. \tag{40}$$

We also want to consider the local truncation error. Just as in the previous section, we choose the implicit midpoint method for explanation purposes. The local truncation error for this method is $\mathcal{O}(h^3)$. Thus, similar to (33), we want to maximize the cost function

$$\hat{J}_3(h) = \frac{\ln\left(h\overline{L}\|A\|\right)}{\ln(\varepsilon/C)} - \kappa h^3 \,. \tag{41}$$

First, we compute

$$\frac{d\hat{J}_3}{dh} = \frac{1}{h\ln(\varepsilon/C)} - 3\kappa h^2 \tag{42}$$

and set it equal to zero. After doing so and rearranging, we get

$$h^3 = \frac{1}{3\kappa\ln(\varepsilon/C)}. \tag{43}$$

Since $\varepsilon \ll C$ in general, (e.g. in our implementations we choose $\varepsilon = 10^{-10}$), it will certainly be the case that $\ln(\varepsilon/C) < 0$. Thus, (43) would imply that it is necessary to have $\kappa < 0$, since it must be that $h > 0$.

Next, we compute

$$\frac{d^2\hat{J}_3}{dh^2} = \frac{-1}{h^2\ln(\varepsilon/C)} - 6\kappa h. \tag{44}$$

Since we are trying to maximize (41), we need $\dfrac{d^2\hat{J}_3}{dh^2} < 0$. However, if we consider each term in (44), we can easily see that this is impossible. Since $\ln(\varepsilon/C) < 0$, it must be that $\dfrac{-1}{h^2\ln(\varepsilon/C)} > 0$. Since we have determined that we must have $\kappa < 0$, it must also be that $-6\kappa h > 0$. Hence, the second derivative would imply that the function we are optimizing is actually concave up and has no local (or absolute) maximum for $h > 0$.

The analysis above shows that this method of choosing variable step-sizes (minimization of a particular efficiency function) fails if one considers local truncation error and local computations for the efficiency function. Although, there certainly are ways of choosing h based on local truncation errors as we described in the Introduction, we simply cannot use local error and computations when using the methods presented in this paper.

3.3. Non-linear IVPs - Lipschitz Constant Unknown. For most initial value problems the Lipschitz constant of f is not easily accessible. In this case, an approach that is slightly different than that of Section 3.1 is taken to solve the problem using this optimization technique. The idea in this case is to linearize the function f at each step of integration by computing the Jacobian of f. We essentially find an optimal h at each step of the integration using the analysis from Section 3.1. The method is described in detail below:

(1) Choose a value for the parameter λ. (A method for choosing λ will be given in Section 3.5.)
(2) Solve equation (35) for x once.
(3) At $t = 0$, let $\overline{L} = \|D_x f\|$, where $D_x f$ is the Jacobian matrix of f in x, and compute $h = \dfrac{x}{\overline{L}\|A\|}$.

(4) Perform one step of integration using the implicit midpoint rule. (Here you may be using another method for integration. If this is the case, you must solve the appropriate equation in step (2) above, as described in Section 3.1.)
(5) Recompute $\overline{L}$ using the new values of the state variables, and use this $\overline{L}$ to find a new optimal h.
(6) Repeat steps four and five until the integration reaches $t = T$.

At this point we must make note of an important point. If the initial value problem being solved happens to be a linear system (stiff or non-stiff), then this arbitrary choice of λ is no different then choosing h arbitrarily. For example, consider the following stiff system (Sewell [22]):

$$\dot{u} = -1000u + \sin(t) \ , \quad u(0) = \frac{-1}{1000001} \ , \tag{45}$$

which has the smooth solution

$$u(t) = \frac{1000\sin(t) - \cos(t)}{1000001}. \tag{46}$$

For this system, choosing λ and then solving (35) to get an optimal value of h is no different than just choosing h to begin with, because $\|D_x f\| = 1000$ is constant. Hence, one is effectively choosing a fixed step-size by choosing λ. For this reason, the proposed variable step-size method is of greatest benefit when solving non-linear systems. In particular, it is useful in solving stiff non-linear IVPs, as shown in Section 4.4.

3.4. Optimization Using Newton's Method. We mentioned in Section 2.1 that one might also use Newton's method to solve for the stage values Y_k. Because of this, the analysis for finding an optimal value of h can be repeated for Newton's method. Convergence properties, and hence convergence theorems, are more complicated for Newton's method than for fixed point iteration. Because of this, one might expect the analysis to be a bit more involved than that of equations (25)-(90).

Before we begin looking at the optimization process for Newton's method, let us first consider the following Lemma.

Lemma 3.1. *If R is an invertible $n \times n$ matrix, then*

$$\left\|R^{-1}\right\| \leq \frac{\|R\|^{n-1}}{|\det R|}. \tag{47}$$

Proof. For proof see Appendix 6.1. □

Again, we consider solving (2) numerically on the interval $[0, T]$ which is partitioned by the following sequence of points: $\{kh\}_{k=0}^{K}$. Stoer and Bulirsch [10] prove a theorem showing that Newton's method is quadratically convergent. We will find an optimal choice of h using the results of that theorem. We begin by defining

$$\rho_k = \frac{\alpha_k \beta_k \gamma}{2} \ , \tag{48}$$

where α_k, β_k and γ are described below. First we let $\overline{C}$ be a convex subset of $\mathbb{R}^{ns}$ and mention that the theorem from [10] and the analysis below is valid only in a

neighborhood, $S_r\left(Y_k^0\right)$, of the initial guess Y_k^0 such that $S_r\left(Y_k^0\right) \subset \overline{C}$. We then let

$$\alpha_k = \left\|Y_k^1 - Y_k^0\right\| , \quad k = 1, \ldots, K \tag{49}$$

$$\beta_k = \sup_{Y \in \overline{C}} \frac{\|DG\left(t_{k-1}, Y\right)\|^{ns-1}}{|\det DG\left(t_{k-1}, Y\right)|} , \quad k = 1, \ldots, K \tag{50}$$

$$\gamma = h\overline{L}\|A\|, \tag{51}$$

where $\overline{L}$ is a constant that satisfies

$$\left\|D\tilde{f}(t,u) - D\tilde{f}(t,v)\right\| \leq \overline{L}\|u - v\|, \tag{52}$$

for any vectors u and v in $\mathbb{R}^{ns}$. Note that this analysis holds whether the Lipschitz constant of $D\tilde{f}$, $\overline{L}$, is given from the beginning or if it is approximated similarly to what was done in the previous section; we shall only require that $\overline{L}$ be a known constant at the k-th step of integration.

It should also be noted that in practice, i.e. in the example we show later, α_k and β_k will not depend on k. For implementation purposes, we elect to choose Y_k^0 to be the same vector for all k. Hence, α_k will be the same for each k. In general, this is not necessary; that is why we define α below for convenience of analysis. Actually, β_k must satisfy $\left\|DG\left(t_{k-1}, Y\right)^{-1}\right\| \leq \beta_k$ for all $Y \in \overline{C}$. We then apply Lemma 3.1 to arrive at the definition given by equation (50). Now, β_k depends on the step-size through the Jacobian matrix of G, i.e. through DG. Since this is a variable step-size method, this implies that β_k should be computed at each step of the integration using the current step-size. We quickly found that computing β_k at each step of the integration makes the process computationally inefficient. Instead, we approximate $\beta = \max_k \beta_k$ before solving a particular example by first solving the system on a much smaller time interval. As we solve the problem, we keep track of the current values of β_k and keep the largest value to use as a global constant to solve the entire example. Putting all this together and using the theorem from Stoer and Bulirsch, we know that for k fixed, the iterations of Newton's method must satisfy

$$\left\|Y_k^{j+1} - Y_k^j\right\| \leq \alpha_k \rho_k^{2^j - 1} \leq \alpha \rho_k^{2^j - 1}, \tag{53}$$

where $\alpha = \max_k \alpha_k$.

Furthermore, we must assume (as do Stoer and Bulirsch) that $\rho_k < 1$. Then for each k, it is sufficient to stop iterating Newton's method at the smallest natural number N_k greater than or equal to M, where M satisfies $\alpha \rho_k^{2^M - 1} = \varepsilon$. Solving for M, we get

$$M = \log_2 \left(\frac{\ln(\rho_k \varepsilon \alpha^{-1})}{\ln \rho} \right). \tag{54}$$

Again, we choose $N_k = \lceil M \rceil$. Also, we are showing the analysis for the implicit midpoint method which has global error $\mathcal{O}(Th^2)$. Thus, we minimize the following cost function:

$$J_4(\rho) = \frac{T\alpha\beta\overline{L}\|A\|}{2\rho_k} \log_2 \left(\frac{\ln(\rho_k \varepsilon \alpha^{-1})}{\ln \rho} \right) + \frac{4\kappa T \rho_k^2}{(\alpha\beta\overline{L}\|A\|)^2} . \tag{55}$$

We now define

$$\kappa = \lambda^2 \frac{(\alpha\beta\overline{L}\|A\|)^3}{16 \ln 2}. \tag{56}$$

Then, after using a little calculus similar to equations (33)-(88), we find the optimal h by first finding the ρ_k that solves

$$\left(\ln(\rho_k \varepsilon \alpha^{-1})\right)^{-1} - (\ln \rho_k)^{-1} - (\ln 2) \log_2 \left(\frac{\ln(\rho_k \varepsilon \alpha^{-1})}{\ln \rho_k} \right) + \lambda^2 \rho_k^3 = 0, \tag{57}$$

and then computing

$$h_{k+1} = \frac{2\rho_k}{\alpha \beta \bar{L} \|A\|}. \tag{58}$$

Because of the complexity of (57), we must solve for ρ_k numerically.

3.5. Milne Device for Choosing λ. Inherent with this variable step-size selection method is the choice of the parameter λ. We offer two approaches for choosing this λ. First, we consider a naive approach. Suppose we are interested in integrating a non-linear system over the time interval $[0, T]$, where T is large, e.g. $T = 1000$. First, we choose a much smaller value for the final time of integration; in this example, let that value be $T^* = 50$. We then integrate the system over the interval $[0, T^*]$ with a fixed step-size and at the same time with various values of λ. Essentially, we analyze how λ affects the solution of this system. We compare this with the solution obtained by the fixed step-size method and choose the smallest λ such that the variable and fixed step-size solutions are similar. This process should be done for any system where the length of the interval over which the integration must be performed is quite large when compared to the evolution of the dynamics of the system.

We could also use what is called a Milne device. Iserles [17] describes the device in detail and uses it to implement a variable step-size method, where the device is used to either double or halve the step-sizes. We suggest a similar technique where the goal is to either double or halve the value of λ.

The idea behind the device is to control the local error per unit step. We use the concept of embedded Runge-Kutta methods. One chooses two Runge-Kutta routines. The first is of order p and is used as the integrator. The second is an explicit Runge-Kutta method also of order p and is used to control the local error of the integrator. We assume that the numerical solution is 'exact' at time t_k. The goal is to control the error as the integrator goes to step t_{k+1}. Let x_{k+1} be the result of the integrator, and let $\bar{x}_{k+1}$ be the result of the local error controller. We have

$$x_{k+1} = x(t_{k+1}) + m h_k^{p+1} + \mathcal{O}(h_k^{p+2}), \tag{59}$$

$$\bar{x}_{k+1} = x(t_{k+1}) + n h_k^{p+1} + \mathcal{O}(h_k^{p+2}), \tag{60}$$

where $x(t_{k+1})$ is the exact solution at time t_{k+1}, and m and n are vectors that depend on the differential equation, but not on h. Subtracting (60) from (59) and ignoring the $\mathcal{O}(h_k^{p+2})$ terms, we get

$$x_{k+1} - \bar{x}_{k+1} \approx (m - n) h_k^{p+1}. \tag{61}$$

We then define the error term κ using (61) as follows:

$$\kappa = \|x_{k+1} - \bar{x}_{k+1}\|. \tag{62}$$

The user will determine a tolerance δ prior to beginning. Then to control the error per unit step, our goal is to enforce the constraint

$$\kappa \leq h_k \delta. \tag{63}$$

One begins by choosing a value for λ. Associated with this value of λ will be the step-size h_k, which is used in the integration. One then computes κ using (62) and determines if (63) is satisfied. If it is not, then one must double λ and reintegrate without advancing the time of integration. If (63) is satisfied then we may advance the integration and repeat the process. Before we advance the integration, we may want to do one further thing. Suppose that it was actually the case that $\kappa \leq \frac{1}{10} h_k \delta$. In this case, we choose to first halve the value of λ, as λ was too conservative, and then we advance the the integration and repeat the process. The entire process is illustrated in Figure 3.5.2.

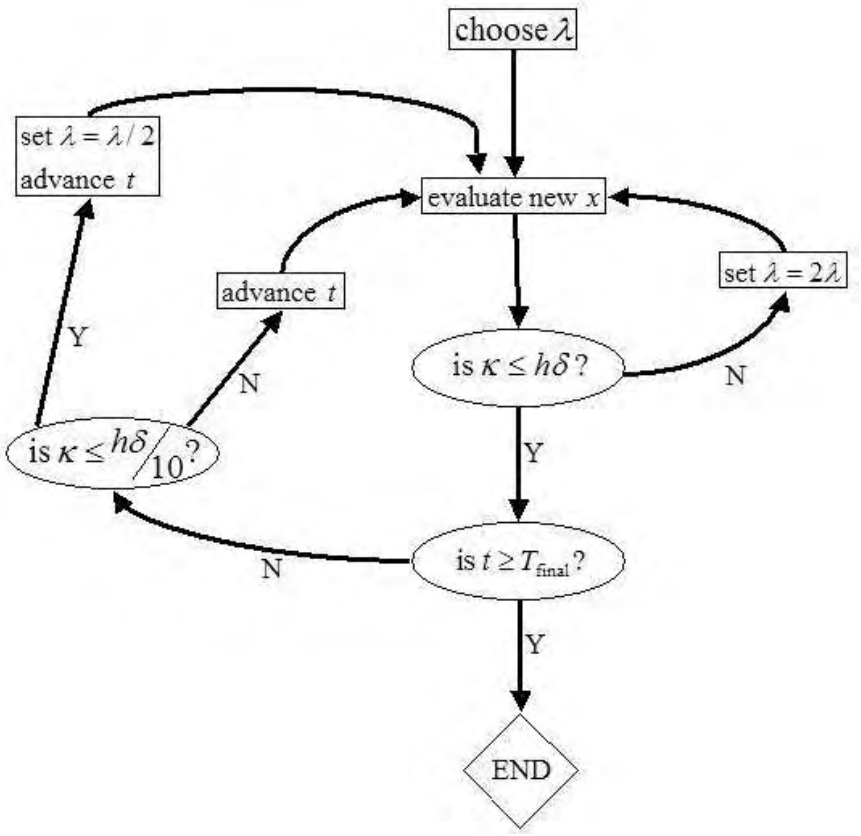

FIGURE 3.5.2. Flowchart for implementing the Milne device for choosing λ

4. Examples

In this section we explore the proposed variable step-size selection method for three problems: the Lotka-Volterra model, the Kepler problem, and the Van der Pol oscillator. It is pivotal to make a note here concerning the Lipschitz continuity of the examples in this section. We explore the proposed method on the nonlinear examples previously mentioned, which do not possess a globally Lipschitz function f. Although Lipschitz continuity was a major assumption in the derivation of the method itself, we have also described how we choose to implement the method on non-linear systems where global Lipschitz continuity is not present.

4.1. The Lotka-Volterra Model. Consider the Lotka-Volterra model of a simple predator-prey system from mathematical biology. This particular example is taken from Hairer, Lubich, and Wanner [1]. Consider the following system:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} u(v-2) \\ v(1-u) \end{bmatrix} = f(u,v); \quad t \in [0,50]. \tag{64}$$

In (64), $u(t)$ denotes the number of predators present at time t, $v(t)$ represents the number of prey present at time t, and the constants one and two have been chosen arbitrarily. This system was integrated numerically using the implicit midpoint rule. Since the system is non-linear, f has no global Lipschitz constant; however, the method described in Section 3.3 does not require one. Only local Lipschitz constants are needed for its implementation.

This procedure was compared to a fixed step-size integration method with random step-sizes chosen. Two measures were chosen for comparison. The first measure, T, was total cpu time (in seconds) for 1000 runs with random initial data uniformly distributed on $[0.1, 10]$. The second measure, E, was the maximum absolute variation of the numerical method from

$$I(u,v) = \ln u - u + 2\ln v - v, \tag{65}$$

an invariant quantity for this system. The initial data for the system in this case was chosen to be $[u(0) \;\; v(0)]^T = [2 \;\; 6]^T$.

We found that for simple systems such as (64), the numerical computational overhead in computing the step-size in the optimal h method renders the method less useful than a simple fixed step-size method. After trying various fixed step-sizes, it was determined that for 1000 runs with random initial data, $h = 0.125$ was the largest fixed step-size attempted that permitted convergence. For $h = 0.125$, $T = 118.3$ and $E = 0.064$. For the optimal h method, various values for λ were tried until a comparable value for E was found. For instance, for $\lambda = 2$ we get $E = 0.143$; for $\lambda = 3$ we get $E = 0.068$; and for $\lambda = 4$ we get $E = 0.037$. Since $\lambda = 3$ yielded a comparable value of E, $\lambda = 3$ was chosen for 1000 runs with random initial data and it was found that $T = 195.6$.

Different results arise when we try more challenging problems. Consider this variation to the Lotka-Volterra problem:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} u^2 v(v-2) \\ v^2 u(1-u) \end{bmatrix} = f(u,v); \quad t \in [0,50]. \tag{66}$$

This system has has the same invariant as (64), but is very sensitive to random initial data. For this reason the initial data is fixed at $[u(0) \;\; v(0)]^T = [2 \;\; 3]^T$ for the computation of both T and E.

Two methods were chosen to solve for the stage value y_1 which is defined implicitly by (10). The first method is the algorithm of Section 2, which is simply a Picard iteration. Secondly, we used Newton's method to compute the stage value y_1. The results from this example are given in Tables 4.1.1 through 4.1.4 and Figures 4.1.3 and 4.1.4.

To compare the fixed step-size method to the variable step-size method, we must locate times that are comparable from the tables and then compare the equivalent error. For example, we first notice that for the fixed step-size $h = 0.1$ in Table 4.1.1, the method took 160.7 seconds to solve the problem using a Picard iteration to solve for y_1. The error involved for this step-size was 0.094. Now we look in Table 4.1.2 and find that when $\lambda = 2$, the problem was solved in 168.4 seconds, which is about eight seconds longer than for $h = 0.1$. However, we notice that the

error has been reduced to 0.004, which is about 4% of the error from when $h = 0.1$. We can locate other instances similar to this from the two tables. For the fixed step-size $h = 0.08$, the problem is solved in 234.4 seconds using Newton's method to find y_1, yielding an error of 0.069. We compare this to $\lambda = 2$ which was solved in 229.9 seconds with an error of 0.004. In addition, for the fixed step-size $h = 0.125$ using Newton's method to solve for y_1, the problem is solved in 155.6 seconds with an error of 0.084. We compare this to $\lambda = 1$ in which the problem is solved in 151.6 seconds with an error of 0.025.

In Table 4.1.2, when we computed the stage value y_1 using Newton's method, we actually used the variable step-sizes determined by the solution of (35) not (57). For comparison, we recreated these table on another machine; this time we use (57) to compute the variable step-sizes when we are solving for the stage value y_1 using Newton's method. The results are given in Table 4.1.3 and Table 4.1.4. There are a couple of interesting things to point out. First, we notice that for fixed step-sizes, Newton's method works faster than the fixed-point iteration. We must note that the MATLAB® code that generated the data for Tables 4.1.1 through 4.1.4 was exactly the same; the only difference is that Tables 4.1.3 and 4.1.4 contain data from a different version of MATLAB® and ran on a different machine. The main goal of this paper is not to compare Newton's method versus Picard iteration as much as it is to compare fixed step-sizes to variable step-sizes. In this regard, the results shown in Tables 4.1.3 and 4.1.4 agree with those given in Tables 4.1.1 and 4.1.2. For example, when we use the fixed step-size $h = 0.05$, the problem is solved in 67.7 seconds with an error of 0.031 when solving for the stage value using a Picard iteration. Using the variable step-size method with $\lambda_P = 3.6$, we see that the problem is solved in 67.4 seconds with an error of 0.003, which is about 90% smaller than the fixed-step size error of 0.031. A similar comparison can be made with $h = 0.125$ and $\lambda_P = 2$.

The second interesting thing we notice is evidenced by Table 4.1.4 when we use (57) to compute the variable step-sizes and solve for the stage value y_1 using Newton's method. We notice that the problem takes longer to solve, but at the same time the error is much smaller. We cannot compare the fixed step-size data with the variable step-size data as above because of the large difference in times. However we can note that when the problem is solved with $\lambda_P = 0.4$ the problem is solved in 27.1 seconds with an error of 0.087; when the problem is solved with $\lambda_N = 10$, it takes 115.7 seconds to get the error down to 0.00113. We can quickly compute that the problem is solved 77% faster using λ_P, but the error is 98% lower using λ_N.

The reason it takes so long to solve the problem using this method is because of the variable step-sizes determined by solving (57). Consider the plot shown in Figure 4.1.5. This graph shows how far the integration of the problem has progressed at each iteration for various values of λ_P and λ_N. Essentially, the graph also shows how large (or how small) of a step-size each method chooses as the integration proceeds from one step to the next. In addition, we can see from the plot that when we use (57) to find the step-size update, the step-sizes determined are much smaller then when (35) is used; hence, many more iterations are required to integrate the system from $t = 0$ to $t = 50$. We would also like to mention something about the existence of a solution to (57). Although, we have not proven that a solution does exist, we can plot the function for reasonable values of the parameters. For example, consider Figure 4.1.6. Here we plot the function $g(\rho)$ where g is the function given by the left hand side of (57). This plot uses the following values

for the parameters given in (57): $\lambda = 10$, $\varepsilon = 10^{-10}$ and $\alpha = \left\|[2\ 3]^T\right\|$. The function appears to be quite smooth in this region and clearly we see that for these parameters, a solution to (57) exists.

All computations, except those that generated Tables 4.1.3 and 4.1.4, were done in MATLAB® version 6.1.0.450 Release 12.1 running in Microsoft Windows XP Professional version 5.1.2600 with an AuthenticAMD processor running at 1544 Mhz.

TABLE 4.1.1. Fixed step-size (Lotka-Volterra model)

$h \rightarrow$	0.05	0.08	0.1	0.125
T (Picard)	240.0	181.1	160.7	159.6
$E \times 10^{-2}$ (Picard)	3.1	6.9	9.4	8.4
T (Newton)	354.4	234.4	187.7	155.6
$E \times 10^{-2}$ (Newton)	3.1	6.9	9.4	8.4

TABLE 4.1.2. Variable step-size (Lotka-Volterra model)

$\lambda \rightarrow$	0.25	0.4	0.5	0.75	1	2
T (Picard)	113.8	119.7	118.5	124.8	127.3	168.4
$E \times 10^{-2}$ (Picard)	11.5	8.7	7.8	4.8	2.5	0.4
T (Newton)	117.6	124.3	127.4	140.7	151.6	229.9
$E \times 10^{-2}$ (Newton)	11.5	8.7	7.8	4.8	2.5	0.4

TABLE 4.1.3. Fixed step-size (Lotka-Volterra model)

$h \rightarrow$	0.05	0.065	0.08	0.1	0.125
T (Picard)	67.7	57.0	50.7	46.2	45.6
$E \times 10^{-2}$ (Picard)	3.1	5.0	6.9	9.4	8.4
T (Newton)	67.8	53.3	44.0	35.6	29.6
$E \times 10^{-2}$ (Newton)	3.1	5.0	6.9	9.4	8.4

TABLE 4.1.4. Variable step-size (Lotka-Volterra model)

$\lambda_P \rightarrow$	0.4	0.6	0.8	1	2	3.6
T (Picard)	27.1	28.4	28.8	33.9	43.6	67.4
$E \times 10^{-2}$ (Picard)	8.7	6.6	4.4	2.5	0.4	0.3
$\lambda_N \rightarrow$	8	10	12	14	16	20
T (Newton)	192.3	115.7	138.1	134.1	140.5	149.9
$E \times 10^{-4}$ (Newton)	4.6	11.3	8.4	8.9	8.2	7.3

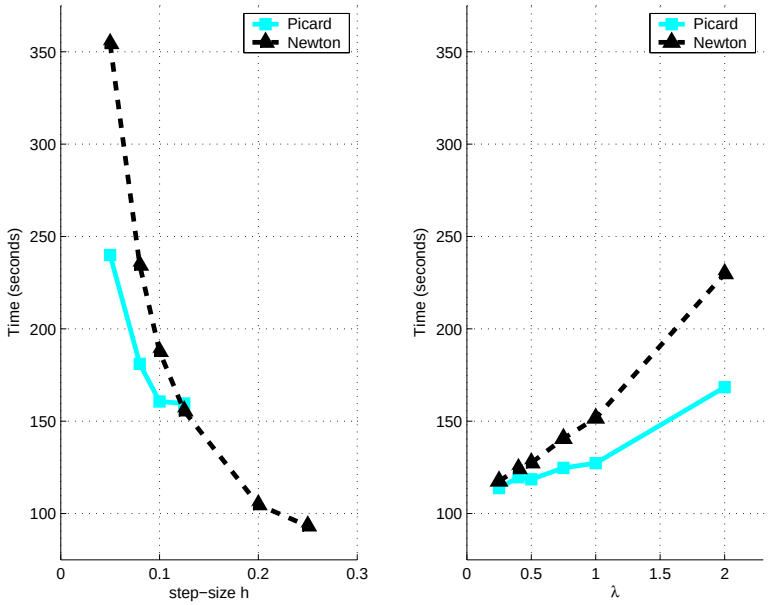

FIGURE 4.1.3. Comparison of execution time for h and λ for the Lotka-Volterra model

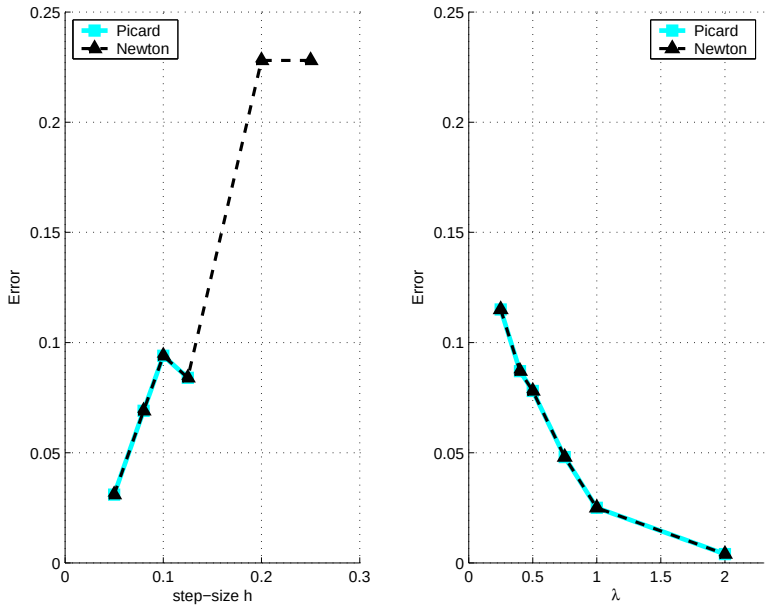

FIGURE 4.1.4. Error comparison for h and λ for the Lotka-Volterra model

4.2. The Kepler Problem. This example, taken from Hairer, Lubich, and Wanner [1], is the well known two-body problem describing planetary motion. Consider

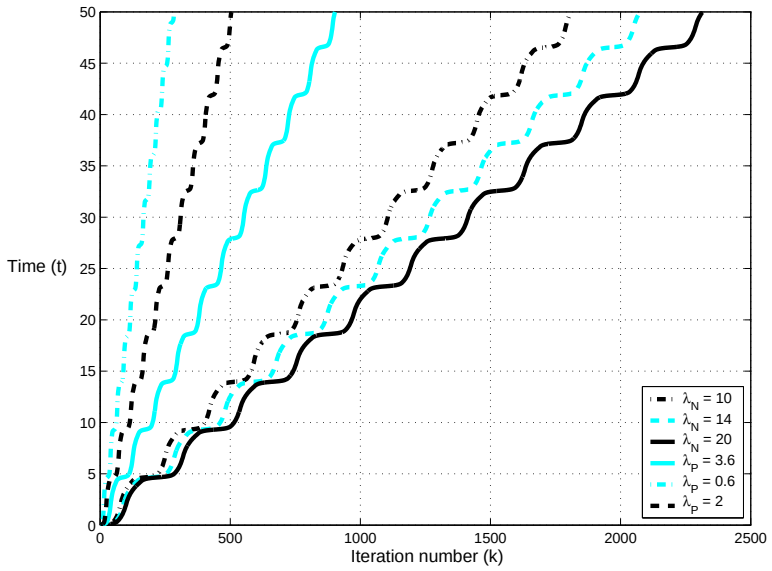

FIGURE 4.1.5. Comparison of the variable step-sizes determined by solving (35) versus (57)

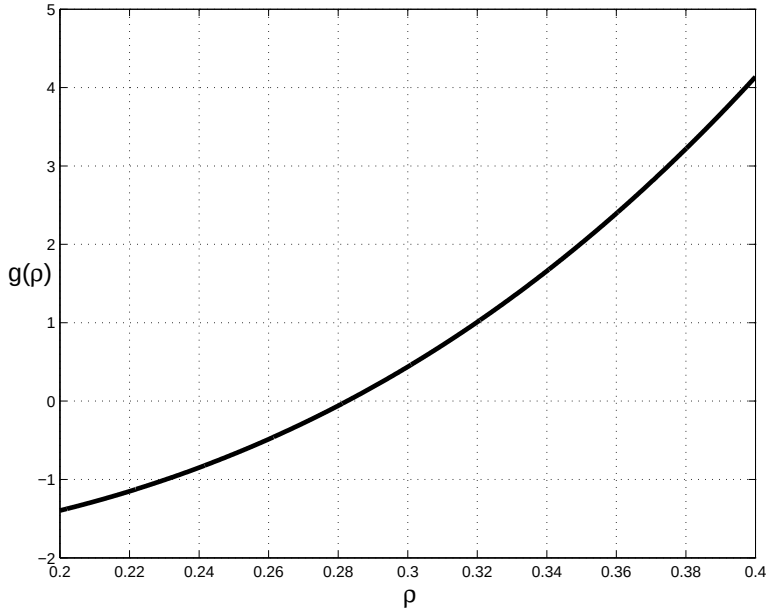

FIGURE 4.1.6. Plot of ρ versus $g(\rho)$, where g is given by the left hand side of (57)

the equations

$$\ddot{q}_1 = -\frac{q_1}{(q_1^2+q_2^2)^{3/2}} \tag{67}$$

$$\ddot{q}_2 = -\frac{q_2}{(q_1^2+q_2^2)^{3/2}}, \tag{68}$$

with the following initial conditions:

$$q_1(0) = 1 - e, \quad q_2(0) = 0, \quad \dot{q}_1(0) = 0, \quad \dot{q}_2(0) = \sqrt{\frac{1+e}{1-e}}, \tag{69}$$

where $0 \leq e < 1$. To describe the motion of two bodies, one of the bodies is taken to be the center of the coordinate system and the position of the second at any time t is given by the two coordinates $q_1(t)$ and $q_2(t)$. Equations (67)-(68) are equivalent to the following Hamiltonian system:

$$\dot{q}_i = p_i \quad i = 1, 2 \tag{70}$$

$$H(p_1, p_2, q_1, q_2) = \frac{\left(p_1^2 + p_2^2\right)}{2} - \frac{1}{\sqrt{q_1^2 + q_2^2}} \tag{71}$$

where $H(p_1, p_2, q_1, q_2)$ is the Hamiltonian of the system.

Before (67)-(68) can be integrated using the implicit midpoint rule, we convert the equations to a system of four first-order ordinary differential equations. Let $z_1 = q_1,\ z_2 = \dot{q}_1,\ z_3 = q_2$ and $z_4 = \dot{q}_2$. Define $z = [z_1\ z_2\ z_3\ z_4]^T$. Then (67)-(68) are equivalent to the following system:

$$\dot{z} = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} z_2 \\ -\frac{z_1}{\left(z_1^2+z_3^2\right)^{3/2}} \\ z_4 \\ -\frac{z_3}{\left(z_1^2+z_3^2\right)^{3/2}} \end{bmatrix}. \tag{72}$$

The above system of equations was solved using the implicit midpoint rule for $t \in [0, 50]$. Just as in the previous example, both a Picard iteration and Newton's method were used to compute the stage value y_1. The two measures we chose for this example are very similar to the those of the previous example. The first measure is T, the total cpu time required to solve the system 1000 times with eccentricity, e, that is uniformly distributed in the interval [0.4,0.8]. The second measure was E, the maximum absolute error that the integration deviates from the exact solution with eccentricity $e = 0.6$. For this measure, we considered the absolute error at every step of the integration.

The results from this numerical experiment are summarized in Tables 4.2.1 and 4.2.2 and also in Figures 4.2.1 and 4.2.2. We compare the performance of the variable step-size method to the fixed-step size method exactly as we did in the previous example. In Table 4.2.1 we begin with the fixed step-size $h = 0.01$. The problem is solved in 84.8 seconds using Picard iteration to find y_1, giving an error of 0.113. We compare this to $\lambda = 8$ in Table 4.2.2 where the problem is solved in 81.4 seconds with an error of 0.067. Also, when the fixed step-size is $h = 0.05$, the problem is solved in 21.2 seconds using Picard iteration to find y_1 and is solved in 29.6 using Newton's method to find y_1. The error in both cases is 1.58. We compare these to when $\lambda = 2$ in Table 4.2.2. Respectively, the times are 21.6 seconds and 29.9 seconds. The error for these two times is 0.64.

All of the computations, up until this point, in this section were performed on the same machine as those from Section 4.1. The computations below were performed in MATLAB® version 5.3.0.10183 Release 11 running in Microsoft Windows XP Professional version 5.1.2600 with an x86 Family 15 Model 4 Stepping 1 GenuineIntel processor running at 3056 Mhz.

Now we would like to compare the performance of the proposed variable step-size selection method versus the accepted variable step-size selection method of Stoer and Bulirsch [10], given by equation (6). For the method from Stoer and

Bulirsch, we chose Φ_1 to be of order two and Φ_2 to be of order three. The method is described in great detail in [10], including coefficients for the function evaluations in computing $\Phi_1(t_k, \bar{x}_k; h)$ and $\Phi_2(t_k, \bar{x}_k; h)$. The only control given to the user of such a method is the initial choice of h. First, we decided to test the method to see what kind of errors and time of execution we would get from various starting values of h. Twelve different values of h, ranging from $h = 5 \times 10^{-5}$ to $h = 2$, were chosen. The error function chosen, is exactly the same from the comparisons above in this section. We found that regardless of the starting value of h, the error ranged only from 2.88×10^{-3} to 2.90×10^{-3}. Furthermore, there seemed to be no significant difference in the times of execution either. The times ranged from 9.5 seconds to 13.0 seconds with no discernable order to them. Random computer processes could be a possible cause for some variation, but the mean time of execution for the twelve runs was 11.7 seconds. The results of this test are given in Table 4.2.3.

Next, we wanted to compare these results to the method proposed in this paper. As the results above suggest, we could not control the variable step-size selection method given by (6), so it was necessary to find a value of λ for the proposed method that would yield a similar error. After a few attempts, it was determined that $\lambda = 26.75$ was a good choice. For comparison purposes, we ran the code twenty times because of one to two second fluctuations in time of executions. We then determined the mean error for all twenty runs and the mean time of execution for all twenty runs. The mean error was 2.90×10^{-3} and the mean time of execution was 13.8 seconds.

Although, the mean time of execution was slightly higher for the proposed method, the method from Stoer and Bulirsch has one major drawback. The error is essentially uncontrollable aside from the tolerance used. The user has no ability to loosen an error requirement to gain speed of execution or vice versa. For the proposed method, this is controlled by the user through the parameter λ, much like a fixed step-size method can be controlled by choosing the step-size. The difference between them is that our results also show that the proposed method performs better than a fixed step-size method as well.

TABLE 4.2.1. Fixed step-size (Kepler problem)

$h \rightarrow$	0.01	0.05	0.1	0.125
T (Picard)	84.8	21.2	13.3	11.9
$E \times 10^{-1}$ (Picard)	1.1	15.8	20.4	24.8
T (Newton)	137.5	29.6	16.5	14.4
$E \times 10^{-1}$ (Newton)	1.1	15.8	20.4	24.8

TABLE 4.2.2. Variable step-size (Kepler problem)

$\lambda \rightarrow$	1	2	4	6	8	10
T (Picard)	20.2	21.6	37.2	58.5	81.4	111.4
$E \times 10^{-1}$ (Picard)	12.5	6.4	3.0	1.3	0.7	0.4
T (Newton)	22.1	29.9	50.6	79.3	114.7	157.2
$E \times 10^{-1}$ (Newton)	12.5	6.4	3.0	1.3	0.7	0.4

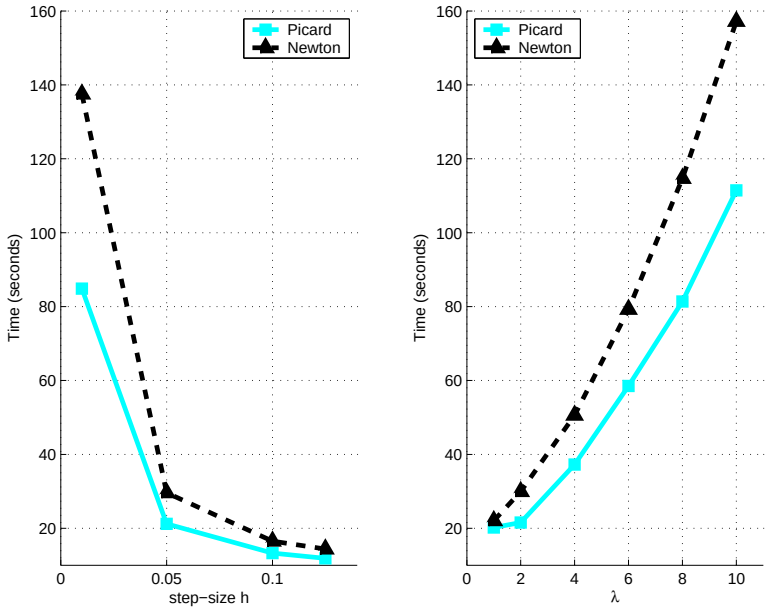

FIGURE 4.2.1. Comparison of execution time for h and λ for the Kepler problem

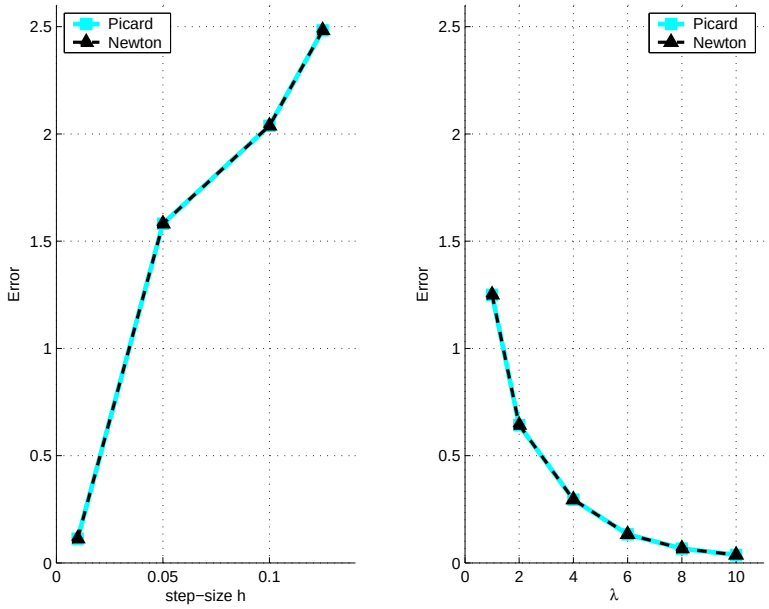

FIGURE 4.2.2. Error comparison for h and λ for the Kepler problem

4.3. The Kepler Problem Using Runge-Kutta-Nyström. Once again, we consider the Kepler problem described by equations (67)-(69). We now solve the problem using a Runge-Kutta-Nyström (RKN) method. The method we use is the

TABLE 4.2.3. Variable Step-Size Data using (6)

h	E (Picard)	Time (for one run)
5×10^{-5}	2.90×10^{-3}	11.4 seconds
1×10^{-4}	2.90×10^{-3}	13.0 seconds
5×10^{-4}	2.90×10^{-3}	12.2 seconds
1×10^{-3}	2.90×10^{-3}	9.5 seconds
5×10^{-3}	2.90×10^{-3}	11.3 seconds
0.01	2.90×10^{-3}	12.1 seconds
0.05	2.92×10^{-3}	10.9 seconds
0.1	2.91×10^{-3}	10.9 seconds
0.2	2.88×10^{-3}	11.4 seconds
0.5	2.91×10^{-3}	12.1 seconds
1	2.90×10^{-3}	12.3 seconds
2	2.91×10^{-3}	12.9 seconds

two-stage Gauss method from [3] which is the RKN method that is induced from the fourth order Gauss method (a Runge-Kutta routine) found in [1]. For notation, we will refer to this integration routine as Gauss4. The constants from equations (19)-(21) are

$$\gamma_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}, \quad \gamma_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}, \quad \beta_1 = \frac{1}{4} + \frac{\sqrt{3}}{12}, \quad \beta_2 = \frac{1}{4} - \frac{\sqrt{3}}{12}, \quad b_1 = \frac{1}{2}, \tag{73}$$

$$b_2 = \frac{1}{2}, \quad a_{11} = \frac{1}{24}, \quad a_{12} = \frac{1}{8} - \frac{\sqrt{3}}{12}, \quad a_{21} = \frac{1}{8} + \frac{\sqrt{3}}{12}, \quad a_{22} = \frac{1}{24}. \tag{74}$$

The analysis leading to equation (90) used the Lipschitz constant from the Runge-Kutta methods, $h\overline{L}\|A\|$. Since we are actually implementing a RKN method to solve this system now, we have to repeat that analysis process with the Lipschitz constant from the RKN methods, $h^2\overline{L}\|A\|$. Following exactly the same process given in equations (25)-(35) and using the substitution

$$x = h^2\overline{L}\|A\|, \tag{75}$$

we find that to find the optimal h we must solve

$$2 + \ln x + \lambda^2 x^{3/2} = 0 \tag{76}$$

for x and then solve for h using (75).

Tan [5] solves the Kepler problem using Gauss4 as a Runge-Kutta routine (not in its induced RKN form) with a fixed step-size of $\frac{\pi}{64}$ for 500,000 iterations. Just for comparison we did the following:

(1) We solve the problem using the fixed step-size $\frac{\pi}{64}$ on the interval [0,50] and determine the maximum absolute variance from the exact solution.
(2) We compute the total cpu time taken to solve the system using this step-size for 500,000 iterations as Tan does.
(3) We solve the system on the interval [0,50] using the optimal step-size selection method to determine the value of λ that gives us a comparable error as determined in step one.
(4) We solve the system on the entire interval using this value of λ and our variable step-size selection method.

When we used $h = \frac{\pi}{64}$ to solve the system on the interval [0,50], we found the maximum absolute variance from the exact solution to be 0.0045. Then using this fixed step-size, we found that it took 101.6 cpu seconds to solve the problem taking 500,000 iterations. We then determined that a comparable error on the interval [0,50] was achieved for $\lambda = 55$; that error was 0.0046. When we solve the problem using our variable step-size selection method and choose $\lambda = 55$ to a final time of $T = \frac{500000\pi}{64}$, we find that the system is solved in 61.5 cpu seconds, which is 39.5% faster than using the fixed step-size of $\frac{\pi}{64}$.

In addition to the above comparison, we also compared the fixed step-size method to the variable step-size method for various values of h and λ for this example. The results of this comparison are given in Tables 4.3.1 and 4.3.2. We only used Picard iteration to solve for the stage values. We used exactly the same two measures as we did in the previous section, when we solved the Kepler problem using the implicit midpoint rule. When we solve the system with the fixed step-size $h = 0.05$, the solution is found in 198.2 seconds with an error of 0.0050. Comparatively, we found that for $\lambda = 100$, the system is solved in 193.3 seconds with an error of 0.0036, which is about 28% lower error. We also found that when the system is solved with the fixed step-size of $h = 0.06$, it took 179.5 seconds with an error of 0.0071. We compare this to the variable step-size with $\lambda = 90$, which took only 182.7 seconds to once again get an error of 0.0036, which is about 49% lower error. In addition to these positive results, we did not notice a large decrease in the error using the variable step-size method as we increased λ. When we used the implicit midpoint rule in the previous two examples, we noticed a steady decline in the error as λ increased. In this case, we actually noticed that the error only goes from 0.0036 when $\lambda = 90$ to 0.0032 when $\lambda = 1000$. We point out that when $\lambda = 1000$, the system is solved in 625.8 seconds which is about 33% faster than the 927.8 seconds it took with the fixed step-size $h = 0.01$, where the error was only 0.0029.

All computations were done in MATLAB® version 6.5.0.180913a Release 13 running in Microsoft Windows XP Professional version 5.1.2600 with an x86 Family 15 Model 2 Stepping 7 GenuineIntel processor running at 2392 Mhz.

TABLE 4.3.1. Fixed step-size (Kepler problem using RKN)

$h \rightarrow$	0.01	0.025	0.05	0.06
T (Picard)	927.8	341.2	198.2	179.5
$E \times 10^{-3}$ (Picard)	2.9	3.0	5.0	7.1

TABLE 4.3.2. Variable step-size (Kepler problem using RKN)

$\lambda \rightarrow$	90	100	250	500	1000
T (Picard)	182.7	193.3	248.6	352.4	625.8
$E \times 10^{-3}$ (Picard)	3.6	3.6	3.2	3.2	3.2

4.4. The Van der Pol Equation. In this example, we would like to demonstrate the usefulness of this variable step-size selection method at solving a well-known stiff differential equation. Let us consider the Van der Pol oscillator:

$$\ddot{x} - \mu(1 - x^2)\dot{x} + \omega^2 x = 0 \,, \tag{77}$$

with $\omega = 1$ and $\mu = 100$. First, we write the equation as system of two first-order equations:

$$\dot{z} = \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} = \begin{bmatrix} z_2 \\ \mu(1 - z_1^2)z_2 - \omega^2 z_1 \end{bmatrix}, \tag{78}$$

where $z_1 = x$ and $z_2 = \dot{x}$.

Many authors [10, 14, 22, 23] suggest that implicitness is pivotal to efficiently solve stiff initial value problems. In addition to this, many agree that variable step-size methods are more beneficial to solving these stiff differential equations than fixed step-size routines. One reason for this in the case of the Van der Pol system is that part of the solution evolves on a much faster time scale than other parts of the solution as it progresses through the limit cycle. To illustrate this point using the proposed variable step-size selection method, we solve (78) for $t \in [0, 400]$ using the fixed step-size implicit midpoint method versus using the variable step-size implicit midpoint method.

Two sets of initial data were chosen for implementation. First, we chose $z(0) = [0 \;\; 1]^T = \hat{z}$, a starting vector that lies inside the loop of the limit cycle. Secondly, we chose $z(0) = [-1 \;\; 100]^T = \tilde{z}$, a starting vector that lies outside the loop of the limit cycle. For $\hat{z}$, $h = 0.004$ was chosen for the fixed step-size method, and the system is solved in 222.5 seconds. A plot of x versus $\dot{x}$ for this method is shown in Figure 4.4.3. Then, for the variable step-size method, a value of λ was chosen that yields a comparable time of execution. That value was $\lambda = 0.01$, and the system was solved in 218.8 seconds. A plot of x versus $\dot{x}$ can be found in Figure 4.4.4. When we choose $\tilde{z}$ as the initial data, we continue to use $\lambda = 0.01$ while solving the system in 313.8 seconds. A plot of the limit cycle for this execution is found in Figure 4.4.6. For comparison we then determine a fixed step-size that yields a comparable execution. This occurred when we let $h = 0.00345$. The system was solved in 316.3 seconds and a plot of the limit cycle is found in Figure 4.4.5. All of the above data is summarized in Table 4.4.1.

Even with the naked eye, it is easily discernable that the variable step-size method performs much better. The solution is smoother and stays closer to the actual limit cycle. Because of the small fixed step-sizes chosen, the fixed step-size routines perform rather well at approximating the solution along the more slowly evolving stretches of the limit cycle. However, there are two places where the dynamics of (78) evolve on a much faster time scale. These two places are at the peak and the valley of the limit cycle. To accurately approximate the solution at these places, it is necessary to have even smaller step-sizes than those chosen in the fixed step-size routine. This is exactly what the proposed variable step-size selection method does. As a result, it manages to approximate the exact solution much more smoothly than is possible for any fixed step-size in a comparable amount of time of execution.

Another goal of considering the stiff system (77) was to see how the variable step-size selection method from Stoer and Bulirsch would perform on the system. We integrated (78) with the initial data $z(0) = [0 \;\; 1]^T$. Although, we integrated the system using an implicit integrator, the step-sizes chosen according to (6) were specifically designed for one-step explicit integration schemes. Using (6), we encountered a problem getting the stage-value for the implicit integration to converge to a real number, just before the integration gets to $t = 0.19$. This problem is unavoidable. No matter what we choose as an initial value for h, the integration stalls at this point. This is exactly the point on Figure 4.4.4 where $[x(t) \;\; \dot{x}(t)]^T \approx [1.75 \;\; 0]^T$.

This is the point in the solution where the dynamics of the system cause the solution to make a very sharp clockwise turn onto the limit cycle. The fixed step-size method used $h = 0.004$ throughout, but using (6) to determine h yields $h = 0.0117$ at this point in the integration, which is too large for the stiff system. We used this step size on the fixed-step size code, only to encounter the same problem at nearly the same point in the integration.

The computations from this section were done in MATLAB® version 5.3.0.10183 Release 11 running in Microsoft Windows XP Professional version 5.1.2600 with an x86 Family 15 Model 4 Stepping 1 GenuineIntel processor running at 3056 Mhz.

TABLE 4.4.1. Fixed versus Variable step-size comparisons

$z(0) = [0 \;\; 1]^T$	$h = 0.004$	222.5 seconds
	$\lambda = 0.01$	218.8 seconds
$z(0) = [-1 \;\; 100]^T$	$h = 0.00345$	316.3 seconds
	$\lambda = 0.01$	313.8 seconds

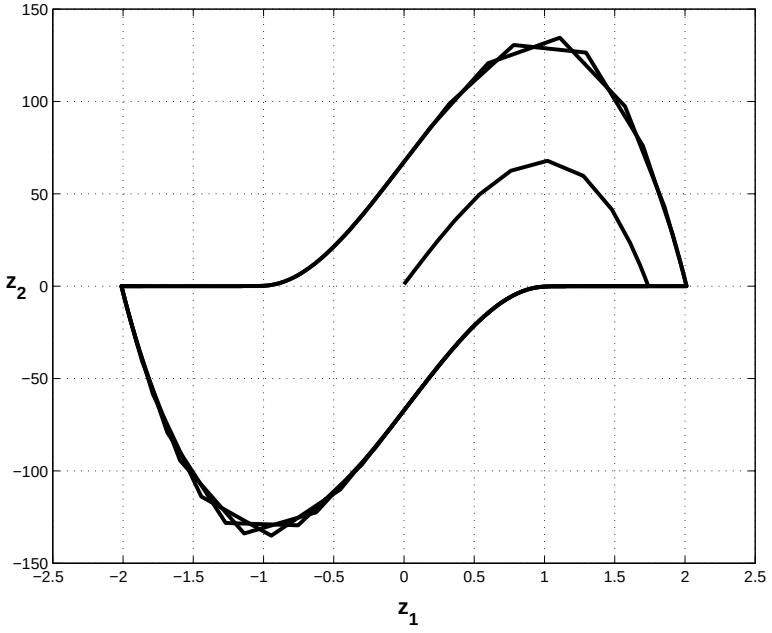

FIGURE 4.4.3. Van der Pol Oscillation: fixed step-size $h = 0.004$

5. Conclusion

The collection of implicit numerical integration routines is vast to say the least. Often times one routine is chosen over another to improve either efficiency or accuracy. In this paper, we have shown that it is possible to wisely choose a variable step-size for these integration schemes.

For linear ordinary differential equations or equations in which the Lipschitz constant for the function f is known, the task becomes quite simple as the optimal value of the step-size will not change from one step of the integration to the next.

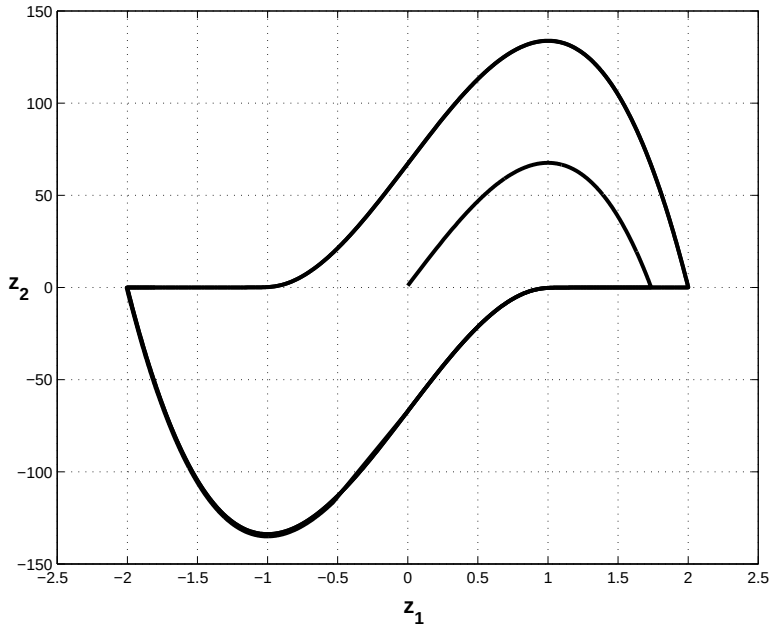

FIGURE 4.4.4. Van der Pol Oscillation: variable step-size $\lambda = 0.01$

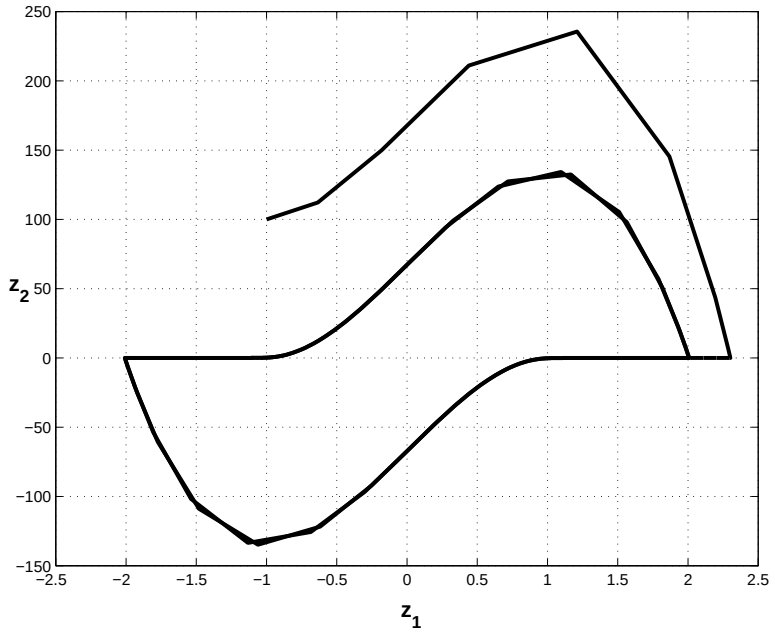

FIGURE 4.4.5. Van der Pol Oscillation: fixed step-size $h = 0.00345$

But, if we are dealing with more complicated non-linear differential equations, we can still choose an optimal time step at each step of integration of the system. As we have shown, this process often involves solving a non-linear equation numerically.

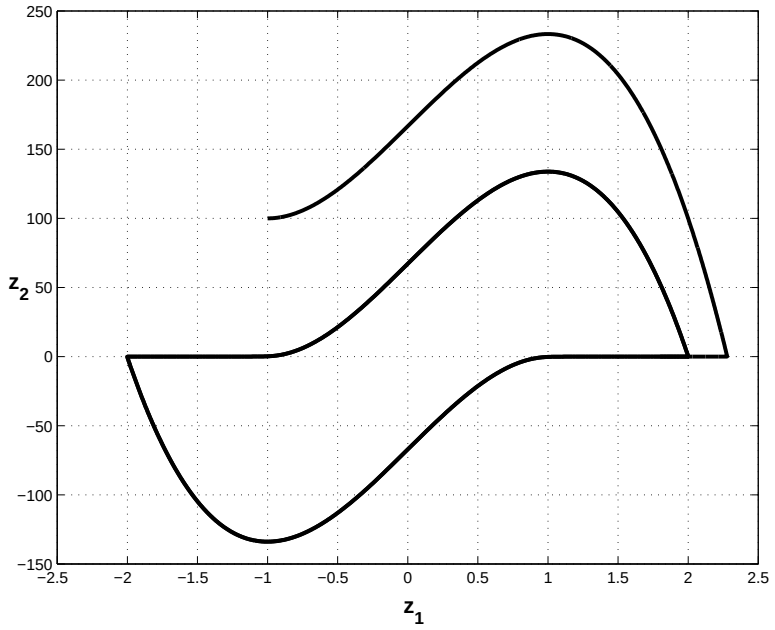

FIGURE 4.4.6. Van der Pol Oscillation: variable step-size $\lambda = 0.01$

Because of this, the computational overhead in using this optimal step-size routine seems to be too much for solving differential equations in which the function f is quite simple. However, our results have shown that this is consistently not the case when f is a complicated function.

The analysis and the derivation of the variable step-size method is described on an interval $[0, T]$ that is divided into uniform subintervals of fixed length h. Furthermore, the initial assumption of global Lipschitz continuity on f is also made. However, we have shown how the optimization may also be performed at each step of the integration to yield a variable step-size method. In addition, the method is applied to non-linear systems that have no global Lipschitz continuity. The results are good, and we summarize them below.

Tables 4.1.1, 4.1.2, 4.1.3, 4.1.4, 4.2.1 and 4.2.2 clearly show that, for comparable integration times, the variable step-size selection method presented in this paper drastically reduces the global error in the solution of the problem. For the Kepler problem, we found that for comparable execution times, the error was reduced 41% to 59% when the variable step-size method is used. In the Lotka-Volterra example, we found that for comparable execution times, the problem is solved with the error reduced 70% to 96% when we use the variable step-size method. From studying the tables we may choose λ so that execution times are comparable, in which case the variable step-size method noticeably reduces error as evidenced from the above discussion. However, λ may also be adjusted to find comparable errors between the fixed step-size and variable step-size methods. When you do this, one notices that the time required to achieve a comparable error for the fixed step-size is much larger.

Furthermore, in Section 4.2 we showed that the proposed variable step-size selection method is comparable to a widely accepted method described by Stoer and Bulirsch. To yield the same error in this example, the two methods took nearly

the same amount of time, with the well-known method having a slight advantage. However, we also argued that the proposed method retains a quality that should be very beneficial to a user. This quality is the ability of the user to control, through a parameter, the emphasis of the numerical integration on global error versus the emphasis on time of execution. This is a quality that was not exhibited by the earlier variable step-size selection method. In addition to this, we also showed that the accepted step-size selection method performs poorly on a very stiff Van der Pol oscillator. This is probably due to the fact that (6) was designed to control step-sizes for explicit integration schemes, which perform poorly on stiff systems.

We must point out that this optimal step-size selection process is dependent upon the scheme being used and we have concentrated on Runge-Kutta and Runge-Kutta-Nyström methods. It should not be too difficult of a task to adapt this process to the ever growing collection of implicit integration routines.

6. Appendix

6.1. Proof of Lemma 3.1.

Proof. Let $\lambda_i\left(R^TR\right)$ denote the i-th eigenvalue of R^TR. Since $R^TR > 0$, we have $\lambda_i\left(R^TR\right) > 0$ for $i = 1, \ldots, n$ and $\left(R^TR\right)^{-1} > 0$. Now using a well-known property of matrices and the Rayleigh-Ritz inequality, we get the following:

$$\left\|R^{-1}\right\|^2 = \lambda_{\max}\left(\left(R^TR\right)^{-1}\right) \tag{79}$$

$$= \frac{1}{\lambda_{\min}\left(R^TR\right)} \tag{80}$$

$$= \frac{\prod_{i=1}^{n}\lambda_i\left(R^TR\right)}{\lambda_{\min}\left(R^TR\right)\cdot\det\left(R^TR\right)} \tag{81}$$

$$\leq \frac{\left[\lambda_{\max}\left(R^TR\right)\right]^{n-1}}{\left(\det R\right)^2} \tag{82}$$

$$= \frac{\|R\|^{2(n-1)}}{\left(\det R\right)^2} \tag{83}$$

$$= \left(\frac{\|R\|^{n-1}}{\det R}\right)^2. \tag{84}$$

Taking square roots on both sides of the above inequality yields the desired result. □

6.2. Derivation of Equation (34). First we compute

$$\frac{dJ_3}{dh} = \frac{\ln(h\overline{L}\|A\|)+1}{T\ln(\varepsilon/C)} - 2\kappa Th \tag{85}$$

and

$$\frac{d^2J_3}{dh^2} = \frac{1}{Th\ln(\varepsilon/C)} - 2\kappa T. \tag{86}$$

In order to find $\max_h J_3(h)$, we must find the $h > 0$ that solves

$$\frac{\ln(h\overline{L}\|A\|)+1}{T\ln(\varepsilon/C)} - 2\kappa Th = 0 \tag{87}$$

such that

$$\frac{1}{Th\ln(\varepsilon/C)} - 2\kappa T < 0. \tag{88}$$

We must have T and h positive, and for a numerical solution to be meaningful, certainly ε must be very small and in general much less than C. Thus, $\ln(\varepsilon/C) < 0$. We then require $\kappa \geq 0$ to ensure that the second derivative of J_3 is negative, which guarantees that the solution to (87) is indeed a maximum.

Now let

$$\kappa = -\lambda^2 \frac{\overline{L}\|A\|}{2T^2\ln(\epsilon/C)} \ , \tag{89}$$

where λ is a free parameter that weights the optimization toward efficiency in time or toward global error. A better understanding of how λ affects the variable step-size selection process can best be explained by studying Table 4.1.2 and Table 4.2.2. By substituting this κ into (87), we find that we must solve

$$\ln(h\overline{L}\|A\|) + 1 + \lambda^2 h\overline{L}\|A\| = 0 \tag{90}$$

for h given an arbitrary value for λ.

Acknowledgments

R. Holsapple was supported by an AFRL Air Vehicles Directorate Graduate Student Assistantship during Summer 2004. R. Iyer was supported by an NRC/AFOSR Summer Faculty Fellowship during Summer 2004. The authors would also like to extend their sincere gratitude to the reviewers. Their many comments and suggestions helped the authors enhance the quality and effectiveness of this work.

References

[1] E. Hairer, C. Lubich, G. Wanner, Geometric Numerical Integration - Structure-Preserving Algorithms for Ordinary Differential Equations, Springer Series in Computational Mathematics, Springer-Verlag, Berlin, Germany, 2002.

[2] D. Lewis, J. Simo, Conserving algorithms for the N-dimensional rigid body, Fields Institute Communications 10 (1996) 121–139.

[3] J. Sanz-Serna, M. Calvo, Numerical Hamiltonian Problems, Applied Mathematics and Mathematical Computation, Chapman & Hall, London, United Kingdom, 1994.

[4] L. W. Roeger, A class of nonstandard symplectic discretization methods for Lotka-Volterra system, preprint - Texas Tech University Department of Mathematics and Statistics (2005).

[5] X. Tan, Almost symplectic Runge-Kutta schemes for Hamiltonian systems, Journal of Computational Physics 203 (1) (2005) 250–273.

[6] E. Hairer, G. Wanner, Algebraically stable and implementable Runge-Kutta methods of high order, SIAM Journal on Numerical Analysis 18, 1981.

[7] E. Hairer, G. Wanner, Characterization of non-linearly stable implicit Runge-Kutta methods, in: J. Hinze (Ed.), Numerical Integration of Differential Equations and Large Linear Systems, Vol. 968 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, Germany, 1982, pp. 207–219, proceedings of Two Workshops Held at the Univeristy of Bielefeld, Spring 1980.

[8] T. Hull, W. Enright, B. Fellen, A. Sedgwick, Comparing numerical methods for ordinary differential equations, SIAM Journal on Numerical Analysis 9 (4) (1972) 603–637.

[9] R. Bulirsch, J. Stoer, Numerical treatment of ordinary differential equations by extrapolation methods, Numerische Mathematik 8 (1966) 1–13.

[10] J. Stoer, R. Bulirsch, Introduction to Numerical Analysis, Third Edition, Texts in Applied Mathematics 12, Springer-Verlag, New York, New York, 2002.

[11] P. V. D. Houwen, Construction Of Integration Formulas For Initial Value Problems, Vol. 19 of North-Holland Series In Applied Mathematics And Mechanics, North-Holland Publishing Company, Amsterdam, 1977.

[12] B. Cano, A. Duran, Analysis of variable-stepsize linear multistep methods with special emphasis on symmetric ones, Mathematics of Computation 72 (244) (2003) 1769–1801.

[13] K. Gustafsson, G. Söderlind, Control strategies for the iterative solution of nonlinear equations in ODE solvers, SIAM J. Sci. Comput. 18 (1) (1997) 23–40.
[14] R. Aiken (Ed.), Stiff Computation, Oxford University Press, New York, 1985.
[15] K. Gustafsson, Control theoretic techniques for stepsize selection in explicit runge-kutta methods, ACM Transactions on Mathematical Software 17 (4) (1991) 533–554.
[16] K. Gustafsson, Control-theoretic techniques for stepsize selection in implicit runge-kutta methods, ACM Transactions on Mathematical Software 20 (4) (1994) 496–517.
[17] A. Iserles, A First Course in the Numerical Analysis of Differential Equations, Cambridge University Press, 1996.
[18] A. Stuart, A. Humphries, Dynamical Systems and Numerical Analysis, Cambridge Monographs on Applied and Computational Mathematics, Cambridge University Press, Cambridge, United Kingdom, 1998.
[19] E. Hairer, S. Nørsett, G. Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems, 2nd Edition, Springer Series in Computational Mathematics, Springer, 2002.
[20] L. Collatz, Functional Analysis and Numerical Analysis, Academic Press, New York, 1966.
[21] J. Ortega, W. Rheinboldt, Iterative solution of non-linear equations in several variables, Academic Press, New York, 1970.
[22] G. Sewell, The Numerical Solution of Ordinary and Partial Differential Equations, Academic Press, 1988.
[23] D. Kincaid, W. Cheney, Numerical Analysis, Second Edition, Brooks/Cole Publishing Company, Pacific Grove, CA, 1996.

Department of Mathematics and Statistics, Texas Tech University, Lubbock, TX 79409, USA
E-mail: raymond.w.holsapple@ttu.edu and ram.iyer@ttu.edu

USAF Research Laboratory, Wright-Patterson Air Force Base, OH 45433, USA
E-mail: david.doman@wpafb.af.mil