A Variable Step-Size Selection Method for Implicit Integration Schemes

Raymond Holsapple, Ram Iyer and David Doman

***Abstract*— Implicit integration schemes, such as Runge-Kutta 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 integration routines. We show the efficacy of this approach on some standard problems found in the literature.**

I. 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 [4], [10], [14], [13]. Tan [17] 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 IV-B. Furthermore, Hairer and Wanner [5], [6] 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 [1]. A goal of that paper was to compare these different types of methods as to how 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. If h is too large or too small the efficiency of an implicit scheme is relatively low. 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 concept in the past [8], [15], [7], [2]. 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 integration routines.

R. Holsapple and R. Iyer are with the Department of Mathematics and Statistics, Texas Tech University, Lubbock, TX 79409 raymond.w.holsapple@ttu.edu and ram.iyer@ttu.edu

D. Doman is with the United States Air Force Research Laboratory, Wright-Patterson Air Force Base, OH 45433-7531 david.doman@wpafb.af.mil

In the rest of this section, we describe the approach taken by Stoer and Bulirsch [15] for computing the variable step-sizes. The method described by Stoer and Bulirsch is quite similar to that of [8]. In order to describe the method, we first note that throughout this paper, we will consider solving the following first order ordinary differential equation:

$$\frac{dx}{dt} = f(t,x), \;\; x(0) = x_0 \in \mathbb{R}^n, \;\; t \in [0,T], \qquad (1)$$

where $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz continuous in the second argument. This means for any $t \in [0,T]$ and any vectors $x, y \in \mathbb{R}^n$, $\|f(t,x) - f(t,y)\| \leq L(t)\|x-y\|$, where $L(\cdot) \in L^\infty[0,T]$. Stoer and Bulirsch consider two discretization methods, Φ_1 and Φ_2, of Runge-Kutta type to solve (1). 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

$$\begin{aligned} \bar{x}_{k+1} &= \bar{x}_k + h_{\text{old}}\Phi_1(t_k, \bar{x}_k; h_{\text{old}}) \\ \hat{x}_{k+1} &= \bar{x}_k + h_{\text{old}}\Phi_2(t_k, \bar{x}_k; h_{\text{old}}). \end{aligned}$$

For a more detailed description of Φ_1 and Φ_2, please consult [15]. Then, denoting the tolerance by ε, they determine $h_{\text{new}} = h_{\text{old}} \left(\varepsilon \left| \bar{x}_{k+1} - \hat{x}_{k+1} \right|^{-1} \right)^{1/(p+1)}$. This process clearly depends on a measure of the local error at the $(k+1)-$st step of integration. Stoer and Bulirsch [15] 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.

II. Runge-Kutta Methods

Numerical methods for solving initial value problems such as (1) 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 (1). We assume the solution exists for $t \in [0, T]$, with $T > 0$.

A.M. Stuart and A.R. Humphries [16] describe a general s-stage fixed time-step Runge-Kutta method for the solution of (1) which may be written as:

$$y_i = x_k + h \sum_{j=1}^{s} a_{ij} f(t_k, y_j), \quad i = 1, ..., s, \tag{2}$$

$$x_{k+1} = x_k + h \sum_{i=1}^{s} b_i f(t_k, y_i), \quad x_0 = x(0). \tag{3}$$

In the above equations, the y_i's are called stage-values, 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.

We now make a few definitions. Let $A(i, j) = a_{ij}$ be the $s \times s$ matrix of a_{ij} entries. Let $Y_k = \begin{bmatrix} y_1^T & \cdots & y_s^T \end{bmatrix}^T$, $X_k = \begin{bmatrix} x_k^T & \cdots & x_k^T \end{bmatrix}^T$, and $\bar{A} = A \otimes I$, 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. Let $\tilde{f}(t_k, Y_k) = \begin{bmatrix} f(t_k, y_1)^T & \cdots & f(t_k, y_s)^T \end{bmatrix}^T$. We can now write the system of ns equations given in equation (2) as

$$Y_k = X_k + h\bar{A}\tilde{f}(t_k, Y_k). \tag{4}$$

Here we must point out that the subscript k in Y_k signifies that each y_i ($i = 1, ..., s$) depends on k. For each k this is an implicit equation involving the vectors $\{y_i\}_{i=1}^{s}$. Equation (4) can be solved using Newton's method or fixed point iteration (Picard iteration). Let's consider Picard iteration. We solve (4) for each k by considering the following iterative scheme:

$$Y_k^{j+1} = X_k + h\bar{A}\tilde{f}\left(t_k, Y_k^j\right) = F\left(t_k, Y_k^j\right). \tag{5}$$

For any fixed k, the iterative scheme given in (5) will converge to the solution of (4) provided that F satisfies a favorable condition. The following theorem addresses this convergence.

Theorem 2.1: Consider the iterative scheme given by equation (5). Let $L(t)$ be the Lipschitz number of f, and let A be the $s \times s$ matrix given above. Also, let $\overline{L} = \sup_{t \in [0,T]} L(t)$. If $h\overline{L}\|A\| < 1$ 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: We begin by referring to a useful result concerning norms of Kronecker products. According to Van Loan [11], $\|\bar{A}\| = \|A \otimes I\| = \|A\| \cdot \|I\| = \|A\|$. It is easily shown that the Lipschitz constant of $\tilde{f}$ is also $\overline{L}$. Now we compute the Lipschitz constant of F. Choose any $t \in [0, T]$ and any vectors u, $v \in \mathbb{R}^{ns}$.

$$\begin{aligned} \|F(t, u) - F(t, v)\| &= h\left\|\bar{A}\left(\tilde{f}(t, u) - \tilde{f}(t, v)\right)\right\| \\ &\leq h\left\|\bar{A}\right\|\left\|\tilde{f}(t, u) - \tilde{f}(t, v)\right\| \\ &\leq h\overline{L}\left\|A\right\|\left\|u - v\right\|. \end{aligned}$$

This shows that F is Lipschitz continuous in the second argument with Lipschitz constant $h\overline{L}\|A\|$. Since $h\overline{L}\|A\| < 1$, we may apply the Contractive Mapping Theorem to F [9]. Hence, there exists a unique point $\overline{Y} \in \mathbb{R}^{ns}$ such that $F(t_k, \overline{Y}) = \overline{Y}$ for any point $t_k \in [0, T]$ that is fixed. The theorem also ensures that $\overline{Y}$ must be the limit of every sequence obtained from (5) with a starting point $Y_k^0 \in \mathbb{R}^{ns}$. ■

Theorem 2.1 suggests how one might implement equations (3) and (5) to solve (1) 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 $\varepsilon > 0$.
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} using (3).

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 $h\overline{L}\|A\| < 1$ in order to guarantee that Newton's method converges. The Newton-Kantorovich Theorem [15], [3], [12] 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.

III. Step-Size Selection

When implementing a Runge-Kutta numerical integration routine, we have shown it is sufficient to assume that $h\overline{L}\|A\| < 1$ to guarantee convergence of the implicit scheme when using a Picard iteration. One might wonder though, is there an optimal choice, in the sense of computational efficiency, for h? If so, how might it be found?

A. Optimization Using Picard Iteration

Consider solving (1) numerically on the interval $[0, T]$ which is partitioned by the following sequence of points: $\{jh\}_{j=0}^{K}$. 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 (5) can be written as a function of the Lipschitz constant of the function F: $N_k = \phi\left(h\overline{L}\|A\|\right)$. Then the total number of computations over the interval $[0, T]$ can be written as: $N(h) = K\phi\left(h\overline{L}\|A\|\right) = (T/h)\phi\left(h\overline{L}\|A\|\right)$.

In the following, we find an explicit expression for $\phi(\cdot)$.

$$\begin{aligned}\left\|Y_k^{j+1}-Y_k^j\right\| &\leq h\overline{L}\|A\|\left\|Y_k^j-Y_k^{j-1}\right\| \cdots \\ &\leq \left(h\overline{L}\|A\|\right)^j\left\|Y_k^1-Y_k^0\right\| \\ &= C_k\left(h\overline{L}\|A\|\right)^j,\end{aligned}$$

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 $h\overline{L}\|A\| < 1$, 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 $C_k\left(h\overline{L}\|A\|\right)^j \leq \varepsilon$. 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 $C_k\left(h\overline{L}\|A\|\right)^M = \varepsilon$. We get $M = \ln\left(\varepsilon/C_k\right)\left(\ln\left(h\overline{L}\|A\|\right)\right)^{-1}$ and $N_k = \lceil M \rceil$. Let $C = \max_k C_k$. In the equation for M, 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 $\overline{L}$ 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 $\overline{L}$ and $\|A\|$. Notice that $C(h\overline{L}\|A\|)^M = \varepsilon$ implies that $C_k(h\overline{L}\|A\|)^M \leq \varepsilon$. We minimize $J_1(h) = K\ln\left(\varepsilon/C\right)\left(\ln\left(h\overline{L}\|A\|\right)\right)^{-1} = T\ln\left(\varepsilon/C\right)\left(h\ln\left(h\overline{L}\|A\|\right)\right)^{-1}$, which is the same as maximizing $J_2(h) = h\ln\left(h\overline{L}\|A\|\right)\left(T\ln\left(\varepsilon/C\right)\right)^{-1}$. 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. The implicit midpoint rule is a one-stage Runge-Kutta method where $a_{11} = 0.5$ and $b_1 = 1$ in (2) and (3). The implicit midpoint method has global error $\mathcal{O}(Th^2)$. Then to find h, we alter the cost function J_2 and maximize $J_3(h) = h\ln(h\overline{L}\|A\|)\left(T\ln(\varepsilon/C)\right)^{-1} - \kappa Th^2$, where κ is a number to be chosen. First we compute $(dJ_3/dh) = \left(\ln(h\overline{L}\|A\|)+1\right)\left(T\ln(\varepsilon/C)\right)^{-1} - 2\kappa Th$ and $(d^2J_3/dh^2) = \left(Th\ln(\varepsilon/C)\right)^{-1} - 2\kappa T$. In order to find $\max_h J_3(h)$, we must find the $h > 0$ that solves

$$\left(\ln(h\overline{L}\|A\|)+1\right)\left(T\ln(\varepsilon/C)\right)^{-1} - 2\kappa Th = 0 \qquad (6)$$

such that $\left(Th\ln(\varepsilon/C)\right)^{-1} - 2\kappa T < 0$. 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 (6) is indeed a maximum.

Now let $\kappa = -\lambda^2\overline{L}\|A\|\left(2T^2\ln(\epsilon/C)\right)^{-1}$, 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 IV-A.2 and Table IV-B.2. By substituting this κ into (6), we find that we must solve

$$\ln(h\overline{L}\|A\|) + 1 + \lambda^2 h\overline{L}\|A\| = 0 \qquad (7)$$

for h given an arbitrary value for λ. In practice, we actually make the substitution $x = h\overline{L}\|A\|$ and solve

$$\ln x + 1 + \lambda^2 x = 0 \qquad (8)$$

for x. We then compute $h = x\left(\overline{L}\|A\|\right)^{-1}$. The solution to this equation exists and is unique. This is because the h that solves (7) is the unique global maximum of the function J_3, which exists because of the concavity of J_3. Furthermore, equation (7) 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 (7) for a Runge-Kutta method other than the implicit midpoint rule, two things will change. First, $\|A\|$ will change as the method changes. It may also be necessary to change the second term in the cost function as well. Suppose the method chosen has global error $\mathcal{O}(Th^r)$. Then, $J_3(h)$ becomes $\tilde{J}_3(h) = h\ln(h\overline{L}\|A\|)\left(T\ln(\varepsilon/C)\right)^{-1} - \kappa Th^r$. If we then define $\kappa = -\lambda^2(\overline{L}\|A\|)^{r-1}\left(2T^2\ln(\varepsilon/C)\right)^{-1}$, we discover that $\ln x + 1 + \lambda^2 x^{r-1} = 0$ must be solved for x after again making the substitution $x = h\overline{L}\|A\|$.

B. 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 III-A is taken to optimize h. 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 III-A. 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 IV-A.)
2) Solve (8) for x once.
3) At $t = 0$, let $\overline{L} = \|Df\|$, where Df is the Jacobian matrix of f, and compute $h = x\left(\overline{L}\|A\|\right)^{-1}$.
4) Perform one step of integration using the implicit midpoint rule.
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 $t = T$.

IV. Examples

In this section we explore this variable step-size selection method for two problems, the Lotka-Volterra model and the Kepler problem.

A. *The Lotka-Volterra Model*

For this example we consider the Lotka-Volterra model of a simple predator-prey system from mathematical biology. This particular example is taken from Hairer, Lubich, and Wanner [4]. Consider the following system:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} u(v-2) \\ v(1-u) \end{bmatrix} = f(u,v); \quad t \in [0,50]. \qquad (9)$$

In (9), $u(t)$ 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 and the Lipschitz constant of the system as a whole is unknown, we will use the method described in Section III-B.

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 $I(u,v) = \ln u - u + 2\ln v - v$, 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 (9), 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 fastest 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:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} u^2 v(v-2) \\ v^2 u(1-u) \end{bmatrix} = f(u,v); \quad t \in [0,50]. \qquad (10)$$

This system has has the same invariant as (9), 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 (2). The first method is the algorithm of Section II, which is simply a Picard iteration. Secondly, we used Newton's method to compute y_1.

The results from this example are given in Table IV-A.1 and Table IV-A.2. In the tables, (P) stands for Picard and (N) stands for Newton, referring to the method used to compute the stage-values. 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 IV-A.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 IV-A.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.

As one can see from the example above, inherent with this variable step-size selection method is the choice of the parameter λ. We will use the system given by equation (10) to explain how one should choose an appropriate value of λ when integrating a system that evolves over a long period of time. Suppose we are interested in integrating the system described by (10) over the interval $[0, 500]$ or $[0, 1000]$. First, we choose a much smaller value for the final time of integration; in this example that value is $T = 50$. We then integrate the system over the interval $[0, 50]$ with a fixed step-size and at the same time with various values of λ. Essentially, we analyze how λ affects this system in particular, just as we did in the above example. After we have integrated the system over the much smaller time interval, we choose the value of λ that works best for this system to integrate the system over the entire time interval. 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.

All computations 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 IV-A.1

FIXED STEP-SIZE

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

B. *The Kepler Problem*

This example, taken from Hairer, Lubich, and Wanner [4], is the well known two-body problem describing planetary motion. Consider the equations

$$\ddot{q}_i = -\frac{q_i}{(q_1^2 + q_2^2)^{3/2}}, \quad i = 1, 2 \qquad (11)$$

TABLE IV-A.2

VARIABLE STEP-SIZE

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

with $q_1(0) = 1 - e$, $q_2(0) = 0$, $\dot{q}_1(0) = 0$, and $\dot{q}_2(0) = ((1+e)/(1-e))^{1/2}$, where $0 \le e < 1$. 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 $q_2(t)$.

Before (11) can be integrated using the implicit midpoint rule, we must 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 (11) is equivalent to the following system:

$$\dot{z} = \left[z_2 \quad \frac{-z_1}{(z_1^2 + z_3^2)^{3/2}} \quad z_4 \quad \frac{-z_3}{(z_1^2 + z_3^2)^{3/2}} \right]^T . \qquad (12)$$

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 computations were performed on the same machine as was the previous example, and the results are summarized in Table IV-B.1 and Table IV-B.2. Just as before, (P) stands for Picard and (N) stands for Newton in the tables. 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 IV-B.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 0.11. We compare this to $\lambda = 8$ in Table IV-B.2 where the problem is solved in 81.4 seconds with an error of 0.07. 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 seconds 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 IV-B.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 IV-A. 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.

TABLE IV-B.1

FIXED STEP-SIZE

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

TABLE IV-B.2

VARIABLE STEP-SIZE

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

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 [15]. 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 [15], 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_0. 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_0. Twelve different values of h_0, ranging from $h_0 = 5\times10^{-5}$ to $h_0 = 2$, 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_0, 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 IV-B.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 described by Stoer and Bulirsch, 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 the best 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 IV-B.3
VARIABLE STEP-SIZE DATA

h_0	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

V. 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. 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. 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 as we describe below.

Tables IV-A.1, IV-A.2, IV-B.1 and IV-B.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% and 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%, 94% and 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.

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 methods. It should not be too difficult of a task to adapt this process to the ever growing collection of implicit integration routines.

VI. ACKNOWLEDGEMENT

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.

REFERENCES

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

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

[3] L. Collatz. *Functional Analysis and Numerical Analysis*. Academic Press, New York, 1966.

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

[5] E. Hairer and G. Wanner. Algebraically stable and implementable runge-kutta methods of high order. *SIAM Journal on Numerical Analysis*, 18, 1981.

[6] E. Hairer and G. Wanner. *Characterization of Non-Linearly Stable Implicit Runge-Kutta Methods*, volume 968 of *Lecture Notes in Mathematics*, pages 207–219. Springer-Verlag, Berlin, Germany, 1982. Proceedings of Two Workshops Held at the Univeristy of Bielefeld, Spring 1980.

[7] P.J. Van Der Houwen. *Construction Of Integration Formulas For Initial Value Problems*, volume 19 of *North-Holland Series In Applied Mathematics And Mechanics*. North-Holland Publishing Company, Amsterdam, 1977.

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

[9] David Kincaid and Ward Cheney. *Numerical Analysis*. Brooks/Cole Publishing Company, Pacific Grove, CA, Second edition, 1996.

[10] D. Lewis and J.C. Simo. Conserving algorithms for the N-dimensional rigid body. *Fields Institute Communications*, 10:121–139, 1996.

[11] C.F. Van Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123:85–100, 2000.

[12] J.M. Ortega and W.C. Rheinboldt. *Iterative solution of non-linear equations in several variables*. Academic Press, New York, 1970.

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

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

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

[16] A.M. Stuart and A.R. Humphries. *Dynamical Systems and Numerical Analysis*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge, United Kingdom, 1998.

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