Applied Mathematics, 2011, 2, 711-717
doi:10.4236/am.2011.26094 Published Online June 2011 (http://www.SciRP.org/journal/am)

Stepsize Selection in Explicit Runge-Kutta Methods for Moderately Stiff Problems

Justin Steven Calder Prentice
Department of Applied Mathematics, *University of Johannesburg*, *Johannesburg*, *South Africa*
E-mail: *jprentice@uj.ac.za*
Received March 27, 2011; *revised April* 9, 2011; *accepted April* 12, 2011

Abstract

We present an algorithm for determining the stepsize in an explicit Runge-Kutta method that is suitable when solving moderately stiff differential equations. The algorithm has a geometric character, and is based on a pair of semicircles that enclose the boundary of the stability region in the left half of the complex plane. The algorithm includes an error control device. We describe a vectorized form of the algorithm, and present a corresponding MATLAB code. Numerical examples for Runge-Kutta methods of third and fourth order demonstrate the properties and capabilities of the algorithm.

Keywords: Moderately Stiff Problems, Runge-Kutta, Stepsize, Jacobian, Stability Region

1. Introduction

Stiff initial-value problems (IVPs) are often solved numerically using implicit A-stable Runge-Kutta (RK) methods. In such methods, there is no need to adjust the stepsize for the sake of stability, since the stability region of the method is the entire left half of the complex plane. These methods are particularly useful when the problem is very stiff. However, implementing an implicit RK method requires the solution of a nonlinear system of stage equations at each step, whereas an explicit RK method does not. It is, therefore, often feasible to use an explicit RK method to solve moderately stiff problems, wherein the stiffness constant λ is not too large. Often, the explicit RK pairs RK (3,4) [1] and RK (4,5) [2,3] are used to solve IVPs, so we will focus our attention on these methods.

The stability region of explicit RK methods is a bounded region S in the complex plane, and the stepsize h must be chosen such that, if the vector $h\lambda$ is in the left half of the complex plane, then it lies within S. Moreover, λ can be complex, so that $h\lambda$ does not necessarily lie along the real axis (even though h is always real). In this paper, we present a simple algorithm for determining h such that $h\lambda$ lies within S, for any relevant λ and S pertaining to an explicit RK3 or RK4 method.

2. Relevant Concepts, Terminology and Notation

An m-dimensional IVP of the form

$$\begin{aligned} \overline{y}' &= \overline{f}\left(x,\overline{y}\right) \\ \overline{y}\left(x_0\right) &= \overline{y}_0 \end{aligned} \tag{1}$$

where

$$\overline{y}' = \begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_m' \end{bmatrix}, f\left(x,y\right) = \begin{bmatrix} f_1\left(x,\overline{y}\right) \\ f_2\left(x,\overline{y}\right) \\ \vdots \\ f_m\left(x,\overline{y}\right) \end{bmatrix} \tag{2}$$

can be solved using an explicit RK method

$$\overline{w}_{i+1} = \overline{w}_i + h\overline{F}\left(x_i,\overline{w}_i\right), \tag{3}$$

where $\overline{w}_i$ denotes the approximate numerical solution at x_i (*i.e.* $\overline{w}_i \approx \overline{y}\left(x_i\right)$), $\overline{F}$ is a function that defines the method, and $h \equiv x_{i+1} - x_i$. From here on, the notation RKr indicates an explicit Runge-Kutta method of order r.

The stability function $R\left(z\right)$ of the RK method is obtained by applying the RK method to the Dahlquist equation

$$\begin{aligned} y' &= \lambda y \\ y\left(x_0\right) &= y_0 \end{aligned} \tag{4}$$

to get, with $w_0 = y_0$,

$$w_1 = w_0 + hF(\lambda w_0), \tag{5}$$

which can be written

$$w_1 = |R(z)| w_0 \tag{6}$$

with $z \equiv h\lambda$. As a simple example, consider the second-order method of Heun [4], which has

$$F(x_i, w_i) = \frac{f(x_i, w_i) + f(x_i + h, w_i + hf(x_i, w_i))}{2}. \tag{7}$$

Applying this method to the Dahlquist equation $(f(x, y) = \lambda y)$, we find

$$\begin{aligned} F(x_1, w_1) &= \frac{\lambda w_0 + \lambda w_0 + h\lambda^2 w_0}{2} = \left(\lambda + \frac{h\lambda^2}{2}\right) w_0 \\ \Rightarrow hF(x_1, w_1) &= \left(h\lambda + \frac{h^2\lambda^2}{2}\right) w_0 = \left(z + \frac{z^2}{2}\right) w_0, \end{aligned} \tag{8}$$

so that

$$w_1 = w_0 + \left(z + \frac{z^2}{2}\right) w_0 = \left(1 + z + \frac{z^2}{2}\right) w_0, \tag{9}$$

from which we identify

$$R(z) = 1 + z + \frac{z^2}{2}. \tag{10}$$

Generally speaking, the stability function for explicit RK methods is a power series in z that represents an approximation to the exponential solution of the Dahlquist equation. Indeed, we see that $R(z)$ in (10) is a low-order Taylor series for the exponential function e^z. For RK3 and RK4 we have [5]

$$R(z) = \begin{cases} 1 + z + \dfrac{z^2}{2} + \dfrac{z^3}{6} & \text{RK3} \\ 1 + z + \dfrac{z^2}{2} + \dfrac{z^3}{6} + \dfrac{z^4}{24} & \text{RK4} \end{cases} \tag{11}$$

and these are the stability functions that will interest us in the remainder of this paper.

The region of stability of the RK method is defined as [6]

$$S \equiv \{z \in \mathbb{C} : |R(z)| < 1\} \tag{12}$$

and we denote the boundary of this region by ∂S. For an explicit RK method, ∂S is a closed contour in the complex plane (see **Figure 1**).

We must consider complex values for z, because it is possible that the stiffness constant λ could be complex; this is particularly true for systems of ODEs, as in (1), where the stiffness constants are determined from the eigenvalues of the Jacobian

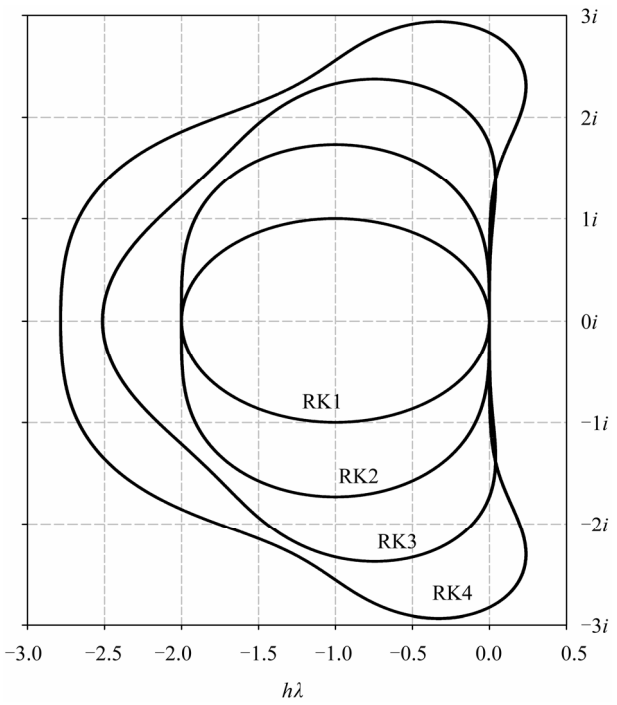

Figure 1. Stability regions for the indicated explicit RK methods.

$$\hat{J}(x, y) \equiv \begin{bmatrix} \dfrac{\partial f_1}{\partial y_1} & \cdots & \dfrac{\partial f_1}{\partial y_m} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial y_1} & \cdots & \dfrac{\partial f_m}{\partial y_m} \end{bmatrix} \tag{13}$$

There are m eigenvalues, and those with negative real part are taken as the stiffness constants of the system.

Now, assume $\lambda < 0$ in (4). The exact solution is therefore expected to be an exponentially decreasing function of x. However, if h is such that $|R(z)| > 1$, then the numerical solution (9) will increase with x, which is quite the opposite behaviour to what is expected. Furthermore, the numerical solution will increase without bound under iteration, whereas the exact solution tends to zero. This is referred to as an unstable solution, or instability with regard to stiff ODEs. It is therefore vital to choose h at each step of the RK method so that $|R(z)| < 1$, *i.e.* $z = h\lambda$ lies within S. An algorithm for determining an appropriate stepsize h is the subject of the next and subsequent sections.

3. Theoretical Description of the Algorithm

We first consider the algorithm for a single stiffness constant (eigenvalue of $\hat{J}$). Consider two semicircles C_1 and C_2, of radius r_1 and r_2, respectively, centered at the origin. These semicircles are such that, in the left half

of the complex plane, C_1 is contained entirely within S, and S is contained entirely within C_2, as shown in **Figure 2**. In this figure, S is the stability region of RK3 (for the sake of example), although the algorithm we describe here holds for the stability region of RK4, as well.

Say λ is an eigenvalue of $\hat{J}$, and λ lies in the left half of the complex plane (*i.e.* λ is a stiffness constant). Define the unit vector

$$\hat{\lambda} \equiv \frac{\lambda}{|\lambda|}, \tag{14}$$

where $|\lambda|$ is the magnitude of λ. Then

$$r_1\hat{\lambda} \text{ and } r_2\hat{\lambda} \tag{15}$$

are vectors of length r_1 and r_2, respectively, in the direction of λ, and

$$D \equiv \left|\left(r_2 - r_1\right)\hat{\lambda}\right| = r_2 - r_1 \tag{16}$$

is the length of the segment of $r_2\hat{\lambda}$ that cuts ∂S (see **Figure 3**).

Let ε be a user-defined tolerance, and compute

$$\begin{aligned} N &= \left\lceil \frac{D}{\varepsilon} \right\rceil \\ \varepsilon^* &= \frac{D}{N} \end{aligned} \tag{17}$$

This gives $\varepsilon^* \le \varepsilon$. This is nothing more than a refinement of the tolerance ε, consistent with an integer value of N.

Now determine, for $j = 0, \cdots, N$,

$$z_j = r_1\hat{\lambda} + j\varepsilon^*\hat{\lambda} \tag{18}$$

and compute

$$\left|R\left(z_j\right)\right| \tag{19}$$

for each j.

Find the largest z_j (in magnitude) such that $\left|R\left(z_j\right)\right| < 1$—call this z_c—and then determine

$$h = \frac{|z_c|}{|\lambda|}. \tag{20}$$

Now, say h^* is such that

$$\left|R\left(h^*\lambda\right)\right| = 1. \tag{21}$$

This gives

$$\begin{aligned} &h^*|\lambda| - h|\lambda| \le \varepsilon^* \\ &\Rightarrow h^* - h \le \frac{\varepsilon^*}{|\lambda|}, \end{aligned} \tag{22}$$

so that the difference between the stepsize estimated in (20) and the exact value h^* (in the sense of (21)) is de-

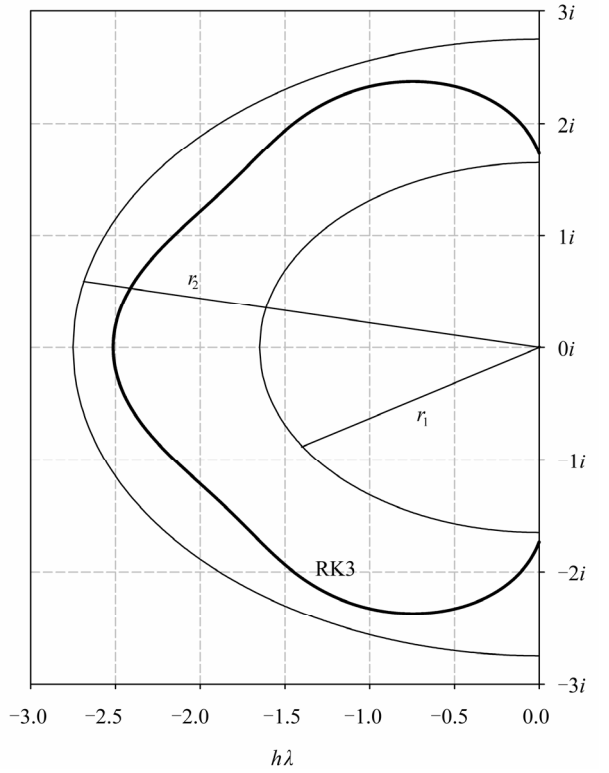

Figure 2. Semicircles of radii r_1 and r_2, for the stability region of RK3.

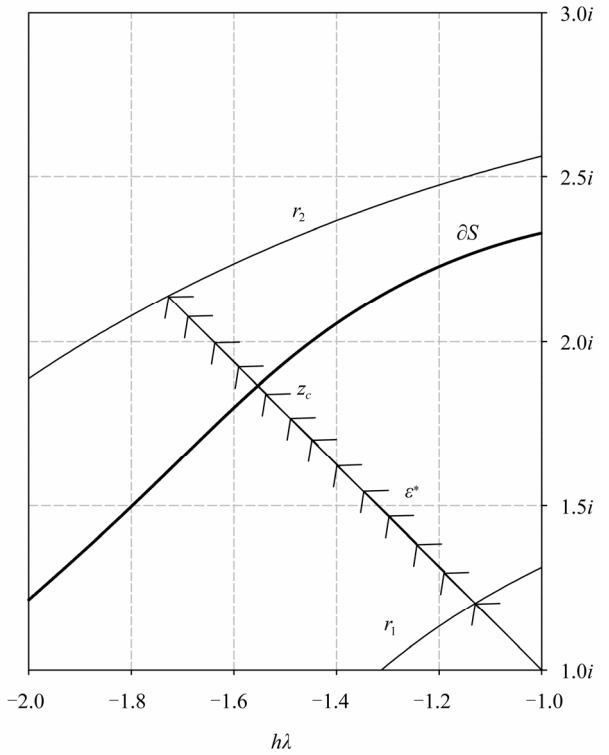

Figure 3. Geometrical representation of the algorithm. The boundary of the stability region is indicated as ∂S.

pendent on $\varepsilon^* \approx \varepsilon$, and $|\lambda|$. Hence, the smaller we choose ε, the closer the endpoint of $h\lambda$ is to ∂S, particularly for large $|\lambda|$.

Note the following: (22) gives

$$\frac{h^* - h}{h} \le \frac{\varepsilon^*}{h|\lambda|} \le \frac{\varepsilon}{h|\lambda|} \le \frac{\varepsilon}{r_1} \tag{23}$$

and if we demand that the relative difference $\frac{h^* - h}{h}$ must be less than some value δ, then choosing

$$\frac{\varepsilon}{r_1} = \delta \Rightarrow \varepsilon = r_1 \delta \tag{24}$$

ensures that this will be case. In other words, (24) provides a form of error control, since r_1 and δ are known a priori.

The algorithm is depicted in **Figure 3**. In this figure, the curves labelled r_1 and r_2 are the semicircles, and ∂S indicates the boundary of the stability region. The arrow touching the r_1 semicircle indicates $r_1\hat{\lambda}$, and the arrow touching the r_2 semicircle indicates $r_2\hat{\lambda}$. The arrows between these two represent z_j for $j = 1, \cdots, N-1$. The separation of two adjacent arrows is ε^*, as indicated. The arrow within S and closest to ∂S is z_c, as shown. It is clear that ∂S cuts $r_2\hat{\lambda}$. The segment between arrowheads that ∂S cuts is the segment of length D, referred to earlier.

For a system of ODEs, in which there is more than one stiffness constant, we would apply the above algorithm for each such constant. This would yield a stepsize for each stiffness constant, and we would choose the minimum of these stepsizes as h in (3).

4. Implementation of the Algorithm

Here we present a vectorized version of the algorithm, catering for a system of ODEs which has more than one stiffness constant.

The parameters r_1, r_2 and ε are input from which N and ε^* are easily determined. Assume that the set of eigenvalues of $\widehat{J}(x_i, \overline{w}_i)$ yields M distinct stiffness constants (note that $M \le m$). Let $\overline{\lambda}$ denote a row vector containing these M stiffness constants, as in

$$\overline{\lambda} \equiv [\lambda_1 \ \lambda_2 \ \cdots \ \lambda_M], \tag{25}$$

and let $\overline{\hat{\lambda}}$ denote the corresponding vector of unit vectors, as in

$$\begin{aligned} \overline{\hat{\lambda}} &= \left[\frac{\lambda_1}{|\lambda_1|} \ \ \frac{\lambda_2}{|\lambda_2|} \ \ \cdots \ \ \frac{\lambda_M}{|\lambda_M|} \right] \\ &= [\lambda_1 \ \ \lambda_2 \ \ \cdots \ \ \lambda_M]. \end{aligned} \tag{26}$$

Define the $N \times M$ matrices

$$\widehat{A} \equiv \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 2 & 2 & & 2 \\ \vdots & & \ddots & \vdots \\ N & N & \cdots & N \end{bmatrix} \tag{27}$$

$$\widehat{L} \equiv \begin{bmatrix} \overline{\hat{\lambda}} \\ \overline{\hat{\lambda}} \\ \vdots \\ \overline{\hat{\lambda}} \end{bmatrix} = \left.\begin{bmatrix} \lambda_1 & \lambda_2 & \cdots & \lambda_M \\ \lambda_1 & \lambda_2 & & \lambda_M \\ \vdots & & \ddots & \vdots \\ \lambda_1 & \lambda_2 & \cdots & \lambda_M \end{bmatrix}\right\} N \text{ times} \tag{28}$$

and hence, determine

$$\widehat{Z} = \left(1_{NM} r_1 + \widehat{A}\varepsilon^*\right) \boxtimes L \tag{29}$$

where 1_{NM} is the $N \times M$ unit matrix, and $\boxtimes$ denotes the element-by-element product of two matrices, sometimes known as the Hadamard product (the matrices must have the same dimensions, of course). The structure of $\widehat{Z}$ is

$$\widehat{Z} = \begin{bmatrix} \left(r_1 + \varepsilon^*\right)\hat{\lambda}_1 & \left(r_1 + \varepsilon^*\right)\hat{\lambda}_2 & \cdots & \left(r_1 + \varepsilon^*\right)\hat{\lambda}_M \\ \left(r_1 + 2\varepsilon^*\right)\hat{\lambda}_1 & \left(r_1 + 2\varepsilon^*\right)\hat{\lambda}_2 & & \left(r_1 + 2\varepsilon^*\right)\hat{\lambda}_M \\ \vdots & & \ddots & \vdots \\ \left(r_1 + N\varepsilon^*\right)\hat{\lambda}_1 & \left(r_1 + N\varepsilon^*\right)\hat{\lambda}_2 & \cdots & \left(r_1 + N\varepsilon^*\right)\hat{\lambda}_M \end{bmatrix}. \tag{30}$$

We then evaluate

$$\widehat{R}\left(\widehat{Z}\right) = \begin{cases} 1 + \widehat{Z} + \dfrac{\widehat{Z} \boxtimes \widehat{Z}}{2} + \dfrac{\widehat{Z} \boxtimes \widehat{Z} \boxtimes \widehat{Z}}{6} & \text{RK3} \\ 1 + \widehat{Z} + \dfrac{\widehat{Z} \boxtimes \widehat{Z}}{2} + \dfrac{\widehat{Z} \boxtimes \widehat{Z} \boxtimes \widehat{Z}}{6} & \\ \quad + \dfrac{\widehat{Z} \boxtimes \widehat{Z} \boxtimes \widehat{Z} \boxtimes \widehat{Z}}{24} & \text{RK4} \end{cases} \tag{31}$$

where $\widehat{R}\left(\widehat{Z}\right)$ is an $N \times M$ matrix, of which the jkth entry is

$$\left(\widehat{R}\left(\widehat{Z}\right)\right)_{jk} = R\left(\left(r_1 + j\varepsilon^*\right)\hat{\lambda}_k\right). \tag{32}$$

We compute $\left|\widehat{R}\left(\widehat{Z}\right)\right|$, the matrix containing the magnitude of each element of $\widehat{R}\left(\widehat{Z}\right)$, and we find the largest entry less than unity in each column of $\left|\widehat{R}\left(\widehat{Z}\right)\right|$. The corresponding elements in $\widehat{Z}$ are the z_c for each stiffness constant. There are M such values of z_c, one for each stiffness constant λ_k (denote them $z_{c,k}$), and we determine

$$h_k = \frac{|z_{c,k}|}{|\lambda_k|} \tag{33}$$

for each $k = 1, \cdots, M$.

Finally, the stepsize h used in the RK iteration (3) is simply

$$h = \min_{k=1,\ldots,M} \{h_k\} \tag{34}$$

We provide a MATLAB code in the Appendix, in the form of a function for determining h for RK3.

5. Comments

Some comments are in order:

1) It may occur that one of the z_j lies on ∂S, in which case $\left|R\left(z_j\right)\right| = 1$. If this is the case, we would still have that z_{j-1} is within ε^* of ∂S and, if we consider ε^* to be acceptably small, then z_{j-1} can be taken as z_c. Nevertheless, the algorithm can be refined to search for those occasions when $\left|R\left(z_j\right)\right| = 1$. It is our opinion that this refinement is not necessary.

2) Although the algorithm has been developed with RK3 and RK4 in mind, it should be clear that it can be applied to any RK method for which semicircles with radii r_1 and r_2 can be constructed (*i.e.* in the left half of the complex plane, one of the semicircles contains S entirely, and the other is contained entirely within S). For methods of order greater than four, however, the stability function $R(z)$ is method-specific so that the appropriate semicircles would also be method-specific. In our numerical examples in the next section, we will give semicircles that are universally applicable to all RK3 and RK4 methods.

3) We have not considered applying the algorithm to RK1 and RK2 because it does not seem possible to define a semicircle interior to S for these methods (see **Figure 1**). Moreover, the low order of these methods probably mitigates against their use in solving moderately stiff IVPs, anyway.

4) The vectorized algorithm described above is not the only way to determine h. We could use the algorithm with $M = 1$ in a for-loop, providing a different λ for each pass through the loop, which would give a stepsize for each pass, and then taking the minimum of these stepsizes. The vectorized algorithm seems, to us, to be more elegant and may also be faster, generally speaking, although this might depend on computational platform.

5) In principle, the algorithm can be applied for stiffness constants of any magnitude, although RK3 and RK4 would most likely be used only for moderately stiff problems, wherein $|\lambda| \lesssim 1000$.

6. Numerical Examples

In our first example, we consider the stiffness constants

$$\begin{aligned} \lambda_1 &= -1000 + 20i \\ \lambda_2 &= -435 + 480i \\ \lambda_3 &= -15 - 910i \end{aligned} \tag{35}$$

which have magnitudes (rounded to nearest integer) of 1000, 647 and 910, respectively. The first of these lies close to the real axis, the second is close to the diagonal, and the third is close to the imaginary axis.

Applying the algorithm to the RK3 stability region, with

$$\begin{aligned} r_1 &= 1.73 \\ r_2 &= 2.52 \end{aligned} \tag{36}$$

and $\varepsilon = 10^{-3}$, gives

$$\begin{aligned} h_1 &= 0.0025 \\ h_2 &= 0.0037 \\ h_3 &= 0.0020 \end{aligned} \tag{37}$$

Also, we find

$$\begin{aligned} R\left(h_1\lambda_1\right) &= 0.9995 \\ R\left(h_2\lambda_2\right) &= 0.9993 \\ R\left(h_3\lambda_3\right) &= 0.9997 \end{aligned} \tag{38}$$

and

$$\begin{aligned} \frac{\varepsilon^*}{h_1|\lambda_1|} &= 0.040\% \\ \frac{\varepsilon^*}{h_2|\lambda_2|} &= 0.042\% \\ \frac{\varepsilon^*}{h_3|\lambda_3|} &= 0.055\% \end{aligned} \tag{39}$$

where $\dfrac{\varepsilon^*}{h_k|\lambda_k|}$ is the upper bound on $\dfrac{h_k^* - h_k}{h_k}$, as per (22). Here, we have

$$\frac{\varepsilon}{r_1} = \frac{10^{-3}}{1.73} = 0.058\% \tag{40}$$

which is greater than the upper bounds in (39), as expected from (23). If we had desired a bound of $\delta = 0.01\%$, say, we would have needed to use $\varepsilon = r_1\delta = 1.73 \times 10^{-4}$.

Applying the algorithm to the stability region of RK4, with

$$\begin{aligned} r_1 &= 2.5 \\ r_2 &= 3.0. \end{aligned} \tag{41}$$

and $\varepsilon = 10^{-3}$, gives

$$\begin{aligned} h_1 &= 0.0028 \\ h_2 &= 0.0041 \\ h_3 &= 0.0031 \end{aligned} \tag{42}$$

Also, we find

$$R(h_1\lambda_1) = 0.9990$$
$$R(h_2\lambda_2) = 0.9989 \quad (43)$$
$$R(h_3\lambda_3) = 0.9987$$

and

$$\frac{\varepsilon^*}{h_1|\lambda_1|} = 0.036\%$$
$$\frac{\varepsilon^*}{h_2|\lambda_2|} = 0.037\% \quad (44)$$
$$\frac{\varepsilon^*}{h_3|\lambda_3|} = 0.035\%$$

These upper bounds are consistent with $\frac{\varepsilon}{r_1} = \frac{10^{-3}}{2.5} = 0.040\%$.

Clearly, the relative "error" in the stepsizes is very small. Of course, since it is proportional to ε^*, it could be made even smaller by choosing ε smaller, which can be done in a controlled way, using (24). Nevertheless, the calculations done here show that the stepsizes determined by the algorithm are very close to optimal, even for a fairly large tolerance of 10^{-3}. Note that the radii r_1 and r_2 used in these calculations are universally applicable to RK3 and RK4.

In our second example, we measure speed of computation for different values of *M*, for both the vectorized algorithm and the for-loop algorithm mentioned in #3 of our earlier comments. Our computational platform is MATLAB 6.5, Windows XP Pro, HP 30AA Mboard, Intel Celeron M 1.73 GHz, 1 MB L2 Cache, 2 GB RAM. Results are shown in **Table 1**.

In this table, all times are in seconds, *v* indicates vectorized algorithm, and *f* indicates for-loop algorithm. We see that the vectorized algorithm is slightly quicker than the for-loop version, in all cases. The algorithm is also faster for RK4 than for RK3, but this is due to the smaller value of $D = r_2 - r_1$, which leads to a smaller value of *N* (we have used $\varepsilon = 10^{-3}$ in all cases). Of course, the values of r_1 and r_2, and hence *D*, are dependent on the geometry of the stability region.

Table 1. Algorithm computation times (in seconds) for the indicated cases.

M	RK3	RK3	RK4	RK4
	v	*f*	*v*	*f*
20	0.015	0.024	0.016	0.024
200	0.194	0.289	0.178	0.203
400	0.383	0.563	0.359	0.398
1000	0.984	1.062	0.890	1.024

7. Conclusions

We have designed an algorithm for determining stepsizes appropriate for stable solutions of stiff IVPs, when such solutions are computed using explicit RK methods. The algorithm, based on a pair of semicircles that "sandwich" the boundary of the relevant stability region in the left half of the complex plane, is simple but robust and effective, and possesses the facility to control the accuracy with which the stepsize is determined. The algorithm caters for complex stiffness constants, which can arise in IVP systems, and can be implemented in vectorized or for-loop form, with the former appearing to be slightly faster in terms of execution time. Features of the algorithm have been ably demonstrated with respect to the RK3 and RK4 methods. Indeed, we expect that the algorithm will be most useful when using RK3 and RK4 to solve moderately stiff problems, although it can easily be used with explicit RK methods of higher order.

8. References

[1] J. C. Butcher, "Numerical Methods for Ordinary Differential Equations," Wiley, Chichester, 2003. doi:10.1002/0470868279

[2] E. Fehlberg, "Low-Order Classical Runge-Kutta Formulas with Step Size Control and Their Application to Some Heat Transfer Problems," *Computing*, Vol. 6, 1970, pp. 61-71. doi:10.1007/BF02241732

[3] L. F. Shampine and M. W. Reichelt, "The MATLAB ODE Suite," *SIAM Journal on Scientific Computing*, Vol. 18, No. 1, 1970, pp. 1-22. doi:10.1137/S1064827594276424

[4] D. Kincaid and W. Cheney, "Numerical Analysis: Mathematics of Scientific Computing," 3rd Edition, Brooks/Cole, Pacific Grove, 2002.

[5] E. Hairer and G. Wanner, "Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems," Springer, Berlin, 2000.

[6] R. L. Burden and J. D. Faires, "Numerical Analysis," 9th Edition, Brooks/Cole, Pacific Grove, 2011.

Appendix

The following is a MATLAB function file implementing the algorithm for RK3. It is easily modified for RK4 by changing the stability function in the tenth line. Text in small font to the right is commentary.

`function [f1] = STIFFSTEPSIZERK3(r1,r2,tol,lambda);`	
	`lambda` must be a row vector
`M = length(lambda);`	number of stiffness constants
`D = r2-r1;`	
`N = ceil(D/tol);`	
`newtol = D/N;`	this is ε^*
`unitlambda = lambda./abs(lambda);`	this is $\bar{\hat{\lambda}}$
`A = repmat([1:N]',1,M);`	this is $\widehat{A}$
`L = repmat(unitlambda,N,1);`	this is $\widehat{L}$
`Z = (A*newtol+r1).*L;`	`.*` is the MATLAB notation for $\boxtimes$
`R = 1 + Z + Z.^2/2 + Z.^3/6;`	this is $\widehat{R}\left(\widehat{Z}\right)$ for RK3
`I = find(abs(R)<1);`	this finds $\left\|\widehat{R}\left(\widehat{Z}\right)\right\|<1$
`B = zeros(N,M);`	
`B(I) = Z(I);`	this places the corresponding entries of $\widehat{Z}$ into a matrix `B` whose other entries are all zero
`h = max(abs(B))./abs(lambda);`	`max(abs(B))` finds the entry in each column of `B` of largest magnitude; these are the $z_{c,k}$.
`[f1] = min(h);`	output

