Comput. chem. Engng, Vol. 12, No. 7, pp. 705–717, 1988
Printed in Great Britain.

0098-1354/88 $3.00 + 0.00

ADAPTIVE RUNGE–KUTTA ALGORITHMS FOR DYNAMIC SIMULATION

I. T. CAMERON
Department of Chemical Engineering, University of Queensland, St Lucia, Queensland 4067, Australia

and

R. GANI
Instituttet for Kemiteknik, Danmarks Tekniske Hojskole, DK 2800, Lyngby, Denmark

(*Received for publication* 5 *January* 1988)

Abstract—This paper presents the development of an adaptive algorithm for the solution of ordinary differential equation systems. The methods used are based on two distinct classes of Runge–Kutta processes which are chosen to match the characteristics of the problem being solved. The principal characteristics related to the problem stability are assessed during the solution procedure and decisions are made on the type of numerical method to use at a given point in time.

It is shown that an adaptive algorithm using Runge–Kutta methods gives modest to significant computational savings over the use of a single implicit Runge–Kutta solution method.

INTRODUCTION

The numerical solution of differential systems arising from the modelling of transient behaviour is commonplace in the process industries and academia. Recourse is normally made to algorithms based on linear multistep methods or to the well-known Runge–Kutta single-step methods. In most cases, fixed formulae methods are used, necessitating the user in the initial choice of method. More recent algorithms based on the linear multistep methods have attempted to provide formulae which handle a wide range of stability characteristics of the problem. (Petzold, 1983).

This paper addresses the parallel development of adaptive Runge–Kutta methods which handle a similar spectrum of problems, thus easing the problem of algorithm choice by users.

THE MATHEMATICAL PROBLEM

Fundamental formulation

The basic problem addressed in this paper is that represented by a pure differential sytstem, whose right-hand-sides may contain numerous procedures or subroutines.

We can write this system in state variable form as:

$$\mathbf{y}' = \mathbf{f}(\mathbf{y}, \mathbf{P}, t), \tag{1}$$

where the procedures $\mathbf{P}$ are given by:

$$\mathbf{P} = \mathbf{g}(\mathbf{y}, \mathbf{z}, t),$$

with:

$$\mathbf{y} \in R^n, \quad \mathbf{z} \in R^m, \quad \mathbf{P} \in R^l, \quad t \in (a, b)$$

and

$$\mathbf{f}: R^n \times R^l \times R^1 \to R^n,$$

$$\mathbf{g}: R^n \times R^m \times R^1 \to R^l.$$

The vector y represents the differential variables and the vector z, the algebraic variables pertaining solely to the procedure functions P. Although the state variable formulation (SVF) given by equation (1) is the most straightforward, it is of course possible to generalize the problem formulation into other forms such as linear implicit (LIF), nonlinear implicit (NLIF) and residual (RF) forms.

BEHAVIOUR AND STABILITY

One of the most important aspects of solving either small- or large-scale dynamic problems is to understand the inherent stability properties of the system equations. This aids enormously in the correct application of appropriate numerical techniques.

Other problem characteristics which may also need consideration are:

- Problem size and sparsity
- Discontinuous behaviour
- Oscillatory behaviour
- Degree of nonlinearity

All of these characteristics play an important role in the efficiency and behaviour of the numerical techniques which are applied to the mathematical problem.

Of the characteristics which have been mentioned, the most significant is that of ultra-stability (or stiffness). It is essential to classify the problem into one of three main categories. Traditionally, we can

observe the behaviour of the linearized variational equations of the original problem, equation (1). These are given by:

$$\mathbf{y}' = \mathbf{f}(\mathbf{w}, \mathbf{P}, t) + \mathbb{J}(\mathbf{w}, t)(\mathbf{y} - \mathbf{w}), \tag{2}$$

where $\mathbf{w}$ is a particular solution.

$\mathbb{J}(\mathbf{w}, t)$ is the Jacobian $(\partial \mathbf{f}/\partial \mathbf{y} + \partial \mathbf{f}/\partial \mathbf{P}\ \partial \mathbf{P}/\partial \mathbf{y})$ of partial derivatives evaluated at $(\mathbf{w}, \mathbf{t})$.

The behaviour of the solution of $\mathbf{y}$ of equation (2) is characterized by the "local" eigenvalues λ_i; $i = 1(1)n$ of $\mathbb{J}$. These generally change with the independent variable t if the problem is nonlinear.

The three main problem categories that we can establish are:

1. Neutrally stable problems.
2. Unstable problems.
3. Ultra-stable (stiff) problems.

Neutrally stable problems are well-conditioned and cause no serious concern for traditional numerical techniques. Unstable problems present special concern to most techniques and as yet no reliable methods exist for their solution. Ultra-stable or stiff problems are extremely common in the physical sciences and put special demands on the type of numerical technique that can be effectively used on such a problem.

It is generally not possible to assess the degree of difficulty in solving general nonlinear differential equation problems before applying a numerical technique. It has been common practice in the past to use a theoretical stiffness ratio as a measure of solution difficulty, this being defined as:

$$S_t(t) = \frac{\max_i |\mathrm{Re}(\lambda_i)|}{\min_i |\mathrm{Re}(\lambda_i)|}. \tag{3}$$

However, this theoretical measure does not in fact help in assessing how difficult the solution procedure will be when using a particular numerical procedure. This is because it neither considers the accuracy demanded by the user nor the accuracy (local truncation error) of the basic method being used.

NUMERICAL PROCEDURES

Desirable characteristics of the methods

The primary characteristics of the methods used to solve problem (1) should be:

(a) robustness—ability to effectively handle a wide range of problems and possess good remedial/corrective action;

(b) efficiency—possess good computational characteristics;

(c) appropriateness—possess the desired characteristics applicable to the mathematical problem.

As well as the above characteristics, general dynamic problems impose the following demands on numerical techniques:

(d) moderate accuracy requirements are normally sought due to modelling inaccuracies in the problem being solved;

(e) output is often demanded at off-mesh points for graphics or hard copy;

(f) possibility of exploiting the problem structure, especially sparsity;

(g) possibility of exploiting component linearity;

(h) desirability of handling discontinuous problems. These may be time events (action at prescribed values of the independent variable) or state events (state variable or function of variables reaches a specified threshold);

(i) the use of analytic partial derivatives where applicable;

(j) the ability to detect or handle unstable problem behaviour.

All the foregoing characteristics often mean that compromises are necessary when developing effective algorithms. The following sections show how some of these characteristics can be addressed using the single-step Runge–Kutta methods.

Adaptive algorithms—a development philosophy

Due to the demands imposed on the numerical routines by the mathematical problem, it is clear that a single method is insufficient to cope with the wide range of demands placed upon it. It is also the contention of the authors that users of numerical software should not be burdened with the added responsibility of choosing an appropriate method. Rather, the choice of the method should be dictated by the local characteristics of the problem, which may change throughout the solution domain. The user of such software is ultimately interested in the solution of the model equations and does not necessarily wish to be involved in the intricacies of the solution technique.

Hence it is essential that any adaptive numerical software incorporates two basic features:

(1) the ability to automatically assess the problem characteristics;

(2) the ability to select the appropriate method from a set of routines in order to deal with those characteristics.

In the following sections we outline some of the development steps of such an algorithm. We examine how some of the characteristics can be assessed and we also develop an adaptive algorithm. Problems associated with both the assessment aspect as well as the method development are addressed. Finally, we illustrate the computational behaviour on some test systems.

Previous work by Petzold (1983) on linear multistep methods has resulted in a code which assessed

stiffness; Shampine (1984) has also addressed this aspect of the problem. Recently, Norsett and Thomsen (1986) have developed a stiff system solver based on type insensitive formulae. This essentially uses an implicit formula with either a functional or Newton iteration depending on the problem characteristics.

Single-step methods

By far the most well-known and widely used single-step methods are from the Runge–Kutta (RK) family. This family of methods possesses, not only a wide range of accuracies, but also an extensive range of stability properties. As such, they are obvious candidates for the development of adaptive algorithms.

We can write the s-stage RK method for the simplified differential system $\mathbf{y}' = \mathbf{f}(\mathbf{y}, t)$ as:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h_n \sum_{i=1}^{s} b_i \mathbf{f}(\mathbf{y}_{n,i}, t_{n,i}), \tag{4}$$

$$\mathbf{y}_{n,i} = \mathbf{y}_n + h_n \sum_{j=1}^{s} a_{ij} \mathbf{f}(\mathbf{y}_{n,j}, t_{n,j}) \quad i = 1(1)s,$$

with

$$t_{n,i} = t_n + c_i h_n.$$

The above method can be conveniently represented in the form of a Butcher block, which displays the system coefficients a_{ij}, b_i and c_i in the following matrix form:

$\mathbf{c}$	$\mathbb{A}$
	$\mathbf{b}^T$

Normally we demand that

$$\sum_j a_{ij} = c_i.$$

By imposing restrictions on the coefficients of the matrix $\mathbb{A}$ we can derive three major classes of RK methods.
These are:

(a) explicit $\quad a_{ij} = 0$ for $j \geqslant i$
(b) semi-explicit $\quad a_{ij} = 0$ for $j > i$
(c) implicit $\quad$ no restrictions on a_{ij}.

As was previously discussed in the section on problem behaviour, the stability properties of the mathematical problem necessitate the use of appropriate numerical methods. The fully-implicit methods can be made to possess the minimum requirement of A-stability and furthermore, it is possible to construct s-stage methods of order $2s$. However, the implementational aspects and their computational performance mitigate against their use. Work of Butcher (1976), Burrage (1978) and Burrage and Chipman (1979) has concentrated on the development of implicit methods with a matrix $\mathbb{A}$ which has a one-point spectrum. These methods can be implemented in a far more efficient manner utilizing a transformation proposed by Butcher (1979). The extra complexity for these methods is not considered justified when compared to the lower-order semi-implicit formulae (SIRK) to be discussed.

The SIRK methods, first suggested by Butcher (1964) were subsequently examined by Norsett (1974) and Alexander (1977). The diagonally implicit (DIRK) methods, where $a_{ii} = \gamma$, $i = 1(1)s$ were favoured because of computational efficiency. Error control for these methods was via a Richardson extrapolation technique (Alexander, 1977) or used embedding (Cash, 1979). Construction of variable step-variable order (VSVO) algorithms based on triples was proposed by Cash and Liem (1980) or via fully embedded formulae (Cameron, 1983).

The great attraction of the DIRK codes is their ability to possess extremely powerful stability properties and yet remain computationally competitive. A-stability is easily achieved as is L-stability. The more restrictive forms of B-stability, S-stability and algebraic stability, based on nonlinear test equations, does mean that the methods are restricted to lower-order formulae. Strong S-stability has been shown to be more than adequate when dealing with difficult, ill-conditioned chemical engineering problems (Cameron, 1981; Cameron *et al.*, 1986).

The Rosenbrock methods, which are essentially a linearized form of the SIRK methods have proved popular within the chemical engineering field (Cailland and Padmanabhan, 1970; Chan *et al.*, 1978; Feng *et al.*, 1984). They rely on the use of an exact, analytic Jacobian in order to maintain solution integrity. They also require the Jacobian to be evaluated at each step for embedded formulations, or twice per step if Richardson extrapolation is used for local error control. In certain circumstances their use can be appealing since the linear problems of the form:

$$\mathbf{y}' = \mathbb{B}\mathbf{y} + \mathbf{q}$$

the Jacobian is simply the matrix of coefficients $\mathbb{B}$ and this is directly available from the evaluation of the functions $\mathbf{f}$. However, it is generally the case that dynamic problems are nonlinear in $\mathbf{y}$ and thus they do not compete favourably with equivalent DIRK methods due to the number of matrix decompositions required.

Explicit RK (ERK) methods have been used successfully for many years on a wide range of neutrally stable problems. Their A-stability boundaries are closed contours in the $h\lambda$-plane and thus they are not suitable for the solution of ultra-stable problems.

Adaptive Runge–Kutta methods

Method development. The purpose of this section is to construct an adaptive algorithm which incorporates RK methods with the desired characteristics. Since modest accuracy requirements are generally demanded, we will restrict our consideration of these

formulae to lower orders, typically order $p = 2$ and 3, although extension to higher orders is also feasible.

As a minimum requirement we demand that there should be both an explicit, nonstiff method as well as an appropriate stiff method.

Explicit method. Consider firstly the construction of an explicit RK method with 2nd- and 3rd-order formulae ($p = 2, 3$). Using the embedding technique, a cheap estimate of the local truncation error of the 2nd-order formula can be made using the 3rd-order solution. If necessary, local extrapolation can be done, with the solution being advanced by use of the higher-order formula.

For the 2nd-order method, in 2 stages we have the following Butcher block ($p = s = 2$):

$$\begin{array}{c|cc} 0 & 0 & \\ c_2 & a_{21} & 0 \\ \hline & b_1 & b_2 \end{array} \tag{5}$$

Adding an additional stage calculation, we can write the coefficients for the 3rd-order method ($p = s = 3$) as:

$$\begin{array}{c|ccc} 0 & 0 & & \\ c_2 & a_{21} & 0 & \\ c_3 & a_{31} & a_{32} & 0 \\ \hline & d_1 & d_2 & d_3 \end{array} \tag{6}$$

To solve for the RK parameters a_{ij}, b_i, c_i, we need the algebraic order conditions which come from the study of the power series in h of the terms y_{n+1} and $y_{n,i}$. These are given by the corresponding Butcher series defined by Hairer and Wanner (1974) as:

$$B(a, y_0) = \sum_{\tau \in T} \frac{h^{\rho(\tau)}}{\rho(\tau)!} a(\tau) F(\tau)(y_0), \tag{7}$$

where a: $T \to R$ and T is the set of all rooted trees. The power $\rho(\tau)$ is the order of the rooted tree, being equal to the number of nodes in the tree. $F(\tau)$ (y_0) are the elementary differentials of the Butcher series.

The true solution of the autonomous problem $y' = f(y)$ is given by:

$$y(x_n + h) = \sum_{\tau \in T} F(\tau)(y_n) \frac{h^{\rho(\tau)}}{\rho(\tau)!} = B(y, y_n), \tag{8}$$

whereas for the numerical solution, we obtain a different Butcher series such that:

$$y_{n+1} = B(y_{n+1}, y_n) \cong y(x_n + h). \tag{9}$$

Here $y_{n+1}(\tau)$ represents the elementary weights which are functions of the RK parameters. Matching $y_{n+1}(\tau)$ with $y(\tau)$ up to a certain order p, ensures that the numerical solution is of order p. For the true solution we have:

$$y(\tau) = 1, \quad \forall \tau \in T. \tag{10}$$

Writing the original RK formulation equation (3) in terms of the Butcher series and then manipulating, we obtain the order conditions as:

$$y_{n+1}(\tau) = \Gamma(\tau) \sum_{i=1}^{s} b_i \xi_i(\tau), \tag{11}$$

where:

$$T(\tau) = \rho(\tau)\Gamma(\tau_1)\Gamma(\tau_2) \ldots \Gamma(\tau_m), \tag{12}$$

$\rho(\tau_i) =$ number of nodes in tree τ_i

and

$$\xi_i(\tau) = \sum_{j_1, j_2 \ldots j_m = 1}^{i} a_{ij_1} a_{ij_2} \ldots a_{ij_m} \xi_{j_1}(\tau_1) \ldots \xi_{j_m}(\tau_m), \tag{13}$$

with the tree τ given by $\tau = [\tau_1, \tau_2, \ldots, \tau_m]$ in Fig. 1.

Thus, from the order conditions, equation (12) and the fact that for an order p method $y_{n+1}(\tau) = y(\tau) = 1$ we have the final order equation:

$$\sum_{i=1}^{s} b_i \xi_i(\tau) = \frac{1}{\Gamma(\tau)}, \quad \forall \tau \in T, \ \rho(\tau) \leqslant p. \tag{14}$$

The advantage of having the order equations in this form is that the derivation of higher order methods is made much easier.

For the low-order, adaptive algorithm given by the Butcher blocks (4, 5) we can write the following order condition equations (after eliminating parameters which are zero and replacing row sums with c_i):

order 2 method:
$$b_1 + b_2 = 1, \qquad b_2 c_2 = 1/2; \tag{15}$$

order 3 method:
$$d_1 + d_2 + d_3 = 1, \quad d_2 c_2 + d_3 c_3 = 1/2, \quad d_2 c_2^2 + d_3 c_3^2 = 1/3, \quad d_3 a_{32} c_2 = 1/6. \tag{16}$$

These equations represent a two-parameter (c_2, c_3) family of explicit RK methods. It is convenient to restrict c_2, $c_3 \in (0,1)$ so that information is not being generated beyond or behind the current step. This has special implications with regard to discontinuous problems where a quadrature point c_i outside of the current step means that detection of the discontinuity becomes difficult.

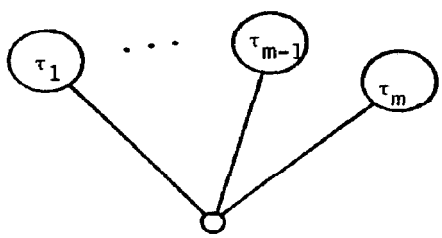

Fig. 1. Rooted tree structure.

The general solution of the order equations (15) and (16) is given by:

$$
\begin{aligned}
b_1 &= (2c_2 - 1)/2c_2,\\
b_2 &= 1/2c_2 = 1 - b_1,\\
d_1 &= [2c_2c_3(c_2 - c_3) + (c_3^2 - c_2^2)\\
&\quad - 2(c_3 - c_2)/3]/2c_2c_3(c_2 - c_3),\\
d_2 &= (c_3 - 2/3)/2c_2(c_3 - c_2),\\
d_3 &= 1 - d_1 - d_2,\\
a_{32} &= c_3(c_2 - c_3)/3c_2(c_2 - 2/3),\\
a_{31} &= c_3 - a_{32}.
\end{aligned} \tag{17}
$$

Selecting values for c_2 and c_3, we can solve the above system of equations for the unknown RK parameters. There are, of course, many possible solutions. An analysis of local truncation error and parameter magnitude can help decide on the most useful formulae.

Local truncation error. In terms of the previously stated Butcher series we define the local truncation error (ξ_p) for the pth-order RK method as:

$$
\xi_p = \frac{h^{\rho(\tau)}}{\rho(\tau)!} \sum_{\substack{\tau \in T \\ \rho(\tau) = p+1}} e(\tau) F(\tau)(y_n) + 0(h^{p+2}), \tag{18}
$$

where $e(\tau) = y_{n+1}(\tau) - y(\tau) = y_{n+1}(\tau) - 1$.

From the above expression we define the maximum error constant (E_p) for an order p method as:

$$
E_p = \max_{\rho(\tau) = p+1} |e(\tau)|. \tag{19}
$$

We can investigate the effect of the variation of c_2 on the truncation error of the various explicit methods. On this basis, we can choose the value of c_2 which minimizes E_p values for the methods.

For the explicit methods under consideration, we can generate the E_p values from the elementary weights $y(\tau)$. These are given in Table 1 as a function of the parameter c_2 with $c_3 = 1.0$.

The choice of $c_2 = 1/2$ is recommended from the above results. This value of c_2 means that several of the 4th-order terms in the Butcher series are also satisfied, thus reducing the magnitude of the local truncation error.

The final Butcher blocks are:

$p = 2$

0	0	
$\frac{1}{2}$	$\frac{1}{2}$	0
	0	1

(20)

$p = 3$

0	0		
$\frac{1}{2}$	$\frac{1}{2}$	0	
1	−1	2	0
	1/6	2/3	1/6

(21)

Table 1. Maximum error constants

	$p = 2$	$p = 3$
c_2	E_2	E_3
0.25	1.1	1.0
0.50	1.0	1.0
0.75	2.5	1.0

Alternatively, we can express the methods in the following form:

2nd-order embedded formula:

$$
\begin{aligned}
y_{n+1} &= y_n + h_n k_2,\\
k_1 &= f(y_n, t_n),\\
k_2 &= f(y_n + 0.5\, h_n k_1, t_n + 0.5\, h_n);
\end{aligned} \tag{22}
$$

3rd-order formula:

$$
\begin{aligned}
y_{n+1} &= y_n + \frac{h_n}{6}(k_1 + 4k_2 + k_3),\\
k_1 &= f(y_n, t_n),\\
k_2 &= f(y_n + 0.5\, h_n k_1, t_n + 0.5\, h_n),\\
k_3 &= f(y_n - h_n k_1 + 2h_n k_2, t_n + h_n).
\end{aligned} \tag{23}
$$

An implicit formula. As well as the explicit RK formula, an implicit formula is also required which possesses the appropriate stability properties. It should also be of the same order so that efficient switching can be made between the two formulae without the need to radically adjust the steplength. Of the implicit or semi-implicit RK methods already discussed, an attractive formula is that due to Cash (1979) being a modification of a formula originally developed by Alexander (1977). This 3rd-order DIRK formula has a Butcher block of the following form:

c_1	γ		
c_2	a_{21}	γ	
c_3	a_{31}	a_{32}	γ
	b_1	b_2	b_3

(24)

where

$$
\begin{aligned}
c_1 = \gamma &= 0.4358665215085,\\
a_{21} &= 0.2820667392458,\\
a_{31} = b_1 &= 1.208496649176,\\
a_{32} = b_2 &= -0.644363170684,\\
a_{33} = b_3 &= \gamma,\\
c_2 &= 0.717933260754,\\
c_3 &= 1.0.
\end{aligned} \tag{25}
$$

The above formula has been used extensively for the solution of both small- and large-scale dynamic simulation problems (Cameron, 1981).

It also has the advantage of possessing strong S-stability as well as allowing interpolation to non-mesh points via a Hermite formula. In order to monitor the error per step we use an embedded method of order 2 given by the Butcher block:

$$\begin{array}{c|cc} c_1 & \gamma & \\ c_2 & a_{21} & \gamma \\ \hline & d_1 & d_2 \end{array}, \tag{26}$$

where c_1, c_2, a_{21} and γ are given by the 3rd-order DIRK method and $d_1 = 0.7726301276676$, $d_2 = 0.2273698723324$.

The error per step can be estimated from the difference between the solutions of the 2nd- and 3rd-order method. Local extrapolation is used by carrying the solution forward using the 3rd-order method.

Using the definition of the local truncation error (18) the maximum error constant E_p for the DIRK ($p = 3$) method is 0.62. The corresponding explicit method has a slightly higher value of 1.0, although some of the 4th-order terms are satisfied.

Equilibrium states of explicit RK methods

When using an explicit method, it is necessary that we have an effective means of detecting when stability is restricting the steplength. Tests have been proposed which examine the steplength history in order to infer the presence of the equilibrium state where the steplength ($h\lambda$) is at the stability boundary of the explicit method.

The general RK method can be written as:

$$y_{n+1} = R(H_n)y_n, \tag{27}$$

where $R(H_n)$ is a polynomial in $h_n\lambda$. The equilibrium state (Hall, 1985) can be defined as:

$$R(H_L) = 1, \tag{28}$$

where H_L is the lower limit of absolute stability in the $h\lambda$-plane. This of course depends on the eigenvalues of the equation system, both their magnitude and distribution. The most effective numerical means to-date of detecting this equilibrium situation was proposed by Shampine (1977).

The technique monitored the local error of both an embedded lower-order method ($p = 1$) at a fractional steplength as well as the higher-order method at the full steplength. Under circumstances where the steplength of the higher-order method is not constrained by stability reasons, the lower-order method will not generally succeed when the higher-order method is successful. However, when stability constrains the steplength of the higher-order method to the boundary where $R(H_L) = 1$ then it can be expected that the lower-order method will also succeed due to the reduced steplength.

For the explicit methods considered in this paper, we have developed a set of fully embedded formulae of orders 1–3. We have already seen in equations (22) and (23) that a local error estimation for the high-order method is available simply by taking the difference of the 2nd- and 3rd-order solutions. What is now required is an appropriate local error estimate for the embedded 1st-order method which uses a steplength of c_2h_n (or $0.5\,h_n$).

Using the Butcher block (20), we can see that a 1st-order ($p = 1$) Euler method with steplength c_2h_n is given by:

$$y^{\mathrm{E}}(t_n + 0.5\,h_n) = y_n + h_n(0.5)f(t_n, y_n) \tag{29}$$

and we can develop a 2nd-order ($p = 2$) Heun method for local error estimation given by:

$$\begin{aligned} y^{\mathrm{H}}(t_n + 0.5\,h_n) &= y_n \\ &+ \frac{h_n}{2}(0.5)f(t_n, y_n) + \frac{h_n}{2}(0.5)[f(t_n) \\ &+ (0.5)h_n, y_n + h_n(0.5)f(y_n, t_n)]. \end{aligned} \tag{30}$$

The local error estimator (E_1) for the 1st-order method is then given by

$$\begin{aligned} E_1 &= y^{\mathrm{H}}(t_n + 0.5\,h_n) - y^{\mathrm{E}}(t_n + 0.5\,h_n) + 0(h_n^3) \\ &= \frac{0.5\,h_n}{2}\{f[t_n + 0.5\,h_n, y_n \\ &+ h_n\,0.5\,f(y_n, t_n)] - f(y_n, t_n)\} \\ &+ 0(h_n^3). \end{aligned} \tag{31}$$

This local error estimator for the embedded 1st-order method can be used in conjunction with the higher-order local error estimator to detect the presence of the equilibrium state as previously outlined.

Figure 2 shows the stability regions for the explicit methods under consideration. Under conditions

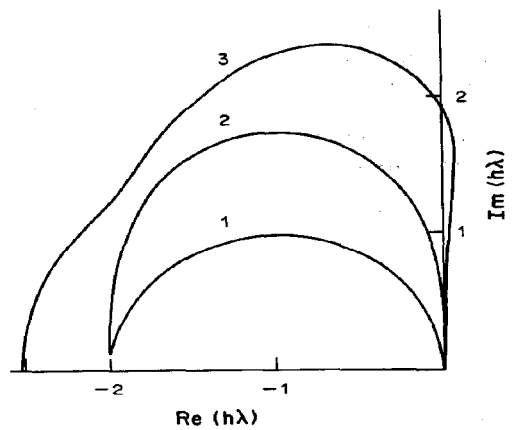

Fig. 2. Stability regions for explicit RK methods.

where the 3rd-order method is not constrained by stability considerations, the value of $h\lambda_{max}$ ($\lambda \in C$) will lie within the 3rd-order boundary. However, as $h\lambda$ increases, the equilibrium state will be reached where $R(H_L) = R(h\lambda) = 1$ and the value of $h\lambda$ will lie on or near the boundary. At this point, the problem has become computationally stiff. This may be due to an increase in h and/or λ which is dependent on the accuracy (order) of the method and the eigenvalues of the problem itself.

Once the value of $h\lambda$ reaches the equilibrium state, accuracy requirements (as set by the user) are no longer controlling steplength selection. A consequence of this is that a larger step could be taken if stability were not a problem. As well, the algorithm is actually achieving more solution accuracy than required, so much so, that the embedded 1st-order method normally achieves the user set local error tolerance. It is this phenomenon which is used as a basis for detecting the equilibrium state, thus indicating a switch to an implicit Runge–Kutta method which includes all the left-hand $h\lambda$-plane in its stability region.

We can formalize this detection mechanism by noting that we can generate a local error estimator (E_2) for the higher-order method using the embedded formulae of orders 2 and 3. This is given by:

$$E_2 = \| \mathbf{y}^{[2]} - \mathbf{y}^{[3]} \| \tag{32}$$

where $\mathbf{y}^{[2]}$ and $\mathbf{y}^{[3]}$ are the solution vectors for the formulae of orders 2 and 3 respectively.

For acceptance of the current step we require that the local error estimator be such that:

$$E_2 \leqslant \Gamma, \tag{33}$$

where Γ is the acceptable local error tolerance. This tolerance within the algorithm is a combination of relative and absolute error on each variable given by:

$$\Gamma = \tau |y_i| + \tau, \tag{34}$$

where τ is the user set tolerance. In this case, each component y_i of the solution is checked.

Steplength adjustment is then made on the basis of the standard adjustment formula

$$h_{new} = 0.9\, h_{old} \left(\frac{\Gamma}{E_2} \right)^{1/3}. \tag{35}$$

Under circumstances where the 2nd-order method is using the maximum steplength possible, based on accuracy considerations we can expect that:

$$E_2 \leqslant \Gamma \leqslant E_1, \tag{36}$$

where E_1 is the 1st-order error estimator. However, when stability controls the steplength selection, then it is usually the case that:

$$E_2 \leqslant E_1 \leqslant \Gamma, \tag{37}$$

implying that both the high- and low-order methods are succeeding and that the steplength is significantly constrained by stability considerations.

The condition given by (37) must be clearly established before any switch is made to an implicit method. To guard against a premature switch to the DIRK method, with subsequent increase in computational work per step, a criterion of 25 1st-order successes in a block of 50 consecutive steps has been used. Although this is based on numerical experience, the evidence suggests that it is a good compromise between premature switching and delayed switching, both of which increase the overall computational effort. This becomes more important as the size of the problem increases.

In order to test how effective the equilibrium detection can be, several problems have been used. The first problem is a simple damped oscillator example given by the following equations:

$$y_1' = y_2, \qquad y_1(0) = 10.0$$
$$y_2' = -k_1 y_2 - k_2 y_1, \qquad y_2(0) = 0.0$$

whose eigenvalues are:

$$\lambda_{1,2} = -0.5\, k_1 \pm [(k_1^2 - 4k_2)/4]^{0.5}.$$

For a local error tolerance of $1.E-3$ the performance of the equilibrium state detection mechanism was checked by varying the eigenvalues with respect to both magnitude and distribution. Table 2 gives the results of such tests and compares the predicted equilibrium state steplength (h_e) with the theoretical value (h_t) based on $(h\lambda)_{max}$ computed from the stability boundary. In this case, we monitor a block of 50 steps. If there are 25 or more 1st-order method successes then the problem is signalled as being stiff. The value t_{sw} represents the time at which the switch was signalled.

From the results in Table 2 it can be seen that the stiffness detection mechanism works well. Examination of the steplength selection history shows that the steplengths oscillate around the equilibrium steplength given by $R(H_L)$. Recently Hall (1985) showed that when using local extrapolation of the 2nd-order method, the equilibrium state is unstable. In the above tests, local extrapolation was used to improve the global error of the solution and this practice has

Table 2. Equilibrium state prediction using the explicit method—Damped Oscillator Problem

Case	k_1	k_2	$\lambda = a + bi$	$\lvert\lambda\rvert$	h_e	h_t	t_{sw}
1	200	1×10^4	$-100 + 0i$	100	2.5×10^{-2}	2.1×10^{-2}	0.98
2	200	2×10^4	$-100 + 100i$	141	1.7×10^{-2}	1.5×10^{-2}	0.54
3	200	5×10^4	$-100 + 200i$	224	1.1×10^{-3}	1.0×10^{-3}	0.36
4	200	1×10^5	$-100 + 300i$	316	8.0×10^{-3}	7.0×10^{-3}	0.25
5	200	1×10^6	$-100 + 995i$	1000	2.2×10^{-3}	2.0×10^{-3}	0.17

been adopted in the rest of the algorithm development, even though some instability in the detection process is encountered.

It appears that the equilibrium state instability does not seriously affect the detection of stiffness. The results also show that the test works well both for purely real eigenvalues as well as those with significant imaginary parts. The stability boundary of the 3rd-order method is quite large near the imaginary axis, so there is no serious deterioration in performance on highly oscillatory problems which would normally require a drastically reduced steplength in order to follow the trajectory.

Switching between methods. An essential decision to be taken in the implementation of adaptive RK methods is related to the switching between methods. Switching can occur in both directions, although numerical experience shows the majority of problems only switch from the explicit to implicit method during the course of the solution. Very few problems continue to switch back and forth between the two basic formulae. Problems with cyclic solution trajectories are amongst this class of problem.

Explicit to implicit switching. The algorithm is always started with the explicit Runge–Kutta method due to the fact that the majority of problems are nonstiff in the initial transient region. A switch is made to the implicit DIRK method only after the equilibrium state of the explicit method has been signalled. This is indicated when out of 50 consecutive steps, there have been at least 25 1st-order method successes as previously outlined.

There are other circumstances in which continuation of the explicit method can be justified. The algorithm can reach a value of the independent variable t which is "close" to the termination value of $t = b$ and be using an explicit steplength h_e, having just indicated that the problem is computationally stiff. Consideration should be given as to whether a switch to the implicit method is worthwhile, since significantly more work per step is demanded. This is especially the case with large problems.

Experience suggests that once in the steady state region of the solution, somewhere between 2 and 4 steps of increasing size of the DIRK method are used to reach the final value of $t = b$. The computational effort using either an explicit or implicit formula is proportional to the number of function evaluations or calls to evaluate the right-hand-sides of the equation set. Given that the explicit method uses three function evaluations per step, the work required by the explicit method to complete the integration is approximated by:

$$W_{\mathrm{ERK}} \simeq 3m, \tag{38}$$

where $m = (b - t_e)/h_e$ is the number of steps required at the equilibrium steplength h_e to complete the integration. This computational work estimate is conservative since the acceptable steplength to be used may decrease due to stiffness considerations.

Based on the observation from a large number of test problems that the DIRK code usually requires about three steps to complete the integration to $t = b$, with step increases and Jacobian evaluations [$(n + 1)$ calls per Jacobian evaluation] at each step, we can approximate the work involved as:

$$W_{\mathrm{DIRK}} \simeq 3(n + 1) + 18, \tag{39}$$

where each stage calculation requires an average of two function evaluations for convergence. This is for dense matrix operations.

Hence an estimate of the number of steps one could take with the explicit method, which demands a similar amount of computational effort as the implicit method, is $(n + 7)$. It should be emphasized that the foregoing procedure is only applicable in the final steady-state region of the solution. Some problems exhibit pseudo-steady states between transient regions (e.g. diurnal kinetics and van der Pohl's equation). The mechanism is not activated under these circumstances. If the number of steps $(n + 7)$ for the explicit method is exceeded then a switch is made to the DIRK formula.

The case of sparse matrix techniques is less clear, especially when sparsity is being assessed numerically rather than stated explicitly.

Implicit to explicit switching. Since we are using an implicit DIRK method, the Jacobian is readily available. This means that an estimate or a bound of the maximum eigenvalue of the system equations can be inferred or calculated from the partial derivative information. Several methods can be used, including a Gerschgorin bound and the Power method to estimate or bound $\lambda_{\max}$. Using the matrix norm $\|J\|_\infty$, we can obtain an upper bound to $\lambda_{\max}$. In most circumstances this can be a conservative value of $\lambda_{\max}$. This in turn means that a possible switch back to the explicit code is delayed since we require that

$$|h\lambda_{\max}| \leqslant \eta H_{\mathrm{L}}, \tag{40}$$

where H_{L} is the lower stability limit for our explicit RK method and η is a fraction (0,1). We do not want to switch back to the explicit RK method if $|h\lambda_{\max}| \simeq H_{\mathrm{L}}$ as this would mean that the explicit method is at its equilibrium state again. A switch is only made when $|h\lambda_{\max}| < 0.5\, H_{\mathrm{L}}$, ensuring that the explicit method is working well within its stability region. We assume that $H_{\mathrm{L}} \simeq 2.5$ for a distribution of eigenvalues. The algorithm does contain a Power method for computing the maximum eigenvalue. However, this method can fail and is sometimes prone to slow convergence. Experience suggests that the matrix norm $\|J\|_\infty$, is adequate as an estimate or bound on $\lambda_{\max}$.

Interpolation of off-mesh points. The ability to interpolate to off-mesh points is made easy by ensuring that stage evaluations are carried out at the end points of the current step. With this in mind,

both explicit and implicit methods require that the quadrature point c_3 be equal to 1.0.

Using the function values f_n, f_{n+1} and their derivatives f'_n, f'_{n+1} at the extremes of the current step $h_n = t_{n+1} - t_n$, we can write the Hermite polynomial $H(t)$ as:

$$H(t) = [1 - 2h'_1(t_n)(t - t_n)]h_1(t)^2 f_n + [1 - 2h'_2(t_{n+1})(t - t_{n+1})]h_2(t)^2 f_{n+1} + (t - t_n)h_1(t)^2 f'_n + (t - t_{n+1})h_2(t)^2 f'_{n+1}, \quad (41)$$

where

$$h_1(t) = -(t - t_{n+1})/h_n, \quad h_2(t) = (t - t_n)/h_n$$

$$h'_1(t) = -1/h_n, \quad h'_2(t) = 1/h_n.$$

Normalizing the polynomial $H(t)$ using:

$$v = (t_i - t_n)/h_n \quad (v:0 \rightarrow 1) \quad (42)$$

and substituting into the expression for $H(t)$ gives the final interpolation formula:

$$H(v) = (1 + 2v)(v - 1)^2 f_n + (3 - 2v)v^2 f_{n+1} + v(v - 1)^2(h_n f'_n) + (v - 1)v^2(h'_n f'_{n+1}). \quad (43)$$

Using the above polynomial, we can interpolate to an off-mesh point, maintaining the same accuracy as the basic numerical routine. Where many off-mesh points are required, significant savings in computational effort can be made.

NUMERICAL EXPERIENCE

The previously developed algorithm was tested on some examples taken from the review paper of Chan *et al.* (1978). As well, the Van der Pohl oscillator problem was included so as to test the algorithm on a problem which constantly changes in stability characteristics and a distillation dynamics problem has also been included. The problems indicate the effectiveness of using the adaptive algorithm, known as VMRK32 (variable method RK 3rd/2nd-order) and also compares this code to the DIRK32 stiff method.

The examples used were:

System 1—ozone decomposition:

$$y'_1 = -y_1 - y_1 y_2 + \epsilon y_2 k; \quad y_1(0) = 1$$

$$\epsilon y'_2 = y_1 - y_1 y_2 - \epsilon y_2 k; \quad y_2(0) = 0$$

$$\epsilon = 1/98, \quad k = 3.0,$$

$$t \in (0, 1000).$$

System 2—fluidized bed:

$$y'_1 = 1.3(y_3 - y_1) + 1.04 \times 10^4 k y_2; \quad y_1(0) = 759.167$$

$$y'_2 = 1.88 \times 10^3[y_4 - y_2(1 + k)]; \quad y_2(0) = 0$$

$$y'_3 = 1752 - 269y_3 + 297y_1; \quad y_3(0) = 600$$

$$y'_4 = 0.1 + 320y_2 - 321y_4; \quad y_4(0) = 0.1$$

$$k = 0.0006 \exp(20.7 - 15000/y_1),$$

$$t \in (0, 1000)$$

System 3—Belousov reaction:

$$y'_1 = 77.27(y_2 - y_1 y_2 + y_1 - 8.375 \times 10^{-6} y_1^2); \quad y_1(0) = 4.0$$

$$y'_2 = (-y_2 - y_1 y_2 + y_3)/77.27; \quad y_2(0) = 1.1$$

$$y'_3 = 0.161(y_1 - y_3); \quad y_3(0) = 4.0$$

$$t \in (0, 100).$$

System 4—Van der Pohl's problem:

$$y'_1 = y_2; \quad y_1(0) = 2.0$$

$$y'_2 = \eta(1 - y_1^2)y_2 - y_1; \quad y_2(0) = 0.0, \quad \eta = 100,$$

$$t \in (0, 550).$$

This system is characterized by an oscillatory response, which for component 1 switches between +2.0 and −2.0. During the transient phases, the problem is essentially nonstiff, with the steplength being controlled by accuracy considerations. On entry to and exit from this phase, the problem is stiff and requires an appropriate stiff method. Hence, the problem provides a reasonable test of the switching ability of the algorithm.

System 5—distillation dynamics:

The system consists of a 7-plate column separating a feed of *iso*-butane, *n*-butane and *iso*-pentane. This is shown in Fig. 3. The column operates at 8.27 bar. A perturbation is made in the reboiler duty (Q_R) from 410 to 322 kW and the transient behaviour of the column is observed. The problem consists of 21 differential equations.

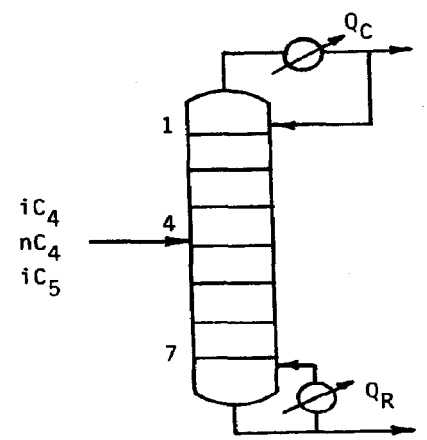

Fig. 3. Three-component distillation system.

Table 3. Solution statistics using VMRK32 (eps = 10^{-3})

System	NS	NFE	NJE	h_e	t_{sw}	λ_{sw}	h_t	Global error
1	95	397	14	$5.04E-2$	0.447[a]	50	$5.00E-2$	$1.8E-4$
2	64	388	25	$1.89E-3$	0.022	2200	$1.20E-3$	$2.4E-5$
3	182	1089	42	$1.41E-3$	1.230	1500	$1.60E-3$	$1.1E-3$

[a]Code switched back to ERK32 at $t = 1.07$, then to DIRK32 at $t = 20.6$.

Solution statistics using DIRK32 (eps = 10^{-3})

System	NS	NFE	NJE	Global error
1	64	431	28	$3.7E-4$
2	51	400	32	$2.4E-5$
3	155	1380	72	$4.6E-4$

Table 4. System 2 solution statistics using VMRK32

Error tolerance (eps)	NS	NFE	NJE	h_e	t_{sw}	λ_{sw}
10^{-3}	64	388	25	$1.89E-3$	0.022	2200
10^{-4}	105	613	28	$8.52E-4$	0.039	2200
10^{-5}	210	1132	31	$1.07E-4$	0.062	2200

System 2 solution statistics using DIRK32

Error tolerance (eps)	NS	NFE	NJE
10^{-3}	51	400	32
10^{-4}	86	686	41
10^{-5}	167	1224	50

The numerical results using the adaptive algorithm are given in Tables 3–7. The following abbreviations have been used:

VMRK32 = variable method RK orders 3 and 2,
DIRK32 = diagonally implicit RK method of orders 3 and 2,
ERK32 = explicit RK method of orders 3 and 2,

(the last two codes make up the adaptive algorithm).

NS = number of steps taken,
NFE = number of function evaluations (including numerically generated Jacobians). i.e. number of calls to the right-hand-side of the equation,
NJE = number of Jacobian evaluations,
h_e = equilibrium state steplength,
h_t = theoretical steplength for switching, based on the eigenvalues,
t_{sw} = time that switching occurred,
λ_{sw} = estimate of maximum eigenvalue at t_{sw},
eps = error/step user set tolerance.

Table 5. Van der Pohl problem (eps = 10^{-3})

Code	NS	NFE	NJE	Global error[b]
DIRK32	631	6019	319	$1.04E-3$
VMRK32	830	5221	257	$3.90E-3$
VMRK32[a]	706	4665	237	$2.46E-3$

[a]Using additional criterion of five successful, consecutive steps of ERK32 to signal stiffness.
[b]Global error at $t = 550$.

Table 6. Van der Pohl problem using VMRK32 (eps = 10^{-3})

t	NS	NFE	NJE
	ERK32 to DIRK32		
0.19	25	90	0
86.76	159	970	30
162.76	285	1823	77
244.04	412	2642	129
325.32	557	3458	180
406.75	689	4490	223
487.99	820	5768	250
	DIRK32 to ERK32		
80.97	49	410	30
162.59	253	1771	77
243.53	310	2267	129
324.82	445	3068	180
406.50	649	4344	223
487.52	709	4779	250

Table 7. Distillation dynamics (eps = 10^{-4})

t (h)	NS	NFE	NJE	h (used)
	Code = DIRK32			
0.18	34	687	23	0.332
0.36	40	723	23	0.332
0.52	43	762	24	0.506
0.84	48	813	25	0.760
1.50	53	884	27	2.044
	Code = VMRK32			
0.037	28	95	—	0.025
0.18	41	327	7	0.238
0.36	47	384	8	0.362
0.52	50	429	9	0.540
0.84	55	474	10	0.850
1.50	60	545	12	2.039

In all cases the error/step criterion was a mixed relative (for $y_i \geqslant 1$) absolute (for $y_i < 1$) error criterion.

Global errors are computed using a "true" solution generated by a high-accuracy solution (eps $= 10^{-6}$) and are quoted as relative errors.

Table 3 shows a comparison between the adaptive algorithm and the use of the stiff solver on the first three test problems. In these cases the adaptive algorithm, although using more steps in each case, reduces the overall number of function evaluations by eliminating Jacobian evaluations within the initial transient phase. The VMRK32 code also detects stiffness correctly in the case of Systems 2 and 3 and switches to the DIRK32 code in order to integrate the stiff region. In the case of System 1, the maximum eigenvalue drops from 100 at $t = 0.0$ to 4 at $t = 10$. Because of this, the code switches back to the explicit code at $t = 1.07$ where the combination of h and λ is less than $0.5\,H_L$. Integration is continued up to $t = 20.6$ (an extra 26 steps) when a further change is needed to the stiff solver.

In the case of System 1 the switch back to ERK32 is not advantageous in minimizing overall work. As well, the ERK32 code needs to take at least 25 steps to test whether the problem is computationally stiff. A modification of this rule is discussed later on with reference to Van der Pohl's problem.

It should be emphasized that the results are based on Jacobian elements being generated numerically. This we believe to be the most general approach, although, when analytic derivatives are available, the work involved in using DIRK32 can be significantly reduced.

For System 2 (fluidized bed) the difference in computational effort as measured by NFE is not really significant. In this case, the switch from ERK32 to DIRK32 occurs quite early in the simulation due to the magnitude of the maximum eigenvalue (~ 2200). As the solution proceeds, the maximum eigenvalue increases to 6550 at $t = 1000$, and thus $|h\lambda_{max}| \gg R(H_L)$ for the explicit method. For a problem that is essentially stiff throughout the whole of the solution domain, this behaviour is to be expected.

In the case of System 3, the maximum eigenvalue is ~ 8.0 at $t = 0.0$ and remains at that order until approx. $t = 0.4$. Beyond $t = 0.4$, $|\lambda_{max}|$ increases to approx. 1×10^5 at $t = 10^3$. The difference in performance of VMRK32 compared to DIRK32 is due to the fact that the problem is computationally nonstiff up to $t = 1.2$ and a significant saving can be made in Jacobian evaluations.

In the case of a problem which is initially stiff, the ERK32 code requires at least 25 steps in order to signal that it is operating at the equilibrium state. In these cases, the DIRK32 code would prove more efficient. The codes being used in the VMRK32 algorithm choose the initial steplength automatically, this being based on initial derivative information. These initial steplengths are slightly pessimistic in order to avoid early error failures.

Table 4 illustrates the effect of the user-set local error tolerance (eps) on the behaviour of the VMRK32 code as well as the comparison with the stiff DIRK32 code. System 2 (fluidized bed) was used for these runs. It can be seen that the time at which switching occurs (t_{sw}) is extended as the error per step (eps) is decreased. Hence, the nonstiff region is extended in each case due to the accuracy demanded in the solution. Differences in performance between VMRK32 and DIRK32 are more pronounced for the tighter tolerances. This is due mainly to the increased number of Jacobian evaluations used by the DIRK32 code in the nonstiff region.

Table 5 gives details of code performance on the Van der Pohl problem at eps $= 10^{-3}$. Here the savings are quite significant when using the adaptive code, due to the number of nonstiff regions within the solution domain. One of the problems alluded to in the previous discussion is the fact that the equilibrium state detection used a block of 50 steps and assessed whether there were at least 25 successful steps using the 1st-order method. It can be advantageous to augment the previously mentioned test with a further condition. This extra condition checks whether 5 consecutive steps have resulted in an equal number of 1st-order method successes. If so, the equilibrium state is signalled and a switch is made.

The results of adding this extra condition leads to a significant reduction in work as shown in Table 5. This criterion however, appears to be useful in the situations represented by Van der Pohl's equation or by problems which are initially computationally stiff. On other classes of problems it can lead to premature switching to the implicit method with a resultant increase in work due to extra Jacobian evaluations as the implicit code changes steplength.

Table 6 illustrates how the adaptive code switches between ERK32 and DIRK32. It can be seen from the table that during the slowly changing periods the adaptive algorithm chooses the stiff solver and then switches back to ERK32 during the integration of the steep transient sections. The results show the effectiveness of having an adaptive algorithm which uses methods appropriate to the stability characteristics of the problem.

The results in Table 6 give an indication of the significant savings to be gained by using the adaptive algorithm. The problem is stiff after only a short period of time ($t = 0.037$), when the explicit code switches over to the diagonally implicit formula. The number of steps and the final steplength are comparable for both runs. However the significant reduction in function evaluations is due to the greatly decreased use of Jacobians during the initial, nonstiff transient phase.

The final results shown in Table 7 relate to a problem of significant size and complexity. The results show that in using an adaptive RK algorithm,

substantial savings can be made in function evaluations and hence computation time. The switch to the DIRK method occurred at $t = 0.037$, this being quite early in the solution procedure and certainly nowhere near the steady-state condition. In this case, the majority of the computational savings again comes from avoiding Jacobian evaluations which occur in the computationally nonstiff region. Because of the inherent nonlinearity of the problem, it is not feasible to choose *a priori* the "best" algorithm. However, the adaptive type algorithms provide the optimal solution to such a complex problem.

CONCLUSIONS AND RECOMMENDATIONS

It has been shown that an effective, adaptive single-step algorithm based on Runge–Kutta formulae can be constructed. The benefits to the general user can be significant in terms of reduced computational effort and the added advantage of not needing to assess the stability of the problem *a priori*.

The assessment of the equilibrium state of the explicit code using the approach of Shampine (1977) has been vindicated and has proved useful in practice. Some changes can be justified for certain classes of problems where significant switching occurs between the methods.

Although this paper has concentrated on dense matrix operations and rather small-scale problems, current work on the sparse matrix version of the above algorithm shows that the benefits of an adaptive algorithm are also significant when applied to large-scale dynamic simulation systems, e.g. distillation systems.

Further work on the algorithm as applied to large-scale problems will be reported in the future.

NOMENCLATURE

a = Lower limit of independent variable t
a_{ij} = Runge–Kutta coefficient
$\mathbb{A}$ = Matrix of a_{ij} values
b = Upper limit of independent variable t
b_i = Runge–Kutta quadrature weights
$\mathbf{b}^T$ = Vector of weights
$\mathbb{B}$ = Matrix of constant coefficients
$B(\ )$ = Butcher Series
c_i = Runge–Kutta quadrature points
$\mathbf{c}$ = Vector of quadrature points
d_i = Runge–Kutta quadrature weights
$\mathbf{d}$ = Vector of quadrature weights
$e(\tau)$ = Truncation error weights
eps = Error per step
E_p = Maximum error constant of order p method
$F(\ )$ = Elementary differentials of the Butcher series
$\mathbf{f}$ = Right-hand-side functions of ordinary differential equations
$f^{[n]}$ = nth order derivative of f
$f(\tau)$ = Elementary differential
$\mathbf{g}$ = Algebraic functions of the procedures
h_e = Equilibrium state steplength
h_i = Numerically indicated equilibrium steplength
h_n = nth steplength
h_t = Theoretical equilibrium steplength
$H(t)$ = Hermite polynomial in t
H_L = Lower stability limit for explicit Runge–Kutta methods
H_n = $h\lambda$
$\mathbb{J}$ = Jacobian matrix
NFE = Number of function evaluations
NJE = Number of Jacobian evaluations
NS = Number of steps taken
$\mathbf{P}$ = Vector of procedures
p = Order of a Runge–Kutta method
R^n = n-dimensional real space
Re( ) = Real part
$R(H_n)$ = Stability function for the Runge–Kutta method
s = Number of Runge–Kutta stages
t = Independent variable, time
t_e = Equilibrium state time
$t_{n,i}$ = nth time step, ith stage value
t_{sw} = Switching time of code
T = Set of all rooted trees
W_{ERK} = Work estimate for explicit Runge–Kutta method
W_{DIRK} = Work estimate for diagonally implicit Runge–Kutta method
y_n = Dependent variable at time t_n
$\mathbf{y}$ = Vector of dependent variables
$y_{n,i}$ = Computed ith stage value of $\mathbf{y}$ on step n
$\mathbf{z}$ = Vector of algebraic variables
y^E = Euler method solution for steplength c_2h_n
y^H = Heun method solution for steplength c_2h_n

Greek letters

ξ = Function of the Runge–Kutta coefficients
λ_i = ith eigenvalue of $\mathbb{J}$
λ_{sw} = Eigenvalue at switching time t_{sw}
ρ = Order of the rooted tree
τ = Rooted tree
Γ = Function of rooted tree order
v = Nondimensional variable in Hermite polynomial

REFERENCES

Alexander R., Diagonally implicit Runge–Kutta methods for stiff ODEs. *SIAM Jl numer. Analysis* **14,** 1006–1022 (1977).

Burrage K., A special family of RK methods for solving stiff systems of ODEs. *BIT* **18,** 22–41 (1978).

Burrage K. and F. Chipman, An implementation of singly-implicit RK methods. Report 19, Dept Mathematics, University of Auckland, New Zealand.

Butcher J. C., Implicit Runge–Kutta methods. *Math Comput.* **18,** 50 (1964).

Butcher J. C., On the implementation of implicit Runge–Kutta methods. *BIT* **16,** 237–240 (1976).

Butcher J. C., A transformed implicit Runge–Kutta method. *J. Ass. Comput. Mach.* **26,** 731–738 (1979).

Cailland J. B. and Padmanabhan, An improved semi-implicit RK method for stiff systems. *Chem. Engng J.* **2,** 227–232 (1970).

Cameron I. T., Numerical solution of differential-algebraic systems in process dynamics. Ph.D. Thesis, University of London (1981).

Cameron I. T., Solution of differential-algebraic systems using DIRK methods. *IMA Jl* **3,** 273–289 (1983).

Cameron I. T., C. A. Ruiz and R. Gani, A generalized dynamic model for distillation columns—II. Numerical and computational aspects. *Comput. chem Engng* **10,** 199–211 (1986).

Cash J. R., Diagonally implicit RK formulae with error estimates. *J. Inst. math. Applic.* **24,** 293–302 (1979).

Cash J. R. and C. B. Liem, On the design of a variable order—variable step DIRK algorithm. *J. Inst. math. Applic.* **26,** 87–91 (1980).

Chan I. Y. N., I. Birnbaum and L. Lapidus, Solution of stiff differential equations and the use of embedding techniques. *Ind. Engng Chem. Fundam.* **17**, 133–148 (1978).

Feng A., C. D. Holland and S. E. Gallum, Development and comparison of a generalized semi-implicit RK method with Gear's method for systems of coupled differential and algebraic equations. *Comput. chem Engng* **8**, 51–59 (1984).

Hairer E. and G. Wanner, On the Butcher group and general multivalue methods. *Computing* **13**, 1–15 (1974).

Hall G., Equilibrium states of Runge–Kutta schemes. *ACM Trans Math. Software* **11**, 289–301 (1985).

Norsett S. P., Semi-explicit Runge–Kutta methods. Report 6/74, Dept Mathematics, University of Trondheim (1974).

Norsett S. P. and P. G. Thomsen, *SIMPLE, a Stiff System Solver*. Dept Numerical Mathematics, NTH, Trondheim, Norway (1986).

Petzold L., Automatic selection of methods for solving stiff and non-stiff systems of ordinary differential equations. *SIAM J. Sci. Statist. Comput.* **4**, 136–148 (1983).

Shampine L. F., Stiffness and non-stiff differential equation solvers, 2: detecting stiffness with Runge–Kutta methods. *ACM Trans Math. Software* **3**, 44–53 (1977).

Shampine L. F., Stiffness and automatic selection of ODEs codes. *J. Comput. Phys.* **54**, 74–86 (1984).