Applied Numerical Mathematics 9 (1992) 1-19
North-Holland

An adaptive theta method for the solution of stiff and nonstiff differential equations

M. Berzins
School of Computer Studies, The University, Leeds LS2 9JT, UK

R.M. Furzeland
Shell Internationale Petroleum Mij B.V., P.O. Box 162, 2501 AN The Hague, Netherlands

Abstract

Berzins, M. and R.M. Furzeland, An adaptive theta method for the solution of stiff and nonstiff differential equations, Applied Numerical Mathematics 9 (1992) 1-19.

This paper describes a new adaptive method that has been developed to give improved efficiency for solving differential equations where the degree of stiffness varies during the course of the integration or is not known beforehand. The method is a modification of the theta method, in which the new adaptive strategy is to automatically select the value of theta and to switch between functional iteration and Newton iteration for the solution of the nonlinear equations arising at each integration step. The criteria for selecting theta and for switching are established by optimising the permissible step size.

The performance of the adaptive methods is demonstrated on a range of test problems including one arising from the method of lines solution of a convection-dominated partial differential equation. In some cases the new approach halves the amount of computational work.

1. Introduction

The theta method is widely used to solve initial value problems for ordinary differential equations. Examples of applications include the method of lines treatment of partial differential equations (Hopkins [7], Berzins, Dew and Furzeland [1]) simulation of electric power systems (Johnson et al. [8]) and the modelling of gas transmission networks (Chua and Dew [4]). The theta method can be applied to both nonstiff and stiff ordinary differential equations (ODEs) depending on the value of theta chosen. The method is an implicit one for all nonzero values of theta, which requires that a system of nonlinear equations be solved at each time step. In order to take reasonably sized steps when solving a stiff problem the convergence criterion requires that a Newton-type iteration be used. The Jacobian evaluations required for the Newton iteration are expensive and often unnecessary for the solution of nonstiff problems where the step size restrictions are less severe and a much cheaper iteration method such as functional iteration can be used instead. An obvious requirement is a single algorithm that can handle both stiff and nonstiff problems by automatically selecting the best value of theta and the most efficient iteration method with respect to step size and cost per step. The theory behind a theta method algorithm which changes iteration methods is discussed by Shampine [15,16] and has also been used by Norsett and Thomsen [11] and by Berzins and Furzeland [2]. There have been a number of attempts, many associated with the problems arising from the electric power industry, to find the best value of theta so that the integration is as accurate and efficient as possible (see Johnson et al. [8]). Many of these attempts attempt to optimally fit theta to a linearisation of the ordinary differential equations being solved. In this paper we shall derive a fully adaptive theta method integrator by directly choosing theta so as to try to minimise the number of time steps, without linearising the system of ordinary differential equations and combine this with the algorithm for switching between iteration methods.

The main aims of the paper are to provide a description of the theta method which is sufficient to allow a software implementation. This description will indicate how theta can be chosen and how the iteration method can be changed adaptively by using the (unpublished) algorithm of Berzins and Furzeland [2]. In Section 2 we describe the numerical methods used as a basis for the new method and in Section 3 we put forward a basic strategy for an adaptive method. In Section 4 we present the new integration method based on the automatic selection of theta and on the switching from Newton to functional iteration, or vice versa. We give criteria for deciding when to change the value of theta and to switch the iteration method according to the error estimates, step sizes and cost per step that the different methods would use. Section 5 deals with the implementation details of the new method. Numerical results are presented in Section 6 and these results illustrate the efficiency of the strategy both on standard test problems and on problems arising from the method of lines treatment of partial differential equations.

2. Numerical methods

This paper is concerned with numerical methods for the solution of solving the system of ordinary differential equations (ODEs):

$$\dot{y} = f(x, y), \quad a \leqslant x \leqslant b, \qquad y(a) \text{ given.} \tag{1}$$

For the rest of this paper we shall not explicitly use vector notation but will consider the scalar form of equation (1).

2.1. The theta method

The theta method is defined as follows. Suppose we have the approximation y_n to $y(x_n)$, and the approximation y_n' to $y'(x_n)$, then the numerical solution at x_{n+1}, where $x_{n+1} = x_n + h$ and h is the step size, is denoted by y_{n+1} and is defined by

$$y_{n+1} = y_n + (1 - \Theta) h y_n' + \Theta h f(x_{n+1}, y_{n+1}). \tag{2}$$

The theta method is S-stable (Prothero and Robinson [13]) for values of $\Theta > 0.5$ but for values of $\Theta < 0.5$ the theta method is only conditionally stable. This formula is implicit in y_{n+1}. In order to calculate y_{n+1} it is necessary to choose values for h and Θ, to select an iteration method and to have a predictor for the iteration method. The choice of iteration method will be discussed first.

2.2. Functional iteration and newton iteration

A simple way to solve the implicit equation (2) is to generate an initial predictor $y_{n+1}^{(0)}$ (the precise form of this will be discussed below) and to generate successive correctors from the *functional iteration* formula

$$y_{n+1}^{(m+1)} = y_n + (1-\Theta)hy_n' + h\Theta f\left(x_{n+1}, y_{n+1}^{(m)}\right). \tag{3}$$

In order to determine whether or not functional iteration converges we expand the function f using Taylor's series to get:

$$f(x, u) - f(x, v) = (u-v)\cdot J,$$

where J is the Jacobian matrix $\{\partial f_i/\partial y_j\}$ evaluated at x and an unknown point y between u and v, then

$$L = \text{Sup}\,\|J\|$$

is a suitable Lipschitz constant for some matrix norm.

On subtracting (3) from (2) and using the Lipschitz condition, we get:

$$\left\| y_{n+1} - y_{n+1}^{(m+1)} \right\| \leqslant hL\Theta \left\| y_{n+1} - y_{n+1}^{(m)} \right\|. \tag{4}$$

This says that if $hL\Theta < r$ then functional iteration converges with a rate of convergence r where r must be < 1.

For the solution of stiff problems, we are, on efficiency grounds, forced to use step size h with $hL\Theta \gg 1$. It is therefore necessary to use a different iteration. The standard one is the simplified *Newton iteration*, which uses an approximate Jacobian matrix J and an iteration matrix W_n defined by

$$J = \frac{\partial f}{\partial y}\left(x_{n+1}, y_{n+1}^{(0)}\right), \qquad W_n + (I - h_n\Theta J) \tag{5}$$

in solving a linear system of equations given by

$$W_n\left(y_{n+1}^{(m+1)} - y_{n+1}^{(m)}\right) = -y_{n+1}^{(m)} + \Theta hf\left(x_{n+1}, y_{n+1}^{(m)}\right) + y_n + h(1-\Theta)y_n' \tag{6}$$

for the correction $(u_{n+1}^{(m+1)} - y_{n+1}^{(m)})$ at each iteration. In comparison with functional iteration, this is computationally expensive since the matrix J must be formed, usually by costly and perhaps inaccurate numerical differencing, and the iteration matrix W_n must be decomposed into triangular factors and a backsubstitution performed on each iteration. However, in comparison with functional iteration, the Newton method may allow larger step sizes h to be used for stiff problems.

2.3. Prediction of iteration values

The accurate prediction of the initial values for the Newton or functional iteration plays an important role in the algorithmic efficiency of the method. The simplest predictor is based on the explicit formula

$$y_{n+1}^{(0)} = y_n + hy_n', \tag{7}$$

where from (2) the derivative y_n' is estimated for $n > 1$ by

$$y_n' = \frac{y_n - y_{n-1} - (1-\Theta)hy_{n-1}'}{\Theta h}. \tag{8}$$

This predict⋅ı has an error of $h_n^2 y_n''$ and although it provides acceptable values it is possible to compute a predicted value closer to that of the theta method by using

$$y_{n+1}^{(0)} = y_n + h_n y_n' + \frac{h_n^2}{h_{n-1}} y_n' - \frac{h_n^2}{h_{n-1}^2}(y_n - y_{n-1}), \tag{9}$$

the two rightmost terms of which form an approximation to the second derivative of the solution. The accuracy of this predictor can be found by noting that the values on the previous step satisfy

$$y_n - y_{n-1} = (1-\Theta)h_{n-1}y_{n+1}' + \Theta h_{n-1}y_n'.$$

Using this equation to substitute for $y_n - y_{n-1}$ in the right-hand side of the predictor equation and collecting terms together enables equation (9) to be rewritten as

$$y_{n+1}^{(0)} = y_n + h_n y_n' + (1-\Theta)h_n^2 \frac{(y_n' - y_{n-1}')}{h_{n-1}}.$$

Assuming that $(y_n' - y_{n-1}')/h_{n-1}$ is an $O(h_{n-1})$ approximation to y_n'' and that

$$y_{n+1} = y_n + h_n y_n' + \tfrac{1}{2}h_n^2 y_n'' + O(h_n^3),$$

the error, $y_{n+1}^0 - y_{n+1}$, in the predictor defined by equation (9) can then be seen to have a form $(\frac{1}{2} - \Theta)h_n^2 y_n''$ that is close in magnitude to the local error of the theta method, see equation (17) below. Alternatively, multiplying the two rightmost terms in equation (9) by $\Theta/(1-\Theta)$ would give a predictor with exactly that property, within $O(h_n^3)$.

In the case when a Newton iteration is used, Prothero and Robinson derived a still better predictor which is used in their code (see Hopkins [7]) and in the codes of Chua and Dew [4] and Berzins and Furzeland [2]. The following derivation shows that the predictor is closely related to the underlying method. Suppose that the value to be predicted is to satisfy the theta method,

$$y_{n+1}^0 = y_n + (1-\Theta)h_n y_n' + \Theta h_n f(x_{n+1}, y_{n+\cdot}^0). \tag{10}$$

By using the Jacobian matrix J to linearise the function $f()$ this equation can be approximated by

$$W_n(y_{n+1}^0 - y_n) \approx h_n y_n' + h_n^2 \Theta f, \tag{11}$$

where y_n' has been used to substitute for $f(x_n, y_n)$. Suppose that the previous step was taken with a step size of h_{n-1} then a similar approximation gives

$$(I - h_{n-1}\Theta J)(y_n - y_{n-1}) \approx h_{n-1}y_{n-1}' + h_{n-1}^2 \Theta f_x, \tag{12}$$

where f_x and J are both evaluated at (x_n, y_n). Addition and subtraction of terms gives

$$W_n(y_n - y_{n-1}) = h_{n-1}y_{n-1}' + \Theta(h_{n-1} - h_n)J(y_n - y_{n-1}) + h_{n-1}^2 \Theta f_x. \tag{13}$$

Divide (11) by h_n and divide (13) by h_{n-1} and subtract the second equation from the first to get

$$W_n\left(\frac{y_{n+1}^0 - y_n}{h_n} - \frac{y_n - y_{n-1}}{h_{n-1}}\right)$$
$$\approx (y_n' - y_{n-1}') - \Theta\left(1 - \frac{h_n}{h_{n-1}}\right)J(y_n - y_{n-1}) + h_n\Theta f_x - h_{n-1}\Theta f_x$$
$$\approx (y_n' - y_{n-1}') - \Theta\left(1 - \frac{h_n}{h_{n-1}}\right)\left[J(y_n - y_{n-1}) + h_{n-1}f_x\right].$$

The right-hand terms involving J and f_x can be rewritten in terms of $f(x_n, y_n')$ and $f(x_{n-1}, y_{n-1}')$, and hence in terms of y_n' and y_{n-1}' to get

$$W_n\left(\frac{y_{n+1}^0 - y_n}{h_n} - \frac{y_n - y_{n-1}}{h_{n-1}}\right) \approx (y_n' - y_{n-1}') - \Theta\left(1 - \frac{h_n}{h_{n-1}}\right)(y_n' - y_{n-1}'). \tag{14}$$

Multiplying by h_n and inverting the matrix W_n gives the predictor used in the codes following the Prothero and Robinson approach and in the new adaptive code discussed here:

$$y_{n+1}^0 = y_n + h_n\left(\frac{y_n - y_{n-1}}{h_{n-1}}\right) + h_n\left[1 - \Theta\left(1 - \frac{h_n}{h_{n-1}}\right)\right]W_n^{-1}(y_n' - y_{n-1}'), \tag{15}$$

where the quantity $W_n^{-1}(y_n' - y_{n-1}')$ is also used in local error estimation and so is stored by the code [7].

2.4. *Choice of theta and the step size h*

The aim of most integrators for stiff ODEs is to control the local error per step so that it is less than a user-supplied tolerance. The smaller the error, the larger the time step that can be taken. The aim of the adaptive method is thus to choose h and Θ so as to optimise the time step. The local solution of the differential equation $u(x)$, about the point x_n is defined to be the solution of the following equation:

$$u' = f(x, u), \qquad u(x_n) = y_n, \tag{16}$$

and the local error τ incurred on the step to x_{n+1} is defined by

$$\tau = y_{n+1} - u(x_{n+1}).$$

It may be shown that the local error satisfies the equation

$$[I - \Theta hJ]\tau = h^2(\Theta - \tfrac{1}{2})\frac{d^2u}{dx^2}(x_n) + \tfrac{1}{2}h^3(\Theta - \tfrac{1}{3})\frac{d^3u}{dx^3}(x_n) + O(h^4). \tag{17}$$

This error estimate leads one to think that the value $\Theta = \frac{1}{2}$ would result in the most accurate theta formula. Such analysis based on just the leading term is, however, incorrect. For the simple test problem

$$y' = \lambda y, \tag{18}$$

Prothero and Robinson [13] showed that, by expanding the error in terms of $p = \lambda h$, the relative error $R(p, \Theta)$, which is defined as $\tau/u(x_{n+1})$, can be written as

$$R(p, \Theta) = -1 + e^{-p}\frac{1 + (1-\Theta)p}{1 - \Theta p} \tag{19}$$

and from this they deduced that when λ is less than 0 the value $\Theta = 0.55$ is a good compromise between a small relative error and a large value of p and hence a large step size relative to the value of λ.

In the case of λ with positive real part the relative error is again defined by equation (19) and it can be shown that

$$R(-p, 1-\Theta) = \frac{-R(p, \Theta)}{1 + R(p, \Theta)}. \tag{20}$$

We are typically interested in $|R(p, \Theta)| < 10^{-3}$ and from the Prothero and Robinson approach it follows that for λ with positive real part a suitable value of Θ is 0.45.

It is possible to choose an optimal value of Θ, say Θ_{opt}, that makes the method have no relative error and hence be exact for the scalar equation (18). Setting $R(p, \Theta)$ to zero in equation (19) and solving for theta gives

$$\Theta_{\mathrm{opt}} = \frac{1}{p} - \frac{1}{e^{p} - 1}, \tag{21}$$

which makes the method exact for the scalar equation (18). This is the basis of the approach adopted by Johnson et al. [8]. In their method values of λ must be found so that the linear equation (18) is a good approximation to the nonlinear equation (1) being solved. In general this means that λ must be estimated numerically. The approach used by Johnson et al. [8] is to approximate λ by λ^* where

$$\lambda^* = \frac{y_n' - y_{n-1}'}{h_n y_n'}. \tag{22}$$

Using the exact values of the derivatives of the solution to the scalar equation (18) then gives

$$\lambda^* = \frac{1 - e^{-h\lambda}}{h_n}. \tag{23}$$

The actual value of Θ used by a code is then given by

$$\Theta_1 = \frac{1}{h\lambda^*} - \frac{1}{e^{h\lambda^*} - 1}. \tag{24}$$

The alternative is to choose Θ to minimise the local error as defined by equation (17) or its numerical estimate. The Prothero and Robinson type codes (see Section 4.1 below) and the adaptive code described here when a Newton iteration is being used, all use the same local error estimate. This local error estimate is defined, in the case of evenly spaced time steps, by

$$\tau_{\mathrm{est}}(\Theta) = (\Theta - \tfrac{1}{2})\Delta_n + (\Theta - \Theta^2 - \tfrac{1}{6})(\Delta_n - \Delta_{n-1}), \tag{25}$$

where

$$\Delta_n = h_n W_n^{-1}(y_n' - y_{n-1}')$$

and Δ_{n-1} is similarly defined and both vectors are stored by the code and also used in the predictor defined by equation (15). In the case when functional iteration is being used with a constant step size the error estimate has the similar form

$$\tau_{\text{est}}(\Theta) = (\Theta - \tfrac{1}{2})h_n(y_n' - y_{n-1}') + (\Theta - \Theta^2 - \tfrac{1}{6})h_n(y_n' - 2y_{n-1}' + y_{n-2}'). \tag{26}$$

The time step chosen by the integrator is taken so as to try and ensure that

$$\| \tau_{\text{est}}(\Theta) \| < \text{TOL},$$

where TOL is a user-supplied tolerance. The approach taken here is to select Θ to try and minimise the local error hence allowing a larger time step to be taken. For example, in the case of one equation, choosing Θ so that τ_{est}, as defined by equation (25), is zero defines a quadratic equation in Θ given by

$$-a\Theta^2 + \Theta(1+a) - (\tfrac{1}{6}a + \tfrac{1}{2}) = 0,$$

where $a = (\Delta_n - \Delta_{n-1})/\Delta_n$. Of the two roots only one provides values of Θ in the required range for accuracy and stability

$$\Theta_2 = \tfrac{1}{2}\left(1 + \frac{1}{a} - \frac{1}{a}\sqrt{1 + \tfrac{1}{3}a^2}\right), \tag{27}$$

where $a = (\Delta_n - \Delta_{n-1})/\Delta_n$. From this equation it can be calculated that for all finite values of a, Θ_2 is approximately bounded above by 0.7887 and below by 0.2113. Thus the form of error indicator limits the range of values of Θ that need be considered. The advantage of this approach is that it is linked directly to the error estimation and step size selection of the integrator and does not rely on the scalar equation (18) being a good model for equation (1).

This new approach can be compared with that of Johnson et al. [8] on the scalar equation (18). On substituting the exact values of the time derivatives into equation (25) (or (26)) and choosing Θ so that τ_{est} is zero the value for Θ is given by equation (27) with the value of a given by $a = (1 - e^{-h\lambda})$. The constant a is thus bounded below by $-\infty$ and above by 1 for all real values of $h\lambda$. Thus from equation (27) the values of Θ_2 for the model problem are further constrained to lie between approximately 0.4226 and 0.7887. The two approximate values of Θ defined by equations (24) and (27) are compared against Θ_{opt} in Fig. 1. The accuracy of the two methods can be compared by evaluating $R(p, \Theta_1)$ and $R(p, \Theta_2)$ for a range of p values. Figure 2 provides such a comparison of the errors that arise when Θ_1 and Θ_2 are used with negative real values of $h\lambda$. The figure shows that Θ_2 gives a smaller error than Θ_1 for values of $p > -8.84$. Thus for a range of values for which the exponential needs to be approximated with accuracy, i.e., $|p| < 10$ (so that the numerical solutions decays by a factor of less than e^{10} over one step) the approach we have adopted is more accurate. For positive values of $h\lambda$ there is remarkably little difference between the errors that arise from using Θ_1 and Θ_2, especially for large positive $h\lambda$. The limit for Θ_1 as $h\lambda$ becomes large and positive being 0.418 while the corresponding value for Θ_2 is 0.4226.

The approach that we have adopted clearly can be extended to use a different value of Θ for each equation in the ODE system. This is considered by Johnson et al. [8] but rejected on the grounds that it will not work well unless the coupling between different equations is not tight. Such a situation may however arise in the method of lines solution of PDEs in which the solution has markedly different characteristics in different spatial regions. In this study though we shall

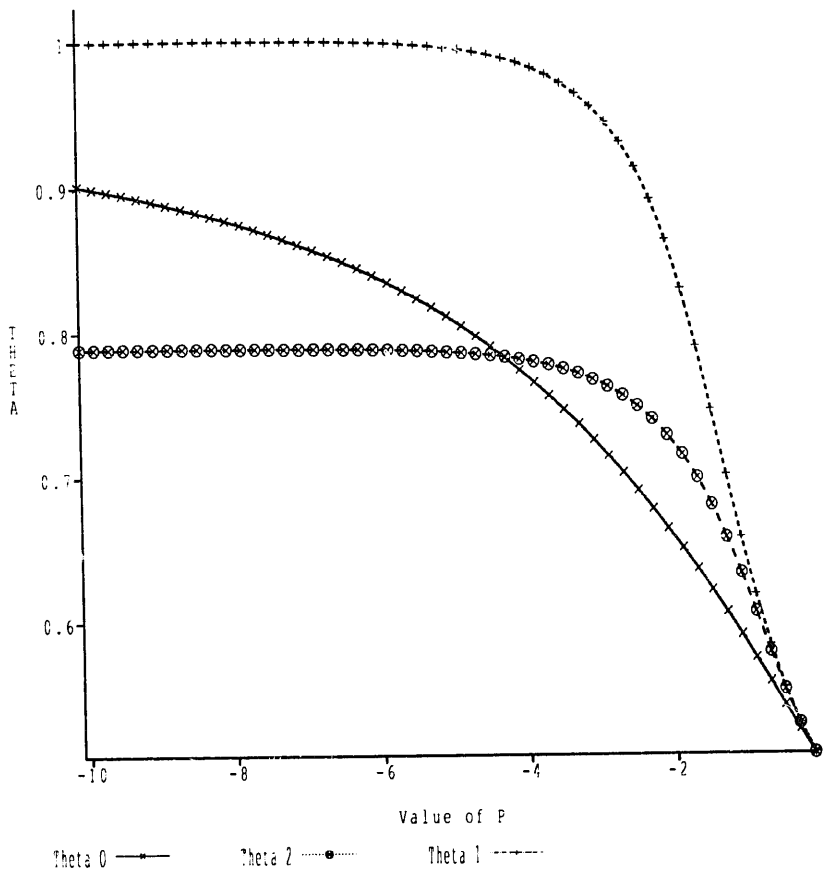

Fig. 1. Comparison of theta values.

attempt to select one "optimal" value. Johnson et al. [8] suggest that it is not necessary to find this value to great precision and that an approximate value is sufficient. The simplest way is to approximately minimise $\| \tau_{est}(\Theta) \|$ by searching over a number of discrete values of Θ. In the case when the ODE system being solved is large in size it may be expensive to perform many norm evaluations. These norm evaluations can be computed inexpensively when the Euclidean norm is being used by splitting the norm calculation into constants depending on the vector parts of equation (25) and constants depending on Θ. As the vector parts need only be computed once, evaluations of $\|\tau(\Theta)\|_2$ are then computationally inexpensive for different values of Θ.

3. A discussion of strategies for an adaptive theta method

The objective is to determine the most efficient method for solving a problem. If a problem changes character, that is, from nonstiff to stiff or vice versa, in the interval of integration, we want to switch automatically to the most efficient iteration method and value of Θ. The difficulty is that any automatic scheme must predict the future behaviour of the integration of a method which it is not currently using. The decision on which iteration method will solve a given problem most efficiently is made by estimating and then comparing:

(a) the step size that each method could use on the next step, and
(b) the cost per step of each method.

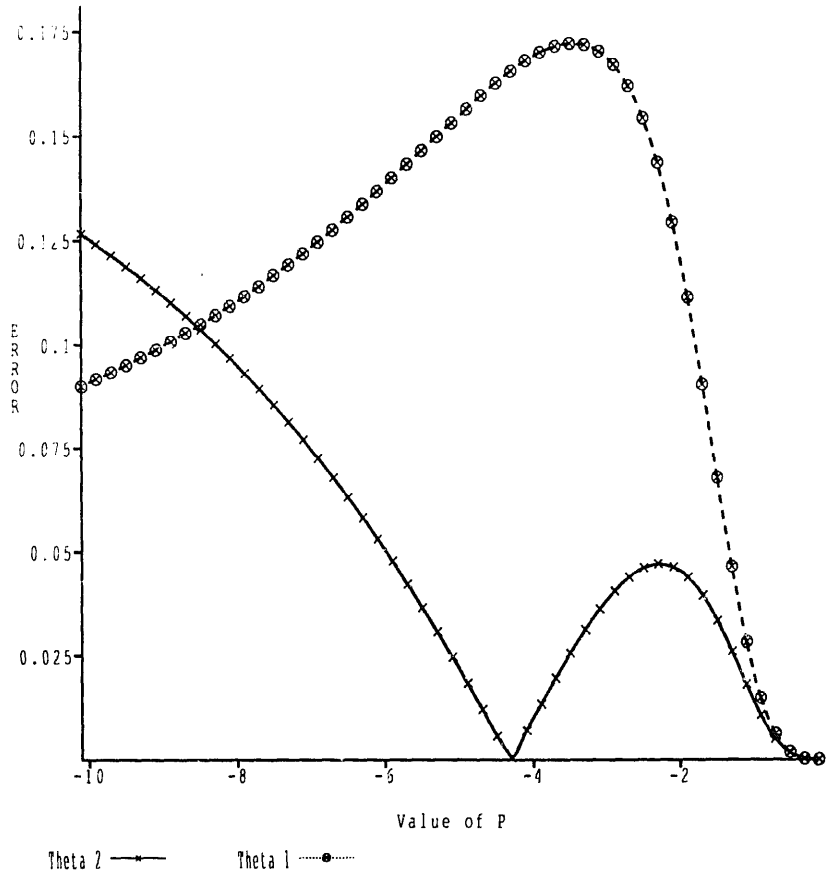

Fig. 2. Comparison of theta errors.

In the case when a nonstiff method is used, equation (2) is solved using functional iteration. The cost per unit step of doing this is the usual code overhead plus two or three iterations each of which involves a function evaluation. In contrast the cost per unit step of Newton's method is more difficult to evaluate. The Jacobian matrix (equation (5)) is not evaluated every time step; it is only estimated and decomposed into its LU factors when this becomes necessary, e.g. due to convergence failures. This cost also depends on whether or not full, banded or sparse matrix routines are used in this decomposition. Each iteration then involves one function evaluation and one backsubstitution, with a maximum of usually three iterations per attempt with a given step size. This range of possibilities makes it difficult to estimate beforehand how much work per step is involved for the Newton iteration.

There are at least two possible ways to measure the relative costs of the two iteration methods. The first is to use the ratio of the function calls per step, averaged out over enough steps to take the periodic updating of the Jacobian into account. The second is to use the CPU time per step as measured by the clock of the computer. Experimentation with these approaches reveals that both have their disadvantages. The first approach takes no account of the time spent in LU decomposition and backsubstitution while the second approach is hampered by the inability of CPU clocks to accurately measure what may be very small amounts of CPU time. The first approach does work well if the cost of a function call is expensive.

For both methods the step size must be chosen so that the formula is accurate over the next step: that is, a norm of the local truncation error is less than some tolerance ε. The step size must be small enough so that the Newton or functional iteration converges rapidly. In addition, for a nonstiff method, the step size must be small enough so that the method will be stable, although in practice instability will be detected by the growth of the local error.

Should a problem start to become stiff there will come a point when the step size is inefficiently small in comparison to the step that the stiff method could use, despite the additional cost of the Newton iteration. The criteria that can be used to decide when the switch should be made are discussed in the next section.

4. An adaptive method for stiff and nonstiff problems

4.1. Introduction

The integrator of Chua and Dew [4] solves the system of differential equations of the form of (1) using the theta method (2) with $\Theta = 0.55$. This integrator was developed from the Prothero and Robinson [13] integrator as described by Hopkins [7]. All these codes use the predictor defined by equation (15) and the error estimator defined by equation (25).

The objective in this paper is to modify the Chua and Dew integrator so that it is adaptive with respect to the value of theta and in the type of iteration method (functional or Newton iteration) used to solve the nonlinear equations. The choice of method is based on criteria established from the error estimates and the step sizes that the different values of theta and the different iteration methods would use.

Shampine [16] has also constructed a switching code based on the theta method for equations in normal form and allows different values of theta though he does not switch between them. Section 2 has indicated that one choice is to use $\Theta = 0.55$ for stiff equations and $\Theta = 0.45$ for nonstiff equations. Following the suggestion in Section 2.4, we have experimented with using values of Θ as low as 0.42 in our algorithm but have found that the algorithm is more efficient if $\Theta > 0.5$. Furthermore, the largest step size that can be used with functional iteration is given by equation (4). On substituting the values $\Theta = 0.45$ and $\Theta = 0.55$ into equation (4) we see that the step size that can be taken with $\Theta = 0.45$ is only 0.55/0.45 or only 22% larger while for values of Θ closer to 0.5 the difference is even less significant. The decision to restrict Θ in this way does seem to depend on the class of problems being solved. Johnson et al. [8] suggest that Θ should be allowed to be less than 0.5. In any case it is straightforward to extend the search for the minimum local error to include values of Θ less than 0.5, see Sections 4.4 and 5.2 below.

4.2. Changing method type from nonstiff to stiff

In considering changing the method type from nonstiff to stiff there are two step sizes of interest:

- h_{iter} : maximum step size for functional iteration to converge;
- h_{accy}: maximum step size possible by using a Newton iteration.

We iterate to convergence with functional iteration and a step size that is chosen to ensure rapid convergence. In this situation, the local error estimate indicates the step size that could be taken if a Newton iteration was being used, h_{accy}, since there is no restriction on the step size due to stability.

The decision to switch is based on comparing the step size needed for accuracy with that needed for convergence. The condition for rapid convergence of functional iteration is, assuming we have an up-to-date estimate of the Jacobian,

$$h_{\text{iter}}\Theta\rho(J) < r < 1, \tag{28}$$

where the convergence rate r is taken to be (say) 0.5, Θ is one of the values mentioned in Section 5.5 below and $\rho(J)$ is the spectral radius of the Jacobian matrix J. The actual rate of convergence of functional iteration may be estimated [15] by:

$$\text{CRATE} = \frac{\| y_{n+1}^{(m+2)} - y_{n+1}^{(m+1)} \|}{\| y_{n+1}^{(m+1)} - y_{n+1}^{(m)} \|} \tag{29}$$

and using (28) with h_{iter} replaced by h and r replaced by CRATE gives

$$\rho(J) \approx \frac{\text{CRATE}}{h\Theta}. \tag{30}$$

This estimate for $\rho(J)$ is substituted into (28) to obtain the inequality used in the code to estimate the functional iteration step size, h_{iter},

$$h_{\text{iter}} \leqslant \frac{h*0.5}{\text{CRATE}}, \tag{31}$$

where the constant 0.5 has been introduced as a safety factor.

If $h_{\text{accy}}/h_{\text{iter}}$ is large, then the step size has to be restricted to make functional iteration converge and it may be cost-effective to switch to Newton's iteration. Whether or not the switch is made depends on the cost per step of the using the two iterations. Let C_{N} be this cost for the Newton method and C_{f} be the cost for functional iteration. The overall computational cost will be reduced if

$$h_{\text{accy}} > h_{\text{iter}}C_{\text{N}}/C_{\text{f}}. \tag{32}$$

In Section 3 we explained why it may be difficult to estimate the ratio of these costs and so we cannot use this criterion exactly. Furthermore, if C_{N} is large compared to C_{f}, i.e., a Newton step is expensive compared to a functional iteration step then the change-over to a Newton method will involve a new step size h_{accy} that is large compared to the present step size h_{iter}. The algorithm will thus attempt to deduce the behaviour of the numerical solution on a much larger time scale than that being used by the functional iteration method. This, in general, will make it harder to accurately predict the error that will be incurred by using the new (much larger) step size. For this reason it seems better to recognise early when a Newton method can be used by restricting the increase in step size at the change-over in iteration methods, and to switch to Newton's method if $h_{\text{accy}} \geqslant 4h_{\text{iter}}$ and if at least twelve steps have been taken since the last switch to functional iteration. The factor 4 was determined by experimentation with a number of test problems and represents a typical value of the ratio $C_{\text{N}}/C_{\text{f}}$ in (32). The code allows the user to change this parameter if this is necessary. The switch is also made if three or more step size reductions have been made on the current step due to convergence failures.

4.3. Changing type from stiff to nonstiff

When Newton's method is being used, an estimate of $\rho(J)$ is required to establish the step size that can be used with functional iteration by applying the convergence condition of equation

(28). Petzold [12] estimates $\rho(J)$ by calculating $\| J \|_\infty$ directly from the Jacobian matrix. This may overestimate the spectral radius and thus stop the code from switching to functional iteration. As the spectral radius is expensive to calculate, the approach we have adopted is a pragmatic one. Four function calls are used in three attempted functional iterations. This allows two rates of convergence to be estimated as in equation (29). No switch is made unless both these rates are less than 0.9, the second rate of convergence is less than 0.7 and the iteration has converged according to the criteria used by the nonlinear equations solver. In the case when the first estimated rate of convergence is > 0.9 the attempt is directly terminated.

The disadvantage of this approach is the cost of the four extra function calls required, but this should be considered in the context of large problems where many function calls may be required to form the Jacobian. In any case the test for switching to functional iteration is only made if the Jacobian matrix is being reformed because the Newton step size is about to be increased. This is an opportune time to test as it marks the point where the existing LU decomposition of the Jacobian is no longer useful and the integrator has not run into difficulties which have required the step size to be reduced. One advantage of this approach is that it does not require the use of matrix norms and so does not rely on the perhaps unnecessarily pessimistic estimate of the spectral radius $\rho(J)$ by using a matrix norm such as $\| J \|_\infty$. An alternative approach is that of Norsett and Thomsen [11] who estimate the stiffness ratio using ratios of the norms of the vector quantities on the left and right sides of equation (6).

Once the switch has been made h_{iter} can then be estimated by

$$h_{\text{iter}} = \frac{0.5h}{\text{CRATE}}, \tag{33}$$

where h is the step size being used. If h_{iter}/h is large, then the code no longer needs a very small step size for the convergence of functional iteration.

The switch back to functional iteration is attempted only under two circumstances. The first is when the Jacobian matrix is about to be reformed because the Newton step size is about to be doubled. (The code automatically forms a new Jacobian whenever the step size is either doubled or halved in a Newton mode.) The second is when the Jacobian is being reformed because twenty steps have been taken with the same Jacobian matrix. In both cases no switch is attempted until at least ten steps after the last switch to a Newton method in order to allow the integration to settle down.

4.4. Choosing a new value of Θ

In order for the procedure for selecting Θ to be reliable, a good estimate of the local error with the proposed new value of Θ is required. It is therefore wise to leave the selection of Θ until the code has taken a number of steps with constant step size (the normal course of events) and has in this sense settled down. It is then possible to estimate

$$\min_i \| \tau(\Theta_i) \|_2,$$

where Θ_i are the possible values used to minimise the norm of the local error τ and the vector norm is weighted to reflect the form of local error test (absolute, relative or mixed) being used for each component. The values of Θ_i used are given in Section 5.2 below. This strategy also prevents Θ from being changed too frequently.

5. Implementation details

The strategy of the integrator is first to predict a solution and then to try and compute the solution at the next time level using the nonlinear equations solver which also estimates the rate of convergence CRATE. If functional iteration is being used the step size h_{iter} is estimated as in equation (31). In the case when the solver fails to converge we take the approach described in Section 5.4.

5.1. Error estimates

After a successful return from the nonlinear solver the weighted local error norm, ERRL, is estimated using equation (25) if Newton iteration is being used or (26) if functional iteration is being used. In the latter case, in addition to the weighted local error norm, another error estimate is computed. This is the estimate of the weighted local error norm we would have, if Newton's method was being used. From this error estimate we know whether or not the step size or Newton's method need to be changed.

Then since the local error estimate is $O(h^2)$, it is expected that the weighted local error norm for Newton's method, ERRN, can be estimated from:

$$\text{ERRN} = \left(\frac{h_{\text{accy}}}{h_{\text{iter}}}\right)^2 \text{ERRF}, \tag{34}$$

where ERRF is the estimate of the weighted norm of the local error using functional iteration.

5.2. Local error tolerance satisfied

If the weighted local error norm ERRL of the current method being used is $\leqslant 1$, then the solution satisfies the local error tolerance. The strategy used is to double the step size, if the weighted local error norm is < 0.25, and to halve the step size if it is > 1.0. The same strategy is used to change the step size h_{accy} depending on the value of ERRN.

After three successful steps with the same step size, we check to see if the step size required for accuracy can be doubled for the next step. If the step size is less than the maximum (integrator dependent) step size, and the weighted local error norm is < 0.25 the step size is doubled provided that it remains less than the maximum allowed step size. The exception to this is when $\Theta < 0.51$—"almost" third-order local error. In this case 0.15 is used instead of 0.25 to prevent the new step size being immediately rejected. In the case of functional iteration, if doubling the step size results in a step size greater than the maximum step size, h_{iter}, then the step size is not changed from its previous value.

In the case when the step size is about to be doubled a test is made to see if the most appropriate value of Θ is being employed. This consists of re-estimating what the local error would be for the values of $\Theta = 0.51$, 0.55, 0.59, and 0.63 and choosing the value of Θ which minimises the error. This range of values is a restriction of the possible values of Θ suggested by Sections 2.4 and 4.1, i.e., $0.50 < \Theta < 0.78$. Although we have experimented with using different ranges of Θ values on a wide class of test problems, the above range of Θ seems to give the most efficient code for the range of stiff and nonstiff test problems that we used. It is possible that different test problems would have suggested different values of Θ.

5.3. *Local error test fails*

If the solution does not satisfy the local error tolerance, then we must reduce the step size and possibly switch methods. From the Chua and Dew [4] strategy, if the weighted local error norm is > 1.0, the step size is halved. If functional iteration is being used, we allow up to three such reductions per step after which a switch to Newton's method is made. This switch has the advantage that the more reliable error estimate for stiff problems defined by equation (25) can be used.

5.4. *Nonlinear equation solver fails*

When the nonlinear equation solver fails to converge, then we must also consider reducing the step size or switching methods. The step size is reduced a maximum of three times per step (six times on the first step). The switch from functional iteration to a Newton method is made after three step size reductions or if $h_{\text{accy}} > 4h$, whichever comes first where the factor 4 is the approximation to the ratio C_N/C_f as in Section 4.2.

6. Numerical results

Numerical testing was conducted using three integrators. The theta method code of Chua and Dew as implemented in the SPRINT software of Berzins, Dew and Furzeland [1], referred to in the numerical testing experiments as STHETA, and the corresponding switching method, code STHETZ, and the SPRINT backward differentiation method, SPGEAR. In Tables 1 and 2, Fun is the total number of ODE function call evaluations including those used in numerical differencing to form the Jacobian matrix while JAC is the number of LU decompositions of the Jacobian matrix. The SPRINT package is designed primarily for differential algebraic equations

Table 1
Results on the van der Pol equation with $\varepsilon = 1000$

TOL	Method	Steps	Fun	JAC	CPU
10^{-2}	STHETA	324	1318	149	0.14
	STHETZ	323	1286	117	0.15
	SPGEAR	348	997	136	0.14
10^{-3}	STHETA	724	2262	177	0.26
	STHETZ	597	1848	109	0.23
	SPGEAR	540	1339	158	0.21
10^{-4}	STHETA	2149	5377	241	0.67
	STHETZ	1240	3405	101	0.44
	SPGEAR	791	1724	170	0.38
10^{-5}	STHETA	6176	14036	422	1.73
	STHETZ	3180	7625	88	1.05
	SPGEAR	1075	2035	76	0.38

Table 2
Results on the Enright et al. B5 test problem using STHETA and STHETZ

TOL	Method	Steps	Fun	JAC	CPU
10^{-2}	STHETA	90	347	21	0.06
	STHETZ	101	279	6	0.05
	SPGEAR	152	681	11	0.12
10^{-3}	STHETA	237	738	34	0.14
	STHETZ	224	583	10	0.12
	SPGEAR	183	357	22	0.10
10^{-4}	STHETA	735	1855	51	0.42
	STHETZ	531	1304	15	0.27
	SPGEAR	322	544	22	0.10
10^{-5}	STHETA	2209	5279	119	1.23
	STHETZ	1367	3094	8	0.69
	SPGEAR	424	682	33	0.22

and has no option to save the Jacobian matrix so the Jacobian is reformulated before each LU decomposition. The CPU time was measured in seconds on an AMDAHL 5850.

6.1. Testing on two small test problems

The first two test problems are designed to test whether or not the code switches as expected, rather than to show dramatic reductions in CPU time.

Problem 1 (Van der Pol's equation). The first problem, van der Pol's equation, [17], was chosen because it alternates between being stiff and nonstiff several times during the interval of integration and so provides a good test of the codes' ability to switch between the two methods.

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 2.0, \\ y_2' &= \varepsilon(1 - y_1^2)y_2 - y_1 & y_2(0) &= 0, \end{aligned}$$

where $\varepsilon = 1000$ and the interval of integration is (0,3000) (see Table 1).

The results show that the STHETZ code is more efficient than STHETA and for low to medium accuracy requirements competes with the BDF code SPGEAR. The number of LU decompositions for STHETZ also compares well with the results obtained by Norsett and Thomsen [11] using their approach.

Problem 2 (Enright et al.). This problem is the well-known B5 test problem from the test set of Enright, Hull and Lindberg [5]. The problem was chosen because the Jacobian matrix of this problem has the eigenvalues $-10-100i$, $-10+100i$, -4, -1, -0.5, -0.1 two of which are close to the imaginary axis. (See Table 2.)

The results for Problems 1 and 2 show that STHETZ uses noticeably less function calls and Jacobian evaluations than STHETA. The decrease in the number of Jacobian evaluations is not significant for these test problems as the cost of computing the Jacobian is not high. In the next section we shall investigate what happens when the two codes are applied to a more typical larger test problem where Jacobian evaluations play a more important role.

6.2. *A larger test problem based on convection–diffusion*

The use of method of lines software for time-dependent partial differential equations in two space dimensions involves the time integration of a large number of (between 10^3 and 10^9) ordinary differential equations. For such large systems it may not be appropriate to use direct sparse matrix routines to solve the large system of linear equations that arise at each step of the Newton iteration. Instead it may be more efficient to use iterative sparse matrix methods which may perform better than traditional direct methods. Of the many iterative sparse methods proposed there are a number of contenders for use within PDE packages, GMRES as used by Brown and Hindmarsh [3], the IOM method as used by Moore and Flaherty [10] and Orthomin as used with SPRINT by Seward [19] in WATSIT. Orthomin is a preconditioned, truncated and restarted, generalised conjugate residual method which can tackle a larger class of problems than the conjugate gradient method. WATSIT also requires the Jacobian matrix to be formed for use in an incomplete LU preconditioner for the Orthomin iteration. The expense of the preconditioning step can be compared to the LU factorisation step of the sparse matrix routines in SPRINT. For reasons of computational efficiency, the preconditioner is reused for several iterations of Newton's method. In general only one or two Orthomin iterations are needed for each Newton iteration. There is an option to try and reduce the size of the system of equations using the red–black ordering scheme applied to the Jacobian matrix as this sometimes leads to a considerable saving in computational effort (see Seward [14]).

The test problem used to demonstrate the performance of the new type insensitive code is the following two-dimensional problem:

$$v_t + u(x, t)v_x + u(y, t)v_y - \nu(v_{xx} + v_{yy}) = 0 \tag{35}$$

with analytic solution, boundary and initial conditions defined by $v(x, y, t) = u(x, t)u(y, t)$ where $u(x, t)$ is defined by

$$u(x, t) = \frac{0.1A + 0.5B + C}{A + B + C},$$

where $A = e^{(-0.05(x-0.5+4.95t)/\nu)}$, $B = e^{-0.25(x-0.5+0.75t)/\nu)}$ and $C = e^{(-0.5(x-0.375)/\nu)}$.

$\nu = 0.0001$ gives a convection-dominated problem, while $\nu = 0.004$ gives a convection problem with rather more diffusion. The problem was discretised in space on a uniform square mesh of NPTS points on the region $(x, y) \in [0, 1] \times [0, 1]$ by using a standard finite volume scheme with the van Leer harmonic mean limiter to preserve monotonicity [9, p. 180], applied to the convective fluxes. This scheme results in a numerical solution that is free of oscillations regardless of the mesh spacing. The steep shock-like wave front is, however, smeared out across two spatial cells. In Table 3 NPTS is the total number of spatial mesh points and hence the total number of ordinary differential equations. The SPRINT time integrators used were the Gear BDF integrator SPGEAR and the STHETZ code which chooses whether or not to switch between Newton and functional iteration. In Tables 3 and 4 the Yale sparse routines and WATSIT when used with the Gear BDF code SPGEAR are referred to as Yale and WATS. In the case of WATSIT, the entry under JAC refers only to the number of formulations of the

Table 3
$\nu = 0.004$ results

NPTS	Method	CPU	Steps	Fun	JAC
625	YALE	116	62	152	5
	WATS	108	61	150	5
	THETZ	112	62	152	5
2500	YALE	836	74	183	6
	WATS	737	74	183	6
	THETZ	875	104	254	0
5625	YALE	2786	79	199	8
	WATS	2415	81	216	9
	THETZ	2134	70	219	4
10000	YALE	7279	79	221	9
	WATS	5137	79	208	8
	THETZ	7368	96	313	7

Jacobian matrix as no full LU decomposition is carried out. In Table 3, CPU refers to the computer time in seconds on a SPARC1 workstation. In both tables the requested local error was 10^{-3} and the error norm used was a maximum norm. In Table 4, CPU refers to the computer time in seconds on a DEC 5000/200 workstation. The results show that WATSIT proves to be superior to the sparse matrix techniques and that as the problem size gets larger, requires less CPU time. Similar results are provided by Seward [19]. In the case when $\nu = 0.004$ the ODEs are only midly stiff so STHETZ uses the Newton iteration for most of the time and competes with the Gear code using the sparse matrix techniques. It is clear from Table 3 that WATSIT should be used with the switching code. Finally in the convection-dominated case ($\nu = 0.0001$) the ODEs are not stiff and the STHETZ code is able to recognise this and to use functional instead

Table 4
$\nu = 0.0001$ results

NPTS	Method	CPU	Steps	Fun	JAC
625	YALE	309	183	1451	77
	WATS	323	194	1638	88
	THETZ	65	140	376	0
2500	YALE	4053	378	2825	160
	WATS	2961	379	2785	157
	THETZ	639	263	677	0
5625	YALE	15860	536	3039	172
	WATS	10040	553	3293	190
	THETZ	2745	386	1016	0
10000	YALE	69900	695	3781	215
	WATS	25210	708	3854	220
	THETZ	7136	474	1229	0

of Newton iteration. This results in large savings in computer time. The ideal combination would thus seem to be to use WATSIT in conjunction with a type insensitive code like STHETZ. This is made possible by the modular structure of SPRINT which allows this combination without difficulty. A similar combination of a type insensitive code and an iterative solver is already used by Hindmarsh and Norsett [6].

7. Conclusions

An adaptive theta method can be used to advantage for the low to medium accuracy solution of differential equation problems where the stiffness of the system may vary widely over the range of integration and in a way that is unknown beforehand. The adaptive method is based on an efficient implementation of the theta method, in which the method used for nonlinear equation solving is switched between functional and Newton iteration as the stiffness varies. The choice of iteration method is based on criteria established from the error estimates and step sizes that the two different methods would use. At the same time the value of Θ used is varied to try and make the time step as large as possible.

The adaptive method can result in improved efficiency in terms of the reduced number of Jacobian evaluations needed compared with a method based on a separate stiff or nonstiff method with a fixed value of Θ. This improvement is particularly noticeable for problems with variable stiffness for example where there is an initial transient (nonstiff) period. For problems in which low to medium accuracy is required the new method appears to compete with variable order multistep codes on the basis of the limited experiments so far.

Acknowledgement

The authors would like to thank Justin Ware for computing the numerical results listed in Table 3 and 4 and Alan Hindmarsh for ending their flirtation with typographical errors.

References

[1] M. Berzins, P.M. Dew and R.M. Furzeland, Developing software for time-dependent problems using the method of lines and differential algebraic integrators, *Appl. Numer. Math.* 5 (1989) 375–397.

[2] M. Berzins and R.M. Furzeland, A type insensitive method for the solution of stiff and non-stiff differential equations, Report 204, University of Leeds, School of Computer Studies (1986).

[3] P.N. Brown and A.C. Hindmarsh, Reduced storage matrix methods in stiff ode systems, *Appl. Math. Comput.* 31 (1989) 40–91.

[4] T.S. Chua and P.M. Dew, The design of a variable-step integrator for the simulation of gas transmission networks, *Internat. J. Numer. Methods Engrg.* (1984) 1797–1813.

[5] W.M. Enright, T.E. Hull and B. Lindberg, Comparing numerical methods for stiff systems of O.D.E.s, *BIT* 15 (1975) 10–48.

[6] A.C. Hindmarsh and S.P. Norsett, KRYSI, An O.D.E. solver combining semi-implicit Runge-Kutta method and a preconditioned Krylov method, Report UCID-21422, Lawrence Livermore National Laboratory, Livermore CA (1988).

[7] T.R. Hopkins, Numerical solution of quasi-linear parabolic P.D.E.s, Ph.D Thesis, Liverpool University (1976).

[8] R.B.I. Johnson, B.J. Cory and M.J. Short, A tunable integration method for the simulation of power system dynamics, IEEE/PES 88 Winter Meeting Paper 88 WM 177-8 (1988).

[9] R.J. Leveque, *Numerical Methods for Conservation Laws*, Lectures in Mathematics Series (Birkhäuser, Basel, 1990).

[10] P.K. Moore and J.E. Flaherty, *Adaptive overlapping grid methods for parabolic systems in two space dimensions*, Report 90-5, Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY (1990).

[11] S.P. Norsett and P.G. Thomsen, Switching between modified Newton and fix-point iteration for implicit o.d.e. solvers, *BIT* 26 (1986) 349–348.

[12] L. Petzold, Automatic selection of methods for solving stiff and non-stiff systems of ordinary differential equations, *SIAM J. Sci. Stat. Comput.* 4 (1) (1983) 136–148.

[13] A. Prothero and A. Robinson, On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations, *Math. Comp.* 28 (1974) 145.

[14] W. Seward, Solving large ODE systems using a reduced system iterative matrix solver, Research Report CS-89-38, Computer Science Department, University of Waterloo, Ont. (1989).

[15] L.F. Shampine, Type-insensitive O.D.E. codes based on implicit A-stable formulas, *Math. Comp.* 36 (1981) 499–510.

[16] L.F. Shampine, Type-insensitive O.D.E. codes based on implicit $A(\alpha)$-stable formulas, *Math. Comp.* 39 (1982) 109–123.

[17] J. Villadsen and M.L. Michaelson, Solution of differential equation models by polynomial approximation (Prentice-Hall, Englewood Cliffs, NJ, 1973).