Journal of Computational and Applied Mathematics 11 (1984) 283–296
North-Holland

Recent advances in methods for numerical solution of O.D.E. initial value problems

T.D. BUI
Dept. of Computer Science, Concordia University, Montreal, Quebec, Canada

A.K. OPPENHEIM
Dept. of Mechanical Engineering, University of California, Berkeley, CA, U.S.A.

D.T. PRATT
Dept. of Mechanical Engineering, University of Washington, Seattle, WA, U.S.A.

Received 9 December 1983
Revised 2 June 1984

Abstract: This is a review paper which describes recent advances in numerical methods and computer codes for solving initial value problems of ordinary differential equations. Particular emphasis is placed upon stiff systems.

Keywords: Initial value problems in O.D.E., stiff systems, measuring stiffness, Runge–Kutta methods, multi-step methods, exponential-fitted methods, computer codes for O.D.E.

1. Introduction

In mathematical modeling of physical systems, very often we are required to solve an initial value problem (IVP), consisting of a system of ordinary differential equations (ODE) which could be written as:

$$y' = f(x, y), \qquad x \in [a, b], \quad y, f \in R^N, \tag{1.1}$$

$$y(a) \text{ given.}$$

A typical program (code) steps through $[a, b]$ and produces approximate solutions at certain mesh points. Proceeding from y_n (the approximate value to $y(x_n)$) it computes y_{n+1} at $x_{n+1} = x_n + h_{n+1}$; h_{n+1} or simply h is the step-size.

If we define $z_n(x)$ as the solution of the following problem:

$$z_n' = f(x, z_n), \qquad z_n(x_n) = y_n. \tag{1.2}$$

Then the program will actually approximate this *local solution* over the step-size h by y_{n+1}. Thus the error $T_{n+1} = z_n(x_{n+1}) - y_{n+1}$ is the *local truncation error*. Almost all existing codes try to control this local error so that at each step

$$\|z_n(x_{n+1}) - y_{n+1}\| \leqslant \tau_{n+1}, \tag{1.3}$$

τ_{n+1} expressing the tolerance.

However, it should be noted, the user is really interested in controlling the *true* or *global error*: $\|y(x_{n+1}) - y_{n+1}\|$.

2. Stiff phenomena

It is customary to define stiff phenomena in terms of the eigenvalues of the Jacobian. However, there are some difficulties with this approach both conceptually and practically. We first define stiffness in terms of the eigenvalues of the Jacobian as follows:

Definition: The system (1.1) is stiff if:

(i) Real $\lambda_i < 0$, $i = 1, 2, \ldots, N$;

(ii) $S \equiv \max_i |\text{Real } \lambda_i| / \min_i |\text{Real } \lambda_i| \gg 1$,

where S is the so-called stiffness ratio.

Comments on the definition of stiffness

(a) Condition (i) does not cover linear problems with variable coefficients and nonlinear problems where one (or more) of the eigenvalues may cross into the region of the positive real axis temporarily.

(b) Condition (ii) becomes ambiguous when the real part of an eigenvalue approaches zero. In this case, the stiffness ratio may be large yet the problem is not stiff since it can be solved effectively by methods with bounded region of stability (explicit methods).

(c) In practice, it is desirable to know if a system is stiff in certain intervals of integration, so that a proper method for stiff equations could be used effectively. Recently, some interest has been paid to developing type-insensitive codes in which implicit (for stiff) and explicit (for non-stiff) methods are used alternatively depending on the stiffness of the problem [40,41]. Monitoring the eigenvalues of the Jacobian at every step of integration is very expensive. However, an estimate of the Lipschitz constant proves to be a very practical way to determine the stiffness of a problem.

(d) A proper way to describe stiffness is as follows: it occurs when *stability* rather than *accuracy* dictates the step size. For example, when solving the constant coefficient linear system $y' = Ay + g(x)$, accuracy may pose a severe restriction on the step size when $g(x)$ is a nasty function, then stability becomes less important and the problem is not stiff.

(e) In some cases, the system (1.1) can be partitioned into stiff and non-stiff sub-systems. This partitioning process allows an efficient numerical method to the problem, since the stiff and non-stiff components are now treated separately (see [17] for more details).

3. Measuring stiffness

In this section we describe a quantitative approach to determine stiffness of a given problem. Unfortunately, the concept of stiffness is rather vague because in practice it involves a number of phenomena. As mentioned earlier, classical measures of stiffness are useful but are known to ignore several important factors. This section presents an approach to refine these measures [42]. We distinguish two kinds of methods for solving the initial value problems of ODE's: explicit methods and implicit methods.

Most methods of order p have local truncation error at x_n of the form:

$$\mathrm{LTE}(x_n) = \zeta h^{p+1} y^{(p+1)}(x_n) + \mathrm{O}(h^{p+2}), \tag{3.1}$$

ζ is a constant. The numerical problem also involves a tolerance τ and a norm in which error is to be measured. We demand that

$$\|\mathrm{LTE}(x_n)\| \leqslant \tau. \tag{3.2}$$

The largest step-size, which would satisfy the local accuracy test (3.2) is given by:

$$h_{\mathrm{acc}} = \left(\frac{\tau}{|\zeta| \cdot \|y^{(p+1)}(x_n)\|} \right)^{1/(p+1)} \tag{3.3}$$

Approximation (3.3) is not valid when $y^{(p+1)}(x)$ vanishes or h is not sufficiently small. In the later case, the leading term in (3.1) does not dominate the remaining terms.

For explicit methods based on polynomial approximations the region of absolute stability is bounded by a half-disc of radius r. Thus, for a stable integration with step-size h we must have:

$$|h\lambda| \leqslant r \tag{3.4}$$

for all eigenvalues λ of the Jacobian $f_y(x_n, y(x_n)) \triangleq f_y^n$ which have nonpositive real parts.

Let $\rho_L(f_y^n) = \max|\lambda|$ (with $\mathrm{Re}(\lambda) < 0$), then the largest stable step size, h_{stab}, is:

$$h_{\mathrm{stab}} = \frac{r}{\rho_L(f_y^n)}. \tag{3.5}$$

A suitable measure of stiffness is then [42]:

$$h_{\mathrm{acc}}/h_{\mathrm{stab}} = \tau^{1/p+1} \rho_L(f_y^n) \|y^{(p+1)}(x_n)\|^{-1/p+1} \left(\frac{|\zeta|^{-1/p+1}}{r} \right). \tag{3.6}$$

Remarks. (a) Reducing τ, decreases stiffness.

(b) Lowering the order p, increases stiffness.

(c) Along the integration curve ζ and r remain unchanged, while $\rho_L(f_y^n)\|y^{(p+1)}(x_n)\|^{-1/p+1}$ computed along the solution curve gives a fair measure of stiffness.

(d) The above criterion is also applicable when we wish to compare stiffness of two problems if both are integrated within the same tolerance using the same class of methods.

For implicit methods, (3.4) is no longer valid since they have no stability restrictions. The implicit equation defining y_{n+1} can be written as:

$$y = h\gamma f(x_{n+1}, y) + \Psi_n, \tag{3.7}$$

where γ is a constant and Ψ_n lumps together information at the previous steps. In order for the simple iteration

$$y^{m+1} = h\gamma f(x_{n+1}, y^m) + \Psi_n \tag{3.8}$$

to work for all starting values y^0 near a solution y^*, it is required that:

$$h|\gamma|\rho\big(f_y(x_{n+1}, y^*)\big) < 1 \tag{3.9}$$

or [42]:

$$h|\gamma|L < 1 \tag{3.10}$$

where

$$L = \|f_y(x_n, y(x_n))\|,$$

hence

$$h_{\text{iter}} = \frac{1}{|\gamma|\|f_y(x_n, y(x_n))\|}, \tag{3.11}$$

thus:

$$\frac{h_{\text{acc}}}{h_{\text{iter}}} = \tau^{1/p+1}\|f_y(x_n, y(x_n))\| \cdot \|y^{(p+1)}(x_n)\|^{-1/p+1}\left(\frac{|\zeta|^{-1/p+1}}{|\gamma|^{-1}}\right). \tag{3.12}$$

4. Runge–Kutta methods

The Runge–Kutta methods (explicit, semi-implicit, and fully implicit) are one-step methods. An s-stage RK method is given by the following formula:

$$y_{n+1} = y_n + h_n \sum_{i=1}^{s} b_i k_i, \tag{4.1a}$$

$$k_i = f\left(x_n + c_i h_n, y_n + h_n \sum_{j=1}^{s'} \beta_{ij} k_j\right), \quad i = 1, 2, \ldots, s. \tag{4.1b}$$

An explicit RK method has $s' = i - 1$ and $k_1 = f(x_n, y_n)$; a semi-implicit method has $s' = i$; and a fully implicit method has $s' = s$. This means that the matrix β_{ij} has a strictly lower triangular form for explicit RK methods, a lower triangular form for semi-implicit, and a full matrix for fully implicit cases.

4.1. Schemes for local error estimates

The RK method is said to be of order $p + 1$ if the local truncation error $y_{n+1} - z(x_{n+1})$ is $O(h_n^{p+2})$, where $z(x)$ is the local solution to the system:

$$z' = f(x, z), \qquad z(x_n) = y_n. \tag{4.2}$$

An imbedded procedure uses a pair of formulae, one of order $p + 1$ and the other order p. The two formulae have the same set of k_i's so that the solution $\bar{y}_{n+1}$ of order p is calculated with very little extra work:

$$\bar{y}_{n+1} = y_n + h_n \sum_{i=1}^{s^*} \bar{b}_i k_i. \tag{4.3}$$

Note that s^* may be different from s—the number of stages for the formula of order $p + 1$. The estimated error is

$$\text{est} \sim \|y_{n+1} - \bar{y}_{n+1}\|. \tag{4.4}$$

This scheme has the advantage of a built-in error estimating capability.

Another approach, which was very popular in the past, and recently has received further attention, is the step-halving procedure. This involves solving the differential equation using step size h_n to obtain y_{n+1}, then solving it again with twice the step size $\frac{1}{2}h_n$, to obtain y^*_{n+1}. The

difference between y^*_{n+1} and y_{n+1} gives an estimate of the local truncation error. It is a general belief that the step-halving procedure requires more work than the imbedded approach. However, it has been shown recently that this is not always true (see [33] for details regarding single-step methods).

4.2. Explicit Runge–Kutta methods

For non-stiff or mildly stiff problems, the explicit RK methods have been very useful. This is because they require very little overhead. One of the most popular methods in this class is Fehlberg's imbedded pair of fourth and fifth-order formulae which requires six stages per step. A good implementation by Shampine and Watts [43,44] called RKF45 was published in 1977. Recently, this code was revised to include some additional capabilities and improvements. This new code called DERKF forms part of a new package called DEPAC developed at Sandia National Laboratories by a group led by Shampine and Watts [45].

The second code of this class is DVERK (available in the IMSL Library) [30]. This code was written by Hull, Enright, and Jackson [29] based on Verner's fifth- and sixth-order imbedded pair of formulae which require eight stages per step.

Runge–Kutta codes with variable orders seem to be useful. A complete set of imbedded RK formulae with order 1 through 6 requiring 9 stages was developed by Bettis of the University of Texas [3]. Recently, Verner [49] developed complete sets of formulae with orders 1 through 5, requiring six stages, orders 1 through 6 requiring eight stages, orders 1 through 7 requiring ten stages, and orders 1 through 9 requiring thirteen stages. Verner's development seems to be optimal as far as the number of stages per step is concerned. However, no computer code was developed based on these formulae.

4.3. Fully implicit Runge–Kutta methods

For stiff and very stiff problems, it is obvious that explicit RK methods are inefficient as a consequence of their bounded regions of stability. It is well known that fully implicit RK methods could be developed for high orders of accuracy and possessing strong stability properties. However, a straightforward implementation of these implicit methods involves solving large systems of nonlinear algebraic equations. For a system of N differential equations, the modified Newton method for solving (4.1b), in the implicit case, is as follows:

Let

$$\Phi_i = k_i - f\left(y_n + h_n \sum_{j=1}^{s} \beta_{ij} k_j \right), \tag{4.5}$$

here for simplicity, we assume an autonomous form. The modified Newton method requires solving repeatedly the linear system:

$$G \cdot \delta k = -\Phi \tag{4.6}$$

where $G = I - h_n(A \otimes J)$—the so-called *iteration matrix*.

$I = I_s \otimes I_N$ with $I_s = s \times s$ unit matrix.

$I_N = N \times N$ unit matrix.

A = matrix of β_{ij} (dimension s).

J = Jacobian matrix evaluated at the previous step.
$k = (k_1, \ldots, k_s)T$.
$\Phi = (\Phi_1, \ldots, \Phi_s)T$.

The solution of the linear system (4.6) by the LU decomposition requires $s^3N^3 + s^2N^2$ operations (multiplications and divisions). This is excessively large. Recently, some ingenious approaches have been devised to overcome this drawback [4,7]. The main idea of these schemes is to decouple the system of sN nonlinear algebraic equations to s systems of N equations each. To accomplish this, we define a similarity transformation T so that $T^{-1}AT$ is a lower triangular matrix. Therefore, the transformed system is clearly uncoupled into s systems of N equations. Further computations could be saved if all the diagonal elements of the lower triangular matrix are the same, then the same iteration matrix G occurs for each of the s systems. Butcher [7] was the first to notice this. He defined a similarity transformation T based on the Laguerre polynomials so that the transformed system has a bi-diagonal structure having constant λ for each entry in the main diagonal and $-\lambda$ for each element in the subdiagonal. A code, STRIDE, developed by Burrage, Butcher and Chipman in 1979 [9] is based on these ideas. The code contains a set of s-stage formulae of order s with s varies from 1 to 15. Some formulae are not A-stable; however, they are all damped at infinitely and the stability regions are quite acceptable. The code was constructed to be of collocation type and output values are produced by interpolating the underlying collocation polynomials. This code is believed to be suitable for stiff problems.

Another idea, which seems to be even more efficient than Bucher's transformation, is to transform the system into Hessenberg matrix. Let T be a similarity transformation so that

$$\tilde{A} = T^{-1}AT = \begin{pmatrix} \lambda_1 \cdots 0 \\ \cdots \\ 0 \cdots \lambda_s \end{pmatrix} \tag{4.7}$$

where λ_i can be complex and distinct. Therefore $\tilde{G} = I - h(\tilde{A} \otimes J)$ is a block diagonal matrix with the ith block being $(I - h_n\lambda_i J)$. If LU decomposition is used, we need s times of decomposition for each iterative step. The idea is then to form a Hessenberg matrix in the following way:

Let $(I - h_n\lambda_i J) = h_n\lambda_i((1/h_n\lambda_i)I - J) = \mu_i^{-1}(\mu_i I - J)$ where $\mu_i = 1/h_n\lambda_i$, then $(\mu_i I - J)$ can be factorized into Hessenberg form:

$$(J - \mu_1 I) = LHL^{-1}. \tag{4.8}$$

This is done only once since

$$(J - \mu_i I) = L[H - (\mu_i - \mu_1)I]L^{-1}.$$

Therefore we just factorize $(J - \mu_1 I)$, then for other blocks we only need to calculate $H - (\mu_i - \mu_1)I$, and the LU decomposition of the Hessenberg matrix H will be done only once. This approach is much more efficient than the decomposition of s blocks into LU forms. This approach was first suggested by Enright [15] and later advocated by Varah [48] for possible effective implementation of implicit RK methods based on Gauss quadrature formulae (Gauss–Legendre, Gauss–Radau, Gauss–Lobatto). The advantage of this scheme is that methods based on Gauss quadrature are of order $2s$ or $2s + 1$ while Butcher's method discussed before is only of order s or $s + 1$. The disadvantage is due to the fact that complex arithmetic is involved.

4.4. Semi-implicit Runge–Kutta methods

In this class, the drawback of having to solve sN nonlinear equations is avoided by imposing that the matrix A be lower triangular form. This automatically results in s systems of N nonlinear equations. Further, by choosing $\beta_{ii} = \beta$ for all i, the same iteration matrix is obtained for each system. This is called diagonally implicit RK methods. A code called DIRK based on this class for $s = 1, 2$, and 3 was developed by Alexander [1]. This code uses the step-halving procedure for error controls. The underlying formulae for DIRK are due to Crouzeix [12] and Alexander [1]. They are all A- or L-stable.

Norsett [38] has derived an L-stable, second-order formula with imbedded error estimate requiring three stages. Houbak and Thomsen [28] implemented this method into a code called SPARKS, which is specifically designed for large systems having sparse Jacobians.

4.5. Rosenbrock methods

The Rosenbrock method could be viewed as one iteration of the semi-implicit RK method. In an autonomous form, it is given by

$$k_i = f\left(y_n + h_n \sum_{j=1}^{i-1} \beta_{ij} k_j\right) + \beta h_n J_n k_i \tag{4.9a}$$

or

$$(I - \beta h_n J_n) k_i = f\left(y_n + h_n \sum_{j=1}^{i-1} \beta_{ij} k_j\right), \quad \text{for } i = 1,\ldots,s. \tag{4.9b}$$

This is a linear system of equations. However, there are s systems of N linear equations. It is important to note that in the implicit (or semi-implicit) case, the Jacobians are not required to be exact since they are only needed for the convergence of the modified Newton iteration. However, in the Rosenbrock methods, the Jacobians appear in the order conditions. Therefore, approximate Jacobians (via finite differences) will directly affect the order of Rosenbrock methods.

There are some computer codes based on this class. Villadsen and Michelsen [50] wrote a code called STIFF 3, which implements a 3-stage third-order L-stable method. Bui [6] has written a program called LSTIFF which implements s-stage formulae of order s for $s = 2, 3, 4$; they are all L-stable. Both codes use the step-halving procedure for error estimates and step-size control.

The original Rosenbrock procedure has been modified by Warner, called ROW-methods, by adding an extra term. This extra term was added to extend the stability properties of the Rosenbrock methods. ROW-methods are given by:

$$(I - \beta h_n J_n) k_i = f\left(y_n + h_n \sum_{j=1}^{i-1} \beta_{ij} k_j\right) + h_n J_n \sum_{j=1}^{i-1} \gamma_{ij} k_j, \quad \text{for } i = 1,\ldots,s. \tag{4.10}$$

Codes based on ROW-methods have been developed by Kaps and Rentrop [32] called GRKA and by Gottwald and Warner [20] called ROW4A. They both contain a pair of imbedded 3rd- and 4th-order formulae (for error estimate); however, only the third order formula is damped at infinity. Actually, ROW4A uses the same pair of formulae in GRK4A but a 'back-step' strategy was included. This back-step strategy is to avoid stepping over sharp peaks or quasi-discontinui-

ties. Recently, Kaps and Wanner [31,34] have been active in developing high order ROW-methods. They have also developed order conditions of ROW-methods for non-autonomous systems. This is, however, not a trivial problem since the number of order conditions increases drastically for non-autonomous ROW-methods. To facilitate the development, they have used Hairer's concepts of monotonically labelled trees and partitioned trees (L- and P-trees), which are useful for developing order conditions in many classes of methods (see [21,22] for details).

5. Multi-step methods

The general linear multistep method may be written as

$$\sum_{i=0}^{k} \alpha_{ki} y_{n-i+1} = h_n \sum_{i=0}^{k} \beta_{ki} f(x_{n-i+1}, y_{n-i+1}) \tag{5.1}$$

where $\alpha_{k0} \neq 0$ and $\alpha_{kk}^2 + \beta_{kk}^2 \neq 0$.

The two best known subclasses are: the Adams class with $\alpha_{k0} = -\alpha_{k1} = 1$, $\alpha_{ki} = 0$ for all $i > 1$ and the backward differentiation formula with $\beta_{k0} \neq 0$ and $\beta_{ki} = 0$ for all $i > 0$. One disadvantage with variable order codes based on multistep methods is that they always start the integration with low order formulae. This makes restarting (over discontinuities) more expensive.

5.1. Codes based on Adams method

The Adams–Bashforth formula of order k can be expressed as

$$y_{n+1} = y_n + h_n \sum_{i=1}^{k} \beta_{ki} f_{n-i+1}. \tag{5.2}$$

This is an explicit formula which is generally used as a predictor for the implicit Adams–Moulton equation of order $k + 1$:

$$y_{n+1} = y_n + h_n \left(\sum_{i=1}^{k} \beta_{ki}^* f_{n-i+1} + \beta_{k0}^* f_{n+1} \right). \tag{5.3}$$

Current Adams codes would perform as follows: predict y_{n+1}^p by (5.2), evaluate $f_{n+1} \equiv f(x_{n+1}, y_{n+1}^p)$. Then correct the value y_{n+1} by (5.3) and follows with another evaluation of $f(x_{n+1}, y_{n+1})$. This scheme is referred to as PECE (Predict, Evaluate, Correct, Evaluate) method, and it is intended for nonstiff problems. Shampine and Gordon [43] have written a variable order code with formulae up to order 12 based on this approach. This code was published in their book [43]. A new version of the code called DEABM was written as part of DEPAC developed at Sandia Laboratories mentioned earlier.

Gear's well-known code, DIFSUB [19], and its successors, GEAR [23], EPISODE [27], and LSODE [25], written by Hindmarsh, contain different implementations of the Adams formulae which are available in the nonstiff option of the code selected by the user. The stiff option of these codes will be discussed below. The code DGEAR in the IMSL library [30] is based on the GEAR code.

5.2. Codes based on backward differentiation formulae

As mentioned earlier, well-known codes, such as EPISODE, contain two families of formulae, one for non-stiff and the other for stiff systems. The formulae are Adams methods and the stiff formulae are given by

$$y_{n+1} = \sum_{i=1}^{k} \alpha_{ki} y_{n-i+1} + h_n \beta_{k0} f_{n+1}. \tag{5.4}$$

This equation can be solved easily for f_{n+1} in terms of the previous and current values of y; thus, it is called backward differentiation formula (BDF). For $k = 1, 2$ the formulae are L-stable, for $3 \leqslant k \leqslant 6$, the formulae are stiffly stable of order k. Therefore, the main drawback of the BDF's is when they are used to solve problems having complex eigenvalues lying near the imaginary axis (for example, problem B5 in the stiff test sets proposed by Enright et al. [16]). The unstable regions extended into the left-half plane get substantially larger for higher k so that most codes implementing BDF restrict $k \leqslant 6$.

The codes DIFSUB, GEAR, LSODE, and DGEAR (in IMSL) are all similar in the stiff option of the packages. In these codes, the stepsize h_n is fixed for a prescribed number of steps. The values of y_i at points which are not former mesh points are obtained by interpolating the previously calculated values of y_i. A modified version of LSODE, called DEBDF, was developed as a member of the Sandia DEPAC package [45].

EPISODE is different from other packages, in that the step-size h_n is allowed to change at each step. This feature makes EPISODE much more effective for problems with sharp fronts (for example, problems involving chemical kinetics systems with diurnally varying reaction rates, which vary like a square wave). GEAR, DIFSUB are completely unreliable for such problems. The fixed stepsize-interpolation strategy does have the advantage that the α's and β's for each family can be computed and stored in tables once and for all, since they do not vary with n. Whereas, in EPISODE, at each step, the α's and β's must be calculated for the formula in use. Furthermore, in EPISODE the iteration matrix involved in the modified Newton scheme for solving the BDF's must be frequently computed and decomposed because the scalar coefficient of the Jacobian has become out of date; whereas, other packages would not require this since the coefficient is varying less frequently. In summary, the variable step strategy of EPISODE permits it to solve certain class of problems effectively. However, the additional overhead involved in computing the coefficients α's, β's and in reevaluating the iteration matrix can cause EPISODE to perform less efficiently than GEAR (DIFSUB, LSODE) for smoothly decaying or linear systems.

Some special codes, EPISODEB, GEARB [24] and an option of the code DEBDF are developed for systems with the Jacobian matrix having a banded structure. These systems appear for example in the method of lines and finite differences to solve PDE's. These packages take advantage of the structure of the Jacobian and reduce both time and space complexities of the modified Newton method for solving the BDF's, therefore EPISODEB could solve a larger banded system than EPISODE.

For large stiff systems of ODE's having a sparse Jacobian structure the code GEARS written by Sherman and Hindmarsh [46] uses the Yale sparse matrix package. The code GEARZ written by Carver [10] uses the Curtis–Reid sparse matrix routines and finally the code FACSIMILE developed by Curtis [13] uses Duff's MA28 sparse matrix routines.

Recently Hindmarsh [26] put together a collection of codes called ODEPACK. One of the most recent additions to ODEPACK is code LSODA. This code automatically determines whether or not a problem is stiff and switches to the most appropriate set of formulae.

6. Other multi-step methods

The cyclic composite multistep method described by

$$\sum_{j=i-k}^{i} \alpha_{ij} y_{ms+j} = h \sum_{j=1}^{i} \beta_{ij} f_{ms+j} \quad \text{for } i = 1,\ldots,s,$$

was studied by Tendler, Bickart, and Picel [47]. These formulae define a block of s forward values of y: $y_{ms+1},\ldots,y_{(m+1)s}$ with each application of the procedure. The matrix β_{ij} has a lower triangular form, thus we have to solve s systems of N nonlinear equations instead of solving sN nonlinear equations. A code named STINT was written by the authors which uses stiffly stable formulae of orders 1 to 7 with better stability properties than BDF's.

The multistep, second derivative methods were investigated by Enright [14]. Formulae of orders 2 to 7 based on the form:

$$y_{n+1} = y_n + h \sum_{i=0}^{k} \beta_{ki} f_{n-i+1} + h^2 \gamma_{k0} y''_{n+1},$$

were developed and implemented in a code SDBASIC. These formulae are all stiffly stable with better stability properties than BDF codes.

Cash [11] uses an extended BDF of the type

$$y_{n+1} = \sum_{i=1}^{k} \alpha_{ki} y_{n-i+1} + h(\beta_{k0} f_{n+1} + \beta_{k1} f_{n+2}).$$

His program includes the conventional BDF's as a predictor and the above extended BDF as the corrector. He was able to develop L-stable schemes of orders up to 4 and $A(\alpha)$-stable schemes of orders up to 9. Recently, he extended the above formula to include second derivatives. He was then able to obtain L-stable formulae up to order 6 and $A(\alpha)$-stable for formulae order 7 to 9.

The major drawback of multistep methods in general is that they are more expensive to get started. All of the codes mentioned in this section start with a low order method and a very small step-size, then gradually increase the order and the step-size as the integration progresses.

7. Exponential-fitted methods

Liniger and Willoughby [36] coined the term 'exponential-fitted' to describe a class of algorithms designed to exactly satisfy the stability test equation $y' = \lambda y$ for systems having one or more large negative eigenvalues—that is, for stiff systems of ODE's. The derivation presented here is a considerably modified version of Liniger and Willoughby's concepts, drawing on subsequent work by Lambert [35], by Brandon [2,5] and by Pratt [39].

Following Lambert, we derive some simple exponential-fitted algorithms for curve-fitting by assuming an interpolating function and determining the free parameters by the method of

undetermined coefficients.

Let us assume a three-parameter exponential interpolant.

$$I(x) = A + Be^{Cx} \tag{7.1}$$

which interpolates the solution of (1.1) over the interval $(x_n, x_n + h)$ as follows:

$$I(0) = y_n = y(x_n), \tag{7.2a}$$

$$I'(0) = f_n = (dy/dx)_{x_n}, \tag{7.2b}$$

$$I(h) = y_{n+1} = y(x_n + h). \tag{7.2c}$$

The first two requirements determine two of the three free parameters:

$$A = y_n - f_n/C, \tag{7.3a}$$

$$B = f_n/C. \tag{7.3b}$$

So that, with (7.3) substituted in (7.1), together with (7.2c), there follows:

$$y_{n+1} = y_n + hf_n\left[\frac{e^{Ch} - 1}{Ch}\right]. \tag{7.4}$$

Miranker (1981) [37] refers to (7.4) as a 'filtered Euler' approximation. We note that the free parameter C has yet to be determined.

Three possible ways for determining C are of interest.

$$I'(h) = f_{n+1} \quad \rightarrow \quad C = h_n^{-1} \ln(f_{n+1}/f_n), \tag{7.5a}$$

$$I'(-h) = f_{n-1} \quad \rightarrow \quad C = h_{n-1}^{-1} \ln(f_n/f_{n-1}), \tag{7.5b}$$

$$I''(0) = f_n' \quad \rightarrow \quad C = f_n'/f_n. \tag{7.5c}$$

With the substitution of (7.5a), (7.4) is an implicit, single-step integration algorithm. With either (7.5b) or (7.5c), (7.4) is an *A-stable explicit* integration algorithm.

Note that the explicit stiffness measure (3.6) does not apply to (7.4) because it has an infinite stability radius for negative C.

It is also interesting to note that, for the conventional assumption of a three parameter polynomial interpolant, the requirements (7.2) and (7.5) result in three familiar second-order integration algorithms: (7.5a) gives the implicit Adams–Moulton method or trapezoidal rule, (7.5b) and (7.5c) result in the explicit Adams–Bashforth and Taylor's methods, respectively.

Our strategy is to take advantage of the filtering or damping factor in (7.4) only when the parameter C is negative, and to use conventional, low-order 'polynomial-fitted' methods when C is positive. To achieve this, we define a 'tunable trapezoid' approximation,

$$y_{n+1} = y_n + h[Uf_{n+1} + (1 - U)f_n] \tag{7.6}$$

where the component-specific 'tuning factor' U is a degree-of-implicitness factor, which is permitted to vary between one-half (trapezoidal rule) and unit (implicit Euler approximation).

Equating (7.6) to (7.4) and solving for U, there results

$$U = \frac{1}{Ch} + \frac{1}{1 - e^{Ch}}, \quad C < 0 \tag{7.7}$$

when $C \geqslant 0$, we use $U = \frac{1}{2}$, and revert to the trapezoidal rule.

With (7.7) to define U in (7.6), together with either of the two explicitly determined constants C, (7.5b) or (7.5c), this yields an exponential-fitted implicit method, in which the degree-of-implicitness factor U is determined *explicitly*.

Liniger and Willoughby give an estimate of the leading-term local truncation error for (7.6),

$$\text{LTE} \sim h^2 f'(\theta)\left(\tfrac{1}{2} - U\right), \qquad U \neq \tfrac{1}{2}, \quad 0 < \theta < h. \tag{7.8}$$

When $U = \frac{1}{2}(C > 0)$, the trapezoidal rule leading term LTE estimate, $\text{LTE} \sim \frac{1}{12}h^3 f''(\theta)$, applies.

A predictor-corrector version of the XFTR (exponential-fitted trapezoidal rule) is appropriate when the system is nonstiff: Equation (7.4) is used as a predictor, with C determined by (7.5b or 7.5c); the corrector, (7.6) with U determined by (7.7), is iterated to convergence by some form of functional iteration: Jacobi, Gauss–Siedel or Jacobi–Newton.

Accuracy is monitored by (7.8), similar to (3.1), with h_{acc} given by

$$h_{\text{acc}} = \left\{ \frac{\tau}{\| f'(\theta)\left(\frac{1}{2} - U\right) \|} \right\}^{1/2} \tag{7.9}$$

and h_{iter} is determined by the rate of convergence of the particular convergence method chosen.

Brandon [5] uses the full-step/half-step algorithm to find h_{acc}, but also (conservatively) assumes effective second-order accuracy to determine h_{acc}. When $(h_{\text{acc}}/h_{\text{iter}})$ is greater than unity, Newton iteration is used directly on (7.6) without a predictor to achieve convergence.

With C determined by (7.5b), it is unnecessary to evaluate the Jacobian except for occasional updating if Newton iteration is used to converge (7.6). If (7.5c) is used to determine C, the Jacobian must be computed at the beginning of each timestep, as with implicit or semi-implicit RK methods.

Brandon evaluated the Jacobian at every iteration of every step in order to improve the accuracy of U by recomputing (7.7) with the implicit approximation:

$$C = \tfrac{1}{2}\left\{ \frac{f_n'}{f_n} + \frac{f_{n+1}'}{f_{n+1}} \right\}. \tag{7.10}$$

However, the LTE estimate (7.8) is not significantly improved by the use of (7.10), so that this practice does not appear to be in general computationally efficient.

Computer codes based on this class of methods were written by Brandon [5] called IMP and by Pratt [39] called CREK-1D. Both codes were developed for solving chemical kinetic problems.

We first observe that all the problems of the test set by Enright et al. [16] are simple to compute the particle derivatives. The chemical kinetics problems can be written in general as:

$$y' = Ap, \quad y(0) \text{ given},$$

where A is an $M \times N$ matrix and $p_j = k_j \prod_{i=1}^{M} y_i^{r_{ij}}$; $j = 1, \ldots, N$, while $r_{ij} \geqslant 0$ and $k_j > 0$ are rate constants. Comparing this equation to equations (1)–(3b) in Pratt's paper [39], we see that they are equivalent, matrix A containing terms dependent on the temperature.

For problems of this kind, the computation of partial derivatives is quite simple, since

$$\partial p_j / \partial y_i = r_{ji}\left(p_j / y_i\right).$$

The Rosenbrock and implicit XFTR methods do not require, therefore, much extra work due to the construction of the Jacobian matrix.

Acknowledgments.

The work of T.D. Bui was supported by the Natural Sciences and Engineering Research Council of Canada under Grant No. A-9265, and the Programme de Formation de Chercheurs et d'Action Concertée (FCAC) of Québec under Grant No. EQ-1438. The work of A.K. Oppenheim was supported by the Office of Energy Research, Basic Energy Science, Engineering, Mathematics, and Geosciences Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098, and by the National Science Foundation under Grant CPE-8115163. The work of D.T. Pratt was supported by the NASA-Lewis Research Center under Grant No. NAG3-227.

References

[1] R. Alexander, Diagonally implicit Runge–Kutta methods for stiff ODEs, *SIAM. J. Numer. Anal.* **6** (1977) 1006–1021.

[2] P.D. Babcock, L.F. Stutzman and D.M. Brandon, Improvements in a single-step integration algorithm, *Simulation* **33** (1979) 1–10.

[3] D.G. Bettis, Efficient embedded Runge–Kutta methods, *Numerical Treatment of Differential Equations*, Lecture Notes in Mathematics **631** (Springer, New York, 1978) 9–18.

[4] T.A. Bickart, An efficient solution process for implicit Runge–Kutta methods, *SIAM J. Numer. Anal.* **6** (1977) 1022–1027.

[5] D.M. Brandon, A new single-step implicit integration algorithm with A-stability and improved accuracy, *Simulation* **23** (1974) 17–29.

[6] T.D. Bui and T.R. Bui, Numerical methods for extremely stiff systems of ordinary differential equations, *Appl. Math. Modeling* **3** (1979) 355–358.

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

[8] J.C. Butcher, A transformed implicit Runge–Kutta method, *Math. Comput.* **26** (1979) 731–738.

[9] J.C. Butcher, K. Burrage and F.H. Chipman, STRIDE: Stable Runge–Kutta integrator for differential equations, Report 150, Dept. of Mathematics, Univ. of Auckland, Auckland New Zealand, 1979.

[10] M.B. Carver, In search of a robust integration algorithm for general library use: some tests, results and recommendations, working papers for the 1979 SIGNUM meeting on numerical ODEs, Report 963, Dept. of Computer Science, Univ. of Illinois, Illinois, 1979.

[11] J.R. Cash, The integration of stiff initial value problems in ODEs using modified extended backward differentiation formulae, Dept. of Math., Imperial College, London, England, 1983.

[12] M. Crouzeix, Sur la B-stabilité des méthodes de Runge–Kutta, *Numer. Math.* **32** (1979) 75–82.

[13] A.R. Curtis, The FACSIMILE numerical integrator for stiff initial value problems, AERE-R.9352, A.E.R.E. Harwell, Oxfordshire, 1978.

[14] W.H. Enright, Second derivative multistep methods for stiff ordinary differential equations, *SIAM J. Numer. Anal.* **2** (1974) 321–331.

[15] W.H. Enright, Improving the efficiency of matrix operations in the numerical solution of stiff ordinary differential equations, *ACM Trans. Math. Software* **4** (1978) 127–136.

[16] W.H. Enright, T.E. Hull and B. Lindberg, Comparing numerical methods for stiff systems of ODEs, *BIT* **15** (1975) 10–48.

[17] W.H. Enright and M.S. Kamel, Automatic partitioning of stiff systems and exploiting the resulting structure, *ACM Trans. Math. Software* **5** (1979) 374–385.

[18] G.E. Forsythe, M.A. Malcolm and C.B. Moler, *Computer Methods for Mathematical Computations* (Prentice-Hall, Englewood Cliffs, NJ, 1977).

[19] C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations* (Prentice-Hall, Englewood Cliffs, NJ, 1971).

[20] B.A. Gottwald and G. Wanner, A reliable Rosenbrock integrator for stiff differential equations, *Computing* **26** (1981) 355–362.

[21] E. Hairer and G. Wanner, Multistep-multistage-multiderivative methods for ordinary differential equations, *Computing* **11** (1973) 287–303.
[22] E. Hairer and G. Wanner, A theory for Nyström methods, *Numer. Math.* **25** (1976) 383–400.
[23] A.C. Hindmarsh, GEAR: Ordinary differential equation system solver, Report UCID-30001, Lawrence Livermore Laboratory, Livermore, California, 1974.
[24] A.C. Hindmarsh, GEARB: Solution of ordinary differential equations having banded Jacobian, Report UCID-30059, Lawrence Livermore Laboratory, Livermore, California, 1976.
[25] A.C. Hindmarsh, LSODE and LSODI, Two new initial value ordinary differential equation solvers, *ACM-SIGNUM Newsletter* **15** (1980) 10–11.
[26] A.C. Hindmarsh, ODEPACK, A systematized collection of ODE solvers, in: R.S. Stepleman, Ed., *Numerical Methods for Scientific Computation* (1983) to appear.
[27] A.C. Hindmarsh and G.D. Byrne, EPISODE: An effective package for the integration of systems of ordinary differential equations, Report UCID-30112, Lawrence Livermore Laboratory, Livermore, California, 1977.
[28] N. Houbak and P.G. Thomsen, SPARKS: A FORTRAN subroutine for the solution of large systems of stiff ODEs with sparse Jacobians, Report NI-79-02, Institute for Numerical Analysis, Tech. Univ. of Denmark, Lungby, Denmark.
[29] T.E. Hull, W.H. Enright and K.R. Jackson, User's guide to DVERK—a subroutine for solving non-stiff ODEs, Report 100, Dept. of Computer Science, Univ. of Toronto, Toronto, Canada, 1976.
[30] International Mathematical and Statistical Library, Houston, Texas.
[31] P. Kaps, Rosenbrock type methods, Numerische Verfahren zum Lösen von steifen Anfangswertproblemen, Mathematisches Forschungsinstitut Oberwolfach, 1981.
[32] P. Kaps and P. Rentrop, Generalized Runge–Kutta methods of order four with step size control for stiff ordinary differential equations, *Numer. Math.* **33** (1979) 55–68.
[33] P. Kaps, S. Poon and T.D. Bui, Rosenbrock methods for stiff ODEs—a comparison of Richardson extrapolation and embedding technique, *Computing* (1984) to appear.
[34] P. Kaps and G. Wanner, A study of Rosenbrock-type methods of high order, *Numer. Math.* **38** (1982) 279–298.
[35] J.D. Lambert, *Computational Methods in Ordinary Differential Equations* (Wiley, London, England, 1973).
[36] W. Liniger and R.A. Willoughby, Efficient Integration methods for stiff systems of ordinary differential equations, *SIAM J. Numer. Anal.* **7** (1970) 47–66.
[37] W.L. Miranker, *Numerical Methods for Stiff Equations and Singular Perturbation Problems* (Reidel, Boston, MA, 1981).
[38] S.P. Norsett, Semi-explicit Runger–Kutta methods, Mathematics and Computation Report 6, Univ. of Trondheim, Trondheim, Norway, 1974.
[39] D.T. Pratt, CREK-1D: A computer code for transient, gas-phase combustion kinetics, Dept. of Mech. Eng., Univ. of Washington, WA, 1983.
[40] L.F. Shampine, Type-insensitive ODE codes based on implicit A-stable formulas, Math. Comput. **36** (1981) 499–510.
[41] L.F. Shampine, Type-insensitive codes based on implicit $A(\alpha)$-stable formulas, Math. Comput. **39** (1982) 109–123.
[42] L.F. Shampine, Measuring stiffness, Sandia National Lab. report SAND 83-119, 1983.
[43] L.F. Shampine and M.K. Gordon, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem* (Freeman, San Francisco, CA, 1975).
[44] L.F. Shampine and H.A. Watts, The art of writing a Runge–Kutta code, III. *Appl. Math. and Comput.* **5** (1979) 93–121.
[45] L.F. Shampine and H.A. Watts, DEPAC—Design of a user oriented package of ODE solvers, Report SAND 79-2374, Sandia National Laboratories, Albuquerque, New Mexico, 1980.
[46] A.H. Sherman and A.C. Hindmarsh, GEARS: A package for the solution of sparse stiff ordinary differential equations, in: A.M. Erisman et al., eds., *Electrical Power Problems: The Mathematical Challenger* (SIAM, Philadelphia, PA, 1980).
[47] J.M. Tender, T.A. Bickart and Z. Piel, A stiffly stable integration process using cyclic composite methods, *ACM Trans. Math. Software* **4** (1978) 339–368.
[48] J.M. Varah, On the efficient implementation of implicit Runge–Kutta methods, *Math. Comput.* **33** (1979) 557–561.
[49] J.H. Verner, Families of imbedded Runge–Kutta methods, *SIAM, J. Numer. Anal.* **16** (1979) 857–875.
[50] J. Villadsen and M.L. Michelsen, *Solution of Differential Equation Models by Polynomial Approximation* (Prentice-Hall, Englewood Cliffs, NJ, 1978).