An MEBDF Package for the Numerical Solution of Large Sparse Systems of Stiff Initial Value Problems

T. J. ABDULLA, J. R. CASH
Department of Mathematics, Imperial College
London SW7 2BZ, England

M. T. DIAMANTAKIS
The Meteorological Office, London Road
Bracknell, Berkshire RG12 2SZ, England

(Received May 2000; revised and accepted September 2000)

Abstract—An efficient algorithm for the numerical integration of large sparse systems of stiff initial value ordinary differential equations (ODEs) and differential-algebraic equations (DAEs) is described. The algorithm is constructed by embedding a standard sparse linear algebraic equation solver into a suitably modified MEBDF code. An important practical application of this algorithm is in the numerical solution of time dependent partial differential equations (PDEs), particularly in two or more space dimensions, using the method of lines (MOL). A code based on this algorithm is illustrated by application to several problems of practical interest and its performance is compared to that of the standard code LSODES. © 2001 Elsevier Science Ltd. All rights reserved.

Keywords—MEBDF, Large sparse systems of stiff IVPs and differential-algebraic equations, Time dependent PDEs.

1. INTRODUCTION

In this paper, we will be concerned with the numerical solution of the initial value problem

\[ M \frac{dy}{dx} = f(x,y), \quad y(x_0) = y_0. \]  

(1.1)

We will be particularly concerned with two important cases.

(i) \( M \equiv I \) and so we have a system of initial value problems which we will assume to be stiff.

(ii) The matrix \( M \) is singular so that we have a system of differential-algebraic equations (DAEs).

Most, if not all, of the efficient numerical methods for the solution of stiff systems of the form (1.1) are implicit and, as a direct result, it is necessary to solve a nonlinear system of algebraic equations at each time step in order to compute the required numerical solution. In the present paper, we will be concerned with the case where the Jacobian matrix \( \frac{df}{dy} \) is very large and sparse. Such
systems can arise quite naturally in practical applications. They also arise from the use of the method of lines (MOL) in the solution of time dependent PDEs. If the PDE being solved has just one space dimension, then the system of ODEs to be solved is normally banded and this is relatively easy to solve using standard software (an efficient banded MEBDF [1] solver is already available from the web page of one of the present authors [2]). If, however, the PDE has more than one space dimension, then the resulting system of ODEs is generally sparse and this presents a much more formidable problem to initial value solvers.

A crucial component in developing an efficient sparse ODE solver is, of course, the derivation of an effective solver for the large sparse system of nonlinear algebraic equations occurring at each time step. There are at least two different approaches to this problem. The first is to use an iterative technique such as Gauss-Seidel, SOR, or Krylov methods [3]. This approach is very efficient as regards storage space, can often be implemented efficiently on a parallel computer, particularly if Jacobi iteration is used, and may often be the only realistic option in three space dimensions. However, the obvious disadvantage is that in some cases, it may experience convergence difficulties. The second approach is to base the algebraic equation solver on a Newton iteration scheme and to use a standard sparse equation solver to solve the resulting system of linear algebraic equations. Such an approach will normally require more storage than is required by an iterative solver, due to the possibility of fill in, but will enjoy the well-known advantages of a direct, as opposed to an iterative, solver. A major disadvantage of this approach is that there may be convergence difficulties with the Newton iteration. In this paper, we will concentrate on the use of a direct solver. However, our code is written in a modular form so that it is relatively straightforward for a user to replace the direct solver by an iterative one if so required.

Most direct solvers for sparse linear algebraic equations are based on Gaussian elimination and they usually employ a pivoting strategy which is a compromise between pivoting solely for stability and pivoting to avoid fill in. In this paper, we will concentrate on two sparse solvers, namely the Yale solver YSMP [4] and the Harwell solver MA28 [5]. As we shall see, we choose the Yale solver for ODEs and the Harwell solver for differential-algebraic equations (DAEs). Another very efficient solver is the special purpose solver developed in [6]. We have not implemented it in our code because it requires the Jacobian to be stored as a full $N \times N$ matrix and this is not practical for the problems we are interested in. In the next section, we will briefly describe the MEBDF approach for differential-algebraic equations. A detailed analysis can be found in [7].

2. MODIFIED EXTENDED BACKWARD DIFFERENTIATION FORMULAE

Modified extended backward differentiation formulae (MEBDF) [1] were originally proposed for the numerical solution of stiff initial value problems (IVPs) in an attempt to derive a class of multistep integration formulae which have better stability properties and higher orders of accuracy than standard backward differentiation formulae (BDF). The basic MEBDF algorithm was subsequently modified to produce the code MEBDFDAE [2] which is applicable to linearly implicit differential-algebraic equations of index $\leq 3$, as well as to stiff IVPs. In an extensive comparison of codes [8], it was found that MEBDFDAE compares very favourably with certain other state of the art codes on a large set of challenging problems. Further evidence of the good performance of the MEBDF codes can be found on the web page of one of the present authors [2]. At the present time, the main options allowed by the MEBDFDAE code are to specify either full or banded Jacobians which are computed either numerically or analytically. The purpose of the present paper is to extend these options to allow the solution of large sparse systems of initial value problems both for ODEs and DAEs. Having done this, our aim is to compare the performance of the sparse version of MEBDF with that of the sparse BDF code LSODES [9]. In particular, we will examine what the more stable, higher-order but more expensive MEBDF
approach has to offer in the method of lines (MOL) solution of time dependent PDEs after the PDE has been semidiscretized in space.

In what follows, we will explain the MEBDF algorithm for differential-algebraic equations. For ease of presentation, we will consider the one-step case, that is, the case where only one past value $y_n$ is used. The general MEBDF algorithm consists of the following three stages.

STAGE 1. Assuming the data $(x_n, y_n)$ are known, we compute $\bar{y}_{n+1}$ from

$$M \left( \bar{y}_{n+1} - y_n \right) = hf \left( x_{n+1}, \bar{y}_{n+1} \right),$$

using a Newton iteration scheme. Once convergence to a solution $\bar{y}_{n+1}$ has been obtained from the Newton iteration, we compute

$$\bar{y}'_{n+1} = \frac{\bar{y}_{n+1} - y_n}{h} \quad \text{and} \quad f \left( x_{n+1}, \bar{y}_{n+1} \right) = M\bar{y}'_{n+1}.$$

STAGE 2. Compute $\bar{y}_{n+2}$ from

$$M \left( \bar{y}_{n+2} - \bar{y}_{n+1} \right) = hf \left( x_{n+2}, \bar{y}_{n+2} \right).$$

Similarly to Stage 1, once convergence has been obtained to $\bar{y}_{n+2}$, we compute

$$\bar{y}'_{n+2} = \frac{\bar{y}_{n+2} - \bar{y}_{n+1}}{h} \quad \text{and} \quad f \left( x_{n+2}, \bar{y}_{n+2} \right) = M\bar{y}'_{n+2}.$$

STAGE 3. Compute $y_{n+1}$ from

$$M \left( y_{n+1} - y_n - \frac{h}{2} \left[ -f \left( x_{n+2}, \bar{y}_{n+2} \right) + f \left( x_{n+1}, \bar{y}_{n+1} \right) \right] \right) = hf \left( x_{n+1}, y_{n+1} \right).$$

Note that this formula is the one-step MEBDF of order 2. This is explained, for example, in Hairer and Wanner [1, pp. 267-270]. As a first approximation to $y_{n+1}$ and $f(x_{n+1}, y_{n+1})$, we use $\bar{y}_{n+1}$ and $f(x_{n+1}, \bar{y}_{n+1})$ from Stage 1. This then completes one step forward of the MEBDF algorithm.

For reasons explained more fully in [10], Stages 1 and 3 are normally relatively cheap compared with Stage 2 and so the total work in implementing a full MEBDF step, consisting of the above three individual stages, is normally not much more than for a single BDF step. Indeed, if the Newton iteration in Stage 3 converges in just one iteration, as is often the case, then Stage 3 requires no extra function evaluations, but simply calls for the solution of a linear system

$$LUy_{n+1} = \text{RHS},$$

where the triangular factors $L$ and $U$ are known. The hope is that the extra accuracy and stability enjoyed by MEBDF will more than compensate for the extra computational effort required compared to a single BDF step. It is clear that MEBDF have higher accuracy than the BDF since a $k$-step MEBDF is of order $k+1$. Also, the MEBDF have considerably enhanced stability properties (see [1, pp. 267-270]) and, in particular, are $A$-stable with order up to and including 4.

In the actual implementation of our sparse solvers, we were guided by the work of Thompson [11]. He looked very carefully at the effect of pivoting in the sparse algebraic equation solver used in LSODES. He came to the important conclusion "that partial pivoting is not necessary in an adaptive solver such as LSODES". This conclusion has also been reached by other researchers who give a partial explanation for this [6]. We too implement our solver without pivoting. We tested two versions of our code for sparse ODEs, one using the Yale solver and the other using
MA28. We found that the Yale solver without pivoting was the more efficient algorithm on all problems we tried and this is the option we chose.

For differential-algebraic equations, however, the situation is quite different. When dealing with sparse systems of ODEs, the argument for not pivoting is essentially that the coefficient matrix of the Newton iteration scheme is

$$ P = I - \alpha h \frac{\partial f}{\partial y}, $$

and if $h$ is reasonably small, then this will be close to the (well-conditioned) identity matrix. However, in the case of differential-algebraic equations, the Newton iteration matrix is

$$ P = MI - \alpha h \frac{\partial f}{\partial y}, $$

where the matrix $M$ is singular. Typically, some of the diagonal elements of $M$ are zero, and in this case,

$$ P_{ii} \approx -\alpha h \frac{\partial f_i}{\partial y_i}, $$

for some $i$. $P_{ii}$ can, of course, be very small for small $h$ and can often be expected to be zero for sparse systems. It follows that in the case of sparse DAEs, the use of partial pivoting is vital for some problems.

We implemented both the MA28 and the Yale sparse solver in a MEBDF code for DAEs. Our practical experience indicated that the MA28 code was often the more efficient of the two and this is the option we chose. Practical experience has also shown that the sparse DAE code can fail miserably if we do not allow the sparse linear algebraic equation solver to pivot [7]. In the next section, we will give some numerical results to illustrate the performance of the sparse MEBDF code in the numerical solution of three time-dependent PDEs.

3. NUMERICAL RESULTS

In this section, we will illustrate the performance of the code MEBDFSO (which is the sparse version of the MEBDF code for solving stiff initial value problems) in solving some sparse systems of ODEs. A full description of these problems, together with many further examples of the performance of the codes for both sparse systems of ODEs and DAEs are available from [2]. Also available from [2] are the codes and drivers which produced these results. Our conclusions to this section will be that if excellent stability is not very important (i.e., it is not vital to use $A$-stable formulae in the time integration), the right-hand side of (1.1) is relatively cheap to evaluate or low accuracy is acceptable, then LSODES is often very effective. However, when excellent stability is essential or high accuracy is needed, then the MEBDF code often performs very well compared with LSODES. We will not give any examples of the solution of DAEs, since we do not know of a widely available code for sparse DAEs with which to compare our results, and results illustrating the performance of MEBDF are already available [2]. Instead, we will be concerned with the solution of the sparse systems of ODEs resulting from the MOL solution of PDEs.

There are already several good software packages available for the solution of time dependent PDEs (including PDECOL [12] and PDETWO [13]). It is not our intention in this paper to compete with these codes, but instead to compare the performance of MEBDFSO and LSODES on some large stiff initial value problems. We wish to emphasise again the importance of our algorithms in the solution of multidimensional problems. Two of our test problems are one dimensional and so relatively easy to solve using standard software. However, these problems still represent useful test problems since we have regarded the systems as being sparse rather than banded so that our sparse algorithms can be used and the extension of our drivers to deal with multidimensional problems is straightforward. Similarly, with Problem 1, the algebraic systems
have a very definite structure since the spatial domain is the unit square. However, we feel that
this is also a useful test problem since we have not sought to exploit this structure in any way
and so any symmetry is immaterial. However, the problem of deriving a sparse solver which can
exploit symmetry is an interesting question for future research.

The first problem we consider is the following.

**Problem 1. Burgers' Equation in Two Dimensions.**

\[
U_t = -UU_x - UU_y + \mu \cdot (U_{xx} + U_{yy}) \tag{3.2}
\]

in the range \(0 < x < 1, 0 < y < 1,\) and \(0 < t < 2.\) The exact solution to this equation (subject
to appropriate initial and boundary conditions) is

\[
U(x,y,t) = \frac{1}{(1 + \exp(x/(2 \cdot \mu) + y/(2 \cdot \mu) - 2t/(4 \cdot \mu)))}. \tag{3.3}
\]

The initial and boundary conditions for (3.2) are specified by imposing the exact conditions for

\[
U(x,y,0), \quad 0 < x < 1, \quad 0 < y < 1,
\]

\[
U(0,y,t), U(1,y,t), \quad \text{for } 0 < y < 1, \quad t > 0,
\]

\[
U(x,0,t), U(x,1,t), \quad \text{for } 0 < x < 1, \quad t > 0.
\]

The results we present are for the case \(\mu = 0.1,\) and we take 31 spatial points in both the \(x\) and
\(y\) directions. This gives rise to a sparse system of 961 ordinary differential equations. We obtained
the space derivative \(U_x\) by using a fourth-order five-point biased upwind approximation and
\(U_{xx}\) was approximated by using a fourth-order five-point centred approximation. These spatial
derivatives were obtained by using Schiesser's library routine DSS034 and subordinate routines
DSS004 and DSS020. These approximations are described in detail in [14,15]. Similar formulae
were used to obtain approximations to the \(y\) derivatives. Having performed the semidiscretization
of (3.2), in this way, we integrated the resulting sparse system of differential equations using the
MEBDFSO code with tolerances

\[
\text{Tol} = 10^{-i}, \quad i = 2, 3, 4, \ldots, 10. \tag{3.4}
\]

![Figure 1](image-url) Numerical solution of Burgers' equation in 2D, with \(\mu = 0.1.\) MEBDFSO
(solid line with +) and LSODES (broken line with *).
We then plotted graphs of time taken against accuracy obtained at $x = 1/2 = y$. The results obtained are given in Figure 1. It is important to realise that the accuracy given is that in solving the system of ODEs resulting from the semidiscretization of (3.2) rather than the accuracy obtained in the solution to (3.2). By adopting this approach, we do not have to be concerned with the effect of spatial discretization errors. This is consistent with the aims of this paper, since we are mainly concerned with the integration of stiff initial value problems and regard the balancing of spatial and temporal errors in PDEs beyond our scope at present. Here CPU denotes the run time in seconds and SCD denotes the number of significant correct digits obtained in the solution at $(1/2, 1/2, t)$ for all $t$ grid points in $0 \leq t \leq 2$. As can be seen, the two codes are competitive with each other with LSODES being superior at low tolerances and MEBDFSO becoming superior at intermediate tolerances.

**PROBLEM 2. THE CHEMICAL FLOODING PROBLEM.** (See [16].) This equation is given by

$$U_t = DU_{xx} - U_x, \quad -2 < x < 2, \quad 0 < t < 2. \tag{3.5}$$

Subject to appropriate boundary conditions, the analytic solution to this problem is

$$U(x, t) = \frac{1}{\sqrt{1 + 4Dt}} e^{-(x-t)^2/(1-4Dt)}. \tag{3.6}$$

We took $D = 0.00002$ and the initial and boundary conditions were obtained by specifying the exact solution (3.6) for

$$U(x, 0), \quad -2 < x < 2,$$

$$U(-2, t), U(2, t) \quad \text{for } t > 0.$$

Of course, it could be argued that with such a small value of $D$, it would be preferable to use an upwinding difference scheme. This is true for advection diffusion problems of this type. However, we wish to consider the case where we do not want to apply any analysis to this problem, but instead wish to apply our discretization scheme and ODE solver as a black box since we feel that many users wish to use numerical integrators in this way. This problem has been used by Hindmarsh [17] to illustrate the problem of instability for ODE solvers. In [17], some suggestions are made to overcome the stability problems, but it is not clear how successful these are in general. Another approach is to use a more stable method such as one based on MEBDF. To obtain the space derivatives in (3.5), we used a centred cubic spline approximation (the DSS038 routine) [15, Chapter 4]. The mesh spacing $\Delta x$, in the space dimension was chosen to be 0.002. This produced a system of 2001 IVPs. (There is no particular reason for this choice of $\Delta x$. We have chosen it simply for illustrative purposes, but any other value would suffice.) Again, we ran this problem using the MEBDFSO code with the tolerances (3.4). The results are given in Figure 2. We see that at low accuracy, the two codes are comparable but that there is a rapid deterioration in the performance of LSODES as increased accuracy is requested. This behaviour is exactly as anticipated with the relatively good performance of the MEBDF being due to the fact that they are more stable than the BDF for order $\geq 3$. In particular, we see from Figure 2 the rapid deterioration in the performance of LSODES at about SCD = 4.25. This is an indication of the stability limitation of the BDF and occurs because LSODES is choosing high-order BDF with poor stability and so is being forced to use a very small steplength of integration.

**PROBLEM 3. THE KDV EQUATION.** The classical KdV equation [18] is given by

$$U_t + 6UU_x + U_{xxx} = 0, \quad -\infty < x < \infty, \quad t > 0. \tag{3.7}$$

Subject to appropriate boundary conditions, the exact solution of (3.7) is given by

$$U(x, t) = \frac{1}{2} c \text{sech}^2 \left( \frac{1}{2} \sqrt{c}(x - ct) \right). \tag{3.8}$$
This solution describes a soliton traveling from left to right with a velocity \( c \) and a height \( (1/2)c \).

Following [19], we replace the given infinite space domain by

\[-30 \leq x \leq 70,\]

and we take as the initial condition

\[U(x, 0) = \frac{1}{2} c \text{sech}^2 \left( \frac{1}{2} \sqrt{c} x \right).\]  \hspace{1cm} (3.9)

Some numerical experiments involving the MOL solution of the KdV equation are given in [20]. These involve the use of a whole variety of approximations to the spatial derivatives, as well as the use of a range of initial conditions. In what follows, we give the results obtained when the derivative \( U_x \) is approximated using a fourth-order five-point biased upwind approximation using Schiesser's library subroutine DSS034 and the derivative \( U_{xxx} \) is approximated using a fourth-order seven-point centred approximation given by Fornberg [21]. This gives rise to a sparse system of ordinary differential equations and we integrated these using MEBDFSO for the following three cases.

CASE 1. We take \( c = 1 \) in the initial condition (3.9). This system describes the motion of a single soliton moving from left to right.

CASE 2. We take the initial condition

\[U(x, 0) = \frac{1}{2} \text{sech}\left( \frac{x}{2\sqrt{2}} \right).\]

This describes the motion of two solitons which are moving at the same speed and do not interact.

CASE 3. We take the initial condition

\[U(x, 0) = \frac{1}{4} \text{sech}^2 \left( \frac{x}{2\sqrt{2}} \right) + \text{sech}^2 \left( \frac{x}{\sqrt{2}} \right).\]

This corresponds to the case where there are two solitons, one moving with speed \( c = 1/2 \) and the other with \( c = 2 \) so that they interact. The space discretization used for this problem was \( \Delta x = 1/4 \), so there are 401 differential equations to be solved and we integrated these for...
$0 < t < 2$. The results for the solution of these three problems are given in Figures 3-5. It can be seen that the MEBDSO and LSODES codes are comparable for low tolerances but, as we would expect, the MEBDSO code becomes superior as more accuracy is required.

This relative behaviour of the two codes is of course to be expected. Due to the better stability and greater accuracy of the MEBDF, we would expect the MEBDSO code to be superior when excellent stability is needed and/or high accuracy is required. When low accuracy is acceptable, the LSODES code often only uses order 1 and 2 formulae and these are $A$-stable. This is in contrast to MEBDF which are $A$-stable up to order 4. This accounts for the observed behaviour of LSODES both on the problems presented in this paper and the many other problems given in [2].

Figure 3. Numerical solution of KdV equation with one soliton. MEBDSO (solid line with +) and LSODES (broken line with x).

Figure 4. Numerical solution of KdV equation with two equal solitons. MEBDSO (solid line with +) and LSODES (broken line with x).
REFERENCES