Applied Numerical Mathematics 5 (1989) 375–397
North-Holland

DEVELOPING SOFTWARE FOR TIME-DEPENDENT PROBLEMS USING THE METHOD OF LINES AND DIFFERENTIAL-ALGEBRAIC INTEGRATORS

M. BERZINS and P.M. DEW
School of Computer Studies, The University, Leeds, United Kingdom LS2 9JT

R.M. FURZELAND *
Thornton Research Centre, Shell Research Limited, Chester, United Kingdom

The method of lines is one of the most powerful tools for the solution of time-dependent coupled ODE/PDE systems. The attraction of this method is that the complex systems of coupled ordinary and partial differential equations arising in mathematical modelling can be solved by using the sophisticated software which has been developed for initial value differential-algebraic equations. The SPRINT software of Berzins, Dew and Furzeland [2] has been developed specifically for the method of lines. This software contains a selection of spatial discretisation methods, time integrators and linear algebra routines. These components together with utility routines for spatial remeshing and discontinuity detection form an open-ended "tool-kit" for the method of lines. The purpose of the paper is to use the SPRINT software to illustrate some of the issues that arise in the use and development of algorithms and software which employs the method of lines.

1. Introduction

The SPRINT package (software for PROblems IN Time), is a general-purpose computer program for the numerical solution of mathematical models that involve mixed systems of time-dependent algebraic, ordinary and partial differential equations (ODEs and PDEs). The software is the result of joint research between Shell Research Limited and the School of Computer Studies at Leeds University. The aim of the research is to provide a flexible and open-ended software tool to enable a user to solve a wide range of problems within a single framework. The design philosophy is described in Berzins, Dew and Furzeland [2].

The constuction of the software involved the identification of a suitable problem class of differential-algebraic equations and the development of algorithms and software to efficiently cater for these problems. The software package consists of a set of well-defined and independent modules that are controlled by a supervisory routine. The internal structure of the package allows the individual modules to be easily replaced and in this way the user has access to different combinations of modules from the three main component areas in the package—the time integration method, the spatial discretisation method and the linear algebra routines. The modules incorporate recent developments in numerical analysis and software such as ODE integrators for differential-algebraic equations (DAEs) and for handling discontinuities, type-insensitive codes for ODEs where the degree of stiffness varies, and adaptive space remeshing methods for PDEs in one space dimension.

* Present address: Koninklijke/Shell Laboratorium, Amsterdam, Netherlands.

The paper provides an overview of the SPRINT software and discusses the difficulties in writing such general-purpose software. In particular, the conflicting requirements of developing an open-ended software architecture while providing a concise user interface are partially resolved by providing two main levels of user interface. These interfaces—a general low-level interface which permits the mathematical modeller to have a high degree of flexibility and a high-level interface, named SPRITE, which allows the user easy access to a fixed range of package options—are briefly described in the paper. Further details of the interfaces can be found in [4].

General-purpose software which is based on differential-algebraic equations integrators requires the provision of a number of features which are not needed in standard ODE integrators. The SPRINT software offers the user a range of options for solving differential-algebraic equations. In order to use the software efficiently the user needs to be aware of which options are important for the solution of such equations. The two options described here are a method for the estimation of the initial values of the solution and its time derivative and a local error estimator.

The general applicability of SPRINT has led to many uses in mathematical modelling in the petrochemical industry. We shall use two examples of such models to illustrate the flexibility of the present software and to consider the requirements of future software.

A large problem class of interest is combustion modelling, in which fluid dynamics plays an important role in determining temperature and concentration distributions. In this case the mixed PDE/DAE system is of the diffusion convection reaction nature. These models are used to study the efficiency of combustion both in combustion burners and in internal combustion engines with spark ignition and/or fuel injection. The models also provide diagnosis of hazard conditions e.g. auto-ignition along hot surfaces, and can be used to simulate situations which would be too hazardous to perform experimentally. The flexibility that the software must have to efficiently solve such problems is illustrated by applying some of the options within SPRINT, including spatial remeshing, to a simple model of flame propagation in a combustion chamber.

Another important application area is two-phase fluid flow in which problems such as vapour-liquid evaporation and condensation arise, e.g. bubble growth or collapse in liquefied natural gases. A simplified model of bubble collapse will be used to illustrate the complex nature of such applications and to show the areas in which the SPRINT software needs to be improved to solve such problems reliably.

2. An overview of the SPRINT software

2.1. Differential-algebraic equations problem class

The core of the software package is a versatile set of differential-algebraic integrators with the flexibility to deal with stiff or nonstiff DAEs coupled with algebraic equations and full, banded and sparse Jacobian matrices computed analytically or numerically. Each integrator is designed to solve the class of ODE initial value problems defined by

$$f(\dot{y}, y, t) = g(y, t) - A(y, t)\dot{y} = 0 \tag{2.1}$$

with the initial condition

$$y(0) = K. \tag{2.2}$$

The square matrix A may be singular indicating a differential-algebraic system of equations. In the special case when A is the identity matrix equation (2.1) is said to be written in *normal form*.

The advantage of the problem class defined by (2.1) over fully implicit ODE problems is that equation (2.1) is linear with respect to the time derivative, i.e.

$$\partial f/\partial \dot{y} = -A(y, t). \tag{2.3}$$

This means that the user interface in the software only requires the definition of the matrix-vector product $-A(y, t)\dot{y}$. It is then possible to provide codes based upon (say) the backward differentiation formulas or the theta method of Prothero and Robinson [26] which are almost as efficient as those for normal form problems. This is because there is no need to calculate and store the matrix $\partial f/\partial \dot{y}$.

2.2. Solving differential-algebraic equations by calling the SPRINT driving routine

In order to solve differential-algebraic equations by calling the SPRINT driving routine the user is required to write a FORTRAN-77 program. The SPRINT driving routine is open-ended in that it is largely independent of the time integration and linear algebra routines. This allows extra routines to be added as the need arises without modifying the SPRINT driving routine and provides the user with a choice of linear algebra and time integration routines. The different parameters required by these routines makes it difficult to design a single interface that can deal with all the different possibilities. The solution adopted in the software is to have setup routines for the linear algebra and the time integrator (and for the PDE routines discussed in Section 2.7).

The alternative is for the user to call the SPRITE routine, described below in Section 2.8, which calls the setup routines with default values for the parameters and then calls the SPRINT driving routine. Although this option is suitable for many users there is inevitably some loss of flexibility. For instance Section 3.3 provides an example of an error estimate option that is specified in a call to a setup routine and which the user may require when solving differential-algebraic equations but which may not be needed for ordinary differential equations.

The greatest flexibility is obtained when the user writes a program which calls the SPRINT driving routine directly and consists of the following:

(i) Initialisations of the parameters to be passed into the SPRINT driving routine.

(ii) Calls to the linear algebra module setup routine and the DAE integrator setup routine.

(iii) A RESID routine (see Section 2.3) that describes the form of the differential-algebraic equation and also provide an optional MONITR routine. The MONITR routine is called after each step taken by the integrator and gives the user the opportunity to perform intermediate output or calculations. Alternatively MONITR routines are provided for tasks such as the estimation of the global error, discontinuity handling, spatial remeshing and parameter sensitivites.

(iv) A call to the SPRINT driving routine to perform the integration. This call specifies the names of the time integration module, LU decomposition and backsubstitution routines, RESID routine and the name of the MONITR routine.

The structure of the user's calling program and of the underlying SPRINT software is illustrated in Fig. 1. The diagram reflects the novel internal structure of the software in that the time integration module, the nonlinear equations solver and the problem description routines are quite separate and communicate with each other only through the main driver using the reverse

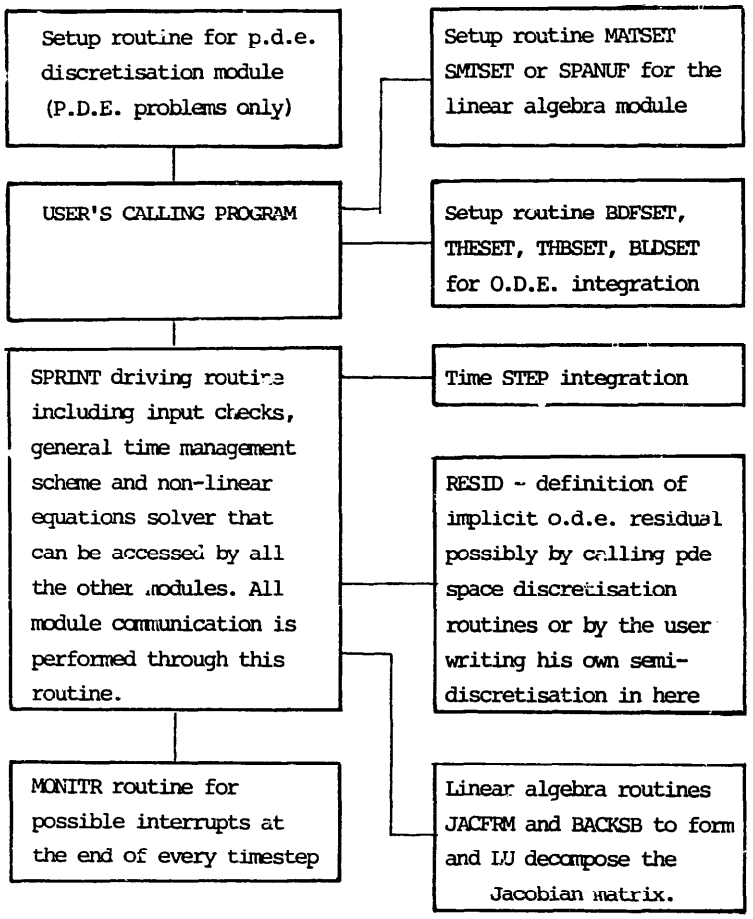

Fig. 1. An overview of the SPRINT package.

communication approach described by Berzins, Dew and Furzeland [2]. In practice low-level user programs are built up from example programs such as those supplied with the software [4].

2.3. The RESID problem description routine

The user-supplied RESID routine defines the system of differential-algebraic equations to be solved. The integrator supplies approximate vectors for the solution and its time derivative, y and $\dot{y}$. The main purpose of the RESID routine is to compute the residual vector r which is obtained by substituting the vectors y and $\dot{y}$ into the DAE system that is being solved. I.e. for (2.1),

$$r = g(y, t) - A(y, t)\dot{y}. \tag{2.4}$$

It is also required that the $\dot{y}$-dependent parts of the residual can be computed by a call to RESID and also returned to SPRINT via the vector r, i.e.

$$r = -A(y, t)\dot{y}. \tag{2.5}$$

One of the parameters that SPRINT passes into the RESID routine is the integer IRES; if this is set to 1 then the user must supply the form of the residual defined by equation (2.4), and if it set

to -1 equation (2.5) must be used. Sections 3.2.1 and 3.3 below provide instances of cases when the RESID routine is called with IRES $= -1$. The integration may be interrupted by the user changing the value of IRES in RESID to force the integrator to either stop the integration, reduce the time-step to avoid a physically impossible solution value or to terminate the current step and enter the MONITR routine.

The form of RESID required is:

```
      SUBROUTINE RESID (NEQ, T, Y, YDOT, R, IRES, WKRES, NWKRES)
      INTEGER NEQ, NWKRES, IRES
      DOUBLE PRECISION T, Y(NEQ), YDOT(NEQ), R(NEQ), WKRES(NWKRES)
C     THE ARRAY WKRES(NWKRES) IS A USER-DEFINED WORKSPACE
      IF (IRES.EQ.-1)
        THEN
          ...for I=1,NEQ set R(I) to be the second form of the residual, as in equation (2.5). Note if
             no time derivatives are present in the Ith equation then set R(I) = 0.0D0.
             RETURN
        ELSE
          ...for I=1,NEQ set R(I) to be the full residual, as in equation (2.4)
             RETURN
      END IF
      END
```

2.4. DAE step integration modules

The first release of the package contains four DAE *step* integrators all of which are capable of solving DAEs of the form (2.1). These are:

(i) The SPGEAR module which implements both the family of Adams' methods up to order 12 and the family of backward differentiation formula (BDF) methods up to order 5. This module was developed from the LSODI code of Hindmarsh [18], but includes a modified step size/order selection algorithm (based on [22]) which improves the performance of the code on the type of ODEs discussed by Gaffney [16] (see Berzins [7]).

(ii) The STHETA module, which is based on the theta method codes of Prothero and Robinson [26] and Chua and Dew [11], is a stiff integrator designed for low to medium accuracy requirements.

(iii) The STHETB module which is a type-insensitive (stiff/nonstiff) Theta method code for problems with variable stiffness during the course of integration (see Berzins and Furzeland [6]).

(iv) The SBLEND module is also designed to cope with both stiff and nonstiff equations by blending the formulas in the SPGEAR code in the manner of Skeel and Kong [29]. This results in formulas with better stability regions than BDF formulas.

Prior to the first call of SPRINT a corresponding setup routine must be called for each of the integrators. The ease with which integrators can be changed encourages the user to experiment to see which is best suited to the problem. Although we have found that all these modules perform well on all the differential-algebraic equations that we have encountered in practical applications we have since found that none of the codes is particularly successful on the index-2 differential-algebraic equations discussed by Gupta, Gear and Leimkuhler [17].

2.5. Linear algebra modules

All the time integration methods above are implicit methods and so at each time step a system of nonlinear equations must be solved, usually by a variant of Newton's method. This means that a system of linear equations must be solved at each iteration. The sparsity pattern of these equations may vary from being completely full to very sparse. This is taken into account by the full, banded and sparse linear algebra routines in SPRINT. For each type of Jacobian matrix the matrix handling is split into three routines—a setup routine for validation of inputs, a lower/upper decomposition routine, and a backsubstitution (solution) routine.

The subset of full and banded matrix LINPACK routines [13] and the Yale sparse matrix package (YSMP) of Eisenstat et al. [14] as used in the LSOD* integrators of Hindmarsh [18] and the MA28 sparse matrix routines of Duff as used by Berzins, Brankin and Gladwell [8] are implemented in SPRINT. Before the first call to SPRINT the common setup routine (MATSET) for the full and banded matrix routines must be called. Alternatively, before the first call to SPRINT the sparse matrix setup routine SMTSET for the Yale routines or SPANUF for MA28 must be called. In both the sparse cases the user can force the sparsity pattern to be automatically updated by a call to an auxiliary routine and can also obtain diagnostic information about the sparse Jacobian matrix.

2.6. Partial differential equations problem class

In the method of lines the partial differential equations are spatially discretised over NPTS points using finite difference, finite element or collocation methods. This discretisation results in a system of NPTS nonlinear, coupled DAEs for each given PDE and provides a unified approach to solving mixed systems of DAEs and PDEs. The SPRINT software provides routines to spatially discretise PDEs with one space dimension. However, no SPRINT routines are available for problems in two or more space dimensions and in this case the user must supply the discretisation.

In the case of one-space-dimensional problems it is expected that most users of SPRINT will use the system routines to perform the spatial discretisation. This involves writing a few simple subroutines to describe the PDE in terms of a *master equation* format given by

$$\sum_{p=1}^{\mathrm{NPDE}} C_{j,p}(x, t, \boldsymbol{u}, \boldsymbol{u}_x, \boldsymbol{v})\frac{\partial u_p}{\partial t} + Q_j(x, t, \boldsymbol{u}, \boldsymbol{u}_x, \boldsymbol{v}, \dot{\boldsymbol{v}})$$

$$= x^{-m}\frac{\partial}{\partial x}\left(x^m R_j(x, t, \boldsymbol{u}, \boldsymbol{u}_x, \boldsymbol{v})\right),$$

$$j = 1, \ldots, \mathrm{NPDE}, \tag{2.6}$$

where m is an integer (usually $m = 0$, 1 or 2) which denotes the space geometry type, and the dots denote time derivatives. The vector $\boldsymbol{v}$ and its time derivative $\dot{\boldsymbol{v}}$ are assumed to be defined by a coupled ODE system (see equation (2.8)). The vector $\boldsymbol{u}(x, t)$ is defined by

$$\boldsymbol{u}(x, t) = [u_1(x, t), \ldots, u_{\mathrm{NPDE}}(x, t)]^{\mathrm{T}},$$

the vector $\boldsymbol{u}_x(x, t)$ is similarly defined, The function $R_j(\ )$ can be thought of as a flux which is

used also in the definition of the boundary conditions. For the jth PDE the boundary conditions have the form:

$$\beta_j(t)R_j(x, t, u, u_x, v) = \gamma_j(x, t, u, u_x, v, \dot{v}). \tag{2.7}$$

The master equation for the coupled ODEs is

$$f(v, \dot{v}, \xi, u^*, u_x^*, R^*, u_t^*, u_{xt}^*) = 0, \tag{2.8}$$

where this system of equations is assumed to be linear in the derivatives $\dot{v}$, u_t^* and u_{xt}^*. The arrays

$$u^*, \quad u_x^*, \quad R^*, \quad u_t^*, \quad u_{xt}^*$$

are all of dimension (NPDE, NXI) and hold the solution, flux and derivative values at the array of NXI coupling points ξ which are a set of NXI distinct points defined by

$$a \leqslant \xi_1 < \xi_2 < \cdots < \xi_{\mathrm{NXI}} \leqslant b.$$

These coupling points are independent of the spatial mesh points used by the spatial discretisation routines.

This choice of problem class was influenced by the work of Schryer [28] who shows that multi-phase PDE problems and PDE problems with coupled moving boundaries can all be formulated as coupled systems of ODEs and PDEs. The problem class described above is more restrictive than that of Schryer [28] as we wish to ensure that the ODE system is of the form of equation (2.1). An example of the broad range of problems that fall within this problem class is provided in Section 5.

2.7. *The* SPRINT *spatial discretisation routines*

There are currently two discretisation modules in the software: SPDIFF, a lumped finite element method developed by Skeel and Berzins [30] and SGENCO, the C^0 collocation discretisation of Berzins and Dew [3]. Each of these modules has a setup routine which performs the initialisation tasks and which must be called before the SPRINT driving routine is entered. The three other main components are the RESID routine discussed above, the MONITR routine that is called at the end of every time step and an interpolation routine that can be used to generate extra solution values after the required output time has been reached.

The SPDIFF discretisation method is analogous to the usual central, three-point finite difference formula for problems in Cartesian coordinates. However, for problems in polar and spherical coordinates the three-point formula is suitably modified to maintain second-order accuracy. An option is provided within this module to allow the user to adaptively vary the space mesh in time (see Section 4).

The collocation discretisation module SGENCO offers a family of high-order formulae based on Chebyshev polynomials. The user can select the order of approximation to be used and also whether the approximation to the solution of the PDE consists of one global polynomial or of a piecewise polynomial. It is necessary to define the degree of polynomial ($\geqslant 1$) used to approximate the solution between the breakpoints. The formulae ensure that the solution possesses only C^0 continuity at each breakpoint regardless of the degree of polynomial used. The power of the collocation method, however, lies in the ability to use high-order polynomials and there is no

pre-set upper limit to the degree of polynomial that can be used. The user interface to this routine is almost identical to that of the finite difference module; however, no adaptive space mesh option has yet been developed.

In order to use the spatial discretisation routines the user has to perform a similar set of operations to those needed for solving ODEs except that:

(i) An initialisation routine must be called for the spatial discretisation module.

(ii) It is no longer necessary for the user to provide a RESID routine or a MONITR routine.

(iii) The user must provide one routine to describe the PDE (equation (2.6)), one routine to describe the boundary conditions (2.7) and one (optional) routine to describe the coupled ODEs (2.8) (if any). In addition routines for initial conditions of the PDE variables and a routine (optional) for the ODE variables must also be provided.

(iv) In the case of the SPDIFF finite difference module when spatial remeshing is being used a routine must be provided to describe the form of the remeshing indicator (monitor) function and also a remeshing setup routine must be called.

In practice user programs for PDE problems are built up from a catalogue of example programs supplied with the software [4].

2.8. The SPRITE interface

SPRITE is an "easy to use" high-level interface routine to the SPRINT driving routine which avoids some of the setup calls and is thus more in line with standard library packages such as the NAG Library. The routine contains the setup calls for the linear algebra routines and the time integrators as well as the call to the SPRINT driving routine itself. This allows the routine to be used as a high-level FORTRAN interface to the software but has the disadvantage that the default parameters used in the setup calls inside SPRITE may not be suitable for all applications. A companion routine for initialising PDE setup routines (SETPDE) is also provided to allow SPRITE to be used for PDEs. Appendix A contains a skeleton driving program which shows how straightforward it is to solve the example PDE in Section 5 using SETPDE and SPRITE. It is expected that once users have become familiar with using SPRITE they will make use of the greater flexibility provided by the low-level interface and call the SPRINT driving routine directly.

3. Practical aspects of using SPRINT to solve DAEs

In order to help the user cope with the particular difficulties that may arise in trying to solve differential-algebraic equations a number of options and features within SPRINT have been devised which may help the user. In this section two of these features are described as follows. The general form of the equations that are solved within SPRINT is described and this description is used to help explain two options available in the software which the user must be aware of when solving differential-algebraic equations. These are the method of initialising the integration and the method of estimating the local error.

3.1. The nonlinear equations solved in SPRINT

The implicit time integration methods used in SPRINT all approximate the time derivative at a given time t_n by

$$\dot{y}(t_n) = \frac{y(t_n) - z}{h\gamma}, \quad h = t_n - t_{n-1}. \tag{3.1}$$

where the vector z depends on solution values at previous times and the constant γ depends on the integration method being used. For all these methods the following system of nonlinear equations has to be solved at each time step for the new solution vector y.

$$-h\gamma\left[g(y, t_n) - A(y, t_n)\frac{(y - z)}{h\gamma}\right] = 0. \tag{3.2}$$

The algorithm used in SPRINT to solve this system of equations is based on our practical experience of solving differential-algebraic equations and on the theoretical justification provided by Petzold and Lotstedt [25]. In the case when algebraic equations are present in (2.1) one or more of the equations may not depend upon any of the time derivatives. In this case the equations of the system (3.2) should not be multiplied by $-h\gamma$ as is normally done [25]. This row scaling procedure is implemented by defining a vector $\boldsymbol{d}$ in the following way:

$$d_i = \begin{cases} 1, & \text{if the } i\text{th equation contains a time derivative,} \\ 0, & \text{otherwise.} \end{cases}$$

The use of this vector enables us to treat the algebraic equations more efficiently when solving the system of equations (3.2). The ith equation of (3.2) can then be rewritten as

$$\left[-\sum_{j=1}^{neq} A_{i,j}(y(t), t)\frac{(y_j - z_j)}{h\gamma} + g_i(y, t)\right](-h\gamma d_i - 1 + d_i) = 0. \tag{3.3}$$

The indicator array, $\boldsymbol{d}$, is checked periodically throughout the integration to ensure that the equations have not changed from being algebraic to differential or vice versa. This is not expensive as we only have to determine which of the equations do not depend upon any of the time derivatives. In the case when the matrix $A(y, t)$ is singular but has no empty rows, no algebraic equations are isolated and the procedure reduces to that used by Petzold [22].

The system of equations (3.3) is solved by using the modified Newton's method used by most stiff ODE integrators (see Shampine [27]). The (i, j)th component of the Jacobian matrix, $J_{i,j}$, is then defined by

$$J_{i,j} = \left[-\sum_{k=1}^{neq} \frac{\partial A_{i,k}}{\partial y_j}\frac{y_k - z_k}{h\gamma} - \frac{1}{h\gamma}A_{ij}(y, t) + \frac{\partial g_i}{\partial y_j}\right](-h\gamma d_i + 1 - d_i). \tag{3.4}$$

Further details of how the Newton method is implemented are provided by Berzins et al. [7].

3.2. Calculating the initial values and starting integration

The difficulties encountered in estimating the initial values of differential-algebraic equations are documented by Gupta, Gear and Leimkuhler [17]. In particular the user-supplied initial values may not satisfy the algebraic equations and all the time derivatives may not be explicitly

defined as the matrix $A(y, t)$ in equation (2.1) may be singular. The need for good approximations to the initial solution values and their derivatives is so that the integrator can move away from possibly inconsistent initial values as smoothly as possible. In this section we shall describe the approaches used by others to overcome this problem, describe the approach adopted in SPRINT, explain the difficulties that may arise and describe the option that is available to deal with these difficulties.

The approach used by Dew and Walsh [12] was to calculate the initial values of the algebraic part of the differential-algebraic system of equations separately. This approach is only applicable when the equations can be clearly split into an algebraic and a differential part, as when the matrix $A(y, t)$ in equation (2.1) has linearly dependent nonzero rows. An alternative approach used by Petzold [22] is to take a small step with the backward Euler method and to solve the resulting system of equations for the solution values and the approximations to their derivatives by using a damped Newton method. The obvious disadvantage of this procedure is that the derivative values for the algebraic components (and possibly some of the differential components also) will contain errors of $O(1/h)$, where h is the stepsize, when the initial values of the algebraic components are in error. In addition in the case when the matrix A in equation (2.1) is singular but has no zero rows it may not be clear which are the "differential" components and which are the "algebraic" components.

3.2.1. The SPRINT initialisation algorithm

Practical experience with the class of problems handled by SPRINT has suggested the following algorithm. In the first instance an attempt is made to use functional iteration to compute the initial values of the time derivatives and the initial values of the algebraic components as this will rapidly converge for ODE equations in normal form and even for some algebraic equations. If this iteration fails to converge the procedure adopted is to take two equally small steps with the backward Euler method. On each step the system of equations for both the unknown solution (and derivative) values is solved by using a damped Newton iteration. The local error is estimated only on the second step so as to allow for the case when inexact initial values are provided for the "algebraic" components. The damping factor is varied using strategies taken from boundary value problem solvers, see Baker and Phillips [1], many more iterations than usual are allowed and the Jacobian matrix may be updated between iterations. The values of the time derivatives which result are based on the sum of the corrections to the initial values. In the case when the initial values do not satisfy the algebraic equations the solution and derivative values obtained for the algebraic components are again in error on the first step by a factor of $O(1/h)$. At the end of the second step all the solution values should be correct as should the derivatives, unless the index of the DAE system is two or more [17].

In this last case in order to compute more realistic initial derivative values we have devised the following option within SPRINT. Let $\dot{z}$ be the first estimates for the time derivatives at the end of the first backward Euler step. The final estimates which are used are given by $\dot{y}$ where

$$J\dot{y} = \left[A(y, t)\dot{z} + \dot{f}\right], \tag{3.5}$$

and J is the (already factored) Jacobian matrix of equation (3.2) and the ith component of the vector $\dot{f}$ is defined by

$$\dot{f}_i = \frac{\partial f_i(\dot{y}, y, t)}{\partial t}(d_i h - 1 + d_i) \tag{3.6}$$

where the vector $f(\dot{y}, y, t)$ is defined by (2.1) and d_i is the indicator array used in (3.3). This step may be explained as follows. Partition the vectors y in (2.1) into two components U and V so that equation (2.1) can be written as the pair of equations

$$A(t)\dot{U} = g_1(U, V, t) \tag{3.7}$$

and

$$0 = g_2(U, V, t), \tag{3.8}$$

where the matrix $A(t)$ may be singular. Equation (3.5) may then be written as

$$\begin{bmatrix} A(t) - h\partial g_1/\partial U & -h\partial g_1/\partial V \\ \partial g_2/\partial U & \partial g_2/\partial V \end{bmatrix} \begin{bmatrix} \dot{U} \\ \dot{V} \end{bmatrix} \approx \begin{bmatrix} A(t)\dot{U} + h\partial g_1/\partial t - h(\partial A/\partial t)\dot{U} \\ -\partial g_2/\partial t \end{bmatrix}. \tag{3.9}$$

The subtraction of the $O(h)$ term $hA(t)\ddot{U}$ from the right side of the top equation turns equation (3.9) into an equality that may be derived by differentiating equations (3.7) and (3.8) in time. The first-order backward Euler approximation used to calculate $\dot{U}$ in the right side of equation (3.9) also leads to an $O(h)$ error, i.e. $\frac{1}{2}h\ddot{U}$. The error in estimating the derivatives $\dot{y}$ by using the method of equation (3.5) could thus be expected to be $O(h)$. However, it is straightforward to construct index-2 DAEs for which the inverse of the J matrix has entries of $O(1/h)$, thus leading to $O(1)$ errors in $\dot{y}$. In this case the main function of the extra step is to try and filter out excessively large initial values caused by inconsistent initial values being supplied.

The cost of the extra step defined by (3.5) is two function evaluations to form the vector f defined by equation (3.6), "half" a function evaluation to calculate $A(t)\dot{U}$ by calling the RESID routine with IRES $= -1$ and one backsubstitution using the already factored Jacobian matrix. This procedure extends naturally to DAEs of the form of equation (2.1) and is also used when integration is restarted after several convergence or error test failures on a single step.

3.2.2. Initialisation example

We shall now use the second example problem of Dew and Walsh [12] to illustrate the effect of varying the way the initial values are calculated. The problem consists of the following pair of elliptic-parabolic PDEs:

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r^2\frac{\partial u}{\partial r}\right) = 4\alpha\left(v + r\frac{\partial v}{\partial r}\right)$$

and

$$(1 - r^2)\frac{\partial v}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial v}{\partial r} - rvu\right),$$

where $(r, t) \in [0, 1] \times [0, 1]$ and the boundary conditions are given by

$$u = \frac{\partial v}{\partial r} = 0 \quad \text{at } r = 0$$

and

$$\frac{\partial}{\partial r}(ru) = 0, \quad v = 0 \qquad \text{at } r = 1.$$

The initial conditions at $t = 0$ are given by

$$u = 2\alpha r, \quad v = 1 \qquad \text{for } r \in [0, 1].$$

Table 1
Initial step taken for Dew and Walsh problem

Initialisation method/Tol	0.1d-2	0.1d-4	0.1d-6
Two backward Euler steps with filter	0.287d-7	0.179d-10	0.489d-13
Two backward Euler steps only	0.349d-7	0.137d-8	0.125d-10
One backward Euler step with filter	0.241d-8	0.480d-12	0.821d-15
One backward Euler step only	0.585d-8	0.237d-10	0.602d-12

The problem was integrated using the SPGEAR integrator and using the SPDIFF spatial discretisation. The value $\alpha = 1$ was used in the problem definition. A mesh of 81 points was used with the circular mesh spacing employed by Dew and Walsh [12] to cluster the points towards the right side of the spatial interval. One difficulty with this problem is that the time derivative of v at the mesh point closest to the boundary is of $O(-1/\Delta x^2)$ which has a value of about -10^7 where Δx is the mesh spacing between the two rightmost mesh points. This means that both the algebraic (elliptic) and ODE (parabolic) components change significantly over the first step which is being used to calculate the initial values. In Table 1 we compare the size of the first time step successfully taken for this PDE. Tol is the relative error tolerance and was used in conjunction with an absolute error tolerance of 0.001 Tol. Table 1 shows that for a range of tolerances when the filter is not used the use of two backward Euler steps results in a larger initial stepsize being used than if a single backward Euler step is used, as in [7,22]. This has been confirmed on a number of other test problems. Although the filter is not beneficial in this case there are contrasting examples of problems for which the use of the filter results in a much larger stepsize being used regardless of whether one or two backward Euler steps are used. It is for this reason that we have included the filter as an option in SPRINT. Regardless of whether or not the filtered derivatives are adopted as the derivative values the norm of the filtered derivatives is used in the algorithm to automatically select the initial stepsize.

The comments in Section 2.3 also apply to the initialisation procedure. Although the procedure works well for differential-algebraic equations of index 1, the same cannot be said with confidence for index-2 problems. The development of an efficient and accurate general-purpose technique for determining the initial values of differential-algebraic equations is an outstanding research problem (see e.g. Leimkuhler, Petzold and Gear [20]).

3.3. Choice of local error estimate

The second option that the user must be aware of when solving differential-algebraic equations is concerned with local error estimation. The option is provided by the setup routine, named BDFSET, to the SPGEAR time integration module. Petzold [23] has shown that for some differential-algebraic equations of the type of equation (2.1) it is essential to modify the usual local error estimate vector of the backward differentiation method of order k, denoted by e^k, by solving the system of equations

$$J e^k_{\text{new}} = \left(A(y, t)e^k\right) \tag{3.10}$$

for the new estimate of the error, e^k_{new}. This estimate is relatively expensive because it involves part of a residual evaluation with IRES $= -1$ (with e_k substituted for $\dot{y}$, see Section (2.2)) to

evaluate the matrix-vector product $A(y, t)e^k$ and a backsubstitution using the LU factors of the Jacobian matrix J as defined by (3.4). It is because of this computational expense that the estimate is provided as an option in the setup routine BDFSET. Although the extra overhead of this error estimator does not sometimes appear to be computationally worthwhile, it is not difficult to find situations (e.g. see Petzold [23]) in which this error estimator is crucial to the success of the integration.

It should be noted that the other time integration methods used in SPRINT automatically have error indicators of the type defined by equation (3.10) (see Prothero and Robinson [26] and Skeel and Kong [29]).

4. Spatial remeshing using the MONITR routine

An important feature of SPRINT is the capability to handle both discrete and continuous remeshing schemes. After each step taken by the SPRINT integrator a routine, generic name MONITR, is called which allows the user to perform intermediate output or calculations (e.g. the integration may be restarted, the stepsize changed and the residual defined by equations (2.4) or (2.5) calculated). The unique feature of the MONITR routine is that it has the power to access the whole of the nonlinear equations solver in SPRINT. The MONITR routine was designed for tasks such as ODE global error estimation, discontinuity detection and discrete time remeshing. In discrete remeshing a new mesh is created at certain times in the integration (based on the current solution profile), the solution and its time derivatives are interpolated onto the new mesh and then the integration continued.

The SPDIFF discrete remeshing option was developed by Furzeland [15]. The user calls a setup routine to define when remeshing should take place and supplies an auxiliary subroutine that specifies the particular aspect—the remeshing monitor function $F_m(x, u)$—of the solution behaviour that he wishes to track. This function typically depends on the solution u and its space derivatives and may represent a measure of the spatial discretisation error. Using this monitor function as a guide, the remesh routines apply the ideas of Kautsky and Nichols [19] to construct a new mesh at the current time step which satisfies certain sensible criteria on the space mesh sizes and adjacent mesh ratios. The user interface to the monitor routine provides the user with solution values at mesh points, the mesh points and with flux values halfway between the mesh points. This provides sufficient flexibility in defining the function that dictates the shape of the mesh.

In order to supply the user with sufficient flexibility with regard to when spatial remeshing should take place the user can specify the following options through the setup routine.

(i) Specify remeshing after a fixed time interval.

(ii) Specify remeshing after a fixed number of integrator time steps.

(iii) Specify remeshing only if remeshing will move one mesh point by more than a given fraction of the old mesh spacing at either side of that point. This means that a possible new mesh must be computed after every time step or after a fixed number of time steps.

(iv) Use a fixed or variable number of mesh points. In the case when the number of points does not vary certain mesh points can be specified as fixed and are not changed by remeshing.

(v) Put a bound on how closely the monitor function is equidistributed by the new mesh.

Once it has been ascertained that remeshing should take place and the remesh routine has determined a new mesh the MONITR routine uses complete cubic spline interpolation to determine the solution, its time derivative and any other higher time derivatives used by the ODE integrator on the new mesh. Time integration then attempts to continue directly using the stepsize and order determined at the end of the step prior to remeshing. There are two cases to consider. The first is when the number of mesh points, and hence the size of the DAE system being integrated in time, is changed by the remeshing routine. In this case it is necessary to recompute the Jacobian matrix before integration can continue. Once the linear algebra routines have been initialised a change in the number of equations will automatically cause the initialisation process to be restarted. It is not necessary to restart the integration at the current time, though the MONITR routine could order this. The more usual case is when the number of mesh points is not changed by the remeshing routine. The conservative approach is to re-evaluate the Jacobian matrix on re-entry to the integrator though as the example problem below shows this is not necessarily the most efficient approach.

4.1. A test problem to illustrate remeshing and the flexibility of SPRINT

This flame propagation test problem arises from the modelling of the onset of ignition and subsequent flame propagation of a pre-mixed fuel/air mixture. The simplified, one-dimensional model presented by Furzeland [15] results in two coupled partial differential equations for the temperature, W, and mass fraction (concentration), V, of the single species undergoing ignition, viz.

$$\frac{\partial V}{\partial t} = \frac{\partial^2 V}{\partial x^2} - V \cdot K(W),$$

$$\frac{\partial W}{\partial t} = \frac{\partial^2 W}{\partial x^2} + V \cdot K(W),$$

where $K(W) = 14.6\, e^{4(1-1/(W+0.1))}$. The initial and boundary conditions are:

$$W(x,0) = 0, \quad V(x,0) = 1, \qquad 0 \leqslant x \leqslant 10$$

$$\frac{\partial V}{\partial x} = 0 \quad \text{at } x = 0, 10.$$

$$\frac{\partial W}{\partial x} = \begin{cases} -t/0.05 & 0 \leqslant t \leqslant 0.05, \\ -\cos(\pi(t-0.05)/1.9), & 0.05 \leqslant t \leqslant 3.08 \end{cases} \qquad \text{at } x = 0$$

$$\frac{\partial W}{\partial x} = 0 \quad \text{at } x = 10.$$

The question of interest is what spark ignition strength, as modelled by the heat input at $x = 0$, is needed to ignite and maintain flame propagation.

A space mesh of 41 equally spaced points is used along with the SPDIFF discretisation. This gives rise to a system of 82 ordinary differential equations in time which are solved using a local error tolerance of 10^{-4} and a mixed error test. The numerical results given in Table 2 compare the theta codes STHETA and STHETB with the backward differentiation module SPGEAR. In the case of SPGEAR the maximum order of the method, as specified in the setup routine, was restricted to

Table 2
Integrator statistics for the flame problem

NPTS	Mesh	Method	Steps	FUN	JAC	CPU	ModCPU
		STHETA	105	381	17	2.94	4.26
41	Fixed	STHETB	124	385	8	1.97	3.31
		SPGEAR	104	435	24	3.26	4.68
	Adapt/	STHETA	139	639	40	1.48	
15	Forced	STHETB	144	680	30	1.55	
	Jacob.	SPGEAR	154	770	49	1.74	
		STHETA	137	454	17	1.11	
15	Adaptive	STHETB	176	570	15	1.34	
		SPGEAR	157	536	25	1.37	
		STHETA	149	517	19	2.63	4.85
41	Adaptive	STHETB	159	567	22	3.10	5.49
		SPGEAR	141	553	25	2.76	5.02

3 so that the storage overhead of the three integrators was identical. In Table 2, ModCPU is the CPU time using the IBM FortVS compiler and CPU is the CPU time with the optimising switch OPT(2) on the same compiler.

In the fixed mesh case the switching code STHETB is more efficient than the STHETA code because it only needed to switch to the Newton method when the boundary condition at $x = 0$ changes at $t = 0.05$ and switched back to functional iteration at $t = 0.66$. The Newton method was used from $t = 1.0$ until the end of integration.

Furzeland [15] has shown how this problem can be solved more efficiently by using the discrete remeshing approach to adapt the spatial mesh every four time steps by using the monitor function

$$F_m(x, V, W) = \left|\frac{\partial^2 W}{\partial x^2}\right| + \left|\frac{\partial^2 V}{\partial x^2}\right|.$$

This procedure results in a much smaller minimum mesh spacing if 41 spatial mesh points are used than the equispaced mesh used earlier. Figure 2 shows typical meshes and solution profiles at times 0.3 and 1.0. The stiffness of the ODE system integrated is proportional to $1/\Delta x_i \Delta x_{i+1}$ where Δx_i is the mesh spacing between x_i and x_{i+1}. The value of this ratio for the fixed mesh of spacing 0.25 is 16 and the maximum value for the adaptive mesh is approximately 500. In order to keep the minimum mesh sizes comparable we use an adaptive mesh of 15 points which results in a minimum space size of 0.085 at about $t = 0.5$ and 0.24 at later time.

The results for the adaptive mesh version of the problem are shown in Table 2 under the heading "Adapt/Forced Jacob.". In this case the Jacobian matrix was re-evaluated after remeshing. The increased stiffness results in the STHETB code switching to the Newton iteration at $t = 0.015$ due to the failure of functional iteration to converge. At a later time, $t = 0.66$, the code correctly takes advantage of the decreasing stiffness of the problem by switching back to functional iteration when the minimum mesh size starts to increase. A final switch back to the Newton method is made about $t = 1.0$.

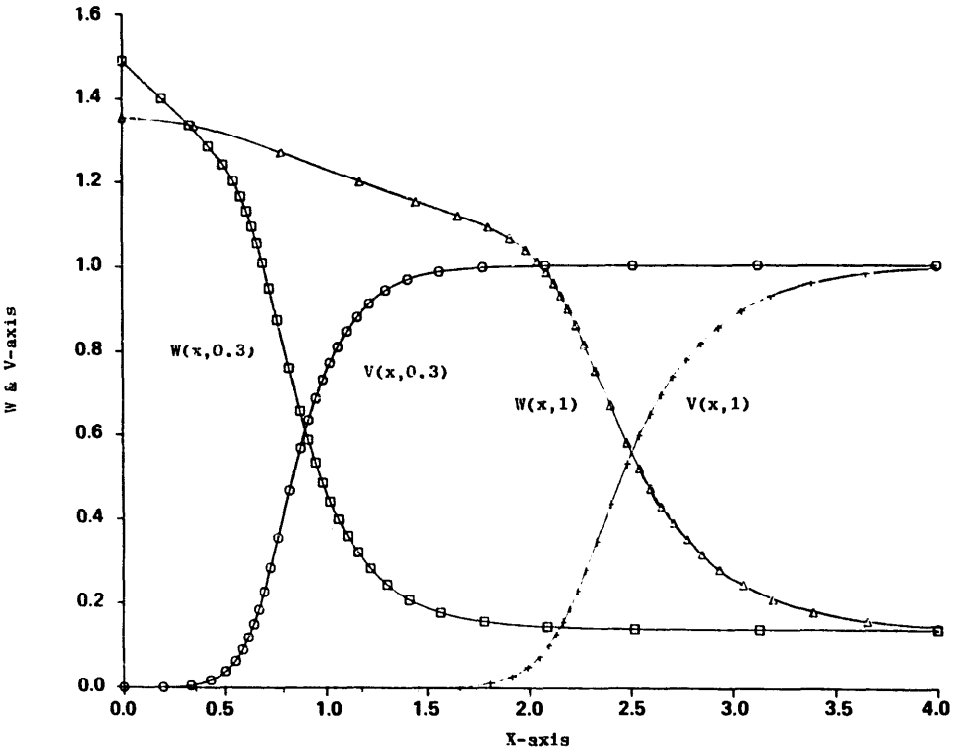

Fig. 2. Temperature (W) and concentration (V) profiles at $t = 0.3$ and 1.0.

The alternative approach of not re-evaluating the Jacobian matrix after the remeshing algorithm is called gives the results in Table 2 under the heading "Adaptive". This shows that for this particular problem the most efficient strategy is not to re-evaluate the Jacobian until this is deemed necessary by the DAE integrator. Using this strategy we can now compare the overhead of remeshing by using 41 mesh points, adapting the mesh and only re-evaluating the Jacobian when necessary. The bottom entries in Table 2 show that an important feature of the method is that the extra computational cost when compared to the fixed mesh approach is about one third. It should also be noted that using the optimising compiler the adaptive mesh option executed more quickly than the fixed mesh option. This unexpected result has been traced to the effect of the optimising compiler on the SPDIFF discretisation module.

Despite the encouraging results that we have obtained with this remeshing scheme it should be noted that the discrete time approach may not be suitable for hyperbolic problems with shocks where space-time characteristic information is important. In this situation, even if discrete remeshing is performed at every time step, the mesh will lag behind the shock wave. For such problems the continuous remeshing approach, as typified by the moving finite element approach of Miller and Miller [21], has proved more effective (see Furzeland [15]).

5. A bubble collapse problem

This section provides an example of a complicated problem that has been solved very effectively using the SPRINT software. The solution of this problem also serves to indicate some of

the areas in which the software could be improved. A simplified model of the collapse of a vapour-filled bubble contained in liquid consists of the following two PDEs and eight coupled differential-algebraic equations. Let $v(y, t)$ be the temperature inside the bubble of radius $s(t)$ where $0 \leqslant y \leqslant s(t)$ and suppose that the bubble is contained in a liquid whose temperature is $w(y, t)$ where $y \geqslant s(t)$. On applying the transformation,

$$x = \begin{cases} \dfrac{y}{s(t)} & \text{for } 0 \leqslant y \leqslant s(t), \\ \dfrac{s(t)}{y} & \text{for } y \geqslant s(t), \end{cases}$$

the PDEs for $v(x, t)$ and $w(x, t)$ are

$$s(t)p(v)\left(s(t)\frac{\partial v}{\partial t} + \left(x^2 U_{vs} - x\dot{s}\right)\frac{\partial v}{\partial x}\right) = \frac{1}{x^2}\frac{\partial}{\partial x}\left\{\frac{x^2}{D_v}\frac{\partial v}{\partial x}\right\} + \beta\dot{p}_v, \tag{5.1}$$

$$s^2(t)\left(\frac{\partial w}{\partial t} - \left(6x^3 - x^2 U_{\ell s} - x\dot{s}\right)\frac{\partial w}{\partial x}\right) = \frac{1}{x^2}\frac{\partial}{\partial x}\left\{\frac{x^2}{D_\ell}x^4\frac{\partial w}{\partial x}\right\}, \tag{5.2}$$

where

$$D_v \approx 18600, \quad D_\ell \approx 205, \quad \beta \approx 95, \quad p(v(x, t)) = \rho_{vs}\left\{\frac{v(1, t) + 3.664}{v(x, t) + 3.664}\right\}. \tag{5.3}$$

The boundary conditions are given by

$$\frac{\partial v}{\partial x} = \frac{\partial w}{\partial x} = 0 \quad \text{at } x = 0, \qquad v(1, t) = w(1, t), \tag{5.4}$$

$$\frac{\partial w}{\partial x} + 0.04\frac{\partial v}{\partial x} + s(t)\rho_{vs}(\dot{s} - U_{vs})\,136.58 = 0 \quad \text{at } x = 1. \tag{5.5}$$

The condensation rate m and the collapse rate of the bubble S (where $S = \dot{s}$) are connected by the relationship

$$3IS + \dot{I}s(t) - m(t)\,1002 = 0 \tag{5.6}$$

where $I(t)$ is the integral of the function $p(v(x, t))$ (see (5.3)) as approximated by equation (5.9) below. The position of the bubble wall, $s(t)$, is governed by a second-order ODE which is written as a pair of first-order equations for s and its time derivative S. The first equation is

$$s(t)\dot{S} + 1.5(S)^2 = \dot{m}s(t) + m(t)\left(S + \tfrac{1}{2}m(t)\right) + 9.7\rho_{vs} - \frac{9.92}{s(t)} - P_\ell(t), \tag{5.7}$$

where $m(t)$ is defined by

$$m(t) = 7.89 \times 10^{-4}\,\frac{(\rho_{vs} - 0.682P_v(t) + 0.02)}{(v(1, t) + 3.664)^{1/2}} \tag{5.8}$$

and $P_\ell(t)$ is defined by

$$P_\ell(t) = \begin{cases} 35.9t, & t \leqslant 0.04, \\ 1.436, & t \geqslant 0.04. \end{cases}$$

The second equation of the pair is defined by equation (5.13) below. The function $I(t)$ in (5.6) is defined by

$$I(t) = \int_0^1 x^2 p(v(x, t))\, dx.$$

This integral thus depends on the temperature inside the bubble and is evaluated using the discrete temperature values and a weighted Gaussian quadrature rule for the x^2-weight, i.e.

$$I(t) - \sum_{i=1}^{N} w_i x_i^2 p(v(x_i, t)) = 0, \tag{5.9}$$

where N is the number of spatial mesh points and w_i are the quadrature points associated with the mesh points. The ODE variable $I(t)$ thus depends on every mesh point solution value of $V(x, t)$. The vapour pressure and density (ρ_{vs} and $P_v(t)$) are related by

$$\rho_{vs} = (1.87P_v(t) + 0.056)/(v(1, t) + 3.664), \tag{5.10}$$

while the vapour velocity at the interface U_{vs} and the liquid velocity there $U_{\ell s}$ are given by

$$U_{vs} = S + \frac{m(t) + 1002}{\rho_{vs}}, \tag{5.11}$$

$$U_{\ell s} = S - m(t); \tag{5.12}$$

and the final equation to connect $\dot{s}$ and S is simply:

$$\dot{s} = S. \tag{5.13}$$

In this case then the coupled ODE system consists of the eight equations numbered from (5.6) to (5.13). The vector $v(t)$ is made up of the eight coupled ODE variables in the following way:

$$v(t) = [s, S, m, I, \rho_{vs}, P_v(t), U_{\ell s}, U_{vs}]^T$$

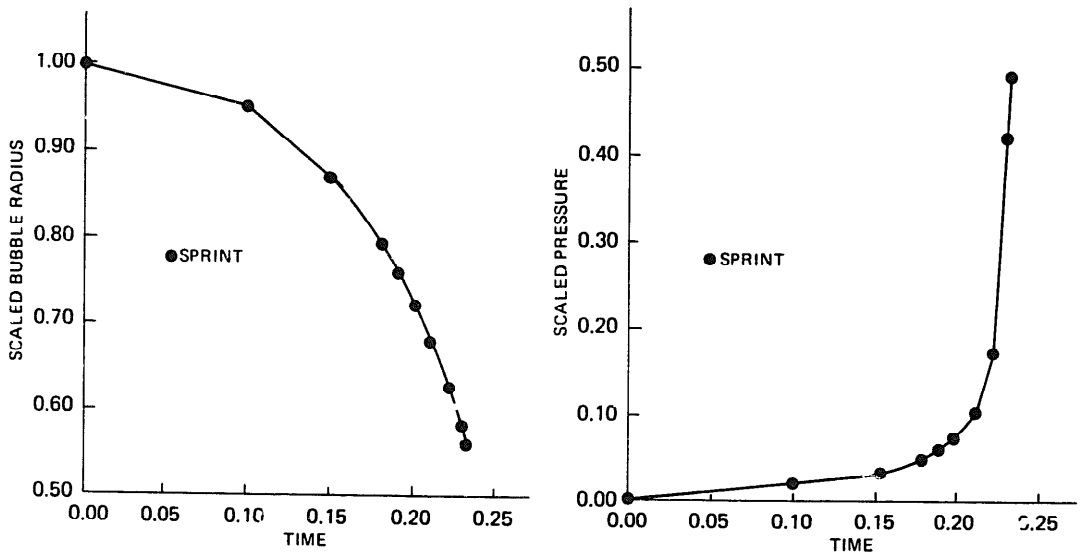

Fig. 3. The collapse of the bubble w.r.t. time and its internal pressure.

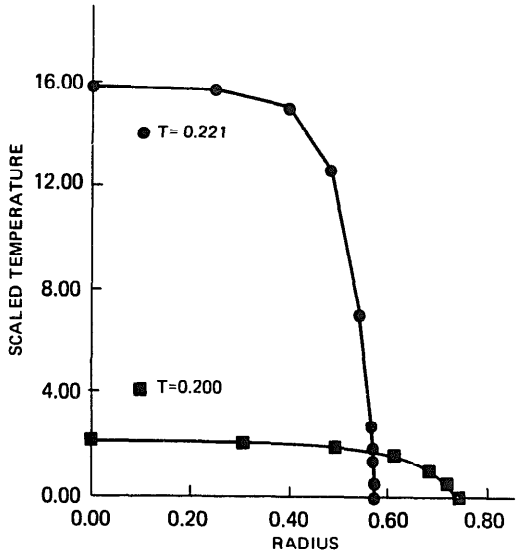

Fig. 4. The temperature profile inside the bubble at t 0.2 and $t = 0.221$.

In this case the spatial coupling points consist of all the spatial mesh points as all the mesh point solution values of $v(x_i, t)$ at a particular time are required in forming the approximate integral defined by (5.9).

The initial temperature is zero in both liquid and vapour phases, $s(0) = 1$, $\dot{s}(0) = 0$, $P_v(0) = 0$ and the initial values of S, m, I, ρ_{vs}, $U_{\ell s}$ and U_{vs} are defined using the initial values already given and equations (5.13), (5.8), (5.9), (5.10), (5.11) and (5.12) respectively.

The problem was integrated using a mesh of 25 points clustered towards $x = 1$ so as to take account of the boundary layer there. The SPDIFF spatial discretisation and the SPGEAR integrator were employed with a mixed local error test of 10^{-5} relative and 10^{-7} absolute. Figures 3 and 4 show the decrease in the bubble radius, the increase in bubble pressure and two examples of temperature profiles inside the bubble.

6. Discussion and conclusions

The bubble collapse problem illustrates several interesting points. To discretise a problem such as this one by hand is time-consuming. Software that allows complex problem formulations to be solved reduces the coding time and allows the scientist to concentrate on the physics of the problem. Appendix A contains a concise outline code which calls SPRITE to solve the bubble collapse problem. In the program the choices of sparse linear algebra, BDF time integration and SPDIFF spatial discretisation routine are specified by setting CHARACTER variables to "SPARSE", "GEAR", and "SKEEL" respectively. The difficulty of coding the problem is thus confined to defining the coupled PDE/ODE system. The convention used by the discretisation routines in ordering the ODE solution vector is that the coupled ODE components are stored after the PDE components. The resulting sparsity pattern takes the form of a banded matrix bordered by rows underneath and columns to its right. The bordering rows and columns may be large in number but may also be sparse in structure. In the case of the bubble collapse problem

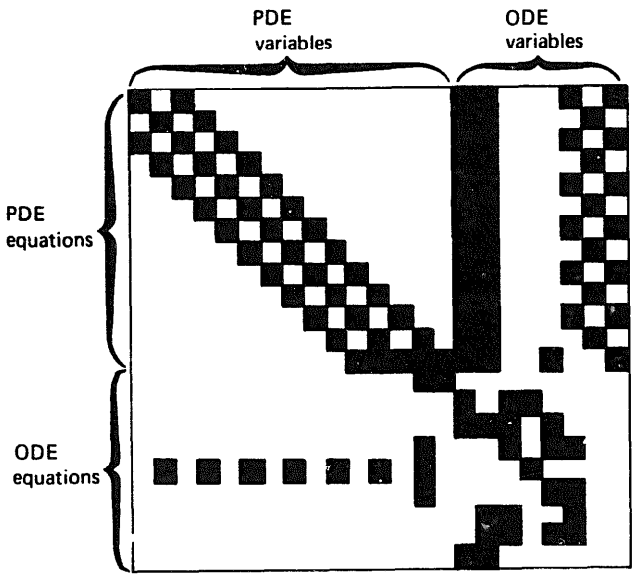

Fig. 5. The Jacobian matrix for the bubble problem (■ indicates a nonzero entry).

Fig. 5 illustrates the nature of the sparsity pattern of the Jacobian matrix for a spatial mesh of seven points only so as to more clearly illustrate the coupling between the different ODE and PDE components. Although it is possible to develop special routines for such matrices or to re-order the matrix to get a banded matrix in some cases, a more straightforward solution is to employ sparse matrix techniques. This also has the advantage that the resulting code can be used to solve large sparse ODE systems and can also be used for PDE problems in two space dimensions. Although there is no software in SPRINT for the solution of PDE problems in two space dimensions many two-dimensional problems have since been solved by users employing their own discretisation methods within the SPRINT framework and by making use of the banded and sparse linear algebra options. It is planned to produce such software in the next phase of the project. The use of this software to solve two-space-dimensional problems within SPRINT will result in very large systems of equations of the form of (3.2) which will make it necessary to use iterative methods, such as those considered by Brown and Hindmarsh [9] and Chan and Jackson [10] to complement the sparse matrix routines already present.

The difficulty of verifying numerical results against experimental results for complicated problems such as the bubble collapse problem suggests that what is required is a robust global error tracking procedure which includes both space and time error components. Berzins and Dew [3] have implemented simple algorithms of this type by using the MONITR facility of SPRINT, but more research is needed in this area, particularly in balancing the space and time components of the global error.

Similar problems also of interest are two-phase vapour-liquid flow in pipes. These pipe flow problems are usually of a hyperbolic PDE nature and are nonconservative in form due to the presence of source or sink terms. These problems are solved with a hybrid approach with a flux-corrected transport algorithm being used to accurately resolve shock waves (pressure pulse propagations) and SPRINT to integrate the source/sink term contributions.

In conclusion it can be said that despite the areas for future research suggested above SPRINT has already proved to be a valuable modelling tool within Shell Research. A modified form of the differential-algebraic part of the software has been released in the D02 (ODE) chapter of the NAG Library (see Berzins, Brankin and Gladwell [8]). Plans are also well advanced to release the remainder of the software in the D03P (parabolic PDE) chapter of the NAG Library.

Appendix A. Outline code for bubble collapse problem using SPRITE

```
      PARAMETER (NPTS = 25, NPDE = 2, NXI = NPTS, NV = 8)
C     NPTS = no. of mesh points, NPDE = no. of PDEs, NV = no. of coupled ODEs
C     and NXI = no. of spatial coupling points (same as mesh points).
      PARAMETER (NEQ = NPTS * NPDE+NV)
C     workspace required for sparse matrix algebra routines
      PARAMETER (NRWK = 20 * NEQ+93+4 * NEQ+11 * NEQ/2)
C     workspace for res-skeel discretisation and dummy MONITR routine.
      PARAMETER (NRESWK = NPTS * 21+35+NV, NDUMWK =1)
C     real workspace and integer workspace
      PARAMETER (NRW = NRWK+NRESWK+NDUMWK, NIWK = 2 * NEQ+14)
      INTEGER I, IBAND, ITRACE, IW(NIWK), M, MAXNPT, NIW(3), NT, NEL, NPTL, IBK
      DOUBLE PRECISION T, TOUT, XI(NPTS), XBK(1),
     1                 Y(NEQ), X(NPTS), RTOL(NEQ), ATOL(NEQ), RW(NRW),
      LOGICAL REMESH
      CHARACTER * 6 RESULT, SNORM, MATZ, STEP, SPACE
C     SKLRES is the name of RESID routine for SPDIFF module, EZMNTR is a
C     simple system-provided monitor. M = 2 for spherical space geometry.
      EXTERNAL SKLRES, EZMNTR
      M = 2
C     Place mesh points in array X(I) and pass into XI(I) for coupling points
      DO 10 I =1, NPTS
        X(I) = {user-defined mesh}
10      XI(I) = X(I)
C     call SPRITE interface using sparse matrix routines and BDF integrator.
      MATZ   ="SPARSE"
      STEP   ="GEAR"
      RESULT="BRIEF"
      T      = 0.0D0
C     Invoke SPDIFF discretisation by calling SETPDE with SPACE ="SKEEL"
      SPACE  ="SKEEL"
      REMESH=.FALSE.
      MAXNPT= NPTS
      CALL SETPDE(NEQ, NPDE, NPTS, X, Y, SPACE, STEP, RW, NRESWK, M, T,
     1            IBAND, 1, REMESH, MAXNPT, XBK, IBK, NEL, NPTL, NV, NXI, XI)
C     Parameters in call to SPRITE, trace level, tolerances, norm, indicators.
      DO 20 I =1, NEQ
        RTOL(I) =1.D-4
20      ATOL(I) =1.D-4
      ITRACE =1
      SNORM ="L2NORM"
      IW(1)  = 0
      IW(3)  = 0
```

```
      NIW(1)  = NIWK
      NIW(2)  = 0
      NIW(3)  = 0
      NT      = 1
      TOUT    = 0.221D0
      CALL SPRITE(NEQ, T, TOUT, NT, RESULT, Y, RTOL, ATOL, ITRACE,
     1                  SNORM, MATZ, STEP, SKLRES, EZMNTR, RW, NRW, IW, NTW)
      STOP
      END
```

Acknowledgment

The authors would like to express their gratitude to the Thornton Research Centre of Shell Research Ltd. for funding the development of the SPRINT software.

References

[1] C.T.H. Baker and C. Phillips, *Numerical Solution of Non-linear Problems* (Clarendon Press, Oxford, 1981) 11–12.

[2] M. Berzins, P.M. Dew and R.M. Furzeland, Software for time-dependent problems, in: B. Engquist and T. Smedsaas, eds., *PDE Software Modules, Interfaces and Systems* (IFIP/North-Holland, Amsterdam, 1984) 309–324.

[3] M. Berzins and P.M. Dew, C^0 Chebyshev methods for parabolic P.D.E.s, *IMA J. Numer. Anal.* 7 (1987) 15–37.

[4] M. Berzins and R.M. Furzeland, A user's manual for SPRINT, Parts 1 and 2, Repts. 199 and 202, Department of Computer Studies, Leeds University, England (1984).

[5] M. Berzins, A C^1 interpolant for codes based upon backward differentiation formulae, *Appl. Numer. Math.* 2 (1986) 109–118.

[6] M. Berzins and R.M. Furzeland, A type-insensitive method for the solution of stiff and non-stiff differential equations, Rept. No. 204, Department of Computer Studies, Leeds University, England (1986).

[7] M. Berzins, P.M. Dew and R.M. Furzeland, Developing P.D.E. software using the method of lines and differential-algebraic integrators, Rept. No. 220, Department of Computer Studies, Leeds University, England (1986).

[8] M. Berzins, R. Brankin and I. Gladwell, The design of stiff integrators in the NAG library, Department of Mathematics Report, University of Manchester, England (1987).

[9] P.N. Brown and A.C. Hindmarsh, Matrix-free methods for stiff systems of O.D.E.s, *SIAM J. Numer. Anal.* 23 (1986) 610–638.

[10] T. Chan, and K.R. Jackson, The use of iterative linear equation solvers for large systems of O.D.E.s, *SIAM J. Sci. Statist. Comput.* 7 (1986).

[11] 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.* 20 (1984) 1797–1813.

[12] P.M. Dew and J.E. Walsh, A set of library routines for solving parabolic equations in one space variable, *ACM Trans. Math. Software* 7 (1981) 295–314.

[13] J.J. Dongarra, C.B. Moler, J.R. Bunch and G.W. Stewart, *LINPACK Users's Guide* (SIAM, Philadelphia, PA, 1979).

[14] S.C. Eisenstat, M.C. Gursky, M.H. Schultz and A.H. Sherman, Yale sparse matrix package, Repts. 112 and 114, Department of Computer Science, Yale University, New Haven, CT (1977).

[15] R.M. Furzeland, The construction of adaptive space meshes, TNER.85.022, Shell Research Ltd, Thornton Research Centre, Chester, England (1985).

[16] P.W. Gaffney, Using the method of lines technique to solve initial boundary value p.d.e.s., in: R.S. Stepleman, M. Carver, R. Peskin, W.F. Ames and R. Vichnevetsky, eds., *Scientific Computation* (IMACS/North-Holland, Amsterdam, 1983) 79–85.

[17] G.K. Gupta, C.W. Gear and B. Leimkuhler, Implementing multi-step formulas for solving D.A.E.s, Rept. UIUCD-R-85-1205, Department of Computer Science, University of Illinois at Urbana-Champaign, IL (1985).
[18] A.C. Hindmarsh, ODEPACK: A systematized collection of O.D.E. solvers, in: R.S. Stepleman, M. Carver, R. Peskin, W.F. Ames and R. Vichnevetsky, eds., *Scientific Computations* (IMACS/North-Holland, Amsterdam, 1983) 55–64.
[19] J. Katusky and N.K. Nichols, Equidistributing meshes with constraints, *SIAM J. Sci. Statist. Comput.* 1 (1980) 499–511.
[20] B. Leimkuhler, L. Petzold and C.W. Gear, On obtaining a consistent set of initial values for systems of differential-algebraic equations, Rep. UIUCD-R-87-1344, Department of Computer Science, University of Illinois at Urbana-Champaign, IL (1987).
[21] K. Miller and R. Miller, Moving finite elements I, *SIAM J. Numer. Anal.* 18 (1981) 1019–1053.
[22] L. Petzold, A description of DASSL, Rept. SAND82-8367, Applied Math Division. SANDIA National Labs. Livermore, CA (1982).
[23] L. Petzold, Differential/algebraic equations are not ODEs, *SIAM J. Sci. Statist. Comput.* 3 (1982) 367–384.
[24] L. Petzold, Automatic selection of methods for solving stiff and non-stiff systems of ordinary differential equations, *SIAM J. Sci. Statist. Comput.* 4 (1983) 136–148.
[25] J. Petzold and P. Lotstedt, Numerical solution of non-linear differential equations, II: Practical implications, *SIAM J. Sci. Statist Comput.* 7 (1986).
[26] A. Prothero and A. Robinson, On the accuracy and stability of one step methods for solving stiff systems of ordinary differential equations, *Math. Comp.* 28 (1974) 145–162.
[27] L.F. Shampine, Implementation of implicit formulas for the solution of O.D.E.s., *SIAM J. Sci. Statist Comput.* 1 (1) (1980).
[28] N. Schryer, Partial differential equations in one space variable, Computing Science Tech. Rept. No 115, AT & T Laboratories, Murray Hill, NJ (1984).
[29] R.D. Skeel and A. Kong, Blended linear multistep methods, *ACM Trans. Math. Solftware* 3 (1977) 326–345.
[30] R.D. Skeel and M. Berzins, Spatial discretisation methods for parabolic P.D.E.s, *SIAM J. Sci. Statist Comput.* (submitted).