Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.G. Blom, P.A. Zegeling

A moving-grid interface for systems of one-dimensional time-dependent partial differential equations

Department of Numerical Mathematics Report NM-R8904 January

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

A Moving-Grid Interface for Systems of One-Dimensional Time-Dependent Partial Differential Equations

J.G. Blom, P.A. Zegeling
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

In the last decade several numerical techniques have been developed to solve time-dependent Partial Differential Equations (PDEs) in one dimension having solutions with steep gradients in space and in time. One of these techniques, a moving-grid method based on a Lagrangian description of the PDE and a smoothed-equidistribution principle to define the grid positions at each time-level, has been coupled to a spatial discretization method which automatically discretizes the spatial part of the user-defined PDE following the Method of Lines approach. We supply two subroutines, **CWRESU** and **CWRESX**, which compute the residuals of the ODE system obtained from semi-discretizing resp., the PDE and the set of moving-grid equations. These routines are combined in an enveloping routine **SKMRES** which delivers the residuals of the complete ODE system to be used in a **SPRINT**[2, 3] environment. To solve this stiff, nonlinear implicit ODE system, a robust and efficient time-integrator must be applied, such as one of the BDF modules in **SPRINT**. Some numerical examples are shown to illustrate the simple and effective use of this software interface.

1980 Mathematics subject classification: Primary:65V05. Secondary:65M20, 65M50.
1987 CR Categories: G.1.8.
Key Words & Phrases: partial differential equations, time-dependent problems, method of lines, Lagrangian methods, moving grids, mathematical software.

Note: This work has been carried out in connection with a joint CWI/Shell project on 'Adaptive Grids'. For this project Paul Zegeling has received support from the 'Netherlands Foundation for the Technical Sciences' (STW), future Technical Science Branch of the Netherlands Organization for the Advancement of Research (NWO) (Contract no. CWI 55.092).

1. Introduction

In this paper we describe a software interface for the spatial discretization of systems of one-dimensional time-dependent partial differential equations. The problem of solving PDEs using an interface on fixed grids, by which the user only needs to define the PDE and accompanying boundary conditions in two subroutines, has already been treated by, for example, Sincovec & Madsen[8], Berzins & Furzeland[3], and Bakker[1]. However, in many models from physics and chemistry steep spatial and temporal gradients may occur in the solution that cause problems if a numerical solution method on a fixed grid is used. In such cases moving-grid methods are likely to be more successful. In [6] an evaluation has been made of the behavior of three moving-grid methods with respect to efficiency and robustness on three difficult test problems having steep moving fronts. On account of the hopeful results of this investigation we decided to develop a moving-grid interface, i.e. an interface with the possibility to move continuously the grid points in time, and to couple this interface with one of the methods from [6], viz., the method due to Dorfi & Drury[5]. For this purpose, we implemented a finite-element discretization for the Lagrangian form of the PDE. This discretization method, borrowed from Skeel & Berzins[9], is akin to a nonlinear Galerkin method and is of 2-nd order accuracy in space, even for polar problems. The resulting interface consists of two Fortran subroutines, **CWRESU**

and **CWRESX**. **CWRESU** computes the residuals of the semi-discretized Lagrangian PDE and is a modification of the subroutine **SKEEL4** of the **SPRINT** package [2, 3]. **CWRESX** computes the residuals of an ODE system that governs the grid motion. These subroutines are called from a subroutine **SKMRES** which is the moving-grid equivalent of the **SPRINT** routine **SKLRES**. The coupled ODE systems are stiff and are integrated in time with one of the BDF codes **SPGEAR** or **SPDASL** of the **SPRINT** package.

Our paper is divided into six sections and two appendices. Section 2 presents an outline of the considered moving-grid method. In Section 3 we define the class of PDE problems that is allowed in the software interface. Section 4 describes the semi-discrete approximation of the PDE system. In Section 5 we discuss the interface itself. We show with an example the ease of use of our software interface in the **SPRINT** environment. Section 6 contains two numerical examples to underline the performance of the moving-grid method when applied to problems possessing moving transition-layers. Appendix A gives an enumeration of the test functions, the trial functions, and the quadrature points used in Section 4. Finally, in Appendix B a listing of the subroutines is given.

2. The moving-grid method

In this section we will give a brief description of the moving-grid technique that controls the spatial grid-movement in time. This technique is due to Dorfi & Drury[5]. For the theoretical background and some analytical aspects of the method we refer to [10]. We wish to emphasize that this moving-grid method is not in a final stage and that research on this method is still going on.

A system of PDEs

$$u_t = f(u,x,t), \qquad x_L < x < x_R, \quad t > t_0, \tag{2.1}$$

is transformed to its Lagrangian form

$$\dot{u} - u_x \dot{x} = f(u,x,t), \tag{2.2}$$

where $\dot{u}$ denotes the total time-derivative. We then choose N time-dependent grid points:

$$x_L = X_0 < \cdots < X_i(t) < X_{i+1}(t) < \cdots < X_{N+1} = x_R \tag{2.3}$$

and discretize (2.2) in space to obtain

$$\dot{U}_i - \frac{(U_{i+1} - U_{i-1})}{(X_{i+1} - X_{i-1})} \dot{X}_i = F_i, \quad t > t_0, \quad 1 \leqslant i \leqslant N. \tag{2.4}$$

Here, U_i and F_i represent the semi-discrete approximation to the exact PDE solution u, resp. the righthand-side function $f(u,x,t)$, at the point $(x,t) = (X_i(t),t)$. To solve the ODE system (2.4) additional equations are required for the time-dependent grid points X_i which are as yet unknown. For this purpose the next quantities are defined

$$\begin{aligned} &n_i := (\Delta X_i)^{-1}, \quad \Delta X_i := X_{i+1} - X_i, \\ &\tilde{n}_i := n_i - \kappa(\kappa+1)(n_{i+1} - 2n_i + n_{i-1}), \quad 0 \leqslant i \leqslant N, \\ &\qquad (n_{-1} := n_0, \ n_N =: n_{N+1}), \end{aligned} \tag{2.5}$$

where n_i stands for the, so-called, point concentration of the grid and $\kappa \geqslant 0$ denotes a spatial smoothing parameter. Now we define implicitly, in terms of $\tilde{n}_i$, the movement of the grid points X_i

$$\frac{\tilde{n}_{i-1} + \tau\dot{\tilde{n}}_{i-1}}{M_{i-1}} = \frac{\tilde{n}_i + \tau\dot{\tilde{n}}_i}{M_i}, \quad 1 \leqslant i \leqslant N, \tag{2.6}$$

where $\tau \geqslant 0$ is a time-smoothing parameter and M_i is a monitor function, here chosen as follows

$$M_i := \sqrt{\alpha + \text{NPDE}^{-1} \sum_{j=1}^{\text{NPDE}} (U_{i+1}^j - U_i^j)^2 / (\Delta X_i)^2}, \tag{2.7}$$

which is a semi-discrete representation of the first derivative solution functional

$$m(u) = \sqrt{\alpha + \|u_x\|^2}.$$

Shortly, κ determines the level of clustering of the grid points and the arclength monitor M_i determines the shape of the X_i-distribution. The parameter τ prevents the grid movement from adjusting immediately to new values of the monitor function M_i, therefore trying to avoid temporal oscillations in the grid which may cause relatively large errors, when applied to solutions with steep gradients. Equations (2.4) and (2.6) are combined to yield the system of ODEs

$$\begin{aligned} \dot{U} - D\dot{X} &= F \\ \tau B\dot{X} &= g \end{aligned}, \tag{2.8}$$

where definitions (2.5) have been used, D and B are solution-dependent matrices, and F and g solution-dependent vectors, containing all information about the monitor function and the righthand-side of the PDE itself, respectively. System (2.8) can be rearranged in the linearly implicit form

$$A(Y)\dot{Y} = L(Y), \tag{2.9}$$

where

$$Y := (\cdots, U_i^1, \cdots, U_i^{\text{NPDE}}, X_i, \cdots)^T.$$

This stiff ODE system may be solved, e.g., by one of the BDF integrators of **SPRINT**. The user of the interface, who is interested in the details of the moving-grid method is advised to read reference [10].

3. PDE DEFINITION

The class of allowable PDE problems is derived from the class defined in the interface of the **SPRINT** package [2, 3]

$$\sum_{k=1}^{\text{NPDE}} C_{j,k}(x,t,u,u_x)\frac{\partial u^k}{\partial t} = x^{-m}\frac{\partial}{\partial x}(x^m R_j(x,t,u,u_x)) - Q_j(x,t,u,u_x) \tag{3.1}$$

$$\text{for } j = 1,\ldots,\text{NPDE} \quad \text{and} \quad x \in (x_L, x_R), \quad t > t_0, \quad m \in \{0,1,2\},$$

where NPDE denotes the number of PDEs, $u := (u^1, u^2, \cdots, u^{\text{NPDE}})^T$ is the solution vector and R_j and Q_j can be thought of, in special cases, as flux and source terms, respectively. The solution u and the functions $C_{j,k}$, R_j, and Q_j are expected to be continuous functions. For problems in Cartesian coordinates $m = 0$, whereas $m = 1$ indicates a cylindrical polar coordinate system, and $m = 2$ a spherical polar one. In the case $m > 0$ we require $x_L \geq 0$.

The user-specified boundary conditions belonging to system (3.1) have the master equation form

$$\beta_j(x,t)R_j(x,t,u,u_x) = \gamma_j(x,t,u,u_x,u_t) \quad \text{at } x = x_L \text{ and } x = x_R, \quad \text{for } j = 1,\ldots,\text{NPDE}, \tag{3.2}$$

where β_j and γ_j are continuous functions of their variables. If $m > 0$ and $x_L = 0$ the boundedness of the solution near $x = 0$ implies $R_j \mid_{x=x_L} = 0$. Simple boundary conditions, such as Dirichlet or Neumann conditions are very easy to define, using equation (3.2), as will be shown in the numerical example in Section 5. As can be seen in Section 2, the appearance of the time-derivatives in (3.1) is an essential part of the moving-grid technique. This means that the interface, in principle, is aimed at parabolic and first-order hyperbolic systems of equations (in this case a dummy boundary equation has to be supplied). But if one or more equations in a system of PDEs are elliptic of nature this still fits in the class (putting $C_{j,k}$ equal to 0 for some index j; $k = 1,\ldots,\text{NPDE}$) although we did not test this combination.

The initial conditions must satisfy

$$u(x,t_0) = u^0(x) \text{ for } x \in [x_L, x_R], \tag{3.3}$$

where u^0 is a piecewise continuous function of x with NPDE components.

It must be noted, that the possibility of coupling (3.1) to a system of Ordinary Differential Equations, as is allowed in the **SPRINT** interface, is not recommended for the time being, because the effects, arising from moving the spatial grid in time with respect to the fixed coupling points, have not yet been investigated.

Summarizing, the user must define, respectively, the interval $[x_L, x_R]$, the functions $C_{j,k}$, R_j, Q_j, β_j, and γ_j, the initial time t_0, the vector u^0 and the numbers m and NPDE. In our opinion, the class of PDE problems defined by (3.1), (3.2) and (3.3) is sufficiently general to treat a huge number of miscellaneous, of course well-posed, problems stemming from engineering, physics and chemistry.

4. Spatial discretization

In order to reduce accuracy problems that arise for coefficients like x^{-m} in (3.1) when x is near zero and $m>0$, a spatial discretization method is used which is second order in space. The nonlinear Galerkin-based method is extensively described in Skeel & Berzins[9]. In the following we give a summary of this discretization method when applied to the PDE class (3.1) transformed to its Lagrangian form. We omit, however, the error analysis which can be found in Skeel & Berzins.

First we apply the Lagrangian transformation.

Let w be defined by $w := u_x\dot{x}$ and S_j by

$$S_j = S_j(x,t,u,u_x,\dot{u},w) := \sum_{k=1}^{\text{NPDE}} C_{j,k}(\dot{u}^k - w^k) + Q_j.$$

Then system (3.1) becomes

$$S_j = x^{-m}(x^m R_j)_x \quad \text{for } j=1,...,\text{NPDE}, \tag{4.1}$$

with $C_{j,k}$, Q_j, and R_j defined as before. On the spatial grid (2.3) we will formulate the Galerkin method for (4.1). Introduce the approximation U^k of u^k

$$U^k(x) = \sum_{i=0}^{N+1} U_i^k \phi_i^{(m)}(x).$$

Let $\psi_i^{(m)}$ denote the test functions. The trial functions $\phi_i^{(m)}$ and the test functions $\psi_i^{(m)}$ are given in Appendix A. Introduce the weight function x^m and integrate (4.1) on $[x_L, x_R]$ partially, so as to obtain

$$\int_{x_L}^{x_R} x^m \psi_i^{(m)} S_j dx = x_R^m \psi_i^{(m)}(x_R) R_j|_{x=x_R} - x_L^m \psi_i^{(m)}(x_L) R_j|_{x=x_L} - \int_{x_L}^{x_R} x^m \frac{d\psi_i^{(m)}}{dx} R_j dx \tag{4.2}$$

$$\text{for } j=1,...,\text{NPDE and } i=0,...,N+1.$$

Using the fact that $\psi_i^{(m)}(x)=0$ for $x \leqslant X_{i-1}$ and $x \geqslant X_{i+1}$ we get for $i=1,...,N$ and $j=1,...,\text{NPDE}$

$$\int_{X_{i-1}}^{X_{i+1}} x^m \psi_i^{(m)} S_j dx = - \int_{X_{i-1}}^{X_{i+1}} x^m \frac{d\psi_i^{(m)}}{dx} R_j dx \tag{4.3}$$

The integration over an interval $[X_{i-1}, X_i]$ is performed by numerical quadrature using 1 quadrature point $\xi_{i-\frac{1}{2}}$. After applying the numerical integration on $[X_{i-1}, X_i]$ and $[X_i, X_{i+1}]$ and lumping (that is evaluation of $\dot{u}^k$ takes place in X_i rather than in ξ) (4.3) yields

$$S_j^{i-\frac{1}{2}} \int_{X_{i-1}}^{X_i} x^m \psi_i^{(m)} dx + S_j^{i+\frac{1}{2}} \int_{X_i}^{X_{i+1}} x^m \psi_i^{(m)} dx =$$

$$- \xi_{i-\frac{1}{2}}^{\mu} R_j^{i-\frac{1}{2}} \int_{X_{i-1}}^{X_i} x^{m-\mu} \frac{d\psi_i^{(m)}}{dx} dx - \xi_{i+\frac{1}{2}}^{\mu} R_j^{i+\frac{1}{2}} \int_{X_i}^{X_{i+1}} x^{m-\mu} \frac{d\psi_i^{(m)}}{dx} dx + E, \tag{4.4}$$

where

$$S_j^{i-\frac{1}{2}} = S_j(\xi_{i-\frac{1}{2}}, t, U(\xi_{i-\frac{1}{2}}), U_x(\xi_{i-\frac{1}{2}}), \dot{U}_i, W_i),$$
$$S_j^{i+\frac{1}{2}} = S_j(\xi_{i+\frac{1}{2}}, t, U(\xi_{i+\frac{1}{2}}), U_x(\xi_{i+\frac{1}{2}}), \dot{U}_i, W_i), \quad \text{with } W_i = \frac{U_{i+1} - U_{i-1}}{X_{i+1} - X_{i-1}} \dot{X}_i,$$
$$R_j^{i+\frac{1}{2}} = R_j(\xi_{i+\frac{1}{2}}, t, U(\xi_{i+\frac{1}{2}}), U_x(\xi_{i+\frac{1}{2}})),$$

and E stands for the total error due to interpolation, quadrature and lumping. For the definition of μ we make a distinction between two special cases

i) the regular case ($m = 0$ or $x_L > 0$): $\mu = m$

ii) the singular case ($m > 0$ and $x_L = 0$): $\mu = -1$.

The choice of ξ depends on μ. On an interval $[X_i, X_{i+1}]$ we choose $\xi_{i+\frac{1}{2}} = \gamma_{-\mu, i+1/2}$, where $\gamma_{p,i+1/2}$ denotes the Gauß-point for the weight function x^p, i.e.,

$$\int_{X_i}^{X_{i+1}} (x - \gamma_{p,i+1/2})\, x^p dx = 0. \tag{4.5}$$

The numerical integration in (4.4) is then second order accurate. If we neglect the error E in (4.4), we arrive at a semi-discrete approximation of (4.1), for $i = 1,...,N$

$$S_j^{i-\frac{1}{2}} \frac{X_i^{m+1} - \zeta_{i-\frac{1}{2}}^{m+1}}{m+1} + S_j^{i+\frac{1}{2}} \frac{\zeta_{i+\frac{1}{2}}^{m+1} - X_i^{m+1}}{m+1} =$$
$$\zeta_{i+\frac{1}{2}}^{m-\mu}\, \xi_{i+\frac{1}{2}}^{\mu}\, R_j^{i+\frac{1}{2}} - \zeta_{i-\frac{1}{2}}^{m-\mu}\, \xi_{i-\frac{1}{2}}^{\mu}\, R_j^{i-\frac{1}{2}}, \tag{4.6}$$

with

$$\zeta_{i+\frac{1}{2}}^{m+1} = -\int_{X_i}^{X_{i+1}} x^{m+1} \frac{d\psi_i^{(m)}}{dx} dx, \quad \text{and} \quad \zeta_{i+\frac{1}{2}}^{0} = 1.$$

For a list of the test functions $\psi_i^{(m)}$, the trial functions $\phi_i^{(m)}$, the quadrature points $\xi_{i+\frac{1}{2}}$ and the integrals $\zeta_{i+\frac{1}{2}}^{m+1}$ see appendix A. In [9] a justification is given for all choices of the parameters and functions. There it is shown that the spatial discretization method is second order accurate, both in the regular and the singular case.

The right boundary equation in (3.2) $\beta_j(x_R,t)R_j|_{x=x_R} = \gamma_j|_{x=x_R}$ is combined with the semi-discrete approximation of (4.2) with $i = N+1$

$$S_j^{(N+1)-\frac{1}{2}} \frac{x_R^{m+1} - \zeta_{N+\frac{1}{2}}^{m+1}}{m+1} + \zeta_{N+\frac{1}{2}}^{m-\mu}\, \xi_{N+\frac{1}{2}}^{\mu}\, R_j^{N+1/2} = x_R^m\, R_j|_{x=x_R} \tag{4.7}$$

to eliminate $R_j|_{x=x_R}$.

In the regular case the same procedure is applied to the left boundary equation in (3.2) $\beta_j(x_L,t)R_j|_{x=x_L} = \gamma_j|_{x=x_L}$ and

$$S_j^{0+\frac{1}{2}} \frac{\zeta_{\frac{1}{2}}^{m+1} - x_L^{m+1}}{m+1} - \zeta_{\frac{1}{2}}^{m-\mu}\, \xi_{\frac{1}{2}}^{\mu}\, R_j^{\frac{1}{2}} = -x_L^m\, R_j|_{x=x_L}, \tag{4.8}$$

from which we can eliminate $R_j|_{x=x_L}$. If x_L and m are both zero we take $x_L^m = 1$. In the singular case, however, we use just the semi-discrete approximation of (4.2) with $i = 0$ which gives

$$S_j^{0+\frac{1}{2}}/(m+1) - \xi_{\frac{1}{2}}^{-1}\, R_j^{\frac{1}{2}} = 0. \tag{4.9}$$

This means, that for $X_1 \to x_L$ the boundary equation tends to $R_j|_{x=x_L} = 0$, which is a natural constraint for polar problems.

It is easy to derive that, for $m = 0$, (4.6) is analogous to a semi-discretization by central differences. Similar to Section 2, equations (4.6) and (2.6) are combined to obtain the stiff system of ODEs

$$A(Y)\dot{Y} = L(Y). \tag{4.10}$$

In the next section we discuss the interface that implements this spatial discretization.

5. The interface

5.1. Choice of the moving-grid parameters

The moving-grid method as described in Section 2 still has three parameters that should be specified by the user; i.e., the time-smoothing parameter τ, the spatial smoothing parameter κ, and the monitor regularizing parameter α. The choice of the parameters κ and α is not very critical, experiments have showed that a variation of these parameters does not result in a drastic change of performance. A wrong choice of τ can result in starting problems or a grid lagging behind. A, problem dependent, guidance of how to choose τ will be given below. It is felt, however, that the choice of the monitor function (e.g., dependent on solution curvature instead of solution variation) is probably more important than the value of τ (see also [10]). But as stressed before, research on the moving-grid technique is still going on.

We will now give an indication as how the parameters should be chosen.

Time-smoothing parameter τ, $\tau \geqslant 0$. It is obvious that the grid will not be adapted if τ tends to infinity. On the other hand if $\tau = 0$ the equidistribution relation

$$\tilde{n}_i(t) = c(t)M_i(t) \tag{5.1}$$

will be solved. So if the initial grid satisfies (5.1) at t_0, as is the case, e.g., for a uniform grid and a constant or linear initial solution, τ can be chosen equal to zero (cf. the first two numerical examples in [10] and the first example in Section 6). If at t_0 (5.1) does not hold and if the initial grid has to be adapted, or if time-smoothing is desired, a typical value for τ is 10^{-3}. Note however that τ should be related to the critical time scale of the problem.

Spatial smoothing parameter κ, $\kappa \geqslant 0$. An adjacent grid ratio of 1.5 (i.e., $\kappa = 2$) was found to be satisfying for all problems tested. It is perhaps a bit conservative and therefore demanding an unnecessary large number of grid points if the solution possesses sharp spatial gradients. For less spatial smoothing $\kappa = 1$ will suffice. The choice $\kappa = 0$, implying that there will be *no* spatial smoothing, is in general not recommended since it can be shown, in the case $m = 0$, that the spatial discretization is equivalent to a finite-difference approximation and the reliability of such a discretization is dependent on a smooth grid distribution.

Monitor regularizing parameter α, $\alpha > 0$. This parameter is added to the monitor function to ensure that M_i is strictly positive. The choice $\alpha = 1$ results in the arclength monitor. This can be used for almost all problems, unless the solution is very flat in which case α dominates the monitor and therefore a uniform grid will be used, which can result in a slightly worse performance than on a nonuniform grid especially in the case of evolving solutions. In this case α should be chosen smaller. α could be thought of as related to the integral over the monitor function $\int_{x_L}^{x_R} m(u)dx$.

5.2. Implementation and use of the interface

The PDE interface module has been adapted to the **SPRINT** package so that a **SPRINT** user can make use of the moving-grid spatial discretization routine in a completely analogous way as the fixed-grid discretization module **SPDIFF** of **SPRINT**. (For the experienced **SPDIFF** user: Of course the **MVMESH** monitor routine **MONITR** has been left out and the subroutine headers of the required user routines are slightly different as a result of our different problem class.) Below we copy that part of the documentation of the module that describes the use of the moving-grid discretization routine. For more information we refer to Part 2 of the **SPRINT** User's Manual [3] and to the full documentation in the module **SPMDIF**, the analogue of **SPDIFF**, that can be found in Appendix B.

```
C-----------------------------------------------------------------------
C
C How to use this module
C ----------------------
C 1.  Set NPDE = # PDEs to be solved.
C     Set NPTS = # mesh points to be used.
C                (NC=NPTS-2 is # internal points)
C     Set M for space coordinate type
C         = 0 for Cartesian,  = 1 for cylindrical, = 2 for spherical.
C     Specify a workspace of size at least  (NPDE+1)*NPTS+(6+NPDE)*NPDE
C     for use by the routine SKMRES which defines the ODE system being
C     solved by the integrator.
C
C     Call the initialization routine SETSKM, see the documentation at
C     the head of this routine for the precise details of the call.
C
C     Set TS and TOUT for start and end integration times.
C     Set INFORM, WORK arrays as required for time integration,
C                - see SPRINT documentation.
C     Call the SPRINT routine with the residual routine SKMRES.
C
C 2.  Provide a set of routines which describe the precise form of the
C     PDEs to be solved. Three routines must be provided and the names
C     of these routines are fixed. These routines are:
C        SPDEF    forms the functions C, Q and R of the PDE in a
C                 given x-point.
C        BNDR     forms the functions BETA and GAMMA associated with the
C                 boundary conditions for the PDE. W.r.t. the function
C                 GAMMA there is a complication because SPRINT requires
C                 that the part of the residual that depends on the time-
C                 derivative must be supplied separately. Therefore the
C                 GAMMA function has to be split up into:
C                 GAMMA(x,t,u,u ,u ) =
C                           - -x -t
C                                GAMDOT(x,t,u,u ,u ) + GAMMAG(x,t,u,u ).
C                                           - -x -t               - -x
C        UINIT    supplies the initial values of the PDE part.
C                 An initial uniform grid is generated by SKMRES and
C                 provided in Y(NPDE+1,I), I=1,NPTS. If required, a user
C                 can redefine the mesh in a nonuniform way.
C     The headers of these routines are:
C
C      SUBROUTINE SPDEF (T, X, NPDE, U, DUDX, C, Q, R, IRES)
C      INTEGER NPDE, IRES
C      REAL T, X
C      REAL U(NPDE), DUDX(NPDE), C(NPDE,NPDE), Q(NPDE), R(NPDE)
C
C      SUBROUTINE BNDR (T, BETA, GAMDOT, GAMMAG, U, DUDX, UDOT, NPDE,
C     +                 LEFT, IRES)
C      INTEGER NPDE, IRES
C      LOGICAL LEFT
```

```
C       REAL T
C       REAL BETA(NPDE), GAMDOT(NPDE), GAMMAG(NPDE),
C      +     U(NPDE), DUDX(NPDE), UDOT(NPDE)
C
C       SUBROUTINE UINIT (NPDE, NPTS, Y)
C       INTEGER NPDE, NPTS
C       REAL Y(NPDE+1,NPTS)
C
C
C Example problem
C ---------------
C-----------------------------------------------------------------------
```

The easiest way to describe how the problem description routines should be written is by a simple example. Consider the following problem from electrodynamics, which can be found, e.g., in Bakker[1]

$$\begin{aligned} u_t &= \epsilon p\, u_{xx} - g(u-v) \\ v_t &= p\, v_{xx} + g(u-v) \end{aligned} \qquad \text{(so } m = 0 \text{ and NPDE} = 2\text{)}, \tag{5.2}$$

with

$$g(z) = e^{\frac{\eta z}{3}} - e^{\frac{-2\eta z}{3}},$$

$0 \leqslant x \leqslant 1$ and $0 \leqslant t \leqslant 4$; $\epsilon = 0.143$, $p = 0.1743$, and $\eta = 17.19$.
The left boundary condition (`LEFT = .TRUE.`) is given by

$$u_x = 0 \text{ and } v = 0 \text{ at } x = 0,$$

the right boundary condition (`LEFT = .FALSE.`) is

$$u = 1 \text{ and } v_x = 0 \text{ at } x = 1,$$

and the initial conditions are

$$u = 1 \text{ and } v = 0 \text{ at } t = 0.$$

The routines `UINIT`, `SPDEF` and `BNDR` are listed below. The component u of the PDE at the i-th grid point is held as `Y(1,i)` in the package, the component v as `Y(2,i)`; the i-th grid point itself is stored in `Y(3,i)`.

```
      SUBROUTINE UINIT (NPDE, NPTS, Y)
C
C Routine for PDE initial values.
C Entry:
C   Y(NPDE+1,i) = X_i; uniform mesh, generated by package
C Exit:
C   Y(NPDE+1,i) = X_i; mesh, optionally changed by user
C   Y(     k,i) = u_k(X_i,t0); initial value of k-th component
C                                    i = 1,.., NPTS
C
      INTEGER NPDE, NPTS
      REAL Y(NPDE+1,NPTS)
C
      INTEGER I
```

```
      DO 10 I = 1, NPTS
         Y(1,I) = 1.0
         Y(2,I) = 0.0
   10 CONTINUE

      RETURN
      END



      SUBROUTINE SPDEF (T, X, NPDE, U, DUDX, C, Q, R, IRES)
C
C Routine to describe the body of the PDE system.
C The PDE is written as
C   NPDE               k                      -m   m
C   sum C  (x,t,u,u ) u  + Q (x,t,u,u )  = x   (x  R (x,t,u,u ) ) .
C   k=1  jk        x   t    j        x             j        x   x
C The functions C, Q and R must be defined in this routine.
C
      INTEGER NPDE, IRES
      REAL T, X
      REAL U(NPDE), DUDX(NPDE), C(NPDE,NPDE), Q(NPDE), R(NPDE)
C
      INTEGER J, K
      REAL EPS, ETA, GZ, P, Z
      DATA EPS /0.143/, ETA /17.19/, P /0.1743/

      DO 10 K = 1, NPDE
         DO 20 J = 1, NPDE
            C(J,K) = 0.0
   20    CONTINUE
         C(K,K) = 1.0
   10 CONTINUE

      Z  = U(1) - U(2)
      GZ = EXP(ETA*Z/3) - EXP(-2*ETA*Z/3)
      Q(1) =  GZ
      Q(2) = -GZ

      R(1) = EPS*P * DUDX(1)
      R(2) =     P * DUDX(2)

      RETURN
      END



      SUBROUTINE BNDR (T, BETA, GAMDOT, GAMMAG, U, DUDX, UDOT, NPDE,
     +                 LEFT, IRES)
C
C Boundary conditions routine
```

```
C The boundary conditions are written as
C   BETA (x,t) R (x,t,u,u ) = GAMMA (x,t,u,u ,u )
C       j       j        x         j        x  t
C                           = GAMDOT (x,t,u,u ,u ) + GAMMAG (x,t,u,u ).
C                                     j        x  t          j        x
C The functions BETA, GAMDOT and GAMMAG should be defined in this
C routine.
C
      INTEGER NPDE, IRES
      LOGICAL LEFT
      REAL T
      REAL BETA(NPDE), GAMDOT(NPDE), GAMMAG(NPDE),
     +     U(NPDE), DUDX(NPDE), UDOT(NPDE)
C
      IF (LEFT) THEN
         BETA  (1) = 1.0
         GAMDOT(1) = 0.0
         GAMMAG(1) = 0.0
         BETA  (2) = 0.0
         GAMDOT(2) = 0.0
         GAMMAG(2) = U(2)
      ELSE
         BETA  (1) = 0.0
         GAMDOT(1) = 0.0
         GAMMAG(1) = U(1) - 1.0
         BETA  (2) = 1.0
         GAMDOT(2) = 0.0
         GAMMAG(2) = 0.0
      ENDIF

      RETURN
      END
```

Note, that Neumann boundary conditions can be implemented in two ways. The first is to put $\beta = 0$ and $\gamma = u_x$ which will result in equal solution values for the boundary point and its neighbor. The second is setting $\beta = 1$ and $\gamma = 0$, as is done above. Now the solution values will be only the same in the limit case.

For a description of the call of the initialization routine `SETSKM` we copy the header and the documentation of this routine below.

```
C
      SUBROUTINE SETSKM (NEQN, NPDE, NPTS, XL, XR, TAU, KAPPA, ALPHA,
     +                   Y, RWK, NRWK, M, TS, IBAND, IRES)
C
C-----------------------------------------------------------------------
C Purpose:
C -------
C Initializing routine for SPRINT moving-grid spatial discretization.
C
```

```
C Parameters:
C ----------
      INTEGER NEQN, NPDE, NPTS, NRWK, M, IRES
      REAL XL, XR, TAU, KAPPA, ALPHA, TS
      REAL Y(*), RWK(NRWK)
C
C NEQN  Exit: the size of the ODE system generated when the PDE +
C       the grid equations are discretized. This value is (NPDE+1).NPTS.
C NPDE  Entry: the number of PDEs.
C NPTS  Entry: the number of spatial mesh points, including the
C       boundary points.
C XL    Entry: left boundary point.
C XR    Entry: right boundary point.
C TAU   Entry: time-smoothing parameter.
C       If the initial grid satisfies the grid equation with TAU=0 at
C       TS=0, TAU can be chosen equal to zero. If this is not the case
C       and if the initial grid has to be adapted, or if time-smoothing
C       is desired a typical value of TAU = 1E-3, but TAU should be
C       related to the time scale of the problem.
C KAPPA Entry: spatial smoothing parameter (REAL).
C       KAPPA = 2.0 was found to be satisfying for all problems tested.
C       For less spatial smoothing KAPPA = 1.0 will suffice.
C ALPHA Entry: monitor regularizing parameter.
C       ALPHA = 1.0 results in the arclength monitor. However, if the
C       solution is very flat, ALPHA should be taken smaller.
C Y     Exit: array of length >= (NPDE+1).NPTS that contains the initial
C       (uniformly spaced) grid and the corresponding initial PDE
C       solution values. This array must be passed across as a one-
C       dimensional array of length NEQN to the SPRINT package. This
C       array is ordered as
C          PDE comp.  : Y((NPDE+1)*l + j) l=0,...,NPTS-1,
C                                         j=1,...,NPDE
C          grid points: Y((NPDE+1)*(l+1)) l=0,...,NPTS-1.
C RWK   workspace of length NRWK for the residual routine SKMRES which
C       actually performs the semi-discretization of the PDEs and
C       defines the grid equations.
C NRWK  Entry: dimension of workspace RWK.
C       NRWK must be >= (NPDE+1).NPTS + (6+NPDE).NPDE.
C M     Entry: integer >= 0 which determines the coordinate system used.
C          0: Cartesian coordinates,
C          1: cylindrical polar coordinates,
C          2: spherical polar coordinates.
C TS    Entry: the time at which the integration starts.
C IBAND Exit: an upper bound on the half bandwidth of the Jacobian
C       matrix when this module is used. This parameter should be
C       supplied to MATSET if SPRINT is called with banded matrix
C       routines.
C IRES  Exit: this parameter is set to -1 if an error is found by
C       this routine.
C
C-----------------------------------------------------------------------
```

6. Numerical examples

We applied the moving-grid interface to the three problems that are used as numerical examples in [10]. The results were comparable with those in that paper as could be expected since the spatial discretization method used in the interface results for problems in a Cartesian coordinate system ($m = 0$) in a central difference method as was used in [10]. Apart from these three problems we tested our moving-grid interface on the problem stemming from electrodynamics that was also used as example in the description of the usage of the interface, and on a nonlinear diffusion equation that can be formulated as a PDE in cylindrical polar coordinates having a singularity at the axis. We present our numerical results in plots wherein an accurate reference solution is denoted by a solid line while the marks show the grid distribution and the PDE approximation. The integration history is given by

STEPS: total number of successful time-steps,
JACS: total number of Jacobian evaluations,
BS: total number of backsolves.

6.1. Problem I: A problem from electrodynamics

This problem is described in Section 5 (formula (5.2)). The steady-state problem has been discussed in [4], pg. 113-116. It is a singular perturbation problem and it develops very rapidly steep boundary layers and evolves after a certain time (approximately at $t = 3.0$) into a smooth steady-state solution.

A reference solution has been computed using a fixed grid with 500 equidistant points and a time-tolerance value for the ODE solver of 10^{-7}. Bakker[1] solved this problem on a fixed nonuniform grid using a grid-spacing of 0.01 near the boundaries. We have solved it starting with a uniform grid with 21 points and using a time-tolerance value 10^{-3} for the Gear-type ODE solver in **SPRINT**. We specified an initial time-step of 10^{-5}.

The moving-grid parameters were $\tau = 0$, $\kappa = 2$, and $\alpha = 0.01$. We also experimented with $\alpha = 1$ but for this choice the grid was held uniform for too long, resulting in a slightly worse performance at the boundaries for the initial time plots.

Figure 6.1 shows a plot of the grid movement over the total time interval with a close-up at the start of the time integration, and two plots of the PDE solution components at times $t = 0.001$, 0.01, 0.1, and 4.0. At $t = 4.0$ the steady-state solution has been reached. The integration costs were STEPS = 67, JACS = 25, and BS = 168. For the integration from $t = 1.0$ to $t = 4.0$ only 7 steps were needed, which shows that the steady-state performance of the method is quite good.

6.2. Problem II: A nonlinear diffusion problem

Our second example is a two-dimensional nonlinear diffusion equation of the form

$$u_t = \nabla.(u^n \nabla u) \qquad (n > 0).$$

This problem is taken from Johnson[7]. The solution exhibits steep moving fronts which are decaying in time. Assuming radial symmetry this problem can be reduced to the one-dimensional equation

$$u_t = x^{-1} \frac{\partial}{\partial x}(x u^n u_x), \qquad 0 < x < 1, \quad t_0 < t,$$

which is an example of a cylindrical polar problem with a singularity at the axis ($m = 1$). In [7] it is shown that a solution exists (assuming constant total thermal energy); this solution can be expressed as

$$u(x,t) = \frac{\sigma}{\rho^2(t)} \left[1 - \left(\frac{x}{\rho(t)}\right)^2 \right]^{\frac{1}{n}} \qquad \text{for } 0 \leqslant x \leqslant \rho(t), \quad \text{and } 0 \text{ for } x > \rho(t),$$

with

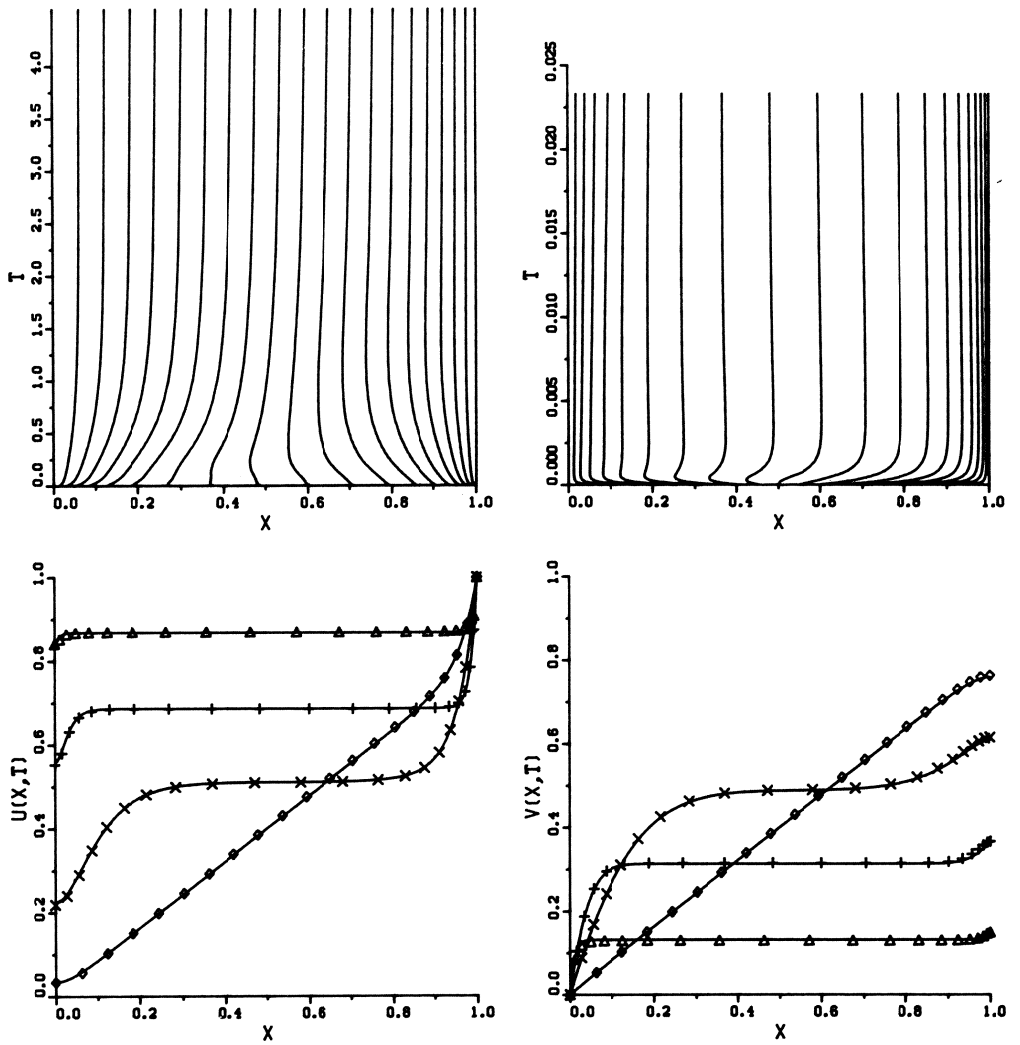

FIGURE 6.1. Problem I (NPTS = 21). Grid + close-up near initial time and solution components at times $t = 0.001, 0.01, 0.1, 4.0$ ($\Delta, +, \times, \square$).

$$\sigma = \frac{2(1+n)}{\pi}\frac{\Gamma(1/n+1)}{\Gamma(1/n)} \quad \text{and} \quad \rho(t) = \left[\sigma^n \frac{4(n+1)}{n} t\right]^{\frac{1}{2(n+1)}}.$$

The boundary conditions for the PDE are

$$u_x(0,t) = 0, \quad u(1,t) = 0, \quad t > t_0.$$

and the initial solution is the exact solution evaluated at t_0.

We solved this problem for $n = 1$ and $t_0 = 10^{-3}$ using 21 grid points and starting again on a uniform mesh with a time-tolerance of 10^{-3} and an initial time-step of 10^{-5}. The moving-grid parameters were $\tau = 10^{-3}$, $\kappa = 2$, and $\alpha = 1$. Observe that the initial uniform mesh does not satisfy the grid equation (2.6).

The integration costs amount to STEPS = 63, JACS = 17, and BS = 152. In Figure 6.2 plots of the grid movement and of the PDE solution at times $t = 0.001$, 0.006, 0.011, and 0.031 are shown. It

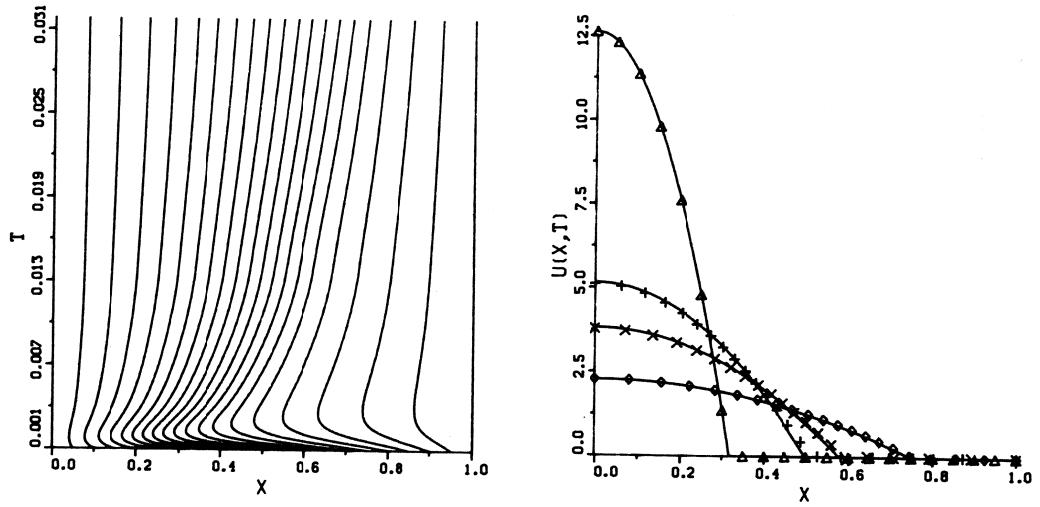

FIGURE 6.2. Problem II. Grid and solution at times $t = 0.001, 0.006, 0.011, 0.031$ ($\Delta, +, \times, \square$).

can be seen that the approximation suffers a bit from smearing. Adjustment of the parameters was no remedy for this problem; adding more points did give a better solution, but we have the feeling that the monitor function should be adapted so as to place more points in the curvature rather than in the slope of the solution.

ACKNOWLEDGEMENTS

We are grateful to Martin Berzins for the helpful discussion during his visit to CWI and for sending us the latest version of the code implementing the Skeel & Berzins discretization on a fixed grid. We want to thank Jan Verwer for his interest and his guidance during this project.

REFERENCES

1. M. BAKKER (1977). *Software for Semi-discretization of Time-dependent Partial Differential Equations in One Space Variable,* Report NW 52/77, Mathematisch Centrum, Amsterdam.
2. M. BERZINS and R.M. FURZELAND (1985). *A User's Manual for SPRINT - A Versatile Software Package for Solving Systems of Algebraic, Ordinary and Partial Differential Equations: Part 1 - Algebraic and Ordinary Differential Equations,* Report TNER.85.058, Thornton Research Centre, Shell Research Ltd., U.K..
3. M. BERZINS and R.M. FURZELAND (1986). *A User's Manual for SPRINT - A Versatile Software Package for Solving Systems of Algebraic, Ordinary and Partial Differential Equations: Part 2 - Solving Partial Differential Equations,* Report No. 202, Department of Computer Studies, The University of Leeds.
4. J.C.P. BUS (ED.) (1976). *Colloquium Numerieke Programmatuur, MC Syllabus 29.1a,* Mathematisch Centrum, Amsterdam.
5. E.A. DORFI and L. O'C. DRURY (1987). Simple Adaptive Grids for 1-D Initial Value Problems, *J. Comput. Phys.*, 69, 175-195.
6. R.M. FURZELAND, J.G. VERWER, and P.A. ZEGELING (1988). *A Numerical Study of Three Moving Grid Methods for One-Dimensional Partial Differential Equations which are based on the Method of Lines,* Report NM-R8806, Centre for Mathematics and Computer Science (CWI), Amsterdam

(submitted to J. Comput. Physics).

7. I.W. JOHNSON (1985). *Moving Finite Elements for Nonlinear Diffusion Problems in One and Two Dimensions,* Numerical Analysis Report no. 12/85, University of Reading.
8. R.F. SINCOVEC and N.K. MADSEN (1975). Software for Nonlinear Partial Differential Equations, *ACM Trans. Math. Software 1*, 232-260.
9. R.D. SKEEL and M. BERZINS (1987). *A Method for the Spatial Discretization of Parabolic Equations in one Space Variable,* Report No. 217, Department of Computer Studies, The University of Leeds.
10. J.G. VERWER, J.G. BLOM, R.M. FURZELAND, and P.A. ZEGELING. *A Moving-Grid Method for One-Dimensional PDEs based on the Method of Lines,* Report NM-R8818, Centre for Mathematics and Computer Science (CWI), Amsterdam. (Paper presented at the 'Workshop on Adaptive Methods for Partial Differential Equations', Rensselaer Polytechnic Institute, Troy, New York, Oct.13-15, 1988. To appear in the proceedings.).

APPENDIX A

Test functions $\psi_i^{(m)}$:

$$\psi_0^{(m)}(x) = \begin{cases} 0 & \text{if singular} \\ \int_x^{X_1} y^{-m}dy \Big/ \int_{x_L}^{X_1} y^{-m}dy & \text{if regular} \end{cases} \qquad x_L \leqslant x \leqslant X_1 \quad ,$$

$$0 \qquad \text{elsewhere}$$

$$\psi_i^{(m)}(x) = \begin{cases} 1 - \psi_{i-1}^{(m)}(x) & X_{i-1} \leqslant x \leqslant X_i \\ \int_x^{X_{i+1}} y^{-m}dy \Big/ \int_{X_i}^{X_{i+1}} y^{-m}dy & X_i \leqslant x \leqslant X_{i+1} \quad i = 1, \ldots, N, \\ 0 & \text{elsewhere} \end{cases}$$

$$\psi_{N+1}^{(m)}(x) = \begin{cases} 1 - \psi_N^{(m)}(x) & X_N \leqslant x \leqslant x_R \\ 0 & \text{elsewhere} \end{cases} .$$

Note that

$$\int_a^b y^{-m}dy = \begin{cases} b - a & m = 0 \\ \ln(b/a) & m = 1 \\ -(b^{-1} - a^{-1}) & m = 2 \end{cases} .$$

Integrals $\zeta_{i+\frac{1}{2}}^{m+1}$:

$$\zeta_{i+\frac{1}{2}}^{m+1} = \begin{cases} 0 & \text{if singular and } i = 0 \\ \int_{X_i}^{X_{i+1}} y dy \Big/ \int_{X_i}^{X_{i+1}} y^{-m}dy = \dfrac{X_{i+1}^2 - X_i^2}{2} \Big/ \int_{X_i}^{X_{i+1}} y^{-m}dy & \text{otherwise} \end{cases} .$$

Trial functions $\phi_i^{(m)}$:
In the regular case

$$\phi_i^{(m)}(x) = \psi_i^{(m)}(x), \quad i = 0, \ldots, N+1$$

and in the singular case

$$\phi_0^{(m)}(x) = \begin{cases} \dfrac{X_1^2 - x^2}{X_1^2 - x_L^2} & x_L \leqslant x \leqslant X_1 \\ 0 & \text{elsewhere} \end{cases} ,$$

$$\phi_i^{(m)}(x) = \begin{cases} 1 - \phi_{i-1}^{(m)}(x) & X_{i-1} \leqslant x \leqslant X_i \\ \dfrac{X_{i+1}^2 - x^2}{X_{i+1}^2 - X_i^2} & X_i \leqslant x \leqslant X_{i+1} \\ 0 & \text{elsewhere} \end{cases} \qquad i = 1, \ldots, N,$$

$$\phi_{N+1}^{(m)}(x) = \begin{cases} 1 - \phi_N^{(m)}(x) & X_N \leqslant x \leqslant x_R \\ 0 & \text{elsewhere} \end{cases} .$$

Quadrature points $\xi_{i+1/2}$:
In the regular case

$$\xi_{i+1/2} = \begin{cases} \dfrac{X_{i+1} + X_i}{2} & m = 0 \\ \dfrac{X_{i+1} - X_i}{\ln(X_{i+1}/X_i)} & m = 1 \\ \dfrac{\ln(X_{i+1}/X_i)}{-(X_{i+1}^{-1} - X_i^{-1})} & m = 2 \end{cases}$$

and in the singular case

$$\xi_{i+1/2} = \frac{2}{3} \frac{X_{i+1}^3 - X_i^3}{X_{i+1}^2 - X_i^2} .$$

APPENDIX B

```
C-----------------------------------------------------------------------
C
C Moving grid discretization module SPMDIF
C ----------------------------------------
C This module discretizes systems of partial differential equations
C in one space variable on a moving grid. The class of equations that
C can be handled is given by
C
C   NPDE                    k                          -m  m
C   sum  C   (x,t, u, u ) u  + Q (x,t, u, u ) =  x  (x  R (x,t, u, u ))
C   k=1   j,k      - -x   t    j      - -x             j      - -x  x
C
C where            1         NPDE T
C           u = ( u , ... , u     )   ,  j = 1,... , NPDE,
C           -
C      k
C and u  is the partial derivative wrt time of the k-th component of u.
C      t
C
C The functions C, Q, and R are assumed to be continuous w.r.t. the
C space variable.
C
C The independent variables x and t satisfy  x  < x < x   with x  and x
C                                             L        R       L      R
C fixed and  t > t .
C                0
C The boundary conditions have the form
C
C   BETA(x,t).R(x,t,u,u ) = GAMMA(x,t,u,u ,u )  at x = x , x  ,
C                  - -x              - -x -t           L   R
C
C where not all of the functions BETA and GAMMA are set to zero.
C
C The initial conditions are given by
C              0
C   u (x,t )  = u (x)  for x  <= x <= x .
C   -     0     -          L          R
C The discretization method for the PDE in Lagrangian formulation
C used by this module is based on a lumped Galerkin / Petrov-Galerkin
C method and evaluates the PDE functions in a point between
C the (moving) grid points.
C
C References:
C   Fixed-grid spatial discretization
C            Skeel R.D. and Berzins M.
C            A Method for the Spatial Discretisation of Parabolic
C            Equations in one Space Variable.
C            Leeds Report no 217,
C            Dept. of Computer Studies, The University.
C   Grid movement
C            Verwer J.G., Blom J.G., Furzeland R.M. and Zegeling P.A.
C            A Moving-Grid Method for One-Dimensional PDEs based on
C            the Method of Lines.
C            Report NM-R8818,
C            Centre for Mathematics and Computer Science, Amsterdam.
C   Interface
C            Blom J.G. and Zegeling P.A.
C            A Moving-Grid Interface for Systems of One-Dimensional
```

```
C            Time-Dependent Partial Differential Equations.
C            Report NM-R89??,
C            Centre for Mathematics and Computer Science, Amsterdam.
C
C-----------------------------------------------------------------------
C
C How to use this module
C ----------------------
C 1.  Set NPDE = # PDEs to be solved.
C     Set NPTS = # mesh points to be used.
C                (NC=NPTS-2 is # internal points)
C     Set M for space coordinate type
C         = 0 for Cartesian,  = 1 for cylindrical, = 2 for spherical.
C     Specify a workspace of size at least  (NPDE+1)*NPTS+(6+NPDE)*NPDE
C     for use by the routine SKMRES which defines the ODE system being
C     solved by the integrator.
C
C     Call the initialization routine SETSKM, see the documentation at
C     the head of this routine for the precise details of the call.
C
C     Set TS and TOUT for start and end integration times.
C     Set INFORM, WORK arrays as required for time integration,
C                - see SPRINT documentation.
C     Call the SPRINT routine with the residual routine SKMRES.
C
C 2.  Provide a set of routines which describe the precise form of the
C     PDEs to be solved. Three routines must be provided and the names
C     of these routines are fixed. These routines are:
C        SPDEF    forms the functions C, Q and R of the PDE in a
C                 given x-point.
C        BNDR     forms the functions BETA and GAMMA associated with the
C                 boundary conditions for the PDE. W.r.t. the function
C                 GAMMA there is a complication because SPRINT requires
C                 that the part of the residual that depends on the time-
C                 derivative must be supplied separately. Therefore the
C                 GAMMA function has to be split up into:
C                 GAMMA(x,t,u,u ,u ) =
C                           - -x -t
C                                 GAMDOT(x,t,u,u ,u ) + GAMMAG(x,t,u,u ).
C                                       - -x -t               - -x
C        UINIT    supplies the initial values of the PDE part.
C                 An initial uniform grid is generated by SKMRES and
C                 provided in Y(NPDE+1,I), I=1,NPTS. If required, a user
C                 can redefine the mesh in a nonuniform way.
C     The headers of these routines are:
C
C      SUBROUTINE SPDEF (T, X, NPDE, U, DUDX, C, Q, R, IRES)
C      INTEGER NPDE, IRES
C      REAL T, X
C      REAL U(NPDE), DUDX(NPDE), C(NPDE,NPDE), Q(NPDE), R(NPDE)
C
C      SUBROUTINE BNDR (T, BETA, GAMDOT, GAMMAG, U, DUDX, UDOT, NPDE,
C     +                 LEFT, IRES)
C      INTEGER NPDE, IRES
C      LOGICAL LEFT
C      REAL T
C      REAL BETA(NPDE), GAMDOT(NPDE), GAMMAG(NPDE),
C     +     U(NPDE), DUDX(NPDE), UDOT(NPDE)
C
C      SUBROUTINE UINIT (NPDE, NPTS, Y)
C      INTEGER NPDE, NPTS
```

```
C      REAL Y(NPDE+1,NPTS)
C
C
C Example problem
C ---------------
C The easiest way to describe how the problem description routines
C should be written is by a simple example. Consider the following
C problem from electrodynamics
C    u  = eps.p.u   - g(u-v)
C     t          xx
C and                                (so m = 0 and NPDE = 2)
C    v  =     p.v   + g(u-v)
C     t          xx
C with
C    g(z) = exp(eta.z/3) - exp(-2.eta.z/3) ,
C    0 <= x <= 1  and  0 <= t <= 4;
C    eps = 0.143, p = 0.1743, and eta = 17.19.
C
C The left boundary condition (LEFT = .TRUE.) is given by
C    u  = 0  and  v = 0   at x = 0,
C     x
C the right boundary condition (LEFT = .FALSE.) is
C    u = 0  and  v  = 0   at x = 1,
C                x
C and the initial conditions are
C    u = 1  and  v = 0  at t = 0.
C
C The routines UINIT, SPDEF and BNDR are listed below.
C The component u of the PDE at the i-th grid point is held as Y(1,i)
C in the package, the component v as Y(2,i); the i-th grid point
C itself is stored in Y(3,i).
C
C
C
C     SUBROUTINE UINIT (NPDE, NPTS, Y)
C
C Routine for PDE initial values.
C Entry:
C   Y(NPDE+1,i) = X_i; uniform mesh, generated by package
C Exit:
C   Y(NPDE+1,i) = X_i; mesh, optionally changed by user
C   Y(     k,i) = u_k(X_i,t0); initial value of k-th component
C                                  i = 1,.., NPTS
C
C     INTEGER NPDE, NPTS
C     REAL Y(NPDE+1,NPTS)
C
C     INTEGER I
C
C     DO 10 I = 1, NPTS
C        Y(1,I) = 1.0
C        Y(2,I) = 0.0
C  10 CONTINUE
C
C     RETURN
C     END
C
C
C
C     SUBROUTINE SPDEF (T, X, NPDE, U, DUDX, C, Q, R, IRES)
C
```

```
C Routine to describe the body of the PDE system.
C The PDE is written as
C   NPDE              k                    -m   m
C   sum C  (x,t,u,u ) u  + Q (x,t,u,u )  = x   (x  R (x,t,u,u ) ) .
C   k=1  jk        x   t    j        x            j       x   x
C The functions C, Q and R must be defined in this routine.
C
C     INTEGER NPDE, IRES
C     REAL T, X
C     REAL U(NPDE), DUDX(NPDE), C(NPDE,NPDE), Q(NPDE), R(NPDE)
C
C     INTEGER J, K
C     REAL EPS, ETA, GZ, P, Z
C     DATA EPS /0.143/, ETA /17.19/, P /0.1743/
C
C     DO 10 K = 1, NPDE
C        DO 20 J = 1, NPDE
C           C(J,K) = 0.0
C  20    CONTINUE
C        C(K,K) = 1.0
C  10 CONTINUE
C
C     Z  = U(1) - U(2)
C     GZ = EXP(ETA*Z/3) - EXP(-2*ETA*Z/3)
C     Q(1) =  GZ
C     Q(2) = -GZ
C
C     R(1) = EPS*P * DUDX(1)
C     R(2) =     P * DUDX(2)
C
C     RETURN
C     END
C
C
C
C     SUBROUTINE BNDR (T, BETA, GAMDOT, GAMMAG, U, DUDX, UDOT, NPDE,
C    +                 LEFT, IRES)
C
C Boundary conditions routine
C The boundary conditions are written as
C   BETA (x,t) R (x,t,u,u ) = GAMMA (x,t,u,u ,u )
C       j       j        x         j        x  t
C                           = GAMDOT (x,t,u,u ,u ) + GAMMAG (x,t,u,u ).
C                                    j        x  t          j        x
C The functions BETA, GAMDOT and GAMMAG should be defined in this
C routine.
C
C     INTEGER NPDE, IRES
C     LOGICAL LEFT
C     REAL T
C     REAL BETA(NPDE), GAMDOT(NPDE), GAMMAG(NPDE),
C    +     U(NPDE), DUDX(NPDE), UDOT(NPDE)
C
C     IF (LEFT) THEN
C        BETA  (1) = 1.0
C        GAMDOT(1) = 0.0
C        GAMMAG(1) = 0.0
C        BETA  (2) = 0.0
C        GAMDOT(2) = 0.0
C        GAMMAG(2) = U(2)
C     ELSE
```

```
C        BETA  (1) = 0.0
C        GAMDOT(1) = 0.0
C        GAMMAG(1) = U(1) - 1.0
C        BETA  (2) = 1.0
C        GAMDOT(2) = 0.0
C        GAMMAG(2) = 0.0
C     ENDIF
C
C     RETURN
C     END
C
C-----------------------------------------------------------------------
C
      SUBROUTINE SETSKM (NEQN, NPDE, NPTS, XL, XR, TAU, KAPPA, ALPHA,
     +                   Y, RWK, NRWK, M, TS, IBAND, IRES)
C
C-----------------------------------------------------------------------
C Purpose:
C -------
C Initializing routine for SPRINT moving-grid spatial discretization.
C
C Parameters:
C ----------
      INTEGER NEQN, NPDE, NPTS, NRWK, M, IRES
      REAL XL, XR, TAU, KAPPA, ALPHA, TS
      REAL Y(*), RWK(NRWK)
C
C NEQN  Exit: the size of the ODE system generated when the PDE +
C       the grid equations are discretized. This value is (NPDE+1).NPTS.
C NPDE  Entry: the number of PDEs.
C NPTS  Entry: the number of spatial mesh points, including the
C       boundary points.
C XL    Entry: left boundary point.
C XR    Entry: right boundary point.
C TAU   Entry: time-smoothing parameter.
C       If the initial grid satisfies the grid equation with TAU=0 at
C       TS=0, TAU can be chosen equal to zero. If this is not the case
C       and if the initial grid has to be adapted, or if time-smoothing
C       is desired a typical value of TAU = 1E-3, but TAU should be
C       related to the time scale of the problem.
C KAPPA Entry: spatial smoothing parameter (REAL).
C       KAPPA = 2.0 was found to be satisfying for all problems tested.
C       For less spatial smoothing KAPPA = 1.0 will suffice.
C ALPHA Entry: monitor regularizing parameter.
C       ALPHA = 1.0 results in the arclength monitor. However, if the
C       solution is very flat, ALPHA should be taken smaller.
C Y     Exit: array of length >= (NPDE+1).NPTS that contains the initial
C       (uniformly spaced) grid and the corresponding initial PDE
C       solution values. This array must be passed across as a one-
C       dimensional array of length NEQN to the SPRINT package. This
C       array is ordered as
C          PDE comp.  : Y((NPDE+1)*l + j) l=0,...,NPTS-1,
C                                         j=1,...,NPDE
C          grid points: Y((NPDE+1)*(l+1)) l=0,...,NPTS-1.
C RWK   workspace of length NRWK for the residual routine SKMRES which
C       actually performs the semi-discretization of the PDEs and
C       defines the grid equations.
C NRWK  Entry: dimension of workspace RWK.
C       NRWK must be >= (NPDE+1).NPTS + (6+NPDE).NPDE.
C M     Entry: integer >= 0 which determines the coordinate system used.
C          0: Cartesian coordinates,
```

```
C          1: cylindrical polar coordinates,
C          2: spherical polar coordinates.
C TS     Entry: the time at which the integration starts.
C IBAND Exit: an upper bound on the half bandwidth of the Jacobian
C       matrix when this module is used. This parameter should be
C       supplied to MATSET if SPRINT is called with banded matrix
C       routines.
C IRES   Exit: this parameter is set to -1 if an error is found by
C       this routine.
C
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
C
C From here the comments are only meant as aid and assistance to
C understand the program. Note that the moving-grid method and therefore
C also the program is still in development. This means, e.g., that not
C all variables in the common blocks are really needed.
C
C
C Four parameters are passed across from here in
C      COMMON /SPSKM/  NPDE1, NC, M, SING
C
C NPDE1 = NPDE+1   the number of PDEs + 1 (for the grid equation).
C NC    = NPTS-2   the number of internal mesh points.
C M     = M_user   = 0,1,2, if resp., Cartesian, cylindrical or
C                            spherical polar coordinates in use.
C SING  = .TRUE.   if PDE has a polar singularity.
C
C Another common block filled with method parameters is initialized
C in this routine
C      COMMON /METPAR/ MONTYP, TAUABS, TAUREL, RKAPPA, ALFA
C
C MONTYP type of monitor: 1: (ALFA+!!ux!!**2 )**(1/2)
C                         2: (ALFA+!!uxx!!**2)**(1/4)
C       (initialized in monitor routine)
C TAUABS abs. part time-smoothing parameter grid equation (= TAU)
C TAUREL rel. part; (old: TAU = TAUABS + TAUREL.DELTA_T)  (= 0.0)
C RKAPPA spatial smoothing parameter grid equation (= KAPPA)
C ALFA   monitor constant (= ALPHA)
C
C At the present stage of development we use only MONTYP = 1 and
C TAUABS = TAU, TAUREL = 0.
C
C
C Detailed description of workspace:
C --------------------------------
C Size:  NRWK must be >=  (NPDE+1)*NPTS + (6+NPDE)*NPDE
C
C RWK(  1:.+(NPDE+1)*NPTS) G(NPDE+1,0:NC+1), part of residual not
C                          dependent on time-derivative.
C RWK(IW1:.+ NPDE)         UKSI(NPDE), solution values at evaluat. point
C RWK(IW2:.+ NPDE)         UXKSI(NPDE), space derivs. at evaluat. point
C RWK(IW3:.+ NPDE)         BETA(NPDE), boundary function BETA
C RWK(IW4:.+ NPDE)         GAMMAG(NPDE), part of boundary function GAMMA
C                          that is not dependent on the time-derivative.
C RWK(IW5:.+ NPDE)         RC(NPDE), flux at evaluation point
C RWK(IW6:.+ NPDE)         QC(NPDE), source term at evaluation point
C RWK(IW7:.+ NPDE*NPDE)    CC(NPDE,NPDE), udot factor at evaluat. point
C
C-----------------------------------------------------------------------
C
```

```
      CHARACTER*6 PDCODE
      COMMON /DISCHK/ PDCODE
C
      INTEGER IOVFLO
      REAL SUNFLO, SRELPR
      COMMON /SCONS1/ SUNFLO, SRELPR, IOVFLO
C
      INTEGER NPDE1, NC, MM
      LOGICAL SING
      COMMON /SPSKM/ NPDE1, NC, MM, SING
C
      INTEGER MONTYP
      REAL TAUABS, TAUREL, RKAPPA, ALFA
      COMMON /METPAR/ MONTYP, TAUABS, TAUREL, RKAPPA, ALFA
C
      SAVE   /DISCHK/, /SCONS1/, /SPSKM/, /METPAR/
C
C-----------------------------------------------------------------------
C
      INTEGER IW1, IW2, IW3, IW4, IW5, IW6, IW7, IWE

      NPDE1 = NPDE+1
      NC = NPTS-2
      MM = M
      SING = M .GE. 1 .AND. ABS(XL) .LE. SRELPR*(XR-XL)

      TAUABS = TAU
      TAUREL = 0.0
      RKAPPA = KAPPA
      ALFA   = ALPHA

C Fill IBAND; ML >= MU=2*NPDE1, dependent from type monitor used
      IF (MONTYP .EQ. 1) THEN
         IBAND = 2*NPDE1
      ELSE IF (MONTYP .EQ. 2) THEN
         IBAND = 3*NPDE1-1
      ELSE
         STOP ' SETSKM: UNKNOWN MONITOR'
      ENDIF

      NEQN = NPDE1*NPTS

      IW1 =   1 + NEQN
      IW2 = IW1 + NPDE
      IW3 = IW2 + NPDE
      IW4 = IW3 + NPDE
      IW5 = IW4 + NPDE
      IW6 = IW5 + NPDE
      IW7 = IW6 + NPDE
      IWE = IW7 + NPDE*NPDE - 1
      IF (IWE .GT. NRWK) THEN
         CALL SERROR(' SETSKM - ERROR REAL WORKSPACE OF SIZE (=I1) IS
     1               SMALLER THAN REQUIRED (=I2)', 1, 2, NRWK, IWE, 0,
     2               0.0E0, 0.0E0)
         IRES = -1
       ENDIF
       IF (M .LT. 0) THEN
         CALL SERROR(' SETSKM- POLAR PARAMETER M (=I1) LESS THAN ZERO',
     1               1, 1, M, 0, 0, 0.0E0, 0.0E0)
         IRES = -1
       END IF
```

```
      IF (IRES .EQ. -1) RETURN
C
C Initialize grid and PDE variables by appropriate calls.
C
      CALL YINIT (NPDE, NPTS, XL, XR, Y)
C
      PDCODE = 'SPSKLM'

      RETURN
      END
      SUBROUTINE YINIT (NPDE, NPTS, XL, XR, Y)
      INTEGER NPDE, NPTS
      REAL XL, XR
      REAL Y(NPDE+1,0:NPTS-1)
C
      INTEGER NPDE1, NC, M
      LOGICAL SING
      COMMON /SPSKM/ NPDE1, NC, M, SING
      SAVE   /SPSKM/
C
      INTEGER I
      REAL DX, XI
C
C Equidistant grid
      DX = (XR-XL)/(NC+1)
      DO 10 I = 0, NC+1
         XI = XL+I*DX
         Y(NPDE1,I) = XI
   10 CONTINUE
      CALL UINIT (NPDE, NPTS, Y)

      RETURN
      END
      SUBROUTINE SKMRES (NEQN, T, Y, YDOT, RES, IRES, RWK, NRWK)
C
C-----------------------------------------------------------------------
C Purpose:
C -------
C SPRINT enveloping routine to compute the residual of the PDE and of
C the grid equations. SKMRES checks on node-crossing, partitions the
C workspace and calls CWRESU for the spatial discretization and the
C computation of the residual of the PDE in Lagrangian formulation and
C CWRESX for the spatial discretization and the residual computation of
C the grid equations.
C
C Parameters:
C ----------
      INTEGER NEQN, IRES, NRWK
      REAL T
      REAL Y(NEQN), YDOT(NEQN), RES(NEQN), RWK(NRWK)
C
C NEQN  Entry: the size of the ODE system generated when the PDE +
C       the grid equations are discretized.
C T     Entry: evaluation time.
C Y     Entry: array of length NEQN containing the ODE vector consisting
C       of the spatial mesh and the corresponding PDE solution
C       values at time T. This array is ordered as
C          PDE comp.  : Y((NPDE+1)*l + j) l=0,...,NPTS-1,
C                                         j=1,...,NPDE
C          grid points: Y((NPDE+1)*(l+1)) l=0,...,NPTS-1.
C RES   Exit: residual vector.
```

```
C       If IRES = -1 RES should contain only the part of the residual
C       dependent on the time-derivative, if IRES /= -1 RES should
C       contain the full residual  A.ydot - g.
C IRES  Entry: see above.
C       Exit: 2, if setup routine SETSKM has not been called.
C             3, if one of the ODE solutions in the vector Y is not
C                acceptable.
C RWK   working storage of length NRWK.
C NRWK  Entry: dimension of RWK. Should be >= NEQN + (6+NPDE)*NPDE.
C
C-----------------------------------------------------------------------
C
      CHARACTER*6 PDCODE
      COMMON /DISCHK/ PDCODE
C
      INTEGER NPDE1, NC, M
      LOGICAL SING
      COMMON /SPSKM/ NPDE1, NC, M, SING
C
      SAVE   /DISCHK/, /SPSKM/
C
C-----------------------------------------------------------------------
C
      INTEGER I, IW1, IW2, IW3, IW4, IW5, IW6, IW7, J, NPDE
      REAL DELTAT

      IF (PDCODE .NE. 'SPSKLM') THEN
         CALL SERROR (' ERROR IN SKLMRES ROUTINE - THE SETUP ROUTINE'//
     1               ' SETSKM WAS NOT CALLED PRIOR TO SPRINT ENTRY',
     2               1,0,0,0,0,0.0E0,0.0E0)
         IRES = 2
         RETURN
      ENDIF

      NPDE = NPDE1-1

C Check on node-crossing
      DO 5 I = NPDE1, NEQN-NPDE1, NPDE1
         IF (Y(I) .GE. Y(I+NPDE1)) THEN
            CALL SERROR (' SKMRES - NON-MONOTONOUS GRID, '//
     +                   'VALUES OF GRID POINTS (I1, I2) ARE (R1, R2)',
     +                   1, 2, I/NPDE1, I/NPDE1+1, 2, Y(I), Y(I+NPDE1))
            IRES = 3
         ENDIF
    5 CONTINUE

C Partition workspace
      IW1 =   1 + NEQN
      IW2 = IW1 + NPDE
      IW3 = IW2 + NPDE
      IW4 = IW3 + NPDE
      IW5 = IW4 + NPDE
      IW6 = IW5 + NPDE
      IW7 = IW6 + NPDE

C Calculate A.ydot and g for Lagrangian PDE
      CALL CWRESU (T, Y, YDOT, NPDE1-1, NC, M, SING,
     +             RWK(IW1), RWK(IW2), RWK(IW3), RWK(IW4),
     +             RWK(IW5), RWK(IW6), RWK(IW7),
     +             RES, RWK(1), IRES)
      IF (IRES .GE. 3) RETURN
```

```
C Calculate A.ydot and g for grid equations
C If TAUREL > 0, DELTAT should contain the current (or previous) time-
C step (can be extracted from SPRINT common /LSIZES/).
      DELTAT = 0.0
      CALL CWRESX (T, DELTAT, Y, YDOT, NPDE1, NC, RES, RWK(1), IRES)
      IF (IRES .GE. 3) RETURN

      IF (IRES .NE. -1) THEN
C      Full residual needed; RES = A.ydot - g
         DO 10 J = 1, NEQN
            RES(J) = RES(J) - RWK(J)
   10    CONTINUE
      ENDIF

      RETURN
      END
      SUBROUTINE CWRESU (T, Y, YDOT, NPDE, NC, M, SING,
     +                   UKSI, UXKSI, BETA, GAMMAG, RC, QC, CC,
     +                   AYDOT, G, IRES)
C
C-----------------------------------------------------------------------
C Purpose:
C -------
C Compute PDE part of residual equations A.ydot - g.
C   Return A.ydot in AYDOT and g in G to satisfy both
C   SPRINT and DASSL.
C
C Method:
C ------
C The Lagrangian form of the PDE is:
C   NPDE                    .k   k .
C   sum  C   (x,t, u, u ) (u - u  x)  + Q (x,t, u, u )
C   k=1   j,k      -  -x        x        j      -  -x
C
C                                    -m   m
C                                 =  x   (x  R (x,t, u, u ))
C                                           j      -  -x  x
C     where        1         NPDE T
C           u = ( u , ... , u    )   ,  j = 1,... , NPDE,
C           -
C         .k                                                        .
C     and u  is the total time-derivative of the k-th comp. of u and x
C     is the derivative wrt to time of x.
C
C This equation is semi-discretized by a lumped finite-element method
C   (udot and ux.xdot lumped).
C   Integration over the l-1_th interval and over the l_th interval
C   both give an expression for the flux in X . Eliminating this value
C                                            l
C   gives a difference equation for l=1,...,N:
C
C                                                    .                .
C fR  R (ksi ) - fR   R (ksi   ) = fS1  S (ksi ,U ) + fSl   S (ksi   ,U )
C   l  j    l      l-1 j    l-1       l  j    l -l       l-1 j    l-1 -l
C
C   with:
C
C          m-mu   mu
C fR = zeta    ksi        , mu = -1 if PDE singular otherwise mu = m
C   l      l      l
C
```

```
C            m+1   m+1
C fS1 = (zeta   - X   ) / (m+1)
C    l       l     l
C
C           m+1      m+1
C fSl   = (X   - zeta   ) / (m+1)
C    l-1    l        l-1
C
C         .     NPDE            .p        p     .
C S (ksi ,U ) = sum C  (ksi ) [ U (X ) - U (X ) X ] + Q (ksi )
C  j    k -l    p=1  jp    k        l     x  l   l     j    k
C
C R , j_th component of flux evaluated at quadrature point ksi,
C  j
C Q , j_th component of source term evaluated at quadrature point ksi,
C  j
C C  , element j,p of matrix multiplying u , evaluated at ksi.
C  jp                                      t
C
C Left boundary equation, if non-singular:
C                                                  .
C BETA (x ,t) R (x ) = GAMMA (x ,t, U , U (x ), U )
C     j  L     j  L         j  L    -0  -x  L   -0
C
C   with
C              m                          .       m
C R (x ) = (ksi R (ksi ) - fS1 S (ksi ,U )) / x
C  j  L        0 j    0       0 j    0 -0      L
C
C                         if singular:
C         .
C S (ksi ,U )/(m+1) - R (ksi )/ksi  = 0
C  j    0 -0           j    0     0
C
C Right boundary equation:
C                                                 .
C BETA (x ,t) R (x ) = GAMMA (x ,t, U   , U (x ), U   )
C     j  R     j  R         j  R    -N+1  -x  R   -N+1
C
C   with
C                                          .         m
C R (x ) = (fR  R (ksi ) + fSl S (ksi ,U   )) / x
C  j  R       N  j    N       N j    N -N+1      R
C
C
C Parameters:
C ----------
      INTEGER NPDE, NC, M, IRES
      LOGICAL SING
      REAL T
      REAL Y(NPDE+1,0:NC+1), YDOT(NPDE+1,0:NC+1),
     +     UKSI(NPDE), UXKSI(NPDE), BETA(NPDE), GAMMAG(NPDE),
     +     RC(NPDE), QC(NPDE), CC(NPDE,NPDE),
     +     AYDOT(NPDE+1,0:NC+1), G(NPDE+1,0:NC+1)
C
C T       Entry: evaluation time
C Y       Entry: solution and grid at time T.
C         (Y(1:NPDE,L): U_l, Y(NPDE+1,L): X_l)
C YDOT    Entry: derivative of Y at T.
C NPDE    Entry: # PDEs.
C NC      Entry: # internal grid points.
```

```
C M      Entry: coordinate system indicator.
C SING   Entry: true, if PDE has a polar singularity (M>0, x_L=0).
C UKSI   workspace to hold the solution value at an evaluation point.
C UXKSI  workspace to hold the space derivatives at an evaluation point.
C BETA   workspace to hold the boundary function BETA.
C GAMMAG workspace to hold that part of the boundary function GAMMA that
C        is not dependent of YDOT.
C RC     workspace to hold the PDE function R.
C QC     workspace to hold the PDE function Q.
C CC     workspace to hold the PDE function C.
C AYDOT  Exit: A.ydot part of the residual.
C G      Exit: g part of the residual.
C IRES   Exit: 3 if some user function indicated that a solution value
C        is unphysical.
C
C Local variables:
C ---------------
      INTEGER MU
      REAL FSCL, FSC1, SCLMQ, SCL1MQ,
     +     DENOMI, KSI, KSIMU, ZETA, ZETAMU, PHIL, PHILX
C MU     if sing. then -1 else m.
C FSCL   fS1_l.
C FSC1   fSl_l+1.
C SCLMQ  S(ksi_l,Udot_l)   - Q(ksi_l).
C SCL1MQ S(ksi_l,Udot_l+1) - Q(ksi_l).
C DENOMI 1 / [ (X_l,X_l+1) int y**(-m) dy ]
C KSI    quadrature point for l_th interval
C KSIMU  ksi**mu
C ZETA   zeta_l ** (m+1); if sing. and l=0 then
C                         zeta = 0.0
C                         else   (X_l,X_l+1) int y dy * denomi
C ZETAMU zeta_l ** (m-mu)
C PHIL   'left' trial function for l_th interval in eval. point ksi
C        NB. phi_l+1 = 1 - phil
C PHILX  deriv. of phi_l wrt x in eval. point ksi
C        NB. phi_l+1_x = - phil_x
C
C ---------------------------------------------------------------------
C
      INTEGER IP, J, L, NPDE1
      REAL DENPHI, RCJDOT, RCJG, UXL, UXL1, XO, XOM, XL, XL1, XN1M

      NPDE1 = NPDE+1

      XO = Y(NPDE1,0)
      IF (SING) THEN
         MU = -1
      ELSE
         MU = M
      ENDIF
C
C First interval, compute contribution to residual eq. in X_1 and
C   left boundary equation
C
      L = 0
         XL  = XO
         XL1 = Y(NPDE1,L+1)
         IF (.NOT. SING) THEN
            IF (M .EQ. 1) THEN
               DENOMI = 1/LOG(XL1/XL)
            ELSE
```

```
            DENOMI = (1-M)/(XL1**(1-M) - XL**(1-M))
         ENDIF
      ENDIF
      IF (SING) THEN
         KSI = 2/3. * (XL1**3-XL**3) / (XL1**2-XL**2)
         KSIMU = 1/KSI
      ELSE IF (M .EQ. 1) THEN
         KSI = (XL1-XL) * DENOMI
         KSIMU = KSI
      ELSE IF (M .EQ. 2) THEN
         KSI = LOG(XL1/XL) * DENOMI
         KSIMU = KSI*KSI
      ELSE
         KSI = (XL1**(2-M)-XL**(2-M)) / (2-M) * DENOMI
         IF (M .EQ. 0) THEN
            KSIMU = 1.0
         ELSE
            KSIMU = KSI**MU
         ENDIF
      ENDIF
      IF (SING) THEN
         ZETA = 0.0
         ZETAMU = ZETA
         DENPHI = 1/(XL1*XL1-XL*XL)
         PHIL = (XL1*XL1-KSI*KSI)*DENPHI
         PHILX = -2*KSI*DENPHI
      ELSE
         ZETA = 0.5*(XL1*XL1-XL*XL) * DENOMI
         ZETAMU = 1.0
         IF (M .EQ. 1) THEN
            PHIL = LOG(XL1/KSI) * DENOMI
            PHILX = -1/KSI * DENOMI
         ELSE IF (M .EQ. 0) THEN
            PHIL = (XL1 - KSI) * DENOMI
            PHILX = -DENOMI
         ELSE
            PHIL = (XL1**(1-M) - KSI**(1-M)) / (1-M) * DENOMI
            PHILX = -KSI**(-M) * DENOMI
         ENDIF
      ENDIF
      IF (M .EQ. 0) THEN
         XOM = 1.0
      ELSE
         XOM = XO**M
      ENDIF

      IF (.NOT. SING) THEN
C      Get left boundary function values
         DO 5 J = 1, NPDE
            UXKSI(J) = (Y(J,1)-Y(J,0)) / (XL1-XL)
 5       CONTINUE
         CALL BNDR (T, BETA, AYDOT(1,0), GAMMAG, Y(1,0), UXKSI,
     +              YDOT(1,0), NPDE, .TRUE., IRES)
         IF (IRES .EQ. 3) RETURN
      ENDIF

C Compute U and Ux in evaluation point
      DO 10 J = 1, NPDE
         UKSI(J)  = Y(J,L)*PHIL  + Y(J,L+1)*(1-PHIL)
         UXKSI(J) = Y(J,L)*PHILX + Y(J,L+1)*(-PHILX)
10    CONTINUE
```

```
C Get C, Q and R in evaluation point
         CALL SPDEF (T, KSI, NPDE, UKSI, UXKSI, CC, QC, RC, IRES)
         IF (IRES .EQ. 3) RETURN

         FSCL = (ZETA- XL**(M+1))/(M+1)
         FSC1 = (XL1**(M+1)-ZETA)/(M+1)
         DO 20 J = 1, NPDE
            SCLMQ  = 0.0
            SCL1MQ = 0.0
            DO 30 IP = 1, NPDE
               UXL1 = (Y(IP,L+2)-Y(IP,L  ))/(Y(NPDE1,L+2)-Y(NPDE1,L  ))
               SCLMQ  = SCLMQ  +
     +                  CC(J,IP)*(YDOT(IP,L  ))
               SCL1MQ = SCL1MQ +
     +                  CC(J,IP)*(YDOT(IP,L+1)-UXL1*YDOT(NPDE1,L+1))
   30       CONTINUE
C Store contribution from l_th interval to residual equation in X_l+1
            AYDOT(J,L+1) = FSC1 * SCL1MQ
            G    (J,L+1) = -ZETAMU*KSIMU*RC(J) - FSC1*QC(J)
C Compute boundary equations
            IF (SING) THEN
C            Bnd.eq. is contribution from 0_th interval to
C            difference eq. in x_L
               AYDOT(J,0) = SCLMQ / (M+1)
               G    (J,0) = RC(J)/KSI - QC(J)/(M+1)
            ELSE
C            Otherwise compute flux in x_L from contribution from 0_th
C            interval to difference equation in x_L, and substitute in
C            user's boundary equation.
C            NB: AYDOT(j,0) = GAMDOT(j).
               RCJDOT = -FSCL * SCLMQ / XOM
               RCJG   = (KSIMU*RC(J) - FSCL*QC(J)) / XOM
               AYDOT(J,0) = -BETA(J)*RCJDOT + AYDOT(J,0)
               G    (J,0) = BETA(J)*RCJG - GAMMAG(J)
            ENDIF
   20    CONTINUE

      DO 100 L = 1, NC-1
C
C Evaluate PDE functions in quadrature point in l_th interval.
C Add contribution from [X_l,X_l+1] to that of previous interval to
C get residual equation in X_l.
C Store contribution from l_th interval to residual equation in X_l+1.
C
         XL  = XL1
         XL1 = Y(NPDE1,L+1)
         IF (M .EQ. 1) THEN
            DENOMI = 1/LOG(XL1/XL)
         ELSE
            DENOMI = (1-M)/(XL1**(1-M) - XL**(1-M))
         ENDIF
         IF (SING) THEN
            KSI = 2/3. * (XL1**3-XL**3) / (XL1**2-XL**2)
            KSIMU = 1/KSI
         ELSE IF (M .EQ. 1) THEN
            KSI = (XL1-XL) * DENOMI
            KSIMU = KSI
         ELSE IF (M .EQ. 2) THEN
            KSI = LOG(XL1/XL) * DENOMI
            KSIMU = KSI*KSI
         ELSE
```

```
               KSI = (XL1**(2-M)-XL**(2-M)) / (2-M) * DENOMI
               IF (M .EQ. 0) THEN
                  KSIMU = 1.0
               ELSE
                  KSIMU = KSI**MU
               ENDIF
            ENDIF
            ZETA = 0.5*(XL1*XL1-XL*XL) * DENOMI
            IF (SING) THEN
               ZETAMU = ZETA
               DENPHI = 1/(XL1*XL1-XL*XL)
               PHIL = (XL1*XL1-KSI*KSI)*DENPHI
               PHILX = -2*KSI*DENPHI
            ELSE
               ZETAMU = 1.0
               IF (M .EQ. 1) THEN
                  PHIL = LOG(XL1/KSI) * DENOMI
                  PHILX = -1/KSI * DENOMI
               ELSE IF (M .EQ. 0) THEN
                  PHIL = (XL1 - KSI) * DENOMI
                  PHILX = -DENOMI
               ELSE
                  PHIL = (XL1**(1-M) - KSI**(1-M)) / (1-M) * DENOMI
                  PHILX = -KSI**(-M) * DENOMI
               ENDIF
            ENDIF

C Compute U and Ux in evaluation point
            DO 110 J = 1, NPDE
               UKSI(J)  = Y(J,L)*PHIL  + Y(J,L+1)*(1-PHIL)
               UXKSI(J) = Y(J,L)*PHILX + Y(J,L+1)*(-PHILX)
  110       CONTINUE
C Get C, Q and R in evaluation point
            CALL SPDEF (T, KSI, NPDE, UKSI, UXKSI, CC, QC, RC, IRES)
            IF (IRES .EQ. 3) RETURN

            FSCL = (ZETA- XL**(M+1))/(M+1)
            FSC1 = (XL1**(M+1)-ZETA)/(M+1)
            DO 120 J = 1, NPDE
               SCLMQ  = 0.0
               SCL1MQ = 0.0
               DO 130 IP = 1, NPDE
                  UXL  = (Y(IP,L+1)-Y(IP,L-1))/(Y(NPDE1,L+1)-Y(NPDE1,L-1))
                  UXL1 = (Y(IP,L+2)-Y(IP,L  ))/(Y(NPDE1,L+2)-Y(NPDE1,L  ))
                  SCLMQ  = SCLMQ  +
     +                     CC(J,IP)*(YDOT(IP,L  )-UXL *YDOT(NPDE1,L  ))
                  SCL1MQ = SCL1MQ +
     +                     CC(J,IP)*(YDOT(IP,L+1)-UXL1*YDOT(NPDE1,L+1))
  130          CONTINUE
C Add contribution over l_th interval to residual equation in X_l
               AYDOT(J,L) = AYDOT(J,L) + FSCL * SCLMQ
               G    (J,L) = G    (J,L) + ZETAMU*KSIMU*RC(J) - FSCL*QC(J)
C Store contribution from l_th interval to residual equation in X_l+1
               AYDOT(J,L+1) = FSC1 * SCL1MQ
               G    (J,L+1) = -ZETAMU*KSIMU*RC(J) - FSC1*QC(J)
  120       CONTINUE
  100    CONTINUE

         L = NC
C
C Add contribution over N_th interval to residual equation in X_N.
```

```
C Compute right boundary equation.
C
         XL  = XL1
         XL1 = Y(NPDE1,L+1)
         IF (M .EQ. 1) THEN
            DENOMI = 1/LOG(XL1/XL)
         ELSE
            DENOMI = (1-M)/(XL1**(1-M) - XL**(1-M))
         ENDIF
         IF (SING) THEN
            KSI = 2/3. * (XL1**3-XL**3) / (XL1**2-XL**2)
            KSIMU = 1/KSI
         ELSE IF (M .EQ. 1) THEN
            KSI = (XL1-XL) * DENOMI
            KSIMU = KSI
         ELSE IF (M .EQ. 2) THEN
            KSI = LOG(XL1/XL) * DENOMI
            KSIMU = KSI*KSI
         ELSE
            KSI = (XL1**(2-M)-XL**(2-M)) / (2-M) * DENOMI
            IF (M .EQ. 0) THEN
               KSIMU = 1.0
            ELSE
               KSIMU = KSI**MU
            ENDIF
         ENDIF
         ZETA = 0.5*(XL1*XL1-XL*XL) * DENOMI
         IF (SING) THEN
            ZETAMU = ZETA
            DENPHI = 1/(XL1*XL1-XL*XL)
            PHIL = (XL1*XL1-KSI*KSI)*DENPHI
            PHILX = -2*KSI*DENPHI
         ELSE
            ZETAMU = 1.0
            IF (M .EQ. 1) THEN
               PHIL = LOG(XL1/KSI) * DENOMI
               PHILX = -1/KSI * DENOMI
            ELSE IF (M .EQ. 0) THEN
               PHIL = (XL1 - KSI) * DENOMI
               PHILX = -DENOMI
            ELSE
               PHIL = (XL1**(1-M) - KSI**(1-M)) / (1-M) * DENOMI
               PHILX = -KSI**(-M) * DENOMI
            ENDIF
         ENDIF
         IF (M .EQ. 0) THEN
            XN1M = 1.0
         ELSE
            XN1M = XL1**M
         ENDIF

C Get right boundary function values
         DO 205 J = 1, NPDE
            UXKSI(J) = (Y(J,NC+1)-Y(J,NC)) / (XL1-XL)
  205    CONTINUE
         CALL BNDR (T, BETA, AYDOT(1,NC+1), GAMMAG, Y(1,NC+1), UXKSI,
     +              YDOT(1,NC+1), NPDE, .FALSE., IRES)
         IF (IRES .EQ. 3) RETURN

C Compute U and Ux in evaluation point
         DO 210 J = 1, NPDE
```

```
            UKSI(J)  = Y(J,L)*PHIL  + Y(J,L+1)*(1-PHIL)
            UXKSI(J) = Y(J,L)*PHILX + Y(J,L+1)*(-PHILX)
  210     CONTINUE
C Get C, Q and R in evaluation point
          CALL SPDEF (T, KSI, NPDE, UKSI, UXKSI, CC, QC, RC, IRES)
          IF (IRES .EQ. 3) RETURN

          FSCL = (ZETA- XL**(M+1))/(M+1)
          FSC1 = (XL1**(M+1)-ZETA)/(M+1)
          DO 220 J = 1, NPDE
             SCLMQ  = 0.0
             SCL1MQ = 0.0
             DO 230 IP = 1, NPDE
                UXL  = (Y(IP,L+1)-Y(IP,L-1))/(Y(NPDE1,L+1)-Y(NPDE1,L-1))
                SCLMQ  = SCLMQ  +
     +                   CC(J,IP)*(YDOT(IP,L  )-UXL *YDOT(NPDE1,L  ))
                SCL1MQ = SCL1MQ +
     +                   CC(J,IP)*(YDOT(IP,L+1))
  230        CONTINUE
C Add contribution over N_th interval to residual equation in X_N
             AYDOT(J,L) = AYDOT(J,L) + FSCL * SCLMQ
             G    (J,L) = G    (J,L) + ZETAMU*KSIMU*RC(J) - FSCL*QC(J)
C Compute flux in x_R and substitute in user's boundary condition
C          NB: AYDOT(j,NC+1) = GAMDOT(j).
             RCJDOT = FSC1 * SCL1MQ / XN1M
             RCJG   = (ZETAMU*KSIMU*RC(J) + FSC1*QC(J)) / XN1M
             AYDOT(J,NC+1) = -BETA(J)*RCJDOT + AYDOT(J,NC+1)
             G    (J,NC+1) = BETA(J)*RCJG - GAMMAG(J)
  220     CONTINUE

      RETURN
      END
      SUBROUTINE CWRESX (T, DELTAT, Y, YDOT, NPDE1, NC, AYDOT, G, IRES)
C
C-----------------------------------------------------------------------
C Purpose:
C -------
C Define grid part of DAE system in general form, i.e. A.ydot and g
C separated to satisfy both SPRINT and DASSL.
C The equations for the moving grid are
C                .                    .
C   nt   + tau.nt        nt + tau.nt
C     i-1        i-1       i        i
C   ----------------  -  ------------  =  0       1 <= i <= N,
C         M                   M
C          i-1                 i
C
C with nt  = n - fac.(n   - 2.n + n   );  fac = rkappa.(rkappa+1)  and
C        i    i        i+1     i   i-1
C
C       n  = 1 / (X   - X ),   n   = n , n    = n .
C        i         i+1   i      -1    0    N+1   N
C
C For simplicity reasons x  and x  are also part of the ODE vector; since
C                         L      R
C the boundaries are fixed we have used as ODEs for these variables
C   .    .
C   X  = X    = 0.
C    0    N+1
C
```

```
C Parameters:
C ----------
      INTEGER NPDE1, NC, IRES
      REAL T, DELTAT
      REAL Y(NPDE1,0:NC+1), YDOT(NPDE1,0:NC+1),
     +     AYDOT(NPDE1,0:NC+1), G(NPDE1,0:NC+1)
C
C T      Entry: evaluation time
C DELTAT Entry: If TAUREL /= 0  a value related to the time-step.
C Y      Entry: solution and grid at time T.
C        (Y(1:NPDE,L): U_l, Y(NPDE+1,L): X_l)
C YDOT   Entry: derivative of Y at T.
C NPDE1  Entry: # PDEs + 1.
C NC     Entry: # internal grid points.
C AYDOT  Exit: A.ydot part of the residual.
C G      Exit: g part of the residual.
C IRES   Exit:  not used.
C
C-----------------------------------------------------------------------
C
      INTEGER MONTYP
      REAL TAUABS, TAUREL, RKAPPA, ALFA
      COMMON /METPAR/ MONTYP, TAUABS, TAUREL, RKAPPA, ALFA
      SAVE   /METPAR/
C
C-----------------------------------------------------------------------
C
      INTEGER I, NPDE
      REAL A0, AM, FAC, G0, GM, NIM1, NI, NIP1, NTI, NTDI, TAU
C
      NPDE = NPDE1-1
C
C ccc Define smoothing factors
      TAU = TAUABS + TAUREL*DELTAT
      FAC = RKAPPA*(RKAPPA+1)
C
C ccc Compute monitor values; store M(I) temp. in G(NPDE1,I), I=0, NC
      CALL XMNTR (Y, G, NPDE, NC)
C
C ccc Compute A.xdot and g for grid equations.
C   Interior equations:
C     A.xdot (I) = TAU/M(I-1).NTDOT(I-1) - TAU/M(I).NTDOT(I)
C     g(I)       = NT(I)/M(I) - NT(I-1)/M(I-1)
C     NT(I) = N(I) - FAC.(N(I-1)-2.N(I)+N(I+1))
C     N (I) = 1 / (X(I+1)-X(I))
      I = 0
         NI   = 1/(Y(NPDE1,I+1)-Y(NPDE1,I  ))
         NIM1 = NI
         NIP1 = 1/(Y(NPDE1,I+2)-Y(NPDE1,I+1))

         NTI  = NI - FAC*(NIM1-2*NI+NIP1)
         NTDI = -
     +          (1+  FAC)*NI  *NI  *(YDOT(NPDE1,I+1)-YDOT(NPDE1,I  )) +
     +               FAC *NIP1*NIP1*(YDOT(NPDE1,I+2)-YDOT(NPDE1,I+1))

         A0 = TAU / G(NPDE1,I) * NTDI
         G0 = NTI / G(NPDE1,I)

      DO 10 I = 1, NC-1
         NIM1 = NI
         NI   = NIP1
```

```
            AM    = AO
            GM    = GO

            NIP1 = 1/(Y(NPDE1,I+2)-Y(NPDE1,I+1))

            NTI   = NI - FAC*(NIM1-2*NI+NIP1)
            NTDI =       FAC *NIM1*NIM1*(YDOT(NPDE1,I   )-YDOT(NPDE1,I-1)) -
     +           (1+2*FAC)*NI   *NI   *(YDOT(NPDE1,I+1)-YDOT(NPDE1,I   )) +
     +                FAC *NIP1*NIP1*(YDOT(NPDE1,I+2)-YDOT(NPDE1,I+1))

            AO = TAU / G(NPDE1,I) * NTDI
            GO = NTI / G(NPDE1,I)

            AYDOT(NPDE1,I) = AM - AO
            G    (NPDE1,I) = GO - GM
         10 CONTINUE
            I = NC
            NIM1 = NI
            NI    = NIP1
            AM    = AO
            GM    = GO

            NIP1 = NI

            NTI   = NI - FAC*(NIM1-2*NI+NIP1)
            NTDI =       FAC *NIM1*NIM1*(YDOT(NPDE1,I   )-YDOT(NPDE1,I-1)) -
     +           (1+  FAC)*NI   *NI   *(YDOT(NPDE1,I+1)-YDOT(NPDE1,I   ))

            AO = TAU / G(NPDE1,I) * NTDI
            GO = NTI / G(NPDE1,I)

            AYDOT(NPDE1,I) = AM - AO
            G    (NPDE1,I) = GO - GM

      C   Boundary equations grid.
      C      Fixed endpoints, xdot=0
            I=0
               AYDOT(NPDE1,I) = YDOT(NPDE1,I)
               G(NPDE1,I) = 0.0
            I=NC+1
               AYDOT(NPDE1,I) = YDOT(NPDE1,I)
               G(NPDE1,I) = 0.0
      C
            RETURN
            END
            SUBROUTINE XMNTR (Y, G, NPDE, N)
            INTEGER NPDE, N
            REAL Y(NPDE+1,0:N+1), G(NPDE+1,0:N+1)
      C
      C-----------------------------------------------------------------------
      C Purpose:
      C -------
      C Compute monitor for grid equation,
      C   M_i = M(x(i+1/2)) = sqrt(alfa + !!ux!!**2)
      C
      C Exit:
      C   G(NPDE+1,i) = M(i)
      C
      C-----------------------------------------------------------------------
      C
            INTEGER MONTYP
```

```
      REAL TAUABS, TAUREL, RKAPPA, ALFA
      COMMON /METPAR/ MONTYP, TAUABS, TAUREL, RKAPPA, ALFA
      SAVE   /METPAR/
C
C-----------------------------------------------------------------------
C
      INTEGER I, K, NPDE1
      REAL DU, DX, SUX2
C
      DATA MONTYP /1/
C
      NPDE1 = NPDE+1

      DO 10 I = 0, N
         SUX2 = 0.0
         DO 20 K = 1, NPDE
            DU = Y(K,I+1)-Y(K,I)
            SUX2 = SUX2 + DU*DU
   20    CONTINUE
         DX = Y(NPDE1,I+1)-Y(NPDE1,I)
         SUX2 = SUX2 / (DX*DX)
         G(NPDE1,I) = SQRT(ALFA + SUX2/NPDE)
   10 CONTINUE

      RETURN
      END
```