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

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. a 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, borrow from Skeel & Berzinc[9], is akin to a nonlinear Galerkin method and is of 2-ad 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 SKEL4 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 SPDAAL 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 Doof & 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

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

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

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

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

\[ n_i := (\Delta X_i)^{-1}, \quad \Delta X_i := X_{i+1} - X_i, \]

\[ \tilde{n}_i := n_i - \kappa (r+1)(n_{i+1} - 2n_i + n_{i-1}), \quad 0 \leq i \leq N, \]

\[ (n_{-1} := n_0, \quad n_N := n_{N+1}), \tag{2.5} \]

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

\[ \frac{\tilde{n}_{i-1} + \tau \tilde{n}_i}{M_{i-1}} = \frac{\tilde{n}_i + \tau \tilde{n}_{i}}{M_{i}}, \quad 1 \leq i \leq N, \tag{2.6} \]

where \( \tau \geq 0 \) is a time-smoothing parameter and \( M_i \) is a monitor function, here chosen as follows

\[ M_i := \sqrt{\alpha + NPD^{-1} \sum_{j=1}^{NPDE} (U_{j+1} - U_j)^2 / (\Delta X_i)^2}, \tag{2.7} \]
which is a semi-discrete representation of the first derivative solution functional
\[ m(u) = \sqrt{\alpha + \|u_x\|^2}. \]

Shortly, \( \kappa \) determines the level of clustering of the grid points and the arclength monitor \( M_t \) determines the shape of the \( X_t \)-distribution. The parameter \( \tau \) prevents the grid movement: from adjusting immediately to new values of the monitor function \( M_t \), 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
\[
\dot{U} - D\dot{X} = F \\
\pi B\dot{X} = g,
\]
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
\[
A(Y)\dot{Y} = L(Y),
\]
where
\[
Y := (\cdots, U_t^1, \cdots, U_t^{\text{NPDE}}, X_t, \cdots)^T.
\]
This stiff ODE system may be solved, e.g., by one of the BDF integrators of \textsc{sprt}. 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 \textsc{sprt} package [2, 3]
\[
\sum_{k=1}^{\text{NPDE}} \lambda^k = x^{-m} R_j(x, t, u, u_x) = x^{-m} \frac{\partial}{\partial x} (x^m R_j(x, t, u, u_x)) = Q_j(x, t, u, u_x)
\]
for \( j = 1, \ldots, \text{NPDE} \) and \( x \in (x_L, x_R), \ t > t_0, \ m \in \{0, 1, 2\} \),
where \text{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 \( x_L > 0 \).

The user-specified boundary conditions belonging to system (3.1) have the master equation form
\[
\beta_j(x, t) R_j(x, t, u, u_x) = \gamma_j(x, t, u, u_x, u_u) \quad \text{at} \ x = x_L \text{ and } x = x_R, \quad \text{for } j = 1, \ldots, \text{NPDE},
\]
where \( \beta_j \) and \( \gamma_j \) are continuous functions of their variables. If \( m > 0 \) and \( x_L = 0 \) the boundedness of the solution near \( x = 0 \) implies \( R_j |_{x=x_L} = 0 \). 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, which 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
\[
u(x,t_0) = u^0(x) \quad \text{for} \ x \in [x_L, x_R],
\]
where \( u^0 \) is a piecewise continuous function of \( x \) with \( \text{NPDE} \) 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 \(C_{jk,0}, R_j, Q_j, \beta_j,\) and \(\gamma_j\), the initial time \(t_0\), the vector \(a^0\) and the numbers \(m\) and \(\text{nPDE}\). 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 \(w := u_t \frac{x}{u} + S_j\) by

\[
S_j = S_j(x, t, \zeta, u, u_t, w) := \sum_{k=1}^{\text{nPDE}} C_{jk,0}(u_t - w) + Q_j.
\]

Then system (3.1) becomes

\[
S_j = x^{-m}(x^m R_j)_x \quad \text{for } j = 1, \ldots, \text{nPDE},
\]

(4.1)

with \(C_{jk,0}, Q_j,\) 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 \(u^k\)

\[
U^k(x) = \sum_{i=0}^{N+1} U^k_i \phi_i(x).
\]

Let \(\psi_i\) denote the test functions. The trial functions \(\phi_i\) and the test functions \(\psi_i\) 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

\[
\int_{x_L}^{x_R} x^m \psi_i S_j dx = \int_{x_L}^{x_R} x^m \psi_i (x^m R_j)_x dx - \int_{x_L}^{x_R} R_j dx.
\]

(4.2)

for \(j = 1, \ldots, \text{nPDE}\) and \(i = 0, \ldots, N + 1\).

Using the fact that \(\psi_i(\xi) = 0\) for \(x \leq X_{i-1}\) and \(x \geq X_{i+1}\) we get for \(i = 1, \ldots, N\) and \(j = 1, \ldots, \text{nPDE}\)

\[
\int_{X_{i-1}}^{X_{i+1}} x^m \psi_i S_j dx = - \int_{X_{i-1}}^{X_{i+1}} x^m \frac{d\psi_i}{dx} R_j dx.
\]

(4.3)

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

\[
S_j^{i-h} \int_{X_{i-1}}^{X_i} x^m \psi_i dx + S_j^{i+h} \int_{X_i}^{X_{i+1}} x^m \psi_i dx =
\]

\[
- \xi_{i-1}^{-h} R_j^{i-h} \int_{X_{i-1}}^{X_i} x^m \frac{d\psi_i}{dx} dx - \xi_{i+1}^{i+h} R_j^{i+h} \int_{X_i}^{X_{i+1}} x^m \frac{d\psi_i}{dx} dx + E,
\]

(4.4)

where
\[
S_j^{\gamma_{i-\frac{1}{2}}} = S_j(\xi_{i-\frac{1}{2}}, t, U(\xi_{i-\frac{1}{2}}), U_\gamma(\xi_{i-\frac{1}{2}}), \dot{U}_\gamma(t), W_i), \quad \text{with } W_i = \frac{U_{i+1} - U_{i-1}}{X_{i+1} - X_{i-1}} \dot{X}_i,
\]
\[
S_j^{\gamma_{i+\frac{1}{2}}} = S_j(\xi_{i+\frac{1}{2}}, t, U(\xi_{i+\frac{1}{2}}), U_\gamma(\xi_{i+\frac{1}{2}}), \dot{U}_\gamma(t), W_i),
\]
\[
R_j^{\gamma_{i+\frac{1}{2}}} = R_j(\xi_{i+\frac{1}{2}}, t, U(\xi_{i+\frac{1}{2}}), U_\gamma(\xi_{i+\frac{1}{2}})),
\]
and \(E\) stands for the total error due to interpolation, quadrature and lumping. For the definition of \(\mu\) we make a distinction between two special cases

i) the regular case \((m = 0 \text{ or } x_L > 0)\): \(\mu = m\)

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

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

\[
\int_{X_i}^{X_{i+1}} (x - \gamma_{\mu,j+1/2}) x^\mu dx = 0. \quad (4.5)
\]

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 \(i = 1, \ldots, N\)

\[
S_j^{\gamma_{i-\frac{1}{2}}} \frac{X_m^{i+1} - X_m^{i-1}}{m+1} + S_j^{\gamma_{i+\frac{1}{2}}} \frac{X_m^{i+\frac{1}{2}} - X_m^{i-\frac{1}{2}}}{m+1} = \xi_{i-\frac{1}{2}}^{\gamma_{i-\frac{1}{2}}} \xi_{i+\frac{1}{2}}^{\gamma_{i+\frac{1}{2}}} R_j^{\gamma_{i+\frac{1}{2}}} - \xi_{i-\frac{1}{2}}^{\gamma_{i-\frac{1}{2}}} \xi_{i-\frac{1}{2}}^{\gamma_{i-\frac{1}{2}}} R_j^{\gamma_{i-\frac{1}{2}}} \quad (4.6)
\]

with

\[
\xi_{i+\frac{1}{2}}^{\gamma_{i+\frac{1}{2}}} = - \frac{X_{i+1}}{X_i} \int_{X_i}^{X_{i+1}} x^{\mu-1} dx, \quad \text{and} \quad \xi_{i-\frac{1}{2}}^{\gamma_{i-\frac{1}{2}}} = 1.
\]

For a list of the test functions \(\psi_j^{(m)}\), the trial functions \(\phi_i^{(m)}\), the quadrature points \(\xi_{i-\frac{1}{2}}\) and the integrals \(\xi_{i+\frac{1}{2}}^{\gamma_{i+\frac{1}{2}}} \) 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) \(\beta_f(x_R,t)R_j|_{x=x_s} = \gamma_f|_{x=x_s}\) is combined with the semi-discrete approximation of (4.2) with \(i = N + 1\)

\[
S_j^{(N+1)-\frac{1}{2}} \frac{X_R^{N+1} - X_R^{N+1}}{m+1} + \xi_{N+\frac{1}{2}}^{\gamma_{N+\frac{1}{2}}} \xi_{N+\frac{1}{2}}^{\gamma_{N+\frac{1}{2}}} R_j^{(N+1)/2} = X_R \ R_j|_{x=x_s} \quad (4.7)
\]

to eliminate \(R_j|_{x=x_s}\).

In the regular case the same procedure is applied to the left boundary equation in (3.2)

\[
\beta_f(x_L,t)R_j|_{x=x_s} = \gamma_f|_{x=x_s}, \quad \text{and}
\]

\[
S_j^{\gamma_{i-\frac{1}{2}}} \frac{X_{L}^{i+\frac{1}{2}} - X_{L}^{i+\frac{1}{2}}}{m+1} - \xi_{i-\frac{1}{2}}^{\gamma_{i-\frac{1}{2}}} \xi_{i-\frac{1}{2}}^{\gamma_{i-\frac{1}{2}}} R_j^{i+\frac{1}{2}} = - X_L \ R_j|_{x=x_s}, \quad (4.8)
\]

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

\[
S_j^{\gamma_{i+\frac{1}{2}}/(m+1)} - \xi_{i+\frac{1}{2}}^{\gamma_{i+\frac{1}{2}}} R_j^{\gamma_{i+\frac{1}{2}}} = 0. \quad (4.9)
\]

This means, that for \(X_1 \to x_L\) the boundary equation tends to \(R_j|_{x=x_s} = 0\), which is a natural constraint for parabolic 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

\[
A(Y) \dot{Y} = L(Y). \quad (4.10)
\]

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 $\tau$, the spatial smoothing parameter $\kappa$, and the monitor regularizing parameter $\alpha$. The choice of the parameters $\kappa$ and $\alpha$ 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 $\tau$ can result in starting problems or a grid lagging behind. A, problem dependent, guidance of how to choose $\tau$ 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 $\tau$ (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 $\tau, \tau \geq 0$. It is obvious that the grid will not be adapted if $\tau$ tends to infinity. On the other hand if $\tau = 0$ the equidistribution relation

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

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, $\tau$ 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 $\tau$ is $10^{-3}$. Note however that $\tau$ should be related to the critical time scale of the problem.

Spatial smoothing parameter $\kappa, \kappa \gg 0$. 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 $\alpha, \alpha > 0$. 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 $\alpha$ 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 case $\alpha$ should be chosen smaller, $\alpha$ could be thought of as related to the integral over the monitor function $\int_0^{x_i} m(u)du$.

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 MVMESSH 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.
How to use this module

1. Set NPDE = # PDEs to be solved.
   Set NPTS = # mesh points to be used.
   (NC=NPTS-2 is # internal points)
   Set M for space coordinate type
   = 0 for Cartesian, = 1 for cylindrical, = 2 for spherical.
   Specify a workspace of size at least (NPDE+1)*NPTS+(6+NPDE)*NPDE
   for use by the routine SKMRES which defines the ODE system being
   solved by the integrator.

   Call the initialization routine SETSKM, see the documentation at
   the head of this routine for the precise details of the call.

   Set TS and TOUT for start and end integration times.
   Set INFORM, WORK arrays as required for time integration,
   - see SPRINT documentation.
   Call the SPRINT routine with the residual routine SKMRES.

2. Provide a set of routines which describe the precise form of the
   PDEs to be solved. Three routines must be provided and the names
   of these routines are fixed. These routines are:

   SPDEF forms the functions $C$, $Q$ and $R$ of the PDE in a
   given $x$-point.

   BNDR forms the functions BETA and GAMMA associated with the
   boundary conditions for the PDE. W.r.t. the function
   GAMMA there is a complication because SPRINT requires
   that the part of the residual that depends on the time-
   derivative must be supplied separately. Therefore the
   GAMMA function has to be split up into:
   \[ \text{GAMMA}(x,t,u,u) = \]
   \[ -x -t \]
   \[ \text{GAMDOT}(x,t,u,u) + \text{GAMMAG}(x,t,u,u). \]

   UINIT supplies the initial values of the PDE part.
   An initial uniform grid is generated by SKMRES and
   provided in \text{Y}(NPDE+1,I), I=1,NPTS. If required, a user
   can redefine the mesh in a nonuniform way.

The headers of these routines are:

SUBROUTINE SPDEF (T, X, NPDE, U, DUDX, C, Q, R, IRES)
INTEGER NPDE, IRES
REAL T, X
REAL U(NPDE), DUDX(NPDE), C(NPDE,NPDE), Q(NPDE), R(NPDE)

SUBROUTINE BNDR (T, BETA, GAMDAT, GAMMAG, U, DUDX, UDRT, NPDE,
                   LEFT, IRES)
INTEGER NPDE, IRES
LOGICAL LEFT
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]

\[ \begin{align*}
    u_t &= \epsilon \rho u_{xx} - g(u - v) \\
    v_t &= p v_{xx} + g(u - v)
\end{align*} \]

(5.2)

with

\[ g(x) = e^{\frac{w}{x}} - e^{-\frac{2w}{x}} \]

where \( w = 0 \leq x \leq 1 \) and \( 0 \leq t \leq 4 \); \( \epsilon = 0.143 \), \( p = 0.1743 \), and \( \eta = 17.19 \).

The left boundary condition (\( \text{LEFT} = \text{.TRUE.} \)) is given by

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

the right boundary condition (\( \text{LEFT} = \text{.FALSE.} \)) 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 \texttt{UINIT, SPDEF} and \texttt{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 \( Y(3,i) \).

\begin{verbatim}
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
C
END
\end{verbatim}
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 \[ \sum_{j=1}^{m} C(x, t, u, u) u_j + Q(x, t, u, u) = \frac{\partial}{\partial x} \left( R(x, t, u, u) \right) \].
C
C The functions C, Q and R must be defined in this routine.
C
INTEGER NPDE, IRES
REAL T, X
REAL UCNPDE, DUDX(NPDE), C(NPDE,NPDE), Q(NPDE), R(NPDE)

INTEGER J, K
REAL EPS, ETA, GI, 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, GAMMAS, U, DUDX, UDOT, NPDE,
                 LEFT, IRES)
C
C Boundary conditions routine
The boundary conditions are written as

\[ \beta_j(x,t,u) R_j(x,t,u,u) = \gamma_j(x,t,u,u,u) \]

\[ = \gamma_j(x,t,u,u) + \gamma_j(x,t,u,u) \]

The functions \( \beta, \gamma \) should be defined in this routine.

```
INTEGER NPDE, IRES
LOGICAL LEFT
REAL T
REAL BETA(NPDE), GAMDOT(NPDE), GAMMAG(NPDE),
     U(NPDE), DUDX(NPDE), UDOT(NPDE)

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 \( \gamma = u \), 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 \textsc{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 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 T=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.
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 \(10^{-5}\). 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

\[ u_t = \nabla \cdot (u^n \nabla u) \quad (n > 0). \]

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

\[ u_t = x^{-1} \frac{d}{dx} (x u^n u_x), \quad 0 < x < 1, \quad t_0 < t, \]

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

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

with
\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure.png}
\caption{Problem I (NPTS = 21). Grid + close-up near initial time and solution components at times $t = 0.001, 0.01, 0.1, 0.4, 4.0$ ($\Delta, +, \times, \square$).}
\end{figure}

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

The boundary conditions for the PDE are

\[
u(x, 0) = 0, \quad u(1, t) = 0, \quad t > t_0.
\]

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
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


Appendix A

Test functions \( \psi^{(m)}_i \):

\[
\psi^{(m)}_0(x) = \begin{cases} 
0 & \text{if singular} \\
\frac{x - x_i}{x_i - x} & \text{if regular} \\
0 & \text{elsewhere}
\end{cases} \quad x_L \leq x \leq X_1,
\]

\[
\psi^{(m)}_i(x) = \begin{cases} 
1 - \psi^{(m)}_i(x) & X_{i-1} \leq x \leq X_i \\
\int_{x}^{X_{i+1}} y^{-m}dy / \int_{x}^{X_{i+1}} y^{-m}dy & X_i \leq x \leq X_{i+1} \\
0 & \text{elsewhere}
\end{cases} \quad i = 1, \ldots, N,
\]

\[
\psi^{(m)}_{N+1}(x) = \begin{cases} 
1 - \psi^{(m)}_N(x) & X_N \leq x \leq x_R \\
0 & \text{elsewhere}
\end{cases}
\]

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 \( \xi^{m+1}_{\frac{a+b}{2}} \):

\[
\xi^{m+1}_{\frac{a+b}{2}} = \begin{cases} 
0 & \text{if singular and } i = 0 \\
\int_{x_i}^{X_{i+1}} ydy / \int_{x_i}^{X_{i+1}} y^{-m}dy = \frac{X_{i+1}^2 - x_i^2}{2} / \int_{x_i}^{x_{i+1}} y^{-m}dy & \text{otherwise}
\end{cases}
\]

Trial functions \( \phi^{(m)}_i \):

In the regular case,

\[
\phi^{(m)}_i(x) = \psi^{(m)}_i(x), \quad i = 0, \ldots, N+1
\]

and in the singular case

\[
\phi^{(m)}_0(x) = \begin{cases} 
\frac{X_1^2 - x^2}{X_i^2 - x_i^2} & X_L \leq x \leq X_1 \\
0 & \text{elsewhere}
\end{cases}
\]
\[ \phi^{(m)}(x) = \begin{cases} 
1 - \phi^{(m)}_1(x) & X_{i-1} \leq x \leq X_i \\
\frac{X_{i+1}^2 - x^2}{X_{i+1}^2 - X_i^2} & X_i \leq x \leq X_{i+1} \\
0 & \text{elsewhere} 
\end{cases} \]

\[ \phi^{(p)}_{N+1}(x) = \begin{cases} 
1 - \phi^{(p)}_N(x) & X_N \leq x \leq x_R \\
0 & \text{elsewhere} 
\end{cases} \]

*Quadrature points* \( \xi_{i+\frac{1}{2}} \):

In the regular case

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

and in the singular case

\[ \xi_{i+\frac{1}{2}} = \frac{2}{3} \frac{X_{i+1}^3 - X_i^3}{X_{i+1}^2 - X_i^2} . \]
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 \[ \sum_{k=1}^{NPDE} \sum_{j,k}^{m} \left( x \frac{\partial^m u}{\partial t^m} \right) \] (x, t, u, \( u_j \)) u + Q(x, \( \frac{\partial}{\partial x} \)) = x (x, t, u, \( u_j \))
C
C where \( u = (u_1, \ldots, u_{NPDE}) \), \( j = 1, \ldots, NPDE \),
C
C and \( u \) is the partial derivative wrt time of the \( k \)-th component of \( u \).
C
C The functions \( C, Q, \) and \( R \) are assumed to be continuous wrt the
C space variable.
C
C The independent variables \( x \) and \( t \) satisfy \( L \leq x \leq R \), \( t \geq 0 \). The fixed and \( t > t^* \).
C
C The boundary conditions have the form
C
C \[ BETA(x_t)R(x_t, u^\prime) = GAMMA(x_t, u^\prime) \] at \( x = L, R \)
C
C where not all of the functions \( BETA \) and \( GAMMA \) are set to zero.
C
C The initial conditions are given by
C
C \[ u(x, t^*) = u(x) \] for \( x^* \leq x \leq x^* \).
C
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
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
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
C Interface
C Blom J.G. and Zegeling P.A.
C A Moving-Grid Interface for Systems of One-Dimensional
How to use this module

1. Set NPDE = # PDEs to be solved.
   Set NPTS = # mesh points to be used.
   (NC=NPTS-2 is # internal points)
   Set M for space coordinate type
     = 0 for Cartesian, = 1 for cylindrical, = 2 for spherical.
   Specify a workspace of size at least (NPDE+1)*NPTS+(6+NPDE)*NPDE
   for use by the routine SKMRES which defines the ODE system being
   solved by the integrator.

   Call the initialization routine SETSKM, see the documentation at
   the head of this routine for the precise details of the call.

   Set TS and TOUT for start and end integration times.
   Set INFORM, WORK arrays as required for time integration,
     see SPRINT documentation.
   Call the SPRINT routine with the residual routine SKMRES.

2. Provide a set of routines which describe the precise form of the
   PDEs to be solved. Three routines must be provided and the names
   of these routines are fixed. These routines are:
   SPDEF forms the functions C, Q and R of the PDE in a
     given x-point.
   BNDR forms the functions BETA and GAMMA associated with the
     boundary conditions for the PDE. W.r.t. the function
     GAMMA there is a complication because SPRINT requires
     that the part of the residual that depends on the time-
     derivative must be supplied separately. Therefore the
     GAMMA function has to be split up into:
     GAMMA(x,t,u,u ,u ) =
     -x
     GAMDOT(x,t,u,u ,u ) + GAMMAG(x,t,u,u ).
     -x
     UINIT supplies the initial values of the PDE part.
     An initial uniform grid is generated by SKMRES and
     provided in Y(NPDE+1), I=1,NPTS. If required, a user
     can redefine the mesh in a nonuniform way.

   The headers of these routines are:

   SUBROUTINE SPDEF (T, X, NPDE, U, DUDX, C, Q, R, IRES)
   SUBROUTINE BNDR (T, BETAM, GAMDOT, GAMMAG, U, DUDX, UD, NPDE,
     LEFT, IRES)
   SUBROUTINE UINIT (NPDE, NPTS, Y)

   INTEGER NPDE, IRES
   REAL T, X
   REAL U(NPDE), DUDX(NPDE), C(NPDE,NPDE), Q(NPDE), R(NPDE)
   INTEGER NPDE, IRES
   LOGICAL LEFT
   REAL T
   REAL BETAM(NPDE), GAMDOT(NPDE), GAMMAG(NPDE),
     U(NPDE), DUDX(NPDE), UD(NPDE)
   INTEGER NPDE, NPTS
REAL Y(NPDE+1,NPTS)

Example problem

The easiest way to describe how the problem description routines should be written is by a simple example. Consider the following problem from electrodynamics

\[ u = \varepsilon_p \cdot u - g(u-v) \]
\[ t \quad \text{xx} \]
\begin{align*}
\text{and} & \quad v = p \cdot v + g(u-v) \\
\text{t} & \quad \text{xx}
\end{align*}

\( g(z) = \exp(\eta_z z/3) - \exp(-2.0 \cdot \eta_z z/3) \),

\( 0 \leq x \leq 1 \) and \( 0 \leq t \leq 4 \);

\( \varepsilon_p = 0.143, \quad p = 0.1743, \quad \) and \( \eta_z = 17.19 \).

The left boundary condition (LEFT = .TRUE.) is given by

\[ u = 0 \quad \text{and} \quad v = 0 \quad \text{at} \quad x = 0, \]

\begin{align*}
\text{x} & \quad \text{xx}
\end{align*}

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

\[ u = 0 \quad \text{and} \quad v = 0 \quad \text{at} \quad x = 1, \]

\begin{align*}
\text{x} & \quad \text{xx}
\end{align*}

and the initial conditions are

\[ u = 1 \quad \text{and} \quad v = 0 \quad \text{at} \quad t = 0. \]

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 \( Y(3,i) \).

\begin{verbatim}
SUBROUTINE UINIT (NPDE, NPTS, Y)

Routine for PDE initial values.

Entry:
\[ Y(1,1) = X_{-1}; \] uniform mesh, generated by package

Exit:
\[ Y(1,1) = X_{-1}; \] mesh, optionally changed by user

\begin{align*}
Y(k,1) & = u(k(X_{-1},t0); \] initial value of \( k \)-th component \\
& \quad i = 1, \ldots, NPTS
\end{align*}

INTEGER NPDE, NPTS
REAL Y(NPDE+1,NPTS)

INTEGER I

DO 10 I = 1, NPTS
\[ Y(1,I) = 1.0 \]
\[ Y(2,I) = 0.0 \]
10 CONTINUE

RETURN
END

SUBROUTINE SPDEF (I, X, NPDE, U, DUDX, C, Q, R, IRES)
\end{verbatim}
Routine to describe the body of the PDE system.
The PDE is written as
\[ u_t + Q(x,t,u,u) + R(x,t,u,u) = f(x,t,u,u) \]
The functions \( C, Q \) and \( R \) must be defined in this routine.

```
INTEGER NPDE, IRES
REAL T, X
REAL U(NPDE), DUDX(NPDE), C(NPDE,NPDE), Q(NPDE), R(NPDE)

INTEGER J, K
REAL EPS, ETA, GZ, P, Z
DATA EPS /D.143/, ETA /17.19/, P /D.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*X/3) - EXP(-Z*ETA*X/3)
Q(1) = GZ
Q(2) = -GZ
R(1) = EPS*P + DUDX(1)
R(2) = P * DUDX(2)
RETURN
END
```

Boundary conditions routine

The boundary conditions are written as
\[ \beta(x,t) R(x,t,u,u) = \Gamma(x,t,u,u) \]
\[ \Gamma(x,t,u,u) = \text{GAMDOT}(x,t,u,u) + \text{GAMMAG}(x,t,u,u) \]
The functions \( \beta, \text{GAMDOT} \) and \( \text{GAMMAG} \) should be defined in this routine.

```
INTEGER NPDE, IRES
LOGICAL LEFT
REAL T
REAL BETANPDE, GAMDOTNPDE, GAMMAGNPDE,
+ U(NPDE), DUDX(NPDE), UDOT(NPDE)

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
```
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       RETURN  
C       END  
C
C--------------------------------------------------------
C
C SUBROUTINE SETSKM (NEQN, NPDE, NPTS, X, XR, TAU, KAPPA, ALPHA,  
+                   Y, RWK, NRWK, M, TS, IBAND, IRES)  
C
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 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,
1: cylindrical polar coordinates, 
2: spherical polar coordinates.

TS Entry: the time at which the integration starts.
IBAND Exit: an upper bound on the half bandwidth of the Jacobian matrix when this module is used. This parameter should be supplied to MATSET if SPRINT is called with banded matrix routines.
IRES Exit: this parameter is set to -1 if an error is found by this routine.

Four parameters are passed across from here in COMMON /SPSKM/ NPDE1, NC, M, SING

NPDE1 = NPDE+1 the number of PDEs + 1 (for the grid equation).
NC = NPTS-2 the number of internal mesh points.
M = Muser = 0,1,2, if resp., Cartesian, cylindrical or spherical polar coordinates in use.
SING = .TRUE. if PDE has a polar singularity.

Another common block filled with method parameters is initialized in this routine

MONTP type of monitor: 1: (ALFA+!uxx!**2)**(1/2)
2: (ALFA+!uxx!**2)**(1/4)
(intialized in monitor routine)
TAUABS abs. part time-smoothing parameter grid equation (= TAU)
TAUREL rel. part; (old; TAU = TAUABS + TAUREL.DELTA T) (= 0.0)
RKAPPA spatial smoothing parameter grid equation (= KAPPA)
ALFA monitor constant (= ALPHA)

At the present stage of development we use only MONTP = 1 and TAUABS = TAU, TAUREL = 0.

Detailed description of workspace:

Size: NRWK must be >= (NPDE+1)*NPTS + (6+NPDE)*NPDE

RWK( 1.+(NPDE+1)*NPTS ) G(NPDE+1,0:NC+1), part of residual not dependent on time-derivative.
RWK(W1.+(NPDE) UKS1(NPDE), solution values at evaluat. point
RWK(W2.+(NPDE) UXKSI(NPDE), space derivs. at evaluat. point
RWK(W3.+(NPDE) BETA(NPDE), boundary function BETA
RWK(W4.+(NPDE) GAMMA(NPDE), part of boundary function GAMMA that is not dependent on the time-derivative.
RWK(W5.+(NPDE) RCR(NPDE), flux at evaluation point
RWK(W6.+(NPDE) QCN(NPDE), source term at evaluation point
RWK(W7.+(NPDE) CC(NPDE,NPDE), udot factor at evaluat. point


CHARACTER*6 PDCODE
COMMON /DISCHK/ PDCODE
C
INTEGER IOVFL0
REAL SUNFL0, SRELPR
COMMON /SCONS1/ SUNFL0, SRELPR, IOVFL0
C
INTEGER NPDE1, NC, MM
LOGICAL SING
COMMON /SPSKM/ NPDE1, NC, MM, SING
C
INTEGER MONTYP
REAL TAUABS, TAUREL, RKAPPA, ALFA
COMMON /MTPAR/ MONTYP, TAUABS, TAUREL, RKAPPA, ALFA
C
SAVE /DISCHK/, /SCONS1/, /SPSKM/, /MTPAR/
C
C-----------------------------------------------------------------------

C INTEGER IW1, IW2, IW3, IW4, IW5, IW6, IW7, IWE

NPDE1 = NPDE1+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 MONITCR'
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 0.000, 0.000)
   IRES = -1
ENDIF

IF (M .LT. 0) THEN
   CALL SERROR(' SETSKM- POLAR PARAMETER M (=I1) LESS THAN ZERO',
1 '1', 1, M, 0, 0, 0.000, 0.000)
   IRES = -1
ENDIF
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 /SPSKLM/ NPDE1, NC, M, SING
SAVE /SPSKLM/
C
INTEGER I
REAL DX, XI
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
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-------
C
INTEGER NEQN, IRES, NRWK
REAL T
REAL Y(NEQN), YDOT(NEQN), RES(NEQN), RWK(NRWK)
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)*I + J) I=0,...,NPDE-1,
C grid points: Y((NPDE+1)*(I+1)) I=0,...,NPDE-1.
C RES Exit: residual vector.
If IRES = -1 RES should contain only the part of the residual
dependent on the time-derivative, if IRES /= -1 RES should
contain the full residual A*ydot - g.
IRES Entry: see above.
Exit: 2, if setup routine SETSKM has not been called.
3, if one of the ODE solutions in the vector Y is not
acceptable.
RWK working storage of length NRWK.
NRWK Entry: dimension of RWK. Should be >= NEQN + (6+NPDE)*NPDE.

CHARACTER*6 PDCODE
COMMON /DISCHK/ PDCODE

INTEGER NPDE1, NC, M
LOGICAL SING
COMMON /SPSKM/ NPDE1, NC, M, SING

SAVE /DISCHK/, /SPSKM/

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",
   "SETSKM WAS NOT CALLED PRIOR TO SPRINT ENTRY",
   "IRES = 2"
   RETURN
ENDIF

NPDE = NPDE-1

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

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

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 CWRESK (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, NNAME
        RES(J) = RES(J) - RWK(J)
    10    CONTINUE
ENDIF
RETURN
END

SUBROUTINE CWRESU (T, Y, YDOT, NPDE, NC, M, SING,
                    + UKSI, UKKSI, BETA, GAMMA, RC, QC, CC,
                    + AYDOT, S, 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
C sum C (x,t,u) (u-u_x) + Q (x,u,u) = x (x R (x,u,u))
C k=1 j,k   -x    x   j  -x
C
C where 1  
C NPDE T
C u = \{ u, ..., u \}, j = 1, ..., NPDE,
C k
C
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 uxx,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 gives a difference equation for l=1,...,N:
C
C e R (ksi) - e R (ksi) = e S (ksi \u) + e S (ksi \u)
C l j l-1 j l-1 l j l-l l-1 j l-l -l
C
C with:
C e = \zeta \ksi \mu
C e = -1 if PDE singular otherwise \mu = m
C l
C l
\[ f_{SI} = \frac{\text{zeta} - X}{(m+1)} \]
\[ f_{SL} = \frac{X - \text{zeta}}{(m+1)} \]

\begin{align*}
C \quad & \text{S (ksi, U) = sum C (ksi) [ U (X) ] + Q (ksi)} \\
C \quad & \text{j k l m p=1 j p k l x l l j k} \\
C \quad & \text{R, j\_th component of flux evaluated at quadrature point ksi,} \\
C \quad & \text{Q, j\_th component of source term evaluated at quadrature point ksi,} \\
C \quad & \text{C, element j\_p of matrix multiplying u, evaluated at ksi.} \\
C \quad & \text{t} \\
C \quad & \text{Left boundary equation, if non-singular:} \\
C \quad & \text{BETA (x, t) R (x) = GAMMA (x, t), U, U (x), U} \\
C \quad & \text{j L j L j L \_0 \_0 \_0 \_0} \\
C \quad & \text{with} \\
C \quad & \text{R (x) = (ksi R (ksi) - f_{SI} S (ksi, U)) / x} \\
C \quad & \text{j L 0 j 0 0 j 0 \_0 \_0} \\
C \quad & \text{if singular:} \\
C \quad & \text{S (ksi, U) / (m+1) - R (ksi) / ksi = 0} \\
C \quad & \text{j 0 \_0 j 0 0} \\
C \quad & \text{Right boundary equation:} \\
C \quad & \text{BETA (x, t) R (x) = GAMMA (x, t), U, U (x), U} \\
C \quad & \text{j R j R j R \_0 \_R \_R} \\
C \quad & \text{with} \\
C \quad & \text{R (x) = (f_{SL} R (ksi) + f_{SI} S (ksi, U)) / x} \\
C \quad & \text{j R N j N N j N \_N+1 \_R} \\
C \quad & \text{Parameters:} \\
C \quad & \text{---------------} \\
C \quad & \text{INTEGER NPDE, NC, M, IRES} \\
C \quad & \text{LOGICAL SING} \\
C \quad & \text{REAL T} \\
C \quad & \text{REAL Y(NPDE+1,0:NC+1), YDOT(NPDE+1,0:NC+1),} \\
C \quad & \text{UKS(NPDE), UKSXI(NPDE), BETA(NPDE), GAMMA(NPDE),} \\
C \quad & \text{RC(NPDE), QC(NPDE), CC(NPDE,NPDE),} \\
C \quad & \text{AYDOT(NPDE+1,0:NC+1), G(NPDE+1,0:NC+1)} \\
C \quad & \text{YDOT Entry: evaluation time} \\
C \quad & \text{Y Entry: solution and grid at time T.} \\
C \quad & \text{Y(1:NPE,L): U_L, Y(NPDE+1,L): X_L} \\
C \quad & \text{YDOT Entry: derivative of Y at T.} \\
C \quad & \text{NPDE Entry: # PDEs.} \\
C \quad & \text{NC Entry: # internal grid points.} \]
ENTRY: coordinate system indicator.

Entry: true, if PDE has a polar singularity (M>0, X_L=0).

workspace to hold the solution value at an evaluation point.

workspace to hold the space derivatives at an evaluation point.

workspace to hold the boundary function BETA.

workspace to hold that part of the boundary function GAMMA that
is not dependent of YDOT.

workspace to hold the PDE function R.

workspace to hold the PDE function Q.

workspace to hold the PDE function C.

Exit: A_ydot part of the residual.

Exit: g part of the residual.

Exit: 3 if some user function indicated that a solution value
is unphysical.

Local variables:

----------

INTEGER MU

REAL FSCL, FSCI, SCLMQ, SCL1MQ,
+ DENOM1, KSI, KSI MU, IETA, ZETAMU, PHIL, PHILX

MU if sing. then -1 else m.

FSCL f$1\_L$.

FSCI f$SL\_L+1$.

SCLMQ $S(ksi\_L, y-dot\_L) - Q(ksi\_L)$.

SCL1MQ $S(ksi\_L, y-dot\_L+1) - Q(ksi\_L)$.

DENOM1 1 / \left( x_{L}x_{L+1} \right) int y^{m}(\text{d}y)

KSI quadrature point for l_th interval

KSI MU ksi**mu

ZETA zeta\_L**(m+1); if sing. and l=0 then

zeta = 0.0

else \left( x_{L}x_{L+1} \right) int y dy * denomi

ZETAMU zeta\_L**(m-mu)

PHIL \textit{left} trial function for l_th interval in eval. point ksi

NB. phi\_L\_L+1 = 1 - phil

PHILX deriv. of phi\_L\_L wrt x in eval. point ksi

NB. phi\_L\_L+1\_x = - phi\_L\_x

----------

INTEGER IP, J, L, NPDE1

REAL DENPHI, RCJDOT, RCJG, UXL, UXL\_L, XO, XOM, XL, XL1, XN1M

NPDE1 = NPDE+1

XO = Y(NPDE1,0)

IF (SING) THEN

MU = -1

ELSE

MU = M

ENDIF

First interval, compute contribution to residual eq. in X_1 and

Left boundary equation

L = 0

XL = XD

X_L1 = Y(NPDE1,L+1)

IF (.NOT. SING) THEN

IF (M_EQ. 1) THEN

DENOM1 = 1/LOG(XL\_L/XL)

ELSE

ENDIF

END
DENOMI = (1-M)/(XL1**2(1-M) - XL**2(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(2-M)-XL**2(2-M)) / (2-M) * DENOMI
  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**2(1-M) - KSI**2(1-M)) / (1-M) * DENOMI
    PHILX = -KSI**2(-M) * DENOMI
  ENDIF
ENDIF

IF (M .EQ. 0) THEN
  XOM = 1.0
ELSE
  XOM = X0**M
ENDIF

C
Get left boundary function values
DO 5 J = 1, NPDE
  UXKSI(J) = (Y(J,1)-Y(J,0)) / (XL1-XL)
5 CONTINUE
C
CALL BNDL (T, BETA, AYDOT(1,0), GAMMAG, Y(1,0), UXKSI,
  + YDOT(1,0), NPDE, .TRUE., IRES)
IF (IRES .EQ. 3) RETURN

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 (i, 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)
   FSC0 = (XL**(M+1)-ZETA)/(M+1)
   DO 20 J = 1, NPDE
      SCLMJ = 0.0
      SCLMQ = 0.0
      DO 30 IF = 1, NPDE
         UXLI = (Y(IP,L+2)-Y(IP,L ))/(Y(NPDE1,L+2)-Y(NPDE1,L ))
         SCLMJ = SCLMJ + CC(J,IP)*YDOT(IP,L )
         SCLMQ = SCLMQ + CC(J,IP)*YDOT(IP,L+1)-UXLI*YDOT(NPDE1,L+1)
      CONTINUE
      30
   CONTINUE

C Store contribution from i_th interval to residual equation in X_L+1
   AYDOT(J,L+1) = FSC0 * SCLMQ
   G (J,L+1) = -ZETAMU*KSI0*RC(J) - FSC0*QC(J)

C Compute boundary equations
   IF (SING) THEN
      Bnd.eq. is contribution from O-th interval to
      difference eq. in x_L
      AYDOT(J,O) = SCLMJ / (M+1)
      G (J,O) = RC(J)/KSI - QC(J)/(M+1)
   ELSE
      Otherwise compute flux in x_L from contribution from O-th
      interval to difference equation in x_L, and substitute in
      user's boundary equation.
      NB: AYDOT(J,O) = GAMDOT(J).
      RCJDOT = -FSCL * SCLMQ / XOM
      RCJG = (KSI0*RC(J) - FSCL*QC(J)) / XOM
      AYDOT(J,O) = -BETA(J)*RCJDOT + AYDOT(J,O)
      G (J,O) = BETA(J)*RCJG - GAMMA(J)
   ENDIF

20
   CONTINUE

DO 100 L = 1, NC-1

C Evaluate PDE functions in quadrature point in i_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 i_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)
      KSI0 = 1/KSI
   ELSE IF (M .eq. 1) THEN
      KSI = (XL1-XL) * DENOMI
      KSI0 = KSI
   ELSE IF (M .eq. 2) THEN
      KSI = LOG(XL1/XL) * DENOMI
      KSI0 = KSI
   ELSE
KSI = (XL1**((2-M)-(L-2-M))) / (2-M) * DENOMI
IF (M.EQ. 0) THEN
  KSI*MU = 1.0
ELSE
  KSI*MU = KSI**M
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-PHI)
  UKSI(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)
FSCL1 = (XL**((M+1)-ZETA))/(M+1)
DO 120 J = 1, NPDE
  SCLMJQ = 0.0
  SCLMJQ = 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))
    SCLMJQ = SCLMJQ +
    CC(J,IP)*(YDOT(IP,L)-UXL*YDOT(NPDE1,L, L))
    SCLMJQ = SCLMJQ +
    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(L,J,L) = AYDOT(J,L) + FSCL*SCLMJQ
C Store contribution from L-th interval to residual equation in X_L+1
  AYDOT(L,J,L+1) = FSCL1 * SCLMJQ
C Add contribution over N-th interval to residual equation in X_N.
L = NC
C
C Compute right boundary equation.

C

XL = XL1
XL1 = Y(NPDE+1,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
  DENS = 1/(XL1*XL1-XL*XL)
  PHIL = (XL1*XL1-KS1*KSI)*DENS
  PHILX = -2*KSI*DENS
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
  XNIM = 1.0
ELSE
  XNIM = 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)
CONTINUE
CALL BNR( T, BETA, AYDOT(1,NC+1), GAMMA, Y(1,NC+1), UIKSI, +
        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)
UXSI(J) = Y(J,L)*PHILX + Y(J,L-1)*(-PHILX)

210 CONTINUE
C Ge: C, Q and R in evaluation point
CALL SPDEF (T, KSI, NPDE, UKSI, UXSI, CC, QC, RC, IRES)
IF (IRES .EQ. 3) RETURN

FSCL = (ZETA- XL***(M+1))/(M+1)
FSCI = (IL1***(M+1)-ZETA)/(M+1)
DO 220 J = 1, NPDE
   SCLMQ = 0.0
   SCLMQ = 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 ))
   SCLMQ = SCLMQ +
   CC(J,IP)*(YDOT(IP,L1))
230 CONTINUE
220 CONTINUE
C Add contribution over N_th interval to residual equation in X_N
   AYDOT(J,L1) = AYDOT(J,L1) + FSCL * SCLMQ
   G (J,L1) = G (J,L1) + ZETAMU*KSINU*RC(J) - FSCL*QC(J)
C Compute flux in x R and substitute in user's boundary condition
C NB: AYDOT(J,NC1) = EAMDOT(J).
   RCJDOT = FSCI * SCLMQ / XN1M
   RCJG = (ZETAMU*KSINU*RC(J) + FSCI*QC(J)) / XN1N
   AYDOT(J,NC1) = -BETA(J)*RCJDOT + AYDOT(J,NC1+1)
   G (J,NC1) = BETA(J)*RCJG - GAMMA(J)

RETURN
END

SUBROUTINE (WRESX (T, DELTAT, Y, YDOT, NPDE1, NC, AYDOT, G, IRES)
C
C Purpose:
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
C     i-1  i-1
C     nt + tau.nt
C     i    i  1 <= i <= N,
C
C     M
C     i-1  i
C
C with nt = n - fac.(n - 2.m + n ); fac = rkappa.(rkappa+1) and
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 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 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(:,NPDE1)) : U, L, (YNPDE1,1) : X
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

 INTEGER MONTYP
 REAL TAUREL, RKAPPA, ALFA
 COMMON /METPAR/ MONTYP, TAUABS, TAUREL, RKAPPA, ALFA
 SAVE /METPAR/

C

 INTEGER I, NPNE
 REAL AO, AM, FAC, GO, GM, NIM1, NI, NIP1, NTI, NTDI, TAU

C NPDE = NPDE1-1

C Define smoothing factors
TAU = TAUABS / TAUREL * DELTAT
FAC = RKAPPA / (RKAPPA + 1)

C Compute monitor values; store M(I) temp. in G(NPDE1,I), I=0, NC
CALL XMNTR (Y, G, NPDE, NC)

C Compute A.xdot: and g for grid equations.
C Interior equations:
A.xdot (I) = TAU/MI-1) * NTDOT(I-1) - TAU/WI) * NTDOT(I)
C g(I) = NT(I-1)/MI - NT(I-1)/WI-1
C NT(I) = NI - FAC*(N(I-1)-2.*N(I)+N(I+1))
C N (I) = 1 / (I(I+1)-X(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)) +
 + AYDOT(NPDE1,I+1)*YDOT(NPDE1,I+1))
AO = TAU / G(NPDE1,I) * NTI
GO = NTI / G(NPDE1,I)

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

10 CONTINUE
AM = AO
GM = GO

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

NTI = N - FAC*(NIM1-2*NI+NIP1)
NTDI = FAC*NIM1*NIM1*(YDOT(NPDE1,I )-YDOT(NPDE1,I-1)) -
+(1+2+FAC)*NI *N1 *(YDOT(NPDE1,I+1)-YDOT(NPDE1,I )) +
+FAC*NIP1*NIP1*(YDOT(NPDE1,I+2)-YDOT(NPDE1,I+1))

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

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

10 CONTINUE
I = NC
NIM1 = N;
NI = N+1
AM = AO
GM = GO

NIP1 = N:

NTI = N - FAC*(NIM1-2*NI+NIP1)
NTDI = FAC*NIM1*NIM1*(YDOT(NPDE1,I )-YDOT(NPDE1,I-1)) -
+(1+2+FAC)*NI *N1 *(YDOT(NPDE1,I+1)-YDOT(NPDE1,I ))

AO = TAU / G(NPDE1,I) * NTI
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
C END
SUBROUTINE XMTR (Y, G, NPDE, N)
INTEGER NPDE, N
REAL Y(NPDE+1,0:N+1), G(NPDE+1,0:N+1)

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 C(NPDE+1,i) = M(i)
C
C...........................................................................
C INTEGER MONTYP
REAL TAUABS, TAUREL, RKAPPA, ALFA
COMMON /MTPAR/ MONTYP, TAUABS, TAUREL, RKAPPA, ALFA
SAVE /MTPAR/

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