1 Purpose

nag_pde_parab_1d_fd_ode_remesh (d03ppc) integrates a system of linear or nonlinear parabolic partial differential equations (PDEs) in one space variable, with scope for coupled ordinary differential equations (ODEs), and automatic adaptive spatial remeshing. The spatial discretization is performed using finite differences, and the method of lines is employed to reduce the PDEs to a system of ODEs. The resulting system is solved using a Backward Differentiation Formula (BDF) method or a Theta method (switching between Newton’s method and functional iteration).

2 Specification

```c
#include <nag.h>
#include <nagd03.h>
void nag_pde_parab_1d_fd_ode_remesh (Integer npde, Integer m, double *ts,
  double tout,
  void (*pdedef)(Integer npde, double t, double x, const double u[],
  const double ux[], Integer ncode, const double v[],
  const double vdot[], double p[], double q[], double r[],
  Integer *ires, Nag_Comm *comm),
  void (*bndary)(Integer npde, double t, const double u[],
  const double ux[], Integer ncode, const double v[],
  const double vdot[], Integer ibnd, double beta[], double gamma[],
  Integer *ires, Nag_Comm *comm),
  void (*uvinit)(Integer npde, Integer npts, Integer nxi,
  const double x[], const double xi[], double u[], Integer ncode,
  double v[], Nag_Comm *comm),
  double u[], Integer npts, double x[], Integer ncode,
  void (*odedef)(Integer npde, double t, Integer ncode, const double v[],
  const double vdot[], Integer nxi, const double xi[],
  const double ucp[],
  const double ucpx[], const double rcp[],
  const double ucp[,], const double ucpx[,],
  double f[],
  Integer *ires, Nag_Comm *comm),
  Integer nxi, const double xi[], Integer neqn, const double rtol[],
  const double atol[], Integer itol, Nag_NormType norm,
  Nag_LinAlgOption laopt, const double algopt[],
  Nag_Boolean remesh,
  Integer nxfix, const double xfix[,], Integer nrmesh, double dxmesh,
  double trmesh, Integer ipminf, double xratio, double con,
  void (*monitf)(double t, Integer npts, Integer npde, const double x[],
  const double u[],
  const double r[],
  double fمونت[],
  Integer *ires, Nag_Comm *comm),
  void (*monitf)(double t, Integer npts, Integer npde, const double x[],
  const double u[],
  const double r[],
  double fمونت[],
  Integer *ires, Nag_Comm *comm),
  double rsave[], Integer lrsave, Integer isave[], Integer lisave,
  Integer itask, Integer itrace, const char *outfile, Integer *ind,
  Nag_Comm *comm, Nag_D03_Save *saved, NagError *fail)
```

3 Description

nag_pde_parab_1d_fd_ode_remesh (d03ppc) integrates the system of parabolic-elliptic equations and coupled ODEs

\[
\sum_{j=1}^{npde} P_{ij} \frac{\partial U_i}{\partial t} + Q_i = x^{-m} \frac{\partial}{\partial x} (x^m R_i), \quad i = 1, 2, \ldots, \text{npde}, \quad a \leq x \leq b, t \geq t_0, \quad (1)
\]

Mark 25
\[ F_i(t, V, \dot{V}, \xi, U^*, U_x^*, R^*, U_t^*, U_{x \tau}^*) = 0, \quad i = 1, 2, \ldots, \text{ncode}, \quad (2) \]

where (1) defines the PDE part and (2) generalizes the coupled ODE part of the problem.

In (1), \( P_{i,j} \) and \( R_i \) depend on \( x, t, U, U_x \), and \( V \); \( Q_i \) depends on \( x, t, U, U_x, V \) and \textbf{linearly} on \( \dot{V} \). The vector \( U \) is the set of PDE solution values

\[ U(x,t) = [U_1(x,t), \ldots, U_{\text{npde}}(x,t)]^T, \]

and the vector \( U_x \) is the partial derivative with respect to \( x \). The vector \( V \) is the set of ODE solution values

\[ V(t) = [V_1(t), \ldots, V_{\text{ncode}}(t)]^T, \]

and \( \dot{V} \) denotes its derivative with respect to time.

In (2), \( \xi \) represents a vector of \( n_\xi \) spatial coupling points at which the ODEs are coupled to the PDEs. These points may or may not be equal to some of the PDE spatial mesh points. \( U^*, U_x^*, R^*, U_t^* \) and \( U_{x \tau}^* \) are the functions \( U, U_x, R, U_t \) and \( U_{x \tau} \) evaluated at these coupling points. Each \( F_i \) may only depend linearly on time derivatives. Hence the equation (2) may be written more precisely as

\[ F = G - A \dot{V} - B \begin{pmatrix} U_t^* \\ U_{x \tau}^* \end{pmatrix}, \]

where \( F = [F_1, \ldots, F_{\text{ncode}}]^T, G \) is a vector of length \( \text{ncode} \), \( A \) is an \( \text{ncode} \) by \( \text{ncode} \) matrix, \( B \) is an \( \text{ncode} \) by \( (n_\xi \times \text{npde}) \) matrix and the entries in \( G, A \) and \( B \) may depend on \( t, \xi, U^*, U_x^* \) and \( V \). In practice you only need to supply a vector of information to define the ODEs and not the matrices \( A \) and \( B \). (See Section 5 for the specification of \texttt{odedef}.)

The integration in time is from \( t_0 \) to \( t_{\text{out}} \), over the space interval \( a \leq x \leq b \), where \( a = x_1 \) and \( b = x_{\text{npts}} \) are the leftmost and rightmost points of a mesh \( x_1, x_2, \ldots, x_{\text{npts}} \) defined initially by you and (possibly) adapted automatically during the integration according to user-specified criteria. The coordinate system in space is defined by the following values of \( m; m = 0 \) for Cartesian coordinates, \( m = 1 \) for cylindrical polar coordinates and \( m = 2 \) for spherical polar coordinates.

The PDE system which is defined by the functions \( P_{i,j}, Q_i \) and \( R_i \) must be specified in \texttt{pdefedef}.

The initial \( (t = t_0) \) values of the functions \( U(x,t) \) and \( V(t) \) must be specified in \texttt{uvinit}. Note that \texttt{uvinit} will be called again following any initial remeshing, and so \( U(x,t_0) \) should be specified for all values of \( x \) in the interval \( a \leq x \leq b \), and not just the initial mesh points.

The functions \( R_i \) which may be thought of as fluxes, are also used in the definition of the boundary conditions. The boundary conditions must have the form

\[ \beta_i(x,t)R_i(x,t;U,U_x,V) = \gamma_i(x,t;U,U_x,V,\dot{V}), \quad i = 1, 2, \ldots, \text{npde}, \]

where \( x = a \) or \( x = b \).

The boundary conditions must be specified in \texttt{bndary}. The function \( \gamma_i \) may depend \textbf{linearly} on \( \dot{V} \).

The problem is subject to the following restrictions:

(i) In (1), \( \dot{V}_j(t) \), for \( j = 1, 2, \ldots, \text{ncode} \), may only appear \textbf{linearly} in the functions \( Q_i \), for \( i = 1, 2, \ldots, \text{npde} \), with a similar restriction for \( \gamma \);

(ii) \( P_{i,j} \) and the flux \( R_i \) must not depend on any time derivatives;

(iii) \( t_0 < t_{\text{out}} \), so that integration is in the forward direction;

(iv) The evaluation of the terms \( P_{i,j}, Q_i \) and \( R_i \) is done approximately at the mid-points of the mesh \( x[i-1] \), for \( i = 1, 2, \ldots, \text{npts} \), by calling the \texttt{pdefedef} for each mid-point in turn. Any discontinuities in these functions \textbf{must} therefore be at one or more of the fixed mesh points specified by \texttt{xfix};

(v) At least one of the functions \( P_{i,j} \) must be nonzero so that there is a time derivative present in the PDE problem;
(vi) If \( m > 0 \) and \( x_1 = 0.0 \), which is the left boundary point, then it must be ensured that the PDE solution is bounded at this point. This can be done by either specifying the solution at \( x = 0.0 \) or by specifying a zero flux there, that is \( \beta_i = 1.0 \) and \( \gamma_i = 0.0 \). See also Section 9.

The algebraic-differential equation system which is defined by the functions \( F_i \) must be specified in \texttt{odeodef}. You must also specify the coupling points \( \xi \) in the array \texttt{xi}.

The parabolic equations are approximated by a system of ODEs in time for the values of \( U_i \) at mesh points. For simple problems in Cartesian coordinates, this system is obtained by replacing the space derivatives by the usual central, three-point finite difference formula. However, for polar and spherical problems, or problems with nonlinear coefficients, the space derivatives are replaced by a modified three-point formula which maintains second order accuracy. In total there are \( \texttt{npde} \times \texttt{npts} + \texttt{ncode} \) ODEs in time direction. This system is then integrated forwards in time using a Backward Differentiation Formula (BDF) or a Theta method.

The adaptive space remeshing can be used to generate meshes that automatically follow the changing time-dependent nature of the solution, generally resulting in a more efficient and accurate solution using fewer mesh points than may be necessary with a fixed uniform or non-uniform mesh. Problems with travelling wavefronts or variable-width boundary layers for example will benefit from using a moving adaptive mesh. The discrete time-step method used here (developed by Furzeland (1984)) automatically creates a new mesh based on the current solution profile at certain time-steps, and the solution is then interpolated onto the new mesh and the integration continues.

The method requires you to supply a \texttt{monitf} which specifies in an analytical or numerical form the particular aspect of the solution behaviour you wish to track. This so-called monitor function is used to choose a mesh which equally distributes the integral of the monitor function over the domain. A typical choice of monitor function is the second space derivative of the solution value at each point (or some combination of the second space derivatives if there is more than one solution component), which results in refinement in regions where the solution gradient is changing most rapidly.

You must specify the frequency of mesh updates together with certain other criteria such as adjacent mesh ratios. Remeshing can be expensive and you are encouraged to experiment with the different options in order to achieve an efficient solution which adequately tracks the desired features of the solution.

Note that unless the monitor function for the initial solution values is zero at all user-specified initial mesh points, a new initial mesh is calculated and adopted according to the user-specified remeshing criteria. \texttt{uvinit} will then be called again to determine the initial solution values at the new mesh points (there is no interpolation at this stage) and the integration proceeds.

4 References


Furzeland R M (1984) The construction of adaptive space meshes TNER.85.022 Thornton Research Centre, Chester


5 Arguments

1: \texttt{npde} – Integer

\textit{Input}

\textit{On entry:} the number of PDEs to be solved.

\textit{Constraint:} \( \texttt{npde} \geq 1 \).
2: \( m \) – Integer

*Input*

*On entry:* the coordinate system used:

\( m = 0 \)
Indicates Cartesian coordinates.

\( m = 1 \)
Indicates cylindrical polar coordinates.

\( m = 2 \)
Indicates spherical polar coordinates.

*Constraint:* \( m = 0, 1 \) or 2.

3: \( ts \) – double *

*Input/Output*

*On entry:* the initial value of the independent variable \( t \).

*On exit:* the value of \( t \) corresponding to the solution values in \( u \). Normally \( ts = \text{tout} \).

*Constraint:* \( ts < \text{tout} \).

4: \( \text{tout} \) – double

*Input*

*On entry:* the final value of \( t \) to which the integration is to be carried out.

5: \( \text{pdef} \) – function, supplied by the user

*External Function*

\( \text{pdef} \) must evaluate the functions \( P_{i,j}, Q_i \) and \( R_i \) which define the system of PDEs. The functions may depend on \( x, t, U, U_x \) and \( V \). \( Q_i \) may depend linearly on \( V \). \( \text{pdef} \) is called approximately midway between each pair of mesh points in turn by \( \text{nag_pde_parab_1d_fd_ode_remesh (d03ppc)} \).

The specification of \( \text{pdef} \) is:

```c
void pdedef (Integer npde, double t, double x, const double u[],
            const double ux[], Integer ncode, const double v[],
            const double vdot[], double p[], double q[], double r[],
            Integer *ires, Nag_Comm *comm)
```

1: \( \text{npde} \) – Integer

*Input*

*On entry:* the number of PDEs in the system.

2: \( \text{t} \) – double

*Input*

*On entry:* the current value of the independent variable \( t \).

3: \( \text{x} \) – double

*Input*

*On entry:* the current value of the space variable \( x \).

4: \( \text{u[npde]} \) – const double

*Input*

*On entry:* \( u[i-1] \) contains the value of the component \( U_i(x,t) \), for \( i = 1, 2, \ldots, \text{npde} \).

5: \( \text{ux[npde]} \) – const double

*Input*

*On entry:* \( \text{ux[i-1]} \) contains the value of the component \( \frac{\partial U_i(x,t)}{\partial x} \), for \( i = 1, 2, \ldots, \text{npde} \).

6: \( \text{ncode} \) – Integer

*Input*

*On entry:* the number of coupled ODEs in the system.
On entry: if \texttt{ncode} > 0, \( v[i-1] \) contains the value of the component \( V_i(t) \), for \( i = 1, 2, \ldots, \text{ncode} \).

On entry: if \texttt{ncode} > 0, \( vdot[i-1] \) contains the value of component \( \dot{V}_i(t) \), for \( i = 1, 2, \ldots, \text{ncode} \).

\textbf{Note:} \( \dot{V}_i(t) \), for \( i = 1, 2, \ldots, \text{ncode} \), may only appear linearly in \( Q_j \), for \( j = 1, 2, \ldots, \text{npde} \).

On exit: \( p[i] \) must be set to the value of \( P_i(x, t, U, U_x, V) \), for \( i = 1, 2, \ldots, \text{npde} \) and \( j = 1, 2, \ldots, \text{npde} \).

On exit: \( q[i-1] \) must be set to the value of \( Q_i(x, t, U, U_x, V, \dot{V}) \), for \( i = 1, 2, \ldots, \text{npde} \).

On exit: \( r[i-1] \) must be set to the value of \( R_i(x, t, U, U_x, V) \), for \( i = 1, 2, \ldots, \text{npde} \).

\textbf{ires} – \texttt{Integer *} \hspace{1cm} \textbf{Input/Output}

On entry: set to \(-1\) or \(1\). On exit: should usually remain unchanged. However, you may set \textbf{ires} to force the integration function to take certain actions as described below:

\begin{itemize}
  \item \textbf{ires} = 2
    Indicates to the integrator that control should be passed back immediately to the calling function with the error indicator set to \texttt{fail.code} = \texttt{NE_USER_STOP}.
  \item \textbf{ires} = 3
    Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set \textbf{ires} = 3 when a physically meaningless input or output value has been generated. If you consecutively set \textbf{ires} = 3, then \texttt{nag_pde_parab_1d_fd_ode_remesh (d03ppc)} returns to the calling function with the error indicator set to \texttt{fail.code} = \texttt{NE_FAILED_DERIV}.
\end{itemize}

\textbf{comm} – \texttt{Nag_Comm *}  
Pointer to structure of type \texttt{Nag_Comm}; the following members are relevant to \texttt{pdedef}.

\textbf{user} – \texttt{double *}  
\textbf{iuser} – \texttt{Integer *}

The type \texttt{Pointer} will be \texttt{void *}. Before calling \texttt{nag_pde_parab_1d_fd_ode_remesh (d03ppc)} you may allocate memory and initialize these pointers with various quantities for use by \texttt{pdedef} when called from \texttt{nag_pde_parab_1d_fd_ode_remesh (d03ppc)} (see Section 3.2.1.1 in the Essential Introduction).

\textbf{bdary} – function, supplied by the user \hspace{1cm} \textbf{External Function}

\textbf{bdary} must evaluate the functions \( \beta_i \) and \( \gamma_i \) which describe the boundary conditions, as given in (4).
The specification of `bndary` is:

```c
void bndary (Integer npde, double t, const double u[],
            const double ux[], Integer ncode, const double v[],
            const double vdot[], Integer ibnd, double beta[],
            double gamma[], Integer *ires, Nag_Comm *comm)
```

1: `npde` – Integer  
   *Input*
   
   *On entry:* the number of PDEs in the system.

2: `t` – double  
   *Input*
   
   *On entry:* the current value of the independent variable `t`.

3: `u[npde]` – const double  
   *Input*
   
   *On entry:* if `ibnd = 0` contains the value of the component `U, i(x, t)` at the boundary specified by `ibnd`, for `i = 1, 2, ..., npde`.

4: `ux[npde]` – const double  
   *Input*
   
   *On entry:* if `ibnd = 0` contains the value of the component `U, i(x, t)` at the boundary specified by `ibnd`, for `i = 1, 2, ..., npde`.

5: `ncode` – Integer  
   *Input*
   
   *On entry:* the number of coupled ODEs in the system.

6: `v[ncode]` – const double  
   *Input*
   
   *On entry:* if `ncode > 0`, contains the value of the component `V, i(t)`, for `i = 1, 2, ..., ncode`.

7: `vdot[ncode]` – const double  
   *Input*
   
   *On entry:* contains the value of component `V, i(t)`, for `i = 1, 2, ..., ncode`.

8: `ibnd` – Integer  
   *Input*
   
   *On entry:* specifies which boundary conditions are to be evaluated.

   - if `ibnd = 0`, `bndary` must set up the coefficients of the left-hand boundary, `x = a`.
   - if `ibnd ≠ 0`, `bndary` must set up the coefficients of the right-hand boundary, `x = b`.

9: `beta[npde]` – double  
   *Output*
   
   *On exit:* must be set to the value of `β, i(x, t)` at the boundary specified by `ibnd`, for `i = 1, 2, ..., npde`.

10: `gamma[npde]` – double  
    *Output*
    
    *On exit:* must be set to the value of `γ, i(x, t, U, Ux, V, V, t)` at the boundary specified by `ibnd`, for `i = 1, 2, ..., npde`.

11: `ires` – Integer  
    *Input/Output*
    
    *On entry:* set to −1 or 1.
On exit: should usually remain unchanged. However, you may set ires to force the integration function to take certain actions as described below:

ires = 2
Indicates to the integrator that control should be passed back immediately to the calling function with the error indicator set to fail.code = NE_USER_STOP.

ires = 3
Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set ires = 3 when a physically meaningless input or output value has been generated. If you consecutively set ires = 3, then nag_pde_parab_1d_fd_ode_remesh (d03ppc) returns to the calling function with the error indicator set to fail.code = NE_FAILED_DERIV.

7: uvinit – function, supplied by the user

uvinit must supply the initial \((t = t_0)\) values of \(U(x, t)\) and \(V(t)\) for all values of \(x\) in the interval \(a \leq x \leq b\).

The specification of uvinit is:

```c
void uvinit (Integer npde, Integer npts, Integer nxi,
            const double x[], const double xi[], double u[], Integer ncode,
            double v[], Nag_Comm *comm)
```

1: npde – Integer

On entry: the number of PDEs in the system.

2: npts – Integer

On entry: the number of mesh points in the interval \([a, b]\).

3: nxi – Integer

On entry: the number of ODE/PDE coupling points.

4: x[npts] – const double

On entry: the current mesh. \(x[i-1]\) contains the value of \(x_i\), for \(i = 1, 2, \ldots, npts\).

5: xi[nxi] – const double

On entry: if \(nxi > 0\), \(xi[i-1]\) contains the value of the ODE/PDE coupling point, \(\xi_i\), for \(i = 1, 2, \ldots, nxi\).

6: u[npde x npts] – double

On exit: if \(nxi > 0\), \(u[npde x (j-1) + i-1]\) contains the value of the component \(U_i(x_j, t_0)\), for \(i = 1, 2, \ldots, npde\) and \(j = 1, 2, \ldots, npts\).
7:  ncode – Integer
   On entry: the number of coupled ODEs in the system.

8:  v[ncode] – double
   On exit: \( v[i-1] \) contains the value of component \( V_i(t_0) \), for \( i = 1, 2, \ldots, \text{ncode} \).

9:  comm – Nag_Comm *
   Pointer to structure of type Nag_Comm; the following members are relevant to \textit{uvinit}.

   user – double *
   iuser – Integer *
   p – Pointer
   The type Pointer will be void *. Before calling
   nag_pde_parab_1d_fd_ode_remesh (d03ppc) you may allocate memory and
   initialize these pointers with various quantities for use by \textit{uvinit} when called from
   nag_pde_parab_1d_fd_ode_remesh (d03ppc) (see Section 3.2.1.1 in the Essential
   Introduction).

8:  u[neqn] – double
    On entry: if \textit{ind} = 1, the value of \( u \) must be unchanged from the previous call.

    On exit: \( u[\text{npde} \times (j-1) + i-1] \) contains the computed solution \( U_i(x_j, t) \), for \( i = 1, 2, \ldots, \text{npde} \)
    and \( j = 1, 2, \ldots, \text{npts} \), and \( u[\text{npts} \times \text{npde} + k-1] \) contains \( V_k(t) \), for \( k = 1, 2, \ldots, \text{ncode} \),
evaluated at \( t = \text{ts} \).

9:  npts – Integer
    On entry: the number of mesh points in the interval \([a, b]\).

    Constraint: \( npts \geq 3 \).

10: x[npts] – double
    On entry: the initial mesh points in the space direction. \( x[0] \) must specify the left-hand boundary,
    \( a \), and \( x[\text{npts} - 1] \) must specify the right-hand boundary, \( b \).

    Constraint: \( x[0] < x[1] < \cdots < x[\text{npts} - 1] \).

    On exit: the final values of the mesh points.

11: ncode – Integer
    On entry: the number of coupled ODE in the system.

    Constraint: \( ncode \geq 0 \).

12: odedef – function, supplied by the user
    External Function

    \textit{odedef} must evaluate the functions \( F \), which define the system of ODEs, as given in (3).

    \textit{odedef} will never be called and the NAG defined null void function pointer, NULLFN, can be
    supplied in the call to nag_pde_parab_1d_fd_ode_remesh (d03ppc).

The specification of \textit{odedef} is:

\begin{verbatim}
void odedef (Integer npde, double t, Integer ncode, const double v[],
            const double vdot[], Integer nxi, const double xi[],
            const double ucp[], const double ucpx[], const double rcp[],
            const double ucpt[], const double ucptx[], double f[],
            Integer *ires, Nag_Comm *comm)
\end{verbatim}

1: \text{npde} – Integer
   \textit{Input}
   
   On entry: the number of PDEs in the system.

2: \text{t} – double
   \textit{Input}
   
   On entry: the current value of the independent variable \text{t}.

3: \text{ncode} – Integer
   \textit{Input}
   
   On entry: the number of coupled ODEs in the system.

4: \text{v[ncode]} – const double
   \textit{Input}
   
   On entry: if \text{ncode} > 0, \text{v[i - 1]} contains the value of the component \( V_i(t) \), for \( i = 1, 2, \ldots, \text{ncode} \).

5: \text{vdot[ncode]} – const double
   \textit{Input}
   
   On entry: if \text{ncode} > 0, \text{vdot[i - 1]} contains the value of component \( \dot{V}_i(t) \), for \( i = 1, 2, \ldots, \text{ncode} \).

6: \text{nxi} – Integer
   \textit{Input}
   
   On entry: the number of ODE/PDE coupling points.

7: \text{xi[nxi]} – const double
   \textit{Input}
   
   On entry: if \text{nxi} > 0, \text{xi[i - 1]} contains the ODE/PDE coupling points, \( \xi_i \), for \( i = 1, 2, \ldots, \text{nxi} \).

8: \text{ucp[npde \times nxi]} – const double
   \textit{Input}
   
   On entry: if \text{nxi} > 0, \text{ucp[npde \times (j + 1) + i - 1]} contains the value of \( U_i(x, t) \) at the coupling point \( x = \xi_j \), for \( i = 1, 2, \ldots, \text{npde} \) and \( j = 1, 2, \ldots, \text{nxi} \).

9: \text{ucpx[npde \times nxi]} – const double
   \textit{Input}
   
   On entry: if \text{nxi} > 0, \text{ucpx[npde \times (j + 1) + i - 1]} contains the value of \( \frac{\partial U_i(x, t)}{\partial x} \) at the coupling point \( x = \xi_j \), for \( i = 1, 2, \ldots, \text{npde} \) and \( j = 1, 2, \ldots, \text{nxi} \).

10: \text{rcp[npde \times nxi]} – const double
    \textit{Input}
    
    On entry: \text{rcp[npde \times (j + 1) + i - 1]} contains the value of the flux \( R_i \) at the coupling point \( x = \xi_j \), for \( i = 1, 2, \ldots, \text{npde} \) and \( j = 1, 2, \ldots, \text{nxi} \).

11: \text{ucpt[npde \times nxi]} – const double
    \textit{Input}
    
    On entry: if \text{nxi} > 0, \text{ucpt[npde \times (j + 1) + i - 1]} contains the value of \( \frac{\partial U_i}{\partial t} \) at the coupling point \( x = \xi_j \), for \( i = 1, 2, \ldots, \text{npde} \) and \( j = 1, 2, \ldots, \text{nxi} \).

12: \text{ucptx[npde \times nxi]} – const double
    \textit{Input}
    
    On entry: \text{ucptx[npde \times (j + 1) + i - 1]} contains the value of \( \frac{\partial^2 U_i}{\partial x \partial t} \) at the coupling point \( x = \xi_j \), for \( i = 1, 2, \ldots, \text{npde} \) and \( j = 1, 2, \ldots, \text{nxi} \).

13: \text{f[ncode]} – double
    \textit{Output}
    
    On exit: \text{f[i - 1]} must contain the \( i \)th component of \( F \), for \( i = 1, 2, \ldots, \text{ncode} \), where \( F \) is defined as
\[ F = G - A\dot{V} - B\left(\frac{U_t}{U_{xt}}\right), \]  
\(5\)

or

\[ F = -A\dot{V} - B\left(\frac{U_t}{U_{xt}}\right). \]  
\(6\)

The definition of \(F\) is determined by the input value of \(\text{ires}\).

14: \(\text{ires} \quad \text{Integer}^* \quad \text{Input/Output}\)

On entry: the form of \(F\) that must be returned in the array \(\text{f}\).

\(\text{ires} = 1\)

Equation (5) must be used.

\(\text{ires} = -1\)

Equation (6) must be used.

On exit: should usually remain unchanged. However, you may reset \(\text{ires}\) to force the integration function to take certain actions as described below:

\(\text{ires} = 2\)

Indicates to the integrator that control should be passed back immediately to the calling function with the error indicator set to \text{fail.code} = \text{NE_USER_STOP}.

\(\text{ires} = 3\)

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. You may wish to set \(\text{ires} = 3\) when a physically meaningless input or output value has been generated. If you consecutively set \(\text{ires} = 3\), then nag_pde_parab_1d_fd_ode_remesh (d03ppc) returns to the calling function with the error indicator set to \text{fail.code} = \text{NE_FAILED_DERIV}.

15: \(\text{comm} \quad \text{Nag_Comm}^*\)

Pointer to structure of type Nag_Comm; the following members are relevant to \text{odedef}.

\(\text{user} \quad \text{double}^*\)
\(\text{iuser} \quad \text{Integer}^*\)
\(\text{p} \quad \text{Pointer}\)

The type Pointer will be \text{void}^*. Before calling nag_pde_parab_1d_fd_ode_remesh (d03ppc) you may allocate memory and initialize these pointers with various quantities for use by \text{odedef} when called from nag_pde_parab_1d_fd_ode_remesh (d03ppc) (see Section 3.2.1.1 in the Essential Introduction).

13: \(\text{nxi} \quad \text{Integer} \quad \text{Input}\)

On entry: the number of ODE/PDE coupling points.

Constraints:

if \(\text{ncode} = 0\), \(\text{nxi} = 0\);  
if \(\text{ncode} > 0\), \(\text{nxi} \geq 0\).

14: \(\text{xi}[\text{nxi}] \quad \text{const double} \quad \text{Input}\)

On entry: if \(\text{xi} > 0\), \(\text{xi}[i-1]\), for \(i = 1,2,\ldots,\text{nxi}\), must be set to the ODE/PDE coupling points.

Constraint: \(\text{x}[0] \leq \text{xi}[0] < \text{xi}[1] < \cdots < \text{xi}[\text{nxi}-1] \leq \text{x}[\text{npts}-1]\).
15: **neqn** – Integer

*On entry:* the number of ODEs in the time direction.

*Constraint:* \( \text{neqn} = \text{npde} \times \text{npts} + \text{ncode}. \)

16: **rtol[dim]** – const double

*Input*

*Note:* the dimension, \( \text{dim} \), of the array \( \text{rtol} \) must be at least

1 when \( \text{itol} = 1 \) or 2;

\( \text{neqn} \) when \( \text{itol} = 3 \) or 4.

*On entry:* the relative local error tolerance.

*Constraint:* \( \text{rtol}[i-1] \geq 0.0 \) for all relevant \( i \).

17: **atol[dim]** – const double

*Input*

*Note:* the dimension, \( \text{dim} \), of the array \( \text{atol} \) must be at least

1 when \( \text{itol} = 1 \) or 3;

\( \text{neqn} \) when \( \text{itol} = 2 \) or 4.

*On entry:* the absolute local error tolerance.

*Constraints:*

\( \text{atol}[i-1] \geq 0.0 \) for all relevant \( i \);

Corresponding elements of \( \text{atol} \) and \( \text{rtol} \) cannot both be 0.0.

18: **itol** – Integer

*Input*

*On entry:* a value to indicate the form of the local error test. \( \text{itol} \) indicates to \n\n\text{d03ppc} (d03ppc) whether to interpret either or both of \( \text{rtol} \) or \( \text{atol} \) as a vector or scalar. The error test to be satisfied is \( \|e_i/w_i\| < 1.0 \), where \( w_i \) is defined as follows:

\[
\text{itol} \quad \text{rtol} \quad \text{atol} \\
1 \quad \text{scalar} \quad \text{scalar} \quad \text{rtol}[0] \times |U_i| + \text{atol}[0] \\
2 \quad \text{scalar} \quad \text{vector} \quad \text{rtol}[0] \times |U_i| + \text{atol}[i-1] \\
3 \quad \text{vector} \quad \text{scalar} \quad \text{rtol}[i-1] \times |U_i| + \text{atol}[0] \\
4 \quad \text{vector} \quad \text{vector} \quad \text{rtol}[i-1] \times |U_i| + \text{atol}[i-1]
\]

In the above, \( e_i \) denotes the estimated local error for the \( i \)th component of the coupled PDE/ODE system in time, \( u[i-1] \), for \( i = 1, 2, \ldots, \text{neqn} \).

The choice of norm used is defined by the argument \( \text{norm} \).

*Constraint:* \( 1 \leq \text{itol} \leq 4 \).

19: **norm** – Nag_NormType

*Input*

*On entry:* the type of norm to be used.

\( \text{norm} = \text{Nag_MaxNorm} \)

Maximum norm.

\( \text{norm} = \text{Nag_TwoNorm} \)

Averaged \( L_2 \) norm.

If \( \text{u}_\text{norm} \) denotes the norm of the vector \( u \) of length \( \text{neqn} \), then for the averaged \( L_2 \) norm

\[
\text{u}_\text{norm} = \sqrt{\frac{1}{\text{neqn}} \sum_{i=1}^{\text{neqn}} (u[i-1]/w_i)^2},
\]

while for the maximum norm

\[
\text{u}_\text{norm} = \max_{i=1}^{\text{neqn}} (u[i-1]/w_i).
\]
\[ u_{\text{norm}} = \max_i |u[i - 1]/w_i|. \]

See the description of `itol` for the formulation of the weight vector \( w \).

**Constraint:** \( \text{norm} = \text{Nag\_MaxNorm} \) or \( \text{Nag\_TwoNorm} \).

20: \( \text{laopt} \) – Nag\_LinAlgOption

*Input*

On entry: the type of matrix algebra required.

- \( \text{laopt} = \text{Nag\_LinAlgFull} \)
  - Full matrix methods to be used.
- \( \text{laopt} = \text{Nag\_LinAlgBand} \)
  - Banded matrix methods to be used.
- \( \text{laopt} = \text{Nag\_LinAlgSparse} \)
  - Sparse matrix methods to be used.

**Constraint:** \( \text{laopt} = \text{Nag\_LinAlgFull}, \text{Nag\_LinAlgBand} \) or \( \text{Nag\_LinAlgSparse} \).

Note: you are recommended to use the banded option when no coupled ODEs are present (i.e., \( ncode = 0 \)).

21: \( \text{algopt}[30] \) – const double

*Input*

On entry: may be set to control various options available in the integrator. If you wish to employ all the default options, then \( \text{algopt}[0] \) should be set to 0.0. Default values will also be used for any other elements of \( \text{algopt} \) set to zero. The permissible values, default values, and meanings are as follows:

- \( \text{algopt}[0] \)
  Selects the ODE integration method to be used. If \( \text{algopt}[0] = 1.0 \), a BDF method is used and if \( \text{algopt}[0] = 2.0 \), a Theta method is used. The default value is \( \text{algopt}[0] = 1.0 \).

- \( \text{algopt}[1] \)
  Specifies the maximum order of the BDF integration formula to be used. \( \text{algopt}[1] \) may be 1.0, 2.0, 3.0, 4.0 or 5.0. The default value is \( \text{algopt}[1] = 5.0 \).

- \( \text{algopt}[2] \)
  Specifies what method is to be used to solve the system of nonlinear equations arising on each step of the BDF method. If \( \text{algopt}[2] = 1.0 \) a modified Newton iteration is used and if \( \text{algopt}[2] = 2.0 \) a functional iteration method is used. If functional iteration is selected and the integrator encounters difficulty, then there is an automatic switch to the modified Newton iteration. The default value is \( \text{algopt}[2] = 1.0 \).

- \( \text{algopt}[3] \)
  Specifies whether or not the Petzold error test is to be employed. The Petzold error test results in extra overhead but is more suitable when algebraic equations are present, such as \( P_{i,j} = 0.0 \), for \( j = 1, 2, \ldots, \text{npde} \), for some \( i \) or when there is no \( V_i(t) \) dependence in the coupled ODE system. If \( \text{algopt}[3] = 1.0 \), then the Petzold test is used. If \( \text{algopt}[3] = 2.0 \), then the Petzold test is not used. The default value is \( \text{algopt}[3] = 1.0 \).

- \( \text{algopt}[4] \)
  Specifies the value of Theta to be used in the Theta integration method. \( 0.51 \leq \text{algopt}[4] \leq 0.99 \). The default value is \( \text{algopt}[4] = 0.55 \).

- \( \text{algopt}[5] \)
  Specifies what method is to be used to solve the system of nonlinear equations arising on each step of the Theta method. If \( \text{algopt}[5] = 1.0 \), a modified Newton iteration is used and...
if \( \text{algopt}[5] = 2.0 \), a functional iteration method is used. The default value is \( \text{algopt}[5] = 1.0 \).

\( \text{algopt}[6] \) Specifies whether or not the integrator is allowed to switch automatically between modified Newton and functional iteration methods in order to be more efficient. If \( \text{algopt}[6] = 1.0 \), then switching is allowed and if \( \text{algopt}[6] = 2.0 \), then switching is not allowed. The default value is \( \text{algopt}[6] = 1.0 \).

\( \text{algopt}[10] \) Specifies a point in the time direction, \( t_{\text{crit}} \), beyond which integration must not be attempted. The use of \( t_{\text{crit}} \) is described under the argument \( \text{itask} \). If \( \text{algopt}[6] \neq 0.0 \), a value of 0.0 for \( \text{algopt}[10] \), say, should be specified even if \( \text{itask} \) subsequently specifies that \( t_{\text{crit}} \) will not be used.

\( \text{algopt}[11] \) Specifies the minimum absolute step size to be allowed in the time integration. If this option is not required, \( \text{algopt}[11] \) should be set to 0.0.

\( \text{algopt}[12] \) Specifies the maximum absolute step size to be allowed in the time integration. If this option is not required, \( \text{algopt}[12] \) should be set to 0.0.

\( \text{algopt}[13] \) Specifies the initial step size to be attempted by the integrator. If \( \text{algopt}[13] = 0.0 \), then the initial step size is calculated internally.

\( \text{algopt}[14] \) Specifies the maximum number of steps to be attempted by the integrator in any one call. If \( \text{algopt}[14] = 0.0 \), then no limit is imposed.

\( \text{algopt}[22] \) Specifies what method is to be used to solve the nonlinear equations at the initial point to initialize the values of \( U, U_t, V \) and \( V_t \). If \( \text{algopt}[22] = 1.0 \), a modified Newton iteration is used and if \( \text{algopt}[22] = 2.0 \), functional iteration is used. The default value is \( \text{algopt}[22] = 1.0 \).

\( \text{algopt}[28] \) and \( \text{algopt}[29] \) are used only for the sparse matrix algebra option, \( \text{laopt} = \text{Nag\_LinAlgSparse} \).

\( \text{algopt}[28] \) Governs the choice of pivots during the decomposition of the first Jacobian matrix. It should lie in the range \( 0.0 < \text{algopt}[28] < 1.0 \), with smaller values biasing the algorithm towards maintaining sparsity at the expense of numerical stability. If \( \text{algopt}[28] \) lies outside this range then the default value is used. If the functions regard the Jacobian matrix as numerically singular then increasing \( \text{algopt}[28] \) towards 1.0 may help, but at the cost of increased fill-in. The default value is \( \text{algopt}[28] = 0.1 \).

\( \text{algopt}[29] \) Is used as a relative pivot threshold during subsequent Jacobian decompositions (see \( \text{algopt}[28] \)) below which an internal error is invoked. If \( \text{algopt}[29] \) is greater than 1.0 no check is made on the pivot size, and this may be a necessary option if the Jacobian is found to be numerically singular (see \( \text{algopt}[28] \)). The default value is \( \text{algopt}[29] = 0.0001 \).

22: \( \text{remesh} \) = Nag_Boolean

\( \text{Input} \)

\( \text{On entry:} \) indicates whether or not spatial remeshing should be performed.

\( \text{remesh} = \text{Nag\_TRUE} \) Indicates that spatial remeshing should be performed as specified.

\( \text{remesh} = \text{Nag\_FALSE} \) Indicates that spatial remeshing should be suppressed.
**Note:** remesh should not be changed between consecutive calls to nag_pde_parab_1d_fd_ode_remesh (d03ppc). Remeshing can be switched off or on at specified times by using appropriate values for the arguments **nrmesh** and **trmesh** at each call.

23: **nxfix** – Integer  
*Input*

*On entry:* the number of fixed mesh points.

*Constraint:*  $0 \leq nxfix \leq \text{npts} - 2$.

*Note:* the end points $x[0]$ and $x[\text{npts} - 1]$ are fixed automatically and hence should not be specified as fixed points.

24: **xfix[dim]** – const double  
*Input*

*Note:* the dimension, dim, of the array xfix must be at least max(1, nxfix).

*On entry:* $xfix[i - 1]$, for $i = 1, 2, \ldots, nxfix$, must contain the value of the $x$ coordinate at the $i$th fixed mesh point.

*Constraints:*

\[ xfix[i - 1] < xfix[i], \text{ for } i = 1, 2, \ldots, nxfix - 1; \]

each fixed mesh point must coincide with a user-supplied initial mesh point, that is $xfix[i - 1] = x[j - 1]$ for some $j$, $2 \leq j \leq \text{npts} - 1$.

*Note:* the positions of the fixed mesh points in the array x remain fixed during remeshing, and so the number of mesh points between adjacent fixed points (or between fixed points and end points) does not change. You should take this into account when choosing the initial mesh distribution.

25: **nrmesh** – Integer  
*Input*

*On entry:* specifies the spatial remeshing frequency and criteria for the calculation and adoption of a new mesh.

- **nrmesh** < 0  
  Indicates that a new mesh is adopted according to the argument **dxmesh**. The mesh is tested every $|nrmesh|$ timesteps.

- **nrmesh** = 0  
  Indicates that remeshing should take place just once at the end of the first time step reached when $t > \text{trmesh}$.

- **nrmesh** > 0  
  Indicates that remeshing will take place every **nrmesh** time steps, with no testing using **dxmesh**.

*Note:* **nrmesh** may be changed between consecutive calls to nag_pde_parab_1d_fd_ode_remesh (d03ppc) to give greater flexibility over the times of remeshing.

26: **dxmesh** – double  
*Input*

*On entry:* determines whether a new mesh is adopted when **nrmesh** is set less than zero. A possible new mesh is calculated at the end of every $|nrmesh|$ time steps, but is adopted only if

\[ x_i^{(\text{new})} > x_i^{(\text{old})} + dxmesh \times (x_{i+1}^{(\text{old})} - x_i^{(\text{old})}) \]

or

\[ x_i^{(\text{new})} < x_i^{(\text{old})} - dxmesh \times (x_i^{(\text{old})} - x_{i-1}^{(\text{old})}) \]

**dxmesh** thus imposes a lower limit on the difference between one mesh and the next.

*Constraint:* $dxmesh \geq 0.0$. 

---

**d03ppc.14**  
Mark 25
27: \texttt{trmesh} – double

\textit{Input}

\textit{On entry:} specifies when remeshing will take place when \texttt{nrmesh} is set to zero. Remeshing will occur just once at the end of the first time step reached when \( t \) is greater than \texttt{trmesh}.

\textbf{Note:} \texttt{trmesh} may be changed between consecutive calls to \texttt{nag_pde_parab_1d_fd_ode_remesh} (d03ppc) to force remeshing at several specified times.

28: \texttt{ipminf} – Integer

\textit{Input}

\textit{On entry:} the level of trace information regarding the adaptive remeshing.

\texttt{ipminf} = 0

No trace information.

\texttt{ipminf} = 1

Brief summary of mesh characteristics.

\texttt{ipminf} = 2

More detailed information, including old and new mesh points, mesh sizes and monitor function values.

\textbf{Constraint:} \texttt{ipminf} = 0, 1 or 2.

29: \texttt{xratio} – double

\textit{Input}

\textit{On entry:} an input bound on the adjacent mesh ratio (greater than 1.0 and typically in the range 1.5 to 3.0). The remeshing functions will attempt to ensure that

\[ (x_i - x_{i-1})/\texttt{xratio} < x_{i+1} - x_i < \texttt{xratio} \times (x_i - x_{i-1}). \]

\textbf{Suggested value:} \texttt{xratio} = 1.5.

\textbf{Constraint:} \texttt{xratio} > 1.0.

30: \texttt{con} – double

\textit{Input}

\textit{On entry:} an input bound on the sub-integral of the monitor function \( F_{\text{mon}}(x) \) over each space step. The remeshing functions will attempt to ensure that

\[ \int_{x_i}^{x_{i+1}} F_{\text{mon}}(x) \, dx \leq \texttt{con} \int_{x_1}^{x_{npts}} F_{\text{mon}}(x) \, dx, \]

(see Furzeland (1984)). \texttt{con} gives you more control over the mesh distribution e.g., decreasing \texttt{con} allows more clustering. A typical value is \( 2/(\texttt{npts} - 1) \), but you are encouraged to experiment with different values. Its value is not critical and the mesh should be qualitatively correct for all values in the range given below.

\textbf{Suggested value:} \texttt{con} = 2.0/(\texttt{npts} - 1).

\textbf{Constraint:} 0.1/(\texttt{npts} - 1) \leq \texttt{con} \leq 10.0/(\texttt{npts} - 1).

31: \texttt{monitf} – function, supplied by the user

\textit{External Function}

\texttt{monitf} must supply and evaluate a remesh monitor function to indicate the solution behaviour of interest.

If you specify \texttt{remesh} = Nag_FALSE, i.e., no remeshing, then \texttt{monitf} will not be called and may be specified as NULLFN.

The specification of \texttt{monitf} is:

\begin{verbatim}
void monitf (double t, Integer npts, Integer npde, const double x[],
            const double u[], const double r[], double fmon[],
            Nag_Comm *comm)
\end{verbatim}
1: \( t \) – double
   \textit{Input}
   
   \textit{On entry:} the current value of the independent variable \( t \).

2: \( \text{npts} \) – Integer
   \textit{Input}
   
   \textit{On entry:} the number of mesh points in the interval \([a, b]\).

3: \( \text{npde} \) – Integer
   \textit{Input}
   
   \textit{On entry:} the number of PDEs in the system.

4: \( x[\text{npts}] \) – const double
   \textit{Input}
   
   \textit{On entry:} the current mesh. \( x[i - 1] \) contains the value of \( x_i \), for \( i = 1, 2, \ldots, \text{npts} \).

5: \( u[\text{npde} \times \text{npts}] \) – const double
   \textit{Input}
   
   \textit{On entry:} \( u[\text{npde} \times (j - 1) + i - 1] \) contains the value of \( U_i(x, t) \) at \( x = x[j - 1] \) and time \( t \), for \( i = 1, 2, \ldots, \text{npde} \) and \( j = 1, 2, \ldots, \text{npts} \).

6: \( r[\text{npde} \times \text{npts}] \) – const double
   \textit{Input}
   
   \textit{On entry:} \( r[\text{npde} \times (j - 1) + i - 1] \) contains the value of \( R_i(x, t, U, U_x, V) \) at \( x = x[j - 1] \) and time \( t \), for \( i = 1, 2, \ldots, \text{npde} \) and \( j = 1, 2, \ldots, \text{npts} \).

7: \( \text{fmon}[\text{npts}] \) – double
   \textit{Output}
   
   \textit{On exit:} \( \text{fmon}[i - 1] \) must contain the value of the monitor function \( F^{\text{mon}}(x) \) at mesh point \( x = x[i - 1] \).
   
   \textit{Constraint:} \( \text{fmon}[i - 1] \geq 0.0 \).

8: \( \text{comm} \) – Nag_Comm *
   
   Pointer to structure of type Nag_Comm; the following members are relevant to \texttt{monitf}.

   \text{user} – double *
   \text{iuser} – Integer *
   \text{p} – Pointer
   
   The type Pointer will be void *. Before calling \texttt{nag_pde_parab_1d_fd_ode_remesh (d03ppc)} you may allocate memory and initialize these pointers with various quantities for use by \texttt{monitf} when called from \texttt{nag_pde_parab_1d_fd_ode_remesh (d03ppc)} (see Section 3.2.1.1 in the Essential Introduction).

32: \( \text{rsave}[\text{lrsave}] \) – double
   \textit{Communication Array}

   If \( \text{ind} = 0 \), \text{rsave} need not be set on entry.

   If \( \text{ind} = 1 \), \text{rsave} must be unchanged from the previous call to the function because it contains required information about the iteration.

33: \( \text{lrsave} \) – Integer
   \textit{Input}
   
   \textit{On entry:} the dimension of the array \text{rsave}. Its size depends on the type of matrix algebra selected.

   If \( \text{laopt} = \text{Nag_LinAlgFull} \), \( \text{lrsave} \geq \text{neqn} \times \text{neqn} + \text{neqn} + \text{nwkres} + \text{lenode} \).

   If \( \text{laopt} = \text{Nag_LinAlgBand} \), \( \text{lrsave} \geq (3 \times \text{mlu} + 1) \times \text{neqn} + \text{nwkres} + \text{lenode} \).

   If \( \text{laopt} = \text{Nag_LinAlgSparse} \), \( \text{lrsave} \geq 4 \times \text{neqn} + 11 \times \text{neqn}/2 + 1 + \text{nwkres} + \text{lenode} \).

Where
mlu is the lower or upper half bandwidths such that
mlu = \(2 \times npde - 1\), for PDE problems only; or
mlu = \(neqn - 1\), for coupled PDE/ODE problems.

\[nwkres = \begin{cases} 
npde \times (3 \times npde - 6 \times nxi + npts + 15) + nxi + ncode + 7 \times npts + nxfix + 1, & \text{when } ncode > 0 \text{ and } nxi > 0; \\
npde \times (3 \times npde - npts + 21) + ncode + 7 \times npts + nxfix + 2, & \text{when } ncode > 0 \text{ and } nxi = 0; \\
npde \times (3 \times npde - npts + 21) + 7 \times npts + nxfix + 3, & \text{when } ncode = 0.
\end{cases}\]

\[lenode = \begin{cases} 
(6 + \text{int(algopt}[1]) \times neqn + 50, & \text{when the BDF method is used}; \\
9 \times neqn + 50, & \text{when the Theta method is used}.
\end{cases}\]

**Note:** when using the sparse option, the value of lrsave may be too small when supplied to the integrator. An estimate of the minimum size of lrsave is printed on the current error message unit if itrace > 0 and the function returns with fail:code = NE_INT_2.

### isave \[\text{isave} - \text{Integer} \]

**Communication Array**

If ind = 0, isave need not be set on entry.

If ind = 1, isave must be unchanged from the previous call to the function because it contains required information about the iteration required for subsequent calls. In particular:

- **isave[0]**
  Contains the number of steps taken in time.

- **isave[1]**
  Contains the number of residual evaluations of the resulting ODE system used. One such evaluation involves computing the PDE functions at all the mesh points, as well as one evaluation of the functions in the boundary conditions.

- **isave[2]**
  Contains the number of Jacobian evaluations performed by the time integrator.

- **isave[3]**
  Contains the order of the ODE method last used in the time integration.

- **isave[4]**
  Contains the number of Newton iterations performed by the time integrator. Each iteration involves residual evaluation of the resulting ODE system followed by a back-substitution using the \(LU\) decomposition of the Jacobian matrix.

The rest of the array is used as workspace.

### lisave - Integer

**Input**

On entry: the dimension of the array isave.

Its size depends on the type of matrix algebra selected:

- if laopt = Nag_LinAlgBand, lisave \(\geq neqn + 25 + nxfix\);
- if laopt = Nag_LinAlgFull, lisave \(\geq 25 + nxfix\);
- if laopt = Nag_LinAlgSparse, lisave \(\geq 25 \times neqn + 25 + nxfix\).

**Note:** when using the sparse option, the value of lisave may be too small when supplied to the integrator. An estimate of the minimum size of lisave is printed if itrace > 0 and the function returns with fail:code = NE_INT_2.

### itask - Integer

**Input**

On entry: specifies the task to be performed by the ODE integrator.

- itask = 1
  Normal computation of output values \(u\) at \(t = tout\).

- itask = 2
  One step and return.
itask = 3
   Stop at first internal integration point at or beyond \( t = \text{tout} \).

itask = 4
   Normal computation of output values \( u \) at \( t = \text{tout} \) but without overshooting \( t = t_{\text{crit}} \) where \( t_{\text{crit}} \) is described under the argument \( \text{algopt} \).

itask = 5
   Take one step in the time direction and return, without passing \( t_{\text{crit}} \), where \( t_{\text{crit}} \) is described under the argument \( \text{algopt} \).

Constraint: itask = 1, 2, 3, 4 or 5.

itrace – Integer

Input

\( \text{On entry:} \) the level of trace information required from nag_pde_parab_1d_fd_ode_remesh (d03ppc) and the underlying ODE solver:

itrace \( \leq -1 \)
   No output is generated.

itrace = 0
   Only warning messages from the PDE solver are printed.

itrace = 1
   Output from the underlying ODE solver is printed. This output contains details of Jacobian entries, the nonlinear iteration and the time integration during the computation of the ODE system.

itrace = 2
   Output from the underlying ODE solver is similar to that produced when itrace = 1, except that the advisory messages are given in greater detail.

itrace \( \geq 3 \)
   Output from the underlying ODE solver is similar to that produced when itrace = 2, except that the advisory messages are given in greater detail.

outfile – const char *

Input

\( \text{On entry:} \) the name of a file to which diagnostic output will be directed. If outfile is NULL the diagnostic output will be directed to standard output.

ind – Integer *

Input/Output

\( \text{On entry:} \) must be set to 0 or 1.

\( \text{ind} = 0 \)
   Starts or restarts the integration in time.

\( \text{ind} = 1 \)
   Continues the integration after an earlier exit from the function. In this case, only the arguments tout and fail and the remeshing arguments nrmesh, dxmesh, trmesh, xratio and con may be reset between calls to nag_pde_parab_1d_fd_ode_remesh (d03ppc).

Constraint: \( 0 \leq \text{ind} \leq 1 \).

\( \text{On exit:} \) ind = 1.

comm – Nag_Comm *

The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

saved – Nag_D03_Save *

Communication Structure

\( \text{saved} \) must remain unchanged following a previous call to a Chapter d03 function and prior to any subsequent call to a Chapter d03 function.
6 Error Indicators and Warnings

NE_ACC_IN_DOUBT
Integration completed, but small changes in \texttt{atol} or \texttt{rtol} are unlikely to result in a changed solution.

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM
On entry, argument \textit{value} had an illegal value.

NE_FAILED_DERIV
In setting up the ODE system an internal auxiliary was unable to initialize the derivative. This could be due to your setting \texttt{ires} = 3 in \texttt{pde}	def or \texttt{bndary}.

NE_FAILED_START
\texttt{atol} and \texttt{rtol} were too small to start integration.
Underlying ODE solver cannot make further progress from the point \texttt{ts} with the supplied values of \texttt{atol} and \texttt{rtol}. \texttt{ts} = \langle value\rangle.

NE_FAILED_STEP
Error during Jacobian formulation for ODE system. Increase \texttt{itrace} for further details.
Repeated errors in an attempted step of underlying ODE solver. Integration was successful as far as \texttt{ts}: \texttt{ts} = \langle value\rangle.

NE_INCOMPAT_PARAM
On entry, \texttt{con} = \langle value\rangle, \texttt{npts} = \langle value\rangle.
Constraint: \texttt{con} \leq 10.0/(\texttt{npts} - 1).
On entry, \texttt{con} = \langle value\rangle, \texttt{npts} = \langle value\rangle.
Constraint: \texttt{con} \geq 0.1/(\texttt{npts} - 1).
On entry, \texttt{m} = \langle value\rangle and \texttt{x}[0] = \langle value\rangle.
Constraint: \texttt{m} \leq 0 or \texttt{x}[0] \geq 0.0
On entry, the point \texttt{xfix}[I - 1] does not coincide with any \texttt{x}[J - 1]: \texttt{I} = \langle value\rangle and \texttt{xfix}[I - 1] = \langle value\rangle.

NE_INT
\texttt{ires} set to an invalid value in call to \texttt{pde}, \texttt{bndary}, or \texttt{ode}.
On entry, \texttt{ind} = \langle value\rangle.
Constraint: \texttt{ind} = 0 or 1.
On entry, \texttt{ipminf} = \langle value\rangle.
Constraint: \texttt{ipminf} = 0, 1 or 2.
On entry, \texttt{itask} = \langle value\rangle.
Constraint: \texttt{itask} = 1, 2, 3, 4 or 5.
On entry, \(itol = \langle\text{value}\rangle\).
Constraint: \(itol = 1, 2, 3\) or \(4\).

On entry, \(m = \langle\text{value}\rangle\).
Constraint: \(m = 0, 1\) or \(2\).

On entry, \(ncode = \langle\text{value}\rangle\).
Constraint: \(ncode \geq 0\).

On entry, \(npde = \langle\text{value}\rangle\).
Constraint: \(npde \geq 1\).

On entry, \(npts = \langle\text{value}\rangle\).
Constraint: \(npts \geq 3\).

On entry, \(nxfix = \langle\text{value}\rangle\).
Constraint: \(nxfix \geq 0\).

**NE_INT_2**

On entry, corresponding elements \(atol[I - 1]\) and \(rtol[J - 1]\) are both zero: \(I = \langle\text{value}\rangle\) and \(J = \langle\text{value}\rangle\).

On entry, \(lisave\) is too small: \(lisave = \langle\text{value}\rangle\). Minimum possible dimension: \(\langle\text{value}\rangle\).

On entry, \(lrsave\) is too small: \(lrsave = \langle\text{value}\rangle\). Minimum possible dimension: \(\langle\text{value}\rangle\).

On entry, \(ncode = \langle\text{value}\rangle\) and \(nxi = \langle\text{value}\rangle\).
Constraint: \(nxi = 0\) when \(ncode = 0\).

On entry, \(ncode = \langle\text{value}\rangle\) and \(nxi = \langle\text{value}\rangle\).
Constraint: \(nxi \geq 0\) when \(ncode > 0\).

On entry, \(nxfix = \langle\text{value}\rangle\), \(npts = \langle\text{value}\rangle\).
Constraint: \(nxfix \leq npts - 2\).

When using the sparse option \(lisave\) or \(lrsave\) is too small: \(lisave = \langle\text{value}\rangle\), \(lrsave = \langle\text{value}\rangle\).

**NE_INT_4**

On entry, \(neqn = \langle\text{value}\rangle\), \(npde = \langle\text{value}\rangle\), \(npts = \langle\text{value}\rangle\) and \(ncode = \langle\text{value}\rangle\).
Constraint: \(neqn = npde \times npts + ncode\).

**NE_INTERNAL_ERROR**

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG. See Section 3.6.6 in the Essential Introduction for further information.

Serious error in internal call to an auxiliary. Increase \(itrace\) for further details.

**NE_ITER_FAIL**

In solving ODE system, the maximum number of steps \(algopt[14]\) has been exceeded.
\(algopt[14] = \langle\text{value}\rangle\).

**NE_NO_LICENCE**

Your licence key may have expired or may not have been installed correctly. See Section 3.6.5 in the Essential Introduction for further information.

**NE_NOT_CLOSE_FILE**

Cannot close file \(\langle\text{value}\rangle\).
NE_NOT_STRICTLY_INCREASING
On entry, \( I = \langle \text{value} \rangle \), \( xfix[I] = \langle \text{value} \rangle \) and \( xfix[I - 1] = \langle \text{value} \rangle \).
Constraint: \( xfix[I] > xfix[I - 1] \).
On entry, \( I = \langle \text{value} \rangle \), \( xi[I] = \langle \text{value} \rangle \) and \( xi[I - 1] = \langle \text{value} \rangle \).
Constraint: \( xi[I] > xi[I - 1] \).
On entry, mesh points \( x \) appear to be badly ordered: \( I = \langle \text{value} \rangle \), \( x[I - 1] = \langle \text{value} \rangle \), \( J = \langle \text{value} \rangle \) and \( x[J - 1] = \langle \text{value} \rangle \).

NE_NOT_WRITE_FILE
Cannot open file \( \langle \text{value} \rangle \) for writing.

NE_REAL
On entry, \( dxmesh = \langle \text{value} \rangle \).
Constraint: \( dxmesh \geq 0.0 \).
On entry, \( xratio = \langle \text{value} \rangle \).
Constraint: \( xratio > 1.0 \).

NE_REAL_2
On entry, at least one point in \( xi \) lies outside \( [x[0], x[npts - 1]]: x[0] = \langle \text{value} \rangle \) and \( x[npts - 1] = \langle \text{value} \rangle \).
On entry, \( tout = \langle \text{value} \rangle \) and \( ts = \langle \text{value} \rangle \).
Constraint: \( tout > ts \).
On entry, \( tout - ts \) is too small: \( tout = \langle \text{value} \rangle \) and \( ts = \langle \text{value} \rangle \).

NE_REAL_ARRAY
On entry, \( I = \langle \text{value} \rangle \) and \( atol[I - 1] = \langle \text{value} \rangle \).
Constraint: \( atol[I - 1] \geq 0.0 \).
On entry, \( I = \langle \text{value} \rangle \) and \( rtol[I - 1] = \langle \text{value} \rangle \).
Constraint: \( rtol[I - 1] \geq 0.0 \).

NE_REMESH_CHANGED
\( \text{remesh} \) has been changed between calls to \( \text{nag_pde_parab_1d_fd_ode_remesh (d03ppc)} \).

NE_SING_JAC
Singular Jacobian of ODE system. Check problem formulation.

NE_TIME_DERIV_DEP
Flux function appears to depend on time derivatives.

NE_USER_STOP
In evaluating residual of ODE system, \( ires = 2 \) has been set in \( \text{pdef}, \text{bndary}, \) or \( \text{ode} \). Integration is successful as far as \( ts: ts = \langle \text{value} \rangle \).

NE_ZERO_WTS
Zero error weights encountered during time integration.
7 Accuracy

nag_pde_parab_1d_fd_ode_remesh (d03ppc) controls the accuracy of the integration in the time direction but not the accuracy of the approximation in space. The spatial accuracy depends on both the number of mesh points and on their distribution in space. In the time integration only the local error over a single step is controlled and so the accuracy over a number of steps cannot be guaranteed. You should therefore test the effect of varying the accuracy arguments, atol and rtol.

8 Parallelism and Performance

nag_pde_parab_1d_fd_ode_remesh (d03ppc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_pde_parab_1d_fd_ode_remesh (d03ppc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

The argument specification allows you to include equations with only first-order derivatives in the space direction but there is no guarantee that the method of integration will be satisfactory for such systems. The position and nature of the boundary conditions in particular are critical in defining a stable problem. It may be advisable in such cases to reduce the whole system to first-order and to use the Keller box scheme function nag_pde_parab_1d_keller_ode_remesh (d03prc).

The time taken depends on the complexity of the parabolic system, the accuracy requested, and the frequency of the mesh updates. For a given system with fixed accuracy and mesh-update frequency it is approximately proportional to neqn.

10 Example

This example uses Burgers Equation, a common test problem for remeshing algorithms, given by

\[ \frac{\partial U}{\partial t} = -U \frac{\partial U}{\partial x} + E \frac{\partial^2 U}{\partial x^2}, \]

for \( x \in [0, 1] \) and \( t \in [0, 1] \), where \( E \) is a small constant.

The initial and boundary conditions are given by the exact solution

\[ U(x, t) = \frac{0.1 \exp(-A) + 0.5 \exp(-B) + \exp(-C)}{\exp(-A) + \exp(-B) + \exp(-C)}, \]

where

\[ A = \frac{50}{E}(x - 0.5 + 4.95t), \]

\[ B = \frac{250}{E}(x - 0.5 + 0.75t), \]

\[ C = \frac{500}{E}(x - 0.375). \]
10.1 Program Text

/* nag_pde_parab_1d_fd_ode_remesh (d03ppc) Example Program. *
* Copyright 2014 Numerical Algorithms Group. *
* Mark 7, 2001. */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd03.h>

#ifdef __cplusplus
extern "C" {
#endif

static void NAG_CALL pdedef(Integer, double, double, const double[],
const double[], Integer, const double[], const double[], double[], double[],
Integer *, Nag_Comm *);

static void NAG_CALL bndary(Integer, double, const double[], const double[],
Integer, const double[], const double[], Integer, double[], double[], Integer *, Nag_Comm *);

static void NAG_CALL uvinit(Integer, Integer, Integer, const double[],
const double[], double[], Integer, double[], Nag_Comm *);

static void NAG_CALL monitf(double, Integer, Integer, const double[],
const double[], const double[], double[], Nag_Comm *);

#ifdef __cplusplus
}
#endif

static void exact(double, double *, Integer, double *, Nag_Comm *);

#define P(I, J) p[npde*((J) -1)+(I) -1]
#define R(I, J) r[npde*((J) -1)+(I) -1]
#define U(I, J) u[npde*((J) -1)+(I) -1]
#define UOUT(I, J, K) uout[npde*(intpts*((K) -1)+(J) -1)+(I) -1]

int main(void)
{
    const Integer npde = 1, npts = 61, ncode = 0, m = 0, nxi = 0, nxfix = 0;
    const Integer itype = 1, neqn = npde*npts+ncode, intpts = 5;
    const Integer lisave = 25+nxfix;
    const Integer nwkres = npde*(npts+3*npde+21)+7*npts+nxfix+3;
    const Integer lenode = 11*neqn+50, lrsave = neqn*neqn+neqn+nwkres+lenode;
    static double ruser[4] = {-1.0, -1.0, -1.0, -1.0};
    double con, dxmesh, e, tout, trmesh, ts, xratio;
    Integer exit_status, i, ind, ipminf, it, itask, itol, itrace, nrmesh;
    Nag_Boolean remesh, theta;
    double *algopt = 0, *atol = 0, *rsave = 0, *rtol = 0, *u = 0, *ue = 0;
    double *uout = 0, *x = 0, *xfix = 0, *xi = 0, *xout = 0;
    Integer *isave = 0;
    NagError fail;
    Nag_Comm comm;
    Nag_D03_Save saved;

    INIT_FAIL(fail);
    exit_status = 0;

    /* Allocate memory */
    if (!(algopt = NAG_ALLOC(30, double))) ||
    
Mark 25

d03ppc.23
!(atol = NAG_ALLOC(1, double)) ||
!(rsave = NAG_ALLOC(lrsave, double)) ||
!(rtol = NAG_ALLOC(1, double)) ||
!(u = NAG_ALLOC(neqn, double)) ||
!(ue = NAG_ALLOC(intpts, double)) ||
!(uout = NAG_ALLOC(npde*intpts*itype, double)) ||
!(x = NAG_ALLOC(npts, double)) ||
!(xi = NAG_ALLOC(1, double)) ||
!(xfix = NAG_ALLOC(1, double)) ||
!(xout = NAG_ALLOC(intpts, double)) ||
!(isave = NAG_ALLOC(lisave, Integer)))
{
printf("Allocation failure\n");
extit_status = 1;
goto END;
}
printf("nag_pde_parab_1d_fd_ode_remesh (d03ppc) Example Program" " Results\n\n");
/* For communication with user-supplied functions: */
comm.user = ruser;
e = 0.005;
comm.p = (Pointer)&e;
itrace = 0;
itol = 1;
atol[0] = 5e-5;
rtol[0] = atol[0];

/* Initialise mesh */
for (i = 0; i < npts; ++i) x[i] = i/(npts-1.0);
/* Set remesh parameters */
remesh = Nag_TRUE;
nrmesh = 3;
dxmesh = 0.5;
trmesh = 0.0;
con = 2.0/(npts-1.0);
xratio = 1.5;
ipminf = 0;

/* Loop over output value of t */
for (i = 0; i < 30; ++i) algopt[i] = 0.0;
if (theta)
{
algotp[0] = 2.0;
}
else
{
algotp[0] = 0.0;
}
/* Set theta to TRUE if the Theta integrator is required */
theta = Nag_FALSE;
for (i = 0; i < 30; ++i) algopt[i] = 0.0;
if (theta)
{
algotp[0] = 2.0;
}
else
{
algotp[0] = 0.0;
}
/* Loop over output value of t */
ts = 0.0;
tout = 0.0;
for (it = 0; it < 5; ++it)
{
    tout = 0.2*(it+1);

    /* nag_pde_parab_1d_fd_ode_remesh (d03ppc).
        * General system of parabolic PDEs, coupled DAEs, method of
        * lines, finite differences, remeshing, one space variable
        */
    nag_pde_parab_1d_fd_ode_remesh(npde, m, &ts, tout, pdedef, bndary,
        uvinit, u, npts, x, ncode, NULLFN, nxif, xi, neqn, rtol, atol, itol,
        Nag_TwoNorm, Nag_LinAlgFull, algopt, remesh, nxfix, nxif, nrmesh,
        dxmesh, trmesh, ipminf, xratio, con, monitf, rsave, lrsave, isave,
        lisave, itask, itrace, 0, &ind, &comm,
        &saved, &fail);

    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_pde_parab_1d_fd_ode_remesh (d03ppc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }

    /* Set output points */
    switch (it)
    {
        case 0:
            for (i = 0; i < 5; ++i) xout[i] = 0.3+0.1*i;
            break;
        case 1:
            for (i = 0; i < 5; ++i) xout[i] = 0.4+0.1*i;
            break;
        case 2:
            for (i = 0; i < 5; ++i) xout[i] = 0.6+0.05*i;
            break;
        case 3:
            for (i = 0; i < 5; ++i) xout[i] = 0.7+0.05*i;
            break;
        case 4:
            for (i = 0; i < 5; ++i) xout[i] = 0.8+0.05*i;
            break;
    }

    printf(" t = %6.3f\n", ts);
    printf(" x ");
    for (i = 0; i < 5; ++i)
    {
        printf("%9.4f", xout[i]);
        printf((i+1)%5 == 0 || i == 4?"\n": "");
    }

    /* Interpolate at output points */
    /* nag_pde_interp_1d_fd (d03pzc). PDEs, spatial interpolation with
        * nag_pde_parab_1d_fd_ode_remesh (d03ppc),
        */
    nag_pde_interp_1d_fd(npde, m, u, npts, x, xout, intpts, itype, uout,
        &fail);

    if (fail.code != NE_NOERROR)
    {
        printf("Error from nag_pde_interp_1d_fd (d03pzc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
    }
}
/* Check against exact solution */
exact(ts, xout, intpts, ue, &comm);
printf(" Approx sol. ");
for (i = 1; i  intpts; ++i)
    {
        printf("%9.4f", UOUT(1, i, 1));
        printf(i%5 == 0 || i == 5?"\n": "");
    }
printf(" Exact sol. ");
for (i = 1; i  intpts; ++i)
    {
        printf("%9.4f", ue[i-1]);
        printf(i%5 == 0 || i == 5?"\n": "");
    }
printf("Number of integration steps in time = %6"NAG_IFMT"\n", isave[0]);
printf("Number of function evaluations = %6"NAG_IFMT"\n", isave[1]);
printf("Number of Jacobian evaluations = %6"NAG_IFMT"\n", isave[2]);
printf("Number of iterations = %6"NAG_IFMT"\n\n", isave[4]);
}

END:
NAG_FREE(algopt);
NAG_FREE(atol);
NAG_FREE(rsave);
NAG_FREE(rtol);
NAG_FREE(ue);
NAG_FREE(uout);
NAG_FREE(x);
NAG_FREE(xfix);
NAG_FREE(xi);
NAG_FREE(xout);
NAG_FREE(isave);
return exit_status;
}

static void NAG_CALL uvinit(Integer npde, Integer npts, Integer nxi,
        const double x[], const double xi[], double u[],
        Integer ncode, double v[], Nag_Comm *comm)
        {
            double  *e = (double *) comm->p;
double a, b, c, t;
            Integer i;

            if (comm->user[0] == -1.0)
                {
                    printf("(User-supplied callback uvinit, first invocation.\n"");
                    comm->user[0] = 0.0;
                }
t = 0.0;
for (i = 1; i  npts; ++i)
    {
        a = (x[i-1] - 0.25 - 0.75*t)/(e*4.0);
b = (0.9*x[i-1] - 0.325 - 0.495*t)/(e*2.0);
        if (a > 0.0 && a > b)
            {
                a = exp(-a);
c = (0.8*x[i-1] - 0.4 - 0.24*t)/(e*4.0);
c = exp(c);
U(1, i) = (0.1*c + 0.5 + a)/(c + 1.0 + a);
}
else if (b > 0.0 && b >= a)
{
    b = exp(-b);
    c = (-0.8*x[i-1] + 0.4 + 0.24*t)/(*e*4.0);
    c = exp(c);
    U(1, i) = (0.5*c + 0.1 + b)/(c + 1.0 + b);
}
else
{
    a = exp(a);
    b = exp(b);
    U(1, i) = (0.5*a + 1.0 + 0.1*b)/(a + 1.0 + b);
}
return;
}

static void NAG_CALL pdedef(Integer npde, double t, double x, const double u[],
const double ux[], Integer ncode, const double v[],
const double vdot[], double p[], double q[],
double r[], Integer *ires, Nag_Comm *comm)
{
    double *e = (double *) comm->p;
    if (comm->user[1] == -1.0)
    {
        printf("(User-supplied callback pdedef, first invocation.)\n");
        comm->user[1] = 0.0;
    }
    P(1, 1) = 1.0;
    r[0] = *e*ux[0];
    q[0] = u[0]*ux[0];
    return;
}

static void NAG_CALL bndary(Integer npde, double t, const double u[],
const double ux[], Integer ncode, const double v[],
const double vdot[], Integer ibnd, double beta[],
double gamma[], Integer *ires, Nag_Comm *comm)
{
    double a, b, c, ue, x;
    double *e = (double *) comm->p;
    if (comm->user[2] == -1.0)
    {
        printf("(User-supplied callback bndary, first invocation.)\n");
        comm->user[2] = 0.0;
    }
    beta[0] = 0.0;
    if (ibnd == 0)
    {
        x = 0.0;
        a = (x - 0.25 - 0.75*t)/(*e*4.0);
        b = (0.9*x - 0.325 - 0.495*t)/(*e*2.0);
        if (a > 0.0 && a > b)
        {
            a = exp(-a);
            c = (0.8*x - 0.4 - 0.24*t)/(*e*4.0);
            c = exp(c);
            ue = (0.1*c + 0.5 + a)/(c + 1.0 + a);
        }
        else if (b > 0.0 && b >= a)
        {
            b = exp(-b);
            c = (-0.8*x + 0.4 + 0.24*t)/(*e*4.0);
            c = exp(c);
            ue = (0.5*c + 0.1 + b)/(c + 1.0 + b);
        }
        else
        { }
```c
{ 
    a = exp(a);
    b = exp(b);
    ue = (0.5*a + 1.0 + 0.1*b)/(a + 1.0 + b);
}
else
{
    x = 1.0;
    a = (x - 0.25 - 0.75*t)/(e*4.0);
    b = (0.9*x - 0.325 - 0.495*t)/(e*2.0);
    if (a > 0.0 && a > b)
    {
        a = exp(-a);
        c = (0.8*x - 0.4 - 0.24*t)/(e*4.0);
        c = exp(c);
        ue = (0.1*c + 0.5 + a)/(c + 1.0 + a);
    }
    else if (b > 0.0 && b >= a)
    {
        b = exp(-b);
        c = (-0.8*x + 0.4 + 0.24*t)/(e*4.0);
        c = exp(c);
        ue = (0.5*c + 0.1 + b)/(c + 1.0 + b);
    }
    else
    {
        a = exp(a);
        b = exp(b);
        ue = (0.5*a + 1.0 + 0.1*b)/(a + 1.0 + b);
    }
}
gamma[0] = u[0] - ue;
return;
}
```

```c
static void exact(double t, double *x, Integer npts, double *u, Nag_Comm *comm)
{
    /* Exact solution (for comparison purposes) */
    double a, b, c;
    double *e = (double *) comm->p;
    Integer i;
    for (i = 0; i < npts; ++i)
    {
        a = (x[i] - 0.25 - 0.75*t)/(e*4.0);
        b = (0.9*x[i] - 0.325 - 0.495*t)/(e*2.0);
        if (a > 0.0 && a > b)
        {
            a = exp(-a);
            c = (0.8*x[i] - 0.4 - 0.24*t)/(e*4.0);
            c = exp(c);
            u[i] = (0.1*c + 0.5 + a)/(c + 1.0 + a);
        }
        else if (b > 0.0 && b >= a)
        {
            b = exp(-b);
            c = (-0.8*x[i] + 0.4 + 0.24*t)/(e*4.0);
            c = exp(c);
            u[i] = (0.5*c + 0.1 + b)/(c + 1.0 + b);
        }
        else
        {
            a = exp(a);
            b = exp(b);
            u[i] = (0.5*a + 1.0 + 0.1*b)/(a + 1.0 + b);
        }
    }
    return;
}
```
static void NAG_CALL monitf(double t, Integer npts, Integer npde,
const double x[], const double u[],
const double r[], double fmon[], Nag_Comm *comm)
{
    double drdx, h;
    Integer i, k, l;
    if (comm->user[3] == -1.0)
    {
        printf("(User-supplied callback monitf, first invocation.)\n");
        comm->user[3] = 0.0;
    }
    for (i = 1; i <= npts-1; ++i)
    {
        k = i-1;
        if (i == 1) k = 1;
        l = i+1;
        h = 0.5* (x[l-1] - x[k-1]);
        /* Second derivative */
        drdx = (R(1, i+1) - R(1, i))/h;
        fmon[i-1] = drdx;
        if (fmon[i-1] < 0) fmon[i-1] = -drdx;
    }
    fmon[npts-1] = fmon[npts-2];
    return;
}

10.2 Program Data
None.

10.3 Program Results

nag_pde_parab_1d_fd_ode_remesh (d03ppc) Example Program Results

Accuracy requirement = 5.000e-05 Number of points = 61
Remeshing every 3 time steps
e = 0.005

(User-supplied callback uvinit, first invocation.)
(User-supplied callback pdedef, first invocation.)
(User-supplied callback monitf, first invocation.)
(User-supplied callback bndary, first invocation.)

<table>
<thead>
<tr>
<th>t</th>
<th>x</th>
<th>0.2000</th>
<th>0.3000</th>
<th>0.4000</th>
<th>0.5000</th>
<th>0.6000</th>
<th>0.7000</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0.3000</td>
<td>0.9968</td>
<td>0.7448</td>
<td>0.4700</td>
<td>0.1667</td>
<td>0.1018</td>
<td></td>
</tr>
<tr>
<td>Approx sol.</td>
<td>0.9967</td>
<td>0.7495</td>
<td>0.4700</td>
<td>0.1672</td>
<td>0.1015</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exact sol.</td>
<td>0.9968</td>
<td>0.7448</td>
<td>0.4700</td>
<td>0.1667</td>
<td>0.1018</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.4000</td>
<td>0.5000</td>
<td>0.6000</td>
<td>0.7000</td>
<td>0.8000</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.4000</td>
<td>1.0003</td>
<td>0.9601</td>
<td>0.4088</td>
<td>0.1154</td>
<td>0.1005</td>
<td></td>
</tr>
<tr>
<td>Approx sol.</td>
<td>0.9997</td>
<td>0.9615</td>
<td>0.4094</td>
<td>0.1157</td>
<td>0.1003</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exact sol.</td>
<td>1.0003</td>
<td>0.9601</td>
<td>0.4088</td>
<td>0.1154</td>
<td>0.1005</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.6000</td>
<td>0.6500</td>
<td>0.7000</td>
<td>0.7500</td>
<td>0.8000</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.6000</td>
<td>0.9966</td>
<td>0.9390</td>
<td>0.3978</td>
<td>0.1264</td>
<td>0.1037</td>
<td></td>
</tr>
<tr>
<td>Approx sol.</td>
<td>0.9964</td>
<td>0.9428</td>
<td>0.4077</td>
<td>0.1270</td>
<td>0.1033</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exact sol.</td>
<td>0.9966</td>
<td>0.9390</td>
<td>0.3978</td>
<td>0.1264</td>
<td>0.1037</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.8000</td>
<td>0.7500</td>
<td>0.8000</td>
<td>0.8500</td>
<td>0.9000</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>0.7000</td>
<td>1.0003</td>
<td>0.9872</td>
<td>0.5450</td>
<td>0.1151</td>
<td>0.1010</td>
<td></td>
</tr>
<tr>
<td>Approx sol.</td>
<td>0.9996</td>
<td>0.9878</td>
<td>0.5695</td>
<td>0.1156</td>
<td>0.1008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exact sol.</td>
<td>1.0003</td>
<td>0.9872</td>
<td>0.5450</td>
<td>0.1151</td>
<td>0.1010</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example Program
Solution of Burgers Equation using Moving Mesh

t = 1.000
x 
Approx sol.  1.0001  0.9961  0.7324  0.1245  0.1004
Exact sol.   0.9999  0.9961  0.7567  0.1273  0.1004

Number of integration steps in time = 205
Number of function evaluations = 4872
Number of Jacobian evaluations = 71
Number of iterations = 518