1 Purpose

The nag_pde_parab_1d_keller_ode_remesh function integrates a system of linear or nonlinear, first-order, time-dependent 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 the Keller box scheme (see Keller (1970)) 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_keller_ode_remesh (Integer npde, double *ts,
               double tout,
               void (*pdedef)(Integer npde, double t, double x, const double u[],
                             const double udot[], const double ux[], Integer ncode,
                             const double v[], const double vdot[], double res[], Integer *ires,
                             Nag_Comm *comm),
               void (*bndary)(Integer npde, double t, Integer ibnd, Integer nobc,
                             const double u[], const double udot[], Integer ncode,
                             const double v[], const double vdot[], double res[], 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 nleft, 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 ucpt[],
                              double r[], 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[], double fmon[], 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

The nag_pde_parab_1d_keller_ode_remesh function integrates the system of first-order PDEs and coupled ODEs given by the master equations:

\[ G_i(x,t,U,U_x,U_t,V,\dot{V}) = 0, \quad i = 1, 2, \ldots, npde, \quad a \leq x \leq b, t \geq t_0, \]  

(1)
In the PDE part of the problem given by (1), the functions $G_i$ must have the general form

$$G_i = \sum_{j=1}^{\text{npde}} P_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} Q_{i,j} V_j + S_i = 0, \quad i = 1, 2, \ldots, \text{npde},$$

where $P_{i,j}$, $Q_{i,j}$ and $S_i$ depend on $x$, $t$, $U$, $U_x$ and $V$.

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 the ODE part given by (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. $U^*$, $U_x^*$ and $U_t^*$ are the functions $U$, $U_x$ and $U_t$ evaluated at these coupling points. Each $R_i$ may only depend linearly on time derivatives. Hence equation (2) may be written more precisely as

$$R = A - B \dot{V} - CU_t^*,$$

where $R = [R_1, \ldots, R_{\text{ncode}}]^T$, $A$ is a vector of length $\text{ncode}$, $B$ is an $\text{ncode}$ by $\text{ncode}$ matrix, $C$ is an $\text{ncode}$ by $(n_\xi \times \text{npde})$ matrix and the entries in $A$, $B$ and $C$ 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 $B$ and $C$. (See Section 5 for the specification of $\text{pdedef}$.)

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 PDE system which is defined by the functions $G_i$ must be specified in $\text{pdedef}$. The initial ($t = t_0$) values of the functions $U(x,t)$ and $V(t)$ must be specified in $\text{uvinit}$. Note that $\text{uvinit}$ will be called again following any 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.

For a first-order system of PDEs, only one boundary condition is required for each PDE component $U_i$. The $\text{npde}$ boundary conditions are separated into $n_a$ at the left-hand boundary $x = a$, and $n_b$ at the right-hand boundary $x = b$ such that $n_a + n_b = \text{npde}$. The position of the boundary condition for each component should be chosen with care; the general rule is that if the characteristic direction of $U_i$ at the left-hand boundary (say) points into the interior of the solution domain, then the boundary condition for $U_i$ should be specified at the left-hand boundary. Incorrect positioning of boundary conditions generally results in initialization or integration difficulties in the underlying time integration functions.

The boundary conditions have the master equation form:

$$G^L_i(x,t,U,U_t,V,\dot{V}) = 0 \quad \text{at } x = a, \quad i = 1, 2, \ldots, n_a,$$

(5)

at the left-hand boundary, and

$$G^R_i(x,t,U,U_t,V,\dot{V}) = 0 \quad \text{at } x = b, \quad i = 1, 2, \ldots, n_b,$$

(6)

at the right-hand boundary.

Note that the functions $G^L_i$ and $G^R_i$ must not depend on $U_x$, since spatial derivatives are not determined explicitly in the Keller box scheme functions. If the problem involves derivative (Neumann) boundary conditions then it is generally possible to restate such boundary conditions in terms of permissible variables. Also note that $G^L_i$ and $G^R_i$ must be linear with respect to time derivatives, so that the boundary
conditions have the general form:

\[ \sum_{j=1}^{\text{npde}} E_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} H_{i,j} V_j + K_i^L = 0, \quad i = 1, 2, \ldots, n_a, \tag{7} \]

at the left-hand boundary, and

\[ \sum_{j=1}^{\text{npde}} E_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} H_{i,j} V_j + K_i^R = 0, \quad i = 1, 2, \ldots, n_b, \tag{8} \]

at the right-hand boundary, where \( E_{i,j}^L, E_{i,j}^R, H_{i,j}^L, H_{i,j}^R, K_i^L \) and \( K_i^R \) depend on \( x, t, U \) and \( V \) only.

The boundary conditions must be specified in `bndary`.

The problem is subject to the following restrictions:

(i) \( P_{i,j}, Q_{i,j} \) and \( S_i \) must not depend on any time derivatives;

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

(iii) The evaluation of the function \( G_i \) is done approximately at the mid-points of the mesh \( x[i-1] \), for \( i = 1, 2, \ldots, \text{npts} \), by calling `pdedef` for each mid-point in turn. Any discontinuities in the function must therefore be at one or more of the fixed mesh points specified by `xfix`;

(iv) At least one of the functions \( P_{i,j} \) must be nonzero so that there is a time derivative present in the PDE problem.

The algebraic-differential equation system which is defined by the functions \( R_i \) must be specified in `odedef`. You must also specify the coupling points \( \xi \) in the array `xi`.

The first-order equations are approximated by a system of ODEs in time for the values of \( U_i \) at mesh points. In this method of lines approach the Keller box scheme is applied to each PDE in the space variable only, resulting in a system of ODEs in time for the values of \( U_i \) at each mesh point. In total there are \( \text{npde} \times \text{npts} + \text{ncode} \) 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 `monitf` which specifies in an analytic or numeric 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 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 along 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. `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: npde – Integer
   
   On entry: the number of PDEs to be solved.
   
   Constraint: npde ≥ 1.

2: ts – double *
   
   On entry: the initial value of the independent variable t.
   
   Constraint: ts < tout.
   
   On exit: the value of t corresponding to the solution values in u. Normally ts = tout.

3: tout – double
   
   On entry: the final value of t to which the integration is to be carried out.

4: pdedef – function, supplied by the user
   
   pdedef must evaluate the functions Gi which define the system of PDEs. pdedef is called approximately midway between each pair of mesh points in turn by nag_pde_parab_1d_keller_ode_remesh (d03prc).

The specification of pdedef is:

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

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

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

3: x – double
   
   On entry: the current value of the space variable x.
4: \( u[npde] \) – const double

\textit{Input}

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

5: \( udot[npde] \) – const double

\textit{Input}

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

6: \( ux[npde] \) – const double

\textit{Input}

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

7: \( ncode \) – Integer

\textit{Input}

On entry: the number of coupled ODEs in the system.

8: \( v[ncode] \) – const double

\textit{Input}

On entry: if \( ncode > 0 \), \( v[i-1] \) contains the value of the component \( V_i(t) \), for \( i = 1,2,\ldots,ncode \).

9: \( vdot[ncode] \) – const double

\textit{Input}

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

10: \( res[npde] \) – double

\textit{Output}

On exit: \( res[i-1] \) must contain the \( i \)th component of \( G \), for \( i = 1,2,\ldots,npde \), where \( G \) is defined as

\[
G_i = \sum_{j=1}^{npde} P_{ij} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{ncode} Q_{ij} \dot{V}_j, \quad (9)
\]

i.e., only terms depending explicitly on time derivatives, or

\[
G_i = \sum_{j=1}^{npde} P_{ij} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{ncode} Q_{ij} \dot{V}_j + S_i, \quad (10)
\]

i.e., all terms in equation (3).

The definition of \( G \) is determined by the input value of \( ires \).

11: \( ires \) – Integer *

\textit{Input/Output}

On entry: the form of \( G_i \) that must be returned in the array \( res \).

\( ires = -1 \)

Equation (9) must be used.

\( ires = 1 \)

Equation (10) must be used.

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 \textit{failcode} = 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_keller_ode_remesh (d03prc) returns to the
calling function with the error indicator set to fail.code = NE_FAILED_DERIV.

12: comm – Nag_Comm *
Pointer to structure of type Nag_Comm; the following members are relevant to pdedef.
user – double *
iuser – Integer *
p – Pointer
The type Pointer will be void *. Before calling
nag_pde_parab_1d_keller_ode_remesh (d03prc) you may allocate memory and
initialize these pointers with various quantities for use by pdedef when called
from nag_pde_parab_1d_keller_ode_remesh (d03prc) (see Section 3.2.1.1 in the
Essential Introduction).

5: bndary – function, supplied by the user
External Function
bndary must evaluate the functions $G_i^L$ and $G_i^R$ which describe the boundary conditions, as given
in (5) and (6).

The specification of bndary is:

```
void bndary (Integer npde, double t, Integer ibnd, Integer nobc,
           const double u[], const double udot[], Integer ncode,
           const double v[], const double vdot[], double res[],
           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: ibnd – Integer
   Input
   On entry: specifies which boundary conditions are to be evaluated.
   ibnd = 0
   bndary must compute the left-hand boundary condition at $x = a$.
   ibnd $\neq$ 0
   bndary must compute of the right-hand boundary condition at $x = b$.

4: nobc – Integer
   Input
   On entry: specifies the number $n_a$ of boundary conditions at the boundary specified by
   ibnd.

5: u[npde] – const double
   Input
   On entry: $u[i-1]$ contains the value of the component $U_i(x,t)$ at the boundary specified
   by ibnd, for $i = 1,2,\ldots,npde$. 


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

7: ncode – Integer

On entry: the number of coupled ODEs in the system.

8: v[ncode] – const double

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

Note: vdot[i-1], for $i = 1, 2, \ldots, \text{ncode}$, may only appear linearly as in (11) and (12).

9: vdot[ncode] – const double

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

Note: vdot[i-1], for $i = 1, 2, \ldots, \text{ncode}$, may only appear linearly as in (11) and (12).

10: res[nobc] – double

On exit: res[i-1] must contain the $i$th component of $G^L$ or $G^R$, depending on the value of ibnd, for $i = 1, 2, \ldots, \text{nobc}$, where $G^L$ is defined as

$$G^L_i = \sum_{j=1}^{\text{npde}} E_{ij} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} H_{ij} \dot{V}_j,$$

(11)

i.e., only terms depending explicitly on time derivatives, or

$$G^L_i = \sum_{j=1}^{\text{npde}} E_{ij} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} H_{ij} \dot{V}_j + K^L_i,$$

(12)

i.e., all terms in equation (7), and similarly for $G^R_i$.

The definitions of $G^L$ and $G^R$ are determined by the input value of ires.

11: ires – Integer *

On entry: the form of $G^L_i$ (or $G^R_i$) that must be returned in the array res.

ires = -1

Equation (11) must be used.

ires = 1

Equation (12) must be used.

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_keller_ode_remesh (d03prc) returns to the calling function with the error indicator set to fail.code = NE_FAILED_DERIV.
12: \texttt{comm} – Nag_Comm *  
Pointer to structure of type Nag_Comm; the following members are relevant to \texttt{bndary}.

\begin{itemize}
  \item \texttt{user} – double *
  \item \texttt{iuser} – Integer *
  \item \texttt{p} – Pointer
\end{itemize}

The type Pointer will be \texttt{void *}. Before calling \texttt{nag_pde_parab_1d_keller_ode_remesh (d03prc)} you may allocate memory and initialize these pointers with various quantities for use by \texttt{bndary} when called from \texttt{nag_pde_parab_1d_keller_ode_remesh (d03prc)} (see Section 3.2.1.1 in the Essential Introduction).

6: \texttt{uvinit} – function, supplied by the user \textit{External Function}

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

The specification of \texttt{uvinit} is:

\begin{verbatim}
void uvinit (Integer npde, Integer npts, Integer nxi,
    const double x[], const double xi[], double u[], Integer ncode,
    double v[], Nag_Comm *comm)
\end{verbatim}

\begin{itemize}
  \item \texttt{npde} – Integer \textit{Input}
    \textbf{On entry:} the number of PDEs in the system.
  \item \texttt{npts} – Integer \textit{Input}
    \textbf{On entry:} the number of mesh points in the interval \([a, b]\).
  \item \texttt{nxi} – Integer \textit{Input}
    \textbf{On entry:} the number of ODE/PDE coupling points.
  \item \texttt{x[npts]} – const double \textit{Input}
    \textbf{On entry:} the current mesh. \texttt{x[i]} contains the value of \(x_i\), for \(i = 1, 2, \ldots, npts\).
  \item \texttt{xi[nxi]} – const double \textit{Input}
    \textbf{On entry:} if \texttt{nxi} > 0, \texttt{xi[i]} contains the ODE/PDE coupling point, \(\xi_i\), for \(i = 1, 2, \ldots, nxi\).
  \item \texttt{u[npde \times npts]} – double \textit{Output}
    \textbf{On exit:} if \texttt{nxi} > 0, \texttt{u[j \times (j - 1) + i - 1]} contains the value of the component \(U_i(x, t_0)\), for \(i = 1, 2, \ldots, npde\) and \(j = 1, 2, \ldots, npts\).
  \item \texttt{ncode} – Integer \textit{Input}
    \textbf{On entry:} the number of coupled ODEs in the system.
  \item \texttt{v[ncode]} – double \textit{Output}
    \textbf{On exit:} if \texttt{ncode} > 0, \texttt{v[i]} contains the value of component \(V_i(t_0)\), for \(i = 1, 2, \ldots, ncode\).
  \item \texttt{comm} – Nag_Comm * \textit{Input}
    Pointer to structure of type Nag_Comm; the following members are relevant to \texttt{uvinit}.
\end{itemize}
### User Defined Functions

- **user** – double *
- **iuser** – Integer *
- **p** – Pointer

The type Pointer will be void *. Before calling 
*nag_pde_parab_1d_keller_ode_remesh (d03pr)* you may allocate memory and
initialize these pointers with various quantities for use by **uvinit** when called from 
*nag_pde_parab_1d_keller_ode_remesh (d03pr)* (see Section 3.2.1.1 in the
Essential Introduction).

### Parameters

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td><code>neqn</code></td>
<td>double</td>
<td>Input/Output</td>
</tr>
</tbody>
</table>
| **On entry:** if `ind` = 1, the value of **u** must be unchanged from the previous call.  
**On exit:** **u[neqn]** = \( (j - 1) + i - 1 \) contains the computed solution \( U_i(x_j,t) \), for \( i = 1, 2, \ldots, npde \) and \( j = 1, 2, \ldots, npts \), evaluated at \( t = ts \). |
| 8 | `npts` | Integer | Input |
| **On entry:** the number of mesh points in the interval \([a,b]\).  
**Constraint:** \( npts \geq 3 \). |
| 9 | `x[npts]` | double | Input/Output |
| **On entry:** the initial mesh points in the space direction. **x[0]** must specify the left-hand boundary, \( a \), and **x[npts] - 1** must specify the right-hand boundary, \( b \).  
**Constraint:** \( x[0] < x[1] < \cdots < x[npts] - 1 \).  
**On exit:** the final values of the mesh points. |
| 10 | `nleft` | Integer | Input |
| **On entry:** the number \( na \) of boundary conditions at the left-hand mesh point **x[0]**.  
**Constraint:** \( 0 \leq nleft \leq npde \). |
| 11 | `ncode` | Integer | Input |
| **On entry:** the number of coupled ODE components.  
**Constraint:** \( ncode \geq 0 \). |
| 12 | `odedef` | function, supplied by the user | External Function |
| **odeodef** must evaluate the functions \( R \), which define the system of ODEs, as given in (4).  
If \( ncode = 0 \), **odeodef** will never be called and the NAG defined null void function pointer,  
NULLFN, can be supplied in the call to 
*nag_pde_parab_1d_keller_ode_remesh (d03pr)*. |

The specification of **odeodef** is:
```
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 ucpt[],
            double r[], Integer *ires, Nag_Comm *comm)
```

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

2: **t** – double  
**On entry:** the current value of the independent variable \( t \).
ncode – Integer

On entry: the number of coupled ODEs in the system.

v[ncode] – const double

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

vdot[ncode] – const double

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

nxi – Integer

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

xi[nxi] – const double

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

ucp[npde \times \text{nxi}] – const double

On entry: if nxi > 0, 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 \( j = 1, 2, \ldots, \text{nxi} \).

ucpx[npde \times \text{nxi}] – const double

On entry: if nxi > 0, 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 \( j = 1, 2, \ldots, \text{nxi} \).

ucpt[npde \times \text{nxi}] – const double

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

r[ncode] – double

On exit: if ncode > 0, r[i - 1] must contain the ith component of \( R \), for \( i = 1, 2, \ldots, \text{ncode} \), where \( R \) is defined as

\[
R = -BV - CU_i^t, 
\]

i.e., only terms depending explicitly on time derivatives, or

\[
R = A - BV - CU_i^t, 
\]

i.e., all terms in equation (4). The definition of \( R \) is determined by the input value of ires.

ires – Integer *

On entry: the form of \( R \) that must be returned in the array r.

ires = -1

Equation (13) must be used.

ires = 1

Equation (14) must be used.
On exit: should usually remain unchanged. However, you may reset 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_keller_ode_remesh (d03prc) returns to the calling function with the error indicator set to fail.code = NE_FAILED_DERIV.

13: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to odedef.

- **user** – double *
- **iuser** – Integer *
- **p** – Pointer

The type Pointer will be void *. Before calling nag_pde_parab_1d_keller_ode_remesh (d03prc) you may allocate memory and initialize these pointers with various quantities for use by odedef when called from nag_pde_parab_1d_keller_ode_remesh (d03prc) (see Section 3.2.1.1 in the Essential Introduction).

13: **nxi** – Integer

*Input*

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

Constraints:

- if ncode = 0, nxi = 0;
- if ncode > 0, nxi ≥ 0.

14: **xi[dim]** – const double

*Input*

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

On entry: xi[i − 1], for i = 1, 2, ..., nxi, must be set to the ODE/PDE coupling points, ξ_i.


15: **neqn** – Integer

*Input*

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

Constraint: neqn = npde × npts + ncode.

16: **rtol[dim]** – const double

*Input*

*Note:* the dimension, dim, of the array rtol must be at least

- 1 when itol = 1 or 2;
- neqn when itol = 3 or 4.

On entry: the relative local error tolerance.

Constraint: rtol[i − 1] ≥ 0.0 for all relevant i.
17: \texttt{atol}[\textit{dim}] -- const double \\
\textbf{Input}

\textbf{Note:} the dimension, \textit{dim}, of the array \texttt{atol} must be at least
1 when \texttt{itol} = 1 or 3;
\texttt{neqn} when \texttt{itol} = 2 or 4.

\textit{On entry:} the absolute local error tolerance.

\textbf{Constraint:} \texttt{atol}[i - 1] \geq 0.0 for all relevant \textit{i}.

\textbf{Note:} corresponding elements of \texttt{rtol} and \texttt{atol} cannot both be 0.0.

18: \texttt{itol} -- Integer \\
\textbf{Input}

A value to indicate the form of the local error test. \texttt{itol} indicates to
\texttt{nag_pde_parab_1d_keller_ode_remesh (d03prc)} whether to interpret either or both of \texttt{rtol} or
\texttt{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:

\textit{On entry:}

\begin{itemize}
  \item \texttt{itol} \quad \texttt{rtol} \quad \texttt{atol} \quad \texttt{w_i}
  \item 1 scalar scalar \texttt{rtol}[0] \times |\texttt{u}[i - 1]| + \texttt{atol}[0]
  \item 2 scalar vector \texttt{rtol}[0] \times |\texttt{u}[i - 1]| + \texttt{atol}[i - 1]
  \item 3 vector scalar \texttt{rtol}[i - 1] \times |\texttt{u}[i - 1]| + \texttt{atol}[0]
  \item 4 vector vector \texttt{rtol}[i - 1] \times |\texttt{u}[i - 1]| + \texttt{atol}[i - 1]
\end{itemize}

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

The choice of norm used is defined by the argument \texttt{norm}.

\textbf{Constraint:} \texttt{itol} = 1, 2, 3 or 4.

19: \texttt{norm} -- Nag_NormType \\
\textbf{Input}

\textit{On entry:} the type of norm to be used.

\begin{itemize}
  \item \texttt{norm} = Nag_MaxNorm \quad Maximum norm.
  \item \texttt{norm} = Nag_TwoNorm \quad Averaged \(L_2\) norm.
\end{itemize}

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

\[ U_{\text{norm}} = \sqrt{\frac{1}{\texttt{neqn}} \sum_{i=1}^{\texttt{neqn}} (U(i)/w_i)^2}, \]

while for the maximum norm

\[ U_{\text{norm}} = \max_i |\texttt{u}[i - 1]/w_i|. \]

See the description of \texttt{itol} for the formulation of the weight vector \(w\).

\textbf{Constraint:} \texttt{norm} = Nag_MaxNorm or Nag_TwoNorm.

20: \texttt{laopt} -- Nag_LinAlgOption \\
\textbf{Input}

\textit{On entry:} the type of matrix algebra required.

\begin{itemize}
  \item \texttt{laopt} = Nag_LinAlgFull \quad Full matrix methods to be used.
\end{itemize}
laopt = Nag_LinAlgBand  
Banded matrix methods to be used.

laopt = Nag_LinAlgSparse  
Sparse matrix methods to be used.

Constraint: laopt = Nag_LinAlgFull, Nag_LinAlgBand or Nag_LinAlgSparse.

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

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 algopt[0] should be set to 0.0. Default values will also be used for any other elements of algopt set to zero. The permissible values, default values, and meanings are as follows:

algopt[0]  
Selects the ODE integration method to be used. If algopt[0] = 1.0, a BDF method is used and if algopt[0] = 2.0, a Theta method is used. The default value is algopt[0] = 1.0.

If algopt[0] = 2.0, then algopt[i - 1], for i = 2, 3, 4, are not used.

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

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 algopt[2] = 1.0 a modified Newton iteration is used and if 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 algopt[2] = 1.0.

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 $P_{ij} = 0.0$, for $j = 1, 2, \ldots, npde$, for some $i$ or when there is no $V_i(t)$ dependence in the coupled ODE system. If algopt[3] = 1.0, then the Petzold test is used. If algopt[3] = 2.0, then the Petzold test is not used. The default value is algopt[3] = 1.0.

If algopt[0] = 1.0, then algopt[i - 1], for i = 5, 6, 7, are not used.

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 algopt[4] = 0.55.

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 algopt[5] = 1.0, a modified Newton iteration is used and if algopt[5] = 2.0, a functional iteration method is used. The default value is algopt[5] = 1.0.

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 algopt[6] = 1.0, then switching is allowed and if algopt[6] = 2.0, then switching is not allowed. The default value is algopt[6] = 1.0.

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 itask. If algopt[0] $\neq 0.0$, a value of 0.0, for algopt[10], say, should be specified even if itask subsequently specifies that $t_{\text{crit}}$ will not be used.
algopt[11] Specifies the minimum absolute step size to be allowed in the time integration. If this option is not required, algopt[11] should be set to 0.0.

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

algopt[13] Specifies the initial step size to be attempted by the integrator. If algopt[13] = 0.0, then the initial step size is calculated internally.

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

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 $\dot{V}$. If algopt[22] = 1.0, a modified Newton iteration is used and if algopt[22] = 2.0, functional iteration is used. The default value is algopt[22] = 1.0.

algopt[28] and algopt[29] are used only for the sparse matrix algebra option, i.e., laopt = Nag_LinAlgSparse.

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 algopt[28] lies outside this range then the default value is used. If the functions regard the Jacobian matrix as numerically singular then increasing algopt[28] towards 1.0 may help, but at the cost of increased fill-in. The default value is algopt[28] = 0.1.

algopt[29] Used as a relative pivot threshold during subsequent Jacobian decompositions (see algopt[28]) below which an internal error is invoked. algopt[29] must be greater than zero, otherwise the default value is used. If 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 algopt[28]). The default value is algopt[29] = 0.0001.

22: remesh – Nag_Boolean

On entry: indicates whether or not spatial remeshing should be performed.

remesh = Nag_TRUE Indicates that spatial remeshing should be performed as specified.

remesh = Nag_FALSE Indicates that spatial remeshing should be suppressed.

Note: remesh should not be changed between consecutive calls to nag_pde_parab_1d_keller_ode_remesh (d03prc). 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

On entry: the number of fixed mesh points.

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

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

24: xfix[ dim] – const double

Note: the dimension, dim, of the array xfix must be at least max(1, nxfix).
On entry: \( x_{\text{fix}}[i - 1] \), for \( i = 1, 2, \ldots, \text{nxfix} \), must contain the value of the \( x \) coordinate at the \( i \)th fixed mesh point.

**Constraint:** \( x_{\text{fix}}[i - 1] < x_{\text{fix}}[i] \), for \( i = 1, 2, \ldots, \text{nxfix} - 1 \), and each fixed mesh point must coincide with a user-supplied initial mesh point, that is \( x_{\text{fix}}[i - 1] = x[j - 1] \) 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: \text{nrmesh} – Integer

**Input**

On entry: indicates the form of meshing to be performed.

- \( \text{nrmesh} < 0 \) indicates that a new mesh is adopted according to the argument \( \text{dxmesh} \). The mesh is tested every \( |\text{nrmesh}| \) timesteps.
- \( \text{nrmesh} = 0 \) indicates that remeshing should take place just once at the end of the first time step reached when \( t > \text{trmesh} \).
- \( \text{nrmesh} > 0 \) indicates that remeshing will take place every \( \text{nrmesh} \) time steps, with no testing using \( \text{dxmesh} \).

**Note:** \( \text{nrmesh} \) may be changed between consecutive calls to \text{nag_pde_parab_1d_keller_ode_remesh (d03prc)} to give greater flexibility over the times of remeshing.

26: \text{dxmesh} – double

**Input**

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

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

or

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

\( \text{dxmesh} \) thus imposes a lower limit on the difference between one mesh and the next.

**Constraint:** \( \text{dxmesh} \geq 0.0 \).

27: \text{trmesh} – double

**Input**

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

**Note:** \( \text{trmesh} \) may be changed between consecutive calls to \text{nag_pde_parab_1d_keller_ode_remesh (d03prc)} to force remeshing at several specified times.

28: \text{ipminf} – Integer

**Input**

On entry: the level of trace information regarding the adaptive remeshing.

- \( \text{ipminf} = 0 \) No trace information.
- \( \text{ipminf} = 1 \) Brief summary of mesh characteristics.
ipminf = 2
More detailed information, including old and new mesh points, mesh sizes and monitor function values.

Constraint: ipminf = 0, 1 or 2.

29: xratio – double

On entry: input bound on 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

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

Suggested value: xratio = 1.5.

Constraint: xratio > 1.0.

30: con – double

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

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

(see Furzeland (1984)). con gives you more control over the mesh distribution e.g., decreasing con allows more clustering. A typical value is \(2/(\text{npts} - 1)\), 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.

Suggested value: con = 2.0/(\text{npts} – 1).

Constraint: 0.1/(\text{npts} – 1) ≤ con ≤ 10.0/(\text{npts} – 1).

31: monitf – function, supplied by the user

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

If ncode = 0, monitf will never be called and the NAG defined null void function pointer, NULLFN, can be supplied in the call to nag_pde_parab_1d_keller_ode_remesh (d03prc).

The specification of monitf is:

void monitf (double t, Integer npts, Integer npde, const double x[],
        const double u[], double fmon[], Nag_Comm *comm)
1: t – double

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

2: npts – Integer

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

3: npde – Integer

On entry: the number of PDEs in the system.

4: x[npts] – const double

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

5: u[npde x npts] – const double

On entry: u[npde x (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 \(j = 1, 2, \ldots, \text{npts}\).
On exit: \( f_{\text{mon}}[i-1] \) must contain the value of the monitor function \( F_{\text{mon}}(x) \) at mesh point \( x = x[i-1] \).

Constraint: \( f_{\text{mon}}[i-1] \geq 0.0 \).

7: \( \text{comm} \) – Nag_Comm *

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

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

The type Pointer will be void *. Before calling nag_pde_parab_1d_keller_ode_remesh (d03prc) you may allocate memory and initialize these pointers with various quantities for use by \( \text{monitf} \) when called from nag_pde_parab_1d_keller_ode_remesh (d03prc) (see Section 3.2.1.1 in the Essential Introduction).

32: \( \text{rsave}\lbrack \text{lrsave} \rbrack \) – double

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

Input

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 \( \text{laopt} = \text{Nag_LinAlgBand}, \text{lrsave} \geq (3 \times m + m + 2) \times \text{neqn} + \text{nwkres} + \text{lenode} \).

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

Where

\( m \) and \( m \) are the lower and upper half bandwidths given by \( m = \text{npde} + \text{nleft} - 1 \) such that \( m = 2 \times \text{npde} - \text{nleft} - 1 \), for problems involving PDEs only; or \( m = \text{neqn} - 1 \), for coupled PDE/ODE problems.

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

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

Note: when using the sparse option, the value of \( \text{lrsave} \) may be too small when supplied to the integrator. An estimate of the minimum size of \( \text{lrsave} \) is printed on the current error message unit if \( \text{itrace} > 0 \) and the function returns with \( \text{fail.code} = \text{NE_INT}_2 \).

34: \( \text{isave}\lbrack \text{lisave} \rbrack \) – Integer

Communication Array

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

If \( \text{ind} = 1 \), \( \text{isave} \) must be unchanged from the previous call to the function because it contains required information about the iteration. In particular the following components of the array \( \text{isave} \) concern the efficiency of the integration:

\( \text{isave}[0] \) 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 evaluating 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.

35: lisave – Integer  
On entry: the dimension of the array isave. Its size depends on the type of matrix algebra selected:

- if laopt = Nag_LinAlgFull, lisave ≥ 25 + nxfix;
- if laopt = Nag_LinAlgBand, lisave ≥ neqn + 25 + nxfix;
- if laopt = Nag_LinAlgSparse, lisave ≥ 25 × 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.

36: itask – Integer  
On entry: the task to be performed by the ODE integrator.

- itask = 1: Normal computation of output values u at t = tout (by overshooting and interpolating).
- itask = 2: Take one step in the time direction and return.
- itask = 3: Stop at first internal integration point at or beyond t = tout.
- itask = 4: Normal computation of output values u at t = tout but without overshooting t = t_crit where 
t_crit is described under the argument algopt.
- itask = 5: Take one step in the time direction and return, without passing t_crit, where t_crit is described 
under the argument algopt.

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

37: itrace – Integer  
On entry: the level of trace information required from nag_pde_parab_1d_keller_ode_remesh 
(d03prc) and the underlying ODE solver as follows:

- itrace ≤ −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 ittrace = 1, except
that the advisory messages are given in greater detail.

itrace ≥ 3
The output from the underlying ODE solver is similar to that produced when ittrace = 2,
except that the advisory messages are given in greater detail.

38: outfile – const char *
Input
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.

39: ind – Integer *
Input/Output
On entry: indicates whether this is a continuation call or a new integration.

    ind = 0
Starts or restarts the integration in time.

    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_keller_ode_remesh (d03prc).

    Constraint: ind = 0 or 1.
On exit: ind = 1.

40: comm – Nag_Comm *
The NAG communication argument (see Section 3.2.1.1 in the Essential Introduction).

41: saved – Nag_D03_Save *
Communication Structure
saved must remain unchanged following a previous call to a Chapter d03 function and prior to
any subsequent call to a Chapter d03 function.

42: fail – NagError *
Input/Output
The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ACC_IN_DOUBT
Integration completed, but small changes in atol or 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 ⟨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 $ires = 3$ in pdedef or bndary.

NE_FAILED_START

$atol$ and $rtol$ were too small to start integration.

NE_FAILED_STEP

Error during Jacobian formulation for ODE system. Increase $itrace$ for further details.

Repeated errors in an attempted step of underlying ODE solver. Integration was successful as far as $ts: ts = \langle value \rangle$.

Underlying ODE solver cannot make further progress from the point $ts$ with the supplied values of $atol$ and $rtol$. $ts = \langle value \rangle$.

NE_INCOMPAT_PARAM

On entry, $con = \langle value \rangle$, $npts = \langle value \rangle$.
Constraint: $con \leq 10.0/(npts - 1)$.

On entry, $con = \langle value \rangle$, $npts = \langle value \rangle$.
Constraint: $con \geq 0.1/(npts - 1)$.

On entry, the point $xfix[I - 1]$ does not coincide with any $x[J - 1]$: $I = \langle value \rangle$ and $xfix[I - 1] = \langle value \rangle$.

NE_INT

$ires$ set to an invalid value in call to pdedef, bndary, or odedef.

On entry, $ind = \langle value \rangle$.
Constraint: $ind = 0$ or 1.

On entry, $ipminf = \langle value \rangle$.
Constraint: $ipminf = 0, 1$ or 2.

On entry, $itask = \langle value \rangle$.
Constraint: $itask = 1, 2, 3, 4$ or 5.

On entry, $itol = \langle value \rangle$.
Constraint: $itol = 1, 2, 3$ or 4.

On entry, $ncode = \langle value \rangle$.
Constraint: $ncode \geq 0$.

On entry, $nleft = \langle value \rangle$.
Constraint: $nleft \geq 0$.

On entry, $npde = \langle value \rangle$.
Constraint: $npde \geq 1$.

On entry, $npts = \langle value \rangle$.
Constraint: $npts \geq 3$.

On entry, $nxfix = \langle value \rangle$.
Constraint: $nxfix \geq 0$.

NE_INT_2

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

On entry, $lisave$ is too small: $lisave = \langle value \rangle$. Minimum possible dimension: $\langle value \rangle$.

On entry, $lrsave$ is too small: $lrsave = \langle value \rangle$. Minimum possible dimension: $\langle value \rangle$. 
On entry, \( \text{ncode} = \langle \text{value} \rangle \) and \( \text{nx} = \langle \text{value} \rangle \).
Constraint: \( \text{nx} = 0 \) when \( \text{ncode} = 0 \).

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

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

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

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

**NE_INT_4**

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

**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 \( \text{itrace} \) for further details.

**NE_ITER_FAIL**

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

**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, \( I = \langle \text{value} \rangle \), \( \text{xfix}[I] = \langle \text{value} \rangle \) and \( \text{xfix}[I - 1] = \langle \text{value} \rangle \).
Constraint: \( \text{xfix}[I] > \text{xfix}[I - 1] \).

On entry, \( I = \langle \text{value} \rangle \), \( \text{xi}[I] = \langle \text{value} \rangle \) and \( \text{xi}[I - 1] = \langle \text{value} \rangle \).
Constraint: \( \text{xi}[I] > \text{xi}[I - 1] \).

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, \( \text{dxmesh} = \langle \text{value} \rangle \).
Constraint: \( \text{dxmesh} \geq 0.0 \).

On entry, \( \text{xratio} = \langle \text{value} \rangle \).
Constraint: \( \text{xratio} > 1.0 \).
NE_REAL_2
On entry, at least one point in $\mathbf{xi}$ lies outside $[\mathbf{x}[0], \mathbf{x}[\text{npts} - 1]]$: $\mathbf{x}[0] = \langle\text{value}\rangle$ and $\mathbf{x}[\text{npts} - 1] = \langle\text{value}\rangle$.

On entry, $\text{tout} = \langle\text{value}\rangle$ and $\text{ts} = \langle\text{value}\rangle$.
Constraint: $\text{tout} > \text{ts}$.

On entry, $\text{tout} - \text{ts}$ is too small: $\text{tout} = \langle\text{value}\rangle$ and $\text{ts} = \langle\text{value}\rangle$.

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

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

NE_REMESH_CHANGED
renmesh has been changed between calls to nag_pde_parab_1d_keller_ode_remesh (d03prc).

NE_SING_JAC
Singular Jacobian of ODE system. Check problem formulation.

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

NE_ZERO_WTS
Zero error weights encountered during time integration.

7 Accuracy
nag_pde_parab_1d_keller_ode_remesh (d03prc) 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_keller_ode_remesh (d03prc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_pde_parab_1d_keller_ode_remesh (d03prc) 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 Keller box scheme can be used to solve higher-order problems which have been reduced to first-order by the introduction of new variables (see the example in Section 10). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite difference scheme (nag_pde_parab_1d_fd_ode_remesh (d03ppc) for example), but at the expense of increased CPU time due to the larger number of function evaluations required.
It should be noted that the Keller box scheme, in common with other central-difference schemes, may be
unsuitable for some hyperbolic first-order problems such as the apparently simple linear advection
equation $U_t + aU_x = 0$, where $a$ is a constant, resulting in spurious oscillations due to the lack of
dissipation. This type of problem requires a discretization scheme with upwind weighting (nag_pde_parab_1d_cd_ode_remesh (d03psc) for example), or the addition of a second-order artificial
dissipation term.

The time taken depends on the complexity of the 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 is the first-order system
\[
\frac{\partial U_1}{\partial t} + \frac{\partial U_1}{\partial x} + \frac{\partial U_2}{\partial x} = 0,
\]
\[
\frac{\partial U_2}{\partial t} + 4\frac{\partial U_1}{\partial x} + \frac{\partial U_2}{\partial x} = 0,
\]
for $x \in [0, 1]$ and $t \geq 0$.

The initial conditions are
\[
U_1(x, 0) = e^x,
\]
\[
U_2(x, 0) = x^2 + \sin(2\pi x^2),
\]
and the Dirichlet boundary conditions for $U_1$ at $x = 0$ and $U_2$ at $x = 1$ are given by the exact solution:
\[
U_1(x, t) = \frac{1}{2}\left\{e^{xt} + e^{x-3t}\right\} + \frac{1}{4}\left\{\sin\left(2\pi(x-3t)^2\right) - \sin\left(2\pi(x+t)^2\right)\right\} + 2t^2 - 2xt,
\]
\[
U_2(x, t) = e^{x-3t} - e^{xt} + \frac{1}{2}\left\{\sin\left(2\pi(x-3t)^2\right) + \sin\left(2\pi(x+t)^2\right)\right\} + x^2 + 5t^2 - 2xt.
\]

10.1 Program Text

/* nag_pde_parab_1d_keller_ode_remesh (d03prc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group.
 * * Mark 7, 2001.
 */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nagd03.h>
#include <nagx01.h>

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL pdedef(Integer, double, double, const double[],
const double[], const double[], Integer, const double[], const double[], double[],
Integer *, Nag_Comm *);
static void NAG_CALL bndary(Integer, double, Integer, Integer, const double[],
const double[], Integer, const double[],
const double[], double[], Integer *, Nag_Comm *);
static void NAG_CALL uvinit(Integer, Integer, Integer, const double[],
const double[], double[], Integer, double[],
Nag_Comm *);

#ifdef __cplusplus
}
#endif
static void NAG_CALL monitf(double, Integer, Integer, const double[],
        const double[], double[], Nag_Comm *);
#endif
static void exact(double, Integer, Integer, double *, double *);
#define UE(I, J) ue[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 = 2, npts = 61, ncode = 0, nxi = 0, nxfix = 0, nleft = 1;
    const Integer itype = 1, intpts = 5, neqn = npde*npts+ncode;
    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, tout, trmesh, ts, xratio;
    Integer exit_status = 0, 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);
    printf("nag_pde_parab_1d_keller_ode_remesh (d03prc) Example Program"
           " Results\n\n");
    /* For communication with user-supplied functions: */
    comm.user = ruser;
    /* Allocate memory */
    if (!(algopt = NAG_ALLOC(30, double)) ||
        !(atol = NAG_ALLOC(1, double)) ||
        !(rsave = NAG_ALLOC(lrsave, double)) ||
        !(rtol = NAG_ALLOC(1, double)) ||
        !(u = NAG_ALLOC(npde*npts, double)) ||
        !(ue = NAG_ALLOC(npde*npts, double)) ||
        !(uout = NAG_ALLOC(npde*intpts*itype, double)) ||
        !(x = NAG_ALLOC(npts, double)) ||
        !(xfix = NAG_ALLOC(1, double)) ||
        !(xi = NAG_ALLOC(1, double)) ||
        !(xout = NAG_ALLOC(intpts, double)) ||
        !(isave = NAG_ALLOC(lisave, Integer)))
    {
        printf("Allocation failure\n\n");
        exit_status = 1;
        goto END;
    }
    itrace = 0;
    itol = 1;
    atol[0] = 5.0e-5;
    rtol[0] = atol[0];
    printf(" Accuracy requirement =\$12.3e", atol[0]);
    printf(" Number of points = %3"NAG_IFMT", npts);
    /* Set remesh parameters */
    remesh = Nag_TRUE;
    nrmesh = 3;

int main(void)
dxmesh = 0.0;
trmesh = 0.0;
con = 5.0/(npts-1.0);
xratio = 1.2;
ipinf = 0;
printf(" Remeshing every %3"NAG_IFMT" time steps\n\n", nrmesh);

/* Initialise mesh */
for (i = 0; i < npts; ++i) x[i] = i/(npts-1.0);

/* 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)
{
algopt[0] = 2.0;
algopt[5] = 2.0;
algopt[6] = 1.0;
}

/* Loop over output value of t */
ts = 0.0;
tout = 0.0;
for (it = 0; it < 5; ++it)
{
tout = 0.05*(it+1);

if (fail.code != NE_NOERROR)
{
printf("Error from "
"nag_pde_parab_1d_keller_ode_remesh (d03prc).\n\n", fail.message);
exit_status = 1;
goto END;
}
/* Interpolate at output points */

/* nag_pde_interp_1d_fd (d03pzc). PDEs, spatial interpolation with
 * nag_pde_parab_1d_keller_ode_remesh (d03prc).
 * 
 * nag_pde_interp_1d_fd(npde, 0, 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 */

effect(ts, npde, intpts, xout, ue);

printf(" t = %6.3f\n", ts);
printf(" Approx u1");
for (i = 1; i <= intpts; ++i)
{
    printf("%10.4f", UOUT(1, i, 1));
    printf(i%5 == 0 || i == 5?"\n":"");
}

printf(" Exact u1");
for (i = 1; i <= 5; ++i)
{
    printf("%10.4f", UE(1, i));
    printf(i%5 == 0 || i == 5?"\n":"");
}

printf(" Approx u2");
for (i = 1; i <= 5; ++i)
{
    printf("%10.4f", UOUT(2, i, 1));
    printf(i%5 == 0 || i == 5?"\n":"");
}

printf(" Exact u2");
for (i = 1; i <= 5; ++i)
{
    printf("%10.4f", UE(2, i));
    printf(i%5 == 0 || i == 5?"\n":"");
}

printf("\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(u);
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)
{  
  Integer i;
  if (comm->user[0] == -1.0)
  {   
    printf("(User-supplied callback uvinit, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  for (i = 1; i <= npts; ++i)
  {   
    U(1, i) = exp(x[i-1]);
    U(2, i) = x[i-1]*x[i-1] + sin(2.0*nag_pi*(x[i-1]*x[i-1]));
  }
  return;
}

static void NAG_CALL pdedef(Integer npde, double t, double x, const double u[],
const double udot[], const double ux[],
Integer ncode, const double v[],
const double vdot[], double res[], Integer *ires,
Nag_Comm *comm)
{  
  if (comm->user[1] == -1.0)
  {   
    printf("(User-supplied callback pdedef, first invocation.)\n");
    comm->user[1] = 0.0;
  }
  if (*ires == -1)
  {   
    res[0] = udot[0];
    res[1] = udot[1];
  }
  else
  {   
    res[0] = udot[0] + ux[0] + ux[1];
    res[1] = udot[1] + 4.0*ux[0] + ux[1];
  }
  return;
}

static void NAG_CALL bndary(Integer npde, double t, Integer ibnd, Integer nobc,
const double u[], const double udot[],
Integer ncode, const double v[],
const double vdot[], double res[], Integer *ires,
Nag_Comm *comm)
{  
  double pp;
  if (comm->user[2] == -1.0)
  {   
    printf("(User-supplied callback bndary, first invocation.)\n");
    comm->user[2] = 0.0;
  }
  pp = 2.0*nag_pi;
  if (ibnd == 0)
if (*ires == -1)
    { 
      res[0] = 0.0;
    } 
else 
    {
      res[0] = u[0] - 0.5*(exp(t) + exp(-3.0*t)) - 0.25*(sin(9.0*pp*t*t) - sin(pp*t*t)) - 2.0*t*t;
    }
} 
else 
    {
      if (*ires == -1)
      { 
        res[0] = 0.0;
      } 
    else 
      {
        res[0] = u[1] - (exp(1.0-3.0*t) - exp(1.0+t) + 0.5*sin(pp*(1.0-3.0*t)*(1.0-3.0*t)) + 0.5*sin(pp*(1.0+t)*(1.0+t)) + 1.0 + 5.0*t*t - 2.0*t);
      }
    }
return;
}

static void NAG_CALL monitf(double t, Integer npts, Integer npde, 
const double x[], const double u[], double fmon[], Nag_Comm *comm)
{
  double d2x1, d2x2, h1, h2, h3;
  Integer i;
  if (comm->user[3] == -1.0)
    {
      printf("(User-supplied callback monitf, first invocation.)\n");
      comm->user[3] = 0.0;
    }
  for (i = 2; i <= npts-1; ++i)
    {
      h1 = x[i - 1] - x[i - 2];
      h2 = x[i] - x[i - 1];
      h3 = 0.5* (x[i] - x[i - 2]);
      /* Second derivatives */
      d2x1 = fabs(((U(1, i+1)-U(1, i))/h2-(U(1, i)-U(1, i-1))/h1)/h3);
      d2x2 = fabs(((U(2, i+1)-U(2, i))/h2-(U(2, i)-U(2, i-1))/h1)/h3);
      fmon[i-1] = d2x1;
      if (d2x2 > d2x1) fmon[i-1] = d2x2;
      fmon[0] = fmon[1];
      fmon[npts- 1] = fmon[npts-2];
    }
  return;
}

static void exact(double t, Integer npde, Integer npts, double *x, double *u)
{
  /* Exact solution (for comparison purposes) */
  double pp;
  Integer i;
  pp = 2.0*nag_pi;
  for (i = 1; i <= npts; ++i)
    {
      U(1, i) = 0.5* (exp(x[i-1]+t) + exp(x[i-1]-3.0*t)) + 0.25*(sin(pp*(x[i-1]-3.0*t)*(x[i-1]-3.0*t)) -
\[
\sin(pp*(x[i-1]+t)*(x[i-1]+t))) + \\
2.0* t^2 - 2.0*x[i-1]*t;
\]

\[U(2, i) = \exp(x[i-1]-3.0*t) - \exp(x[i-1]+t) + \\
0.5*(\sin(pp*((x[i-1]-3.0*t)*(x[i-1]-3.0*t))) + \\
\sin(pp*((x[i-1]+t)*(x[i-1]+t)))) + \\
x[i-1]*x[i-1] + 5.0*t^2 - 2.0*x[i-1]*t;\]

10.2 Program Data

None.

10.3 Program Results

nag_pde_parab_1d_keller_ode_remesh (d03prc) Example Program Results

Accuracy requirement = 5.000e-05 Number of points = 61

Remeshing every 3 time steps

<table>
<thead>
<tr>
<th>t</th>
<th>x</th>
<th>0.0000</th>
<th>0.2500</th>
<th>0.5000</th>
<th>0.7500</th>
<th>1.0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 0.050</td>
<td></td>
<td>0.9923</td>
<td>1.0894</td>
<td>1.4686</td>
<td>2.3388</td>
<td>2.1071</td>
</tr>
<tr>
<td>Approx u1</td>
<td></td>
<td>1.0963</td>
<td>0.9856</td>
<td>1.3120</td>
<td>2.3084</td>
<td>2.1039</td>
</tr>
<tr>
<td>Exact u1</td>
<td></td>
<td>-0.0150</td>
<td>0.1057</td>
<td>0.7180</td>
<td>0.0967</td>
<td>0.2021</td>
</tr>
<tr>
<td>Approx u2</td>
<td></td>
<td>-0.0150</td>
<td>0.1046</td>
<td>0.7193</td>
<td>0.0966</td>
<td>0.2022</td>
</tr>
</tbody>
</table>

| t = 0.100   |     | 1.0613 | 0.9851 | 1.3113 | 2.3092 | 2.1025 |
| Approx u1   |     | 1.1485 | 0.9764 | 1.2654 | 2.0911 | 2.2027 |
| Exact u1    |     | 0.1370 | -0.0250 | -0.4107 | -0.8577 | 0.3096 |
| Approx u2   |     | 0.1366 | -0.0266 | -0.4100 | -0.8567 | 0.3096 |

| t = 0.150   |     | 1.0956 | 1.0529 | 1.3407 | 1.8322 | 2.2035 |
| Approx u1   |     | 1.0956 | 1.0515 | 1.3393 | 1.8327 | 2.2050 |
| Exact u1    |     | 0.0381 | 0.1282 | -0.7979 | -1.1776 | -0.4221 |
| Approx u2   |     | 0.0370 | 0.1247 | -0.7961 | -1.1784 | -0.4221 |

| t = 0.200   |     | 0.9119 | 1.1288 | 1.5163 | 1.6076 | 2.2027 |
| Approx u1   |     | 0.8119 | 1.1276 | 1.5142 | 1.6091 | 2.2035 |
| Exact u1    |     | -0.4968 | 0.2123 | -1.0259 | -1.2149 | -1.3938 |
| Approx u2   |     | -0.4992 | 0.2078 | -1.0257 | -1.2183 | -1.3938 |

Number of integration steps in time = 50
Number of function evaluations = 2579
Number of Jacobian evaluations = 20
Number of iterations = 126
Example Program
Solution of First-order System using Moving Mesh
$U(1,x,t)$

Solution of First-order System using Moving Mesh
$U(2,x,t)$