NAG C Library Function Document

nag_pde_parab_1d_keller_ode (d03pkc)

1 Purpose

nag_pde_parab_1d_keller_ode (d03pkc) 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). The spatial discretisation is performed using the Keller box scheme 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
void nag_pde_parab_1d_keller_ode (Integer npde, double *ts, double tout,
    void (*pdedef)(Integer npde, double t, double x, const double *u[],
                       const double *ut[], const double *ux[], Integer ncode, const double *v[],
                       const double vdot[], Integer *ires, Nag_Comm *comm),
    void (*bndary)(Integer npde, double t, Integer ibnd, Integer nobc, const double *u[],
                        const double *ut[], Integer ncode, const double *v[],
                        const double vdot[], Integer *ires, Nag_Comm *comm),
    double *u[], Integer npts, const 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 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[], double *rsave[], Integer *isave[], Integer *ind,
    Nag_Comm *comm, Nag_D03_Save *saved, NagError *fail)
```

3 Description

nag_pde_parab_1d_keller_ode (d03pkc) integrates the system of first-order PDEs and coupled ODEs

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

\[
F_i(t, V, \dot{V}, \xi, U^*, U_x^*, U_t^*) = 0, \quad i = 1, 2, \ldots, \text{ncode}. \quad (2)
\]

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

\[
G_i = \sum_{j=1}^{\text{npde}} P_{ij} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} Q_{ij} \dot{V}_j + R_i = 0, \quad i = 1, 2, \ldots, \text{npde}, \quad (3)
\]

where \(P_{ij}, Q_{ij}\) and \(R_i\) 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 $n_c$ 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 $F_i$ may only depend linearly on time derivatives. Hence equation (2) may be written more precisely as

$$F = A - BV - CU^*_t,$$

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

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 user-defined mesh $x_1, x_2, \ldots, x_{\text{npts}}$.

The PDE system which is defined by the functions $G_i$ must be specified in the user-supplied function $\text{pdedef}$. The initial values of the functions $U(x, t)$ and $V(t)$ must be given at $t = t_0$.

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 initialisation or integration difficulties in the underlying time integration functions.

The boundary conditions have the form:

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

at the left-hand boundary, and

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

at the right-hand boundary.

Note that the functions $G_i^L$ and $G_i^R$ must not depend on $U_x$, since spatial derivatives are not determined explicitly in the Keller box scheme. 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 $G_i^L$ and $G_i^R$ must be linear with respect to time derivatives, so that the boundary conditions have the general form:

$$\sum_{j=1}^{\text{npde}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} H_{i,j}^L \dot{V}_j + S_i^L = 0, \quad i = 1, 2, \ldots, n_a,$$

at the left-hand boundary, and

$$\sum_{j=1}^{\text{npde}} E_{i,j}^R \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} H_{i,j}^R \dot{V}_j + S_i^R = 0, \quad i = 1, 2, \ldots, n_b,$$

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

The boundary conditions must be specified in a function $\text{bndary}$ provided by the user.

The problem is subject to the following restrictions:

(i) $P_{i,j}$, $Q_{i,j}$ and $R_i$ 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 the function $\text{pdedef}$ for each mid-point in turn. Any discontinuities in the function must therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\text{npts}}$.
At least one of the functions $P_{i,j}$ must be non-zero so that there is a time derivative present in the PDE problem.

The algebraic-differential equation system which is defined by the functions $F_i$ must be specified in the user-supplied function `pdedef`. The user must also specify the coupling points $\xi$ in the array `xi`.

The parabolic 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 (Keller (1970)) 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 $npde \times npts + ncode$ ODEs in time direction. This system is then integrated forwards in time using a Backward Differentiation Formula (BDF) or a Theta method.

4 References


5 Parameters

1: `npde` – Integer

   *Input*

   *On entry:* the number of PDEs to be solved.

   *Constraint:* $npde \geq 1$.

2: `ts` – double *

   *Input/Output*

   *On entry:* the initial value of the independent variable $t$.

   *Constraint:* $ts < tout$.

   *On exit:* the value of $t$ corresponding to the solution in $u$. Normally $ts = tout$.

3: `tout` – double

   *Input*

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

4: `pdedef`

   *Function*

   `pdedef` must evaluate the functions $G_i$ 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` (d03pkc).

   Its specification is:

   ```c
   void pdedef (Integer npde, double t, double x, const double u[], const double ut[],
               const double ux[], 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: \( x \) – double 

*Input*

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

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

*Input*

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

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

*Input*

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

7: \( \text{ncode} \) – Integer 

*Input*

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

8: \( v[\text{ncode}] \) – const double 

*Input*

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

9: \( v[\text{ncode}] \) – const double 

*Input*

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

10: \( \text{res}[\text{npde}] \) – double 

*Output*

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

\[
G_i = \sum_{j=1}^{\text{npde}} P_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} Q_{i,j} \dot{V}_j, \tag{9}
\]

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

\[
G_i = \sum_{j=1}^{\text{npde}} P_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} Q_{i,j} \dot{V}_j + R_i, \tag{10}
\]

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

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

11: \( \text{ires} \) – Integer * 

*Input/Output*

*On entry:* the form of \( G_i \) that must be returned in the array \( \text{res} \). If \( \text{ires} = -1 \), then equation (9) above must be used. If \( \text{ires} = 1 \), then equation (10) above must be used.

*On exit:* should usually remain unchanged. However, the user may set \( \text{ires} \) 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. The user may wish to set \( \text{ires} = 3 \) when a
physically meaningless input or output value has been generated. If the user
consecutively sets $ires = 3$, then nag_pde_parab_1d_keller_ode (d03pkc) returns to
the calling function with the error indicator set to fail.code = NE_FAILED_DERIV.

12: **comm** – NAG_Comm *  
   Input/Output
   The NAG communication parameter (see the Essential Introduction).

---

**Function**

**bndary** must evaluate the functions $G_i^L$ and $G_i^R$ which describe the boundary conditions, as given in
(5) and (6).

Its specification is:

```c
void bndary (Integer *npde, double *t, Integer *ibnd, Integer *nobc, const double (*u)[],
const double (*ut)[], 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. If $ibnd = 0$, then
   **bndary** must compute the left-hand boundary condition at $x = a$. If $ibnd \neq 0$, then
   **bndary** must compute the right-hand boundary condition at $x = b$.

4: **nobc** – Integer  
   Input  
   On entry: **nobc** specifies the number 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 $i = 1, 2, \ldots, npde$.

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

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

8: **v[ncode]** – const double  
   Input  
   On entry: $v[i - 1]$ contains the value of component $V_i(t)$, for $i = 1, 2, \ldots, ncode$.

9: **vdot[ncode]** – const double  
   Input  
   On entry: $vdot[i - 1]$ contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots, ncode$.
   Note: $vdot[i - 1]$, for $i = 1, 2, \ldots, ncode$, may only appear linearly as in (7) and (8).
On exit: \( \text{res}[i - 1] \) must contain the \( i \)th component of \( G^L \) or \( G^R \), depending on the value of \( \text{ibnd} \), for \( i = 1, 2, \ldots, \text{nobc} \), where \( G^L \) is defined as

\[
G^L_i = \sum_{j=1}^{\text{npde}} E^L_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} H^L_{i,j} V_j,
\]

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

\[
G^L_i = \sum_{j=1}^{\text{npde}} E^L_{i,j} \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{ncode}} H^L_{i,j} V_j + S^L_i,
\]

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 \( \text{ires} \).

**ires** – Integer *

On entry: the form of \( G^L_i \) (or \( G^R_i \)) that must be returned in the array \( \text{res} \). If \( \text{ires} = -1 \), then equation (11) above must be used. If \( \text{ires} = 1 \), then equation (12) above must be used.

On exit: should usually remain unchanged. However, the user may set \( \text{ires} \) 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 \( \text{fail.code} = \text{NE_USER_STOP} \).

**ires** = 3

Indicates to the integrator that the current time step should be abandoned and a smaller time step used instead. The user may wish to set \( \text{ires} = 3 \) when a physically meaningless input or output value has been generated. If the user consecutively sets \( \text{ires} = 3 \), then \( \text{nag_pde_parab_1d_keller_ode} \) (d03pkc) returns to the calling function with the error indicator set to \( \text{fail.code} = \text{NE_FAILED_DERIV} \).

**comm** – NAG_Comm *

The NAG communication parameter (see the Essential Introduction).
8: \( \mathbf{x}[\text{npts}] \) – const double  
*Input*

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

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

9: \( \text{nleft} \) – Integer  
*Input*

*On entry:* the number \( n_a \) of boundary conditions at the left-hand mesh point \( \mathbf{x}[0] \).

*Constraint:* \( 0 \leq \text{nleft} \leq \text{npde} \).

10: \( \text{ncode} \) – Integer  
*Input*

*On entry:* the number of coupled ODE components.

*Constraint:* \( \text{ncode} \geq 0 \).

11: \text{odedef}  
*Function*

\text{odedef} must evaluate the functions \( F \), which define the system of ODEs, as given in (4). If the user wishes to compute the solution of a system of PDEs only (i.e., \( \text{ncode} = 0 \)), \text{odedef} must be the dummy function \text{d03pek}. (\text{d03pek} is included in the NAG C Library; however, its name may be implementation-dependent: see the Users’ Note for your implementation for details.)

Its specification is:

```c
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 ucp[],
            double f[], Integer *ires,
            Nag_Comm *comm)
```

1: \( \text{npde} \) – 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: \( \text{ncode} \) – Integer  
*Input*

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

4: \( v[\text{ncode}] \) – const double  
*Input*

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

5: \( vdot[\text{ncode}] \) – const double  
*Input*

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

6: \( \text{nxi} \) – Integer  
*Input*

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

7: \( \text{xi}[\text{nxi}] \) – const double  
*Input*

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

8: \( \text{ucp}[\text{npde} \times \text{nxi}] \) – const double  
*Input*

*Note:* where \( \text{UCP}(i,j) \) appears in this document it refers to the array element \( \text{ucp}[\text{npde} \times (j - 1) + i - 1] \). We recommend using a \#define to make the same definition in your calling program.
On entry: $UCP(i,j)$ contains the value of $U_i(x,t)$ at the coupling point $x = \xi_j$, for $i = 1,2,\ldots,\text{npde}; j = 1,2,\ldots,\text{nxi}$.

9: \texttt{ucp}[\text{npde} \times \text{nxi}] \rightarrow \text{const double} \quad \text{Input}

Note: where $UCP(i,j)$ appears in this document it refers to the array element $\text{ucp}[\text{npde} \times (j-1) + i-1]$. We recommend using a \#define to make the same definition in your calling program.

On entry: $UCP(i,j)$ contains the value of $(\partial U_i(x,t))/(\partial x)$ at the coupling point $x = \xi_j$, for $i = 1,2,\ldots,\text{npde}; j = 1,2,\ldots,\text{nxi}$.

10: \texttt{ucpt}[\text{npde} \times \text{nxi}] \rightarrow \text{const double} \quad \text{Input}

Note: where $UCPT(i,j)$ appears in this document it refers to the array element $\text{ucpt}[\text{npde} \times (j-1) + i-1]$. We recommend using a \#define to make the same definition in your calling program.

On entry: $UCPT(i,j)$ contains the value of $(\partial U_i)/(\partial t)$ at the coupling point $x = \xi_j$, for $i = 1,2,\ldots,\text{npde}; j = 1,2,\ldots,\text{nxi}$.

11: \texttt{f}[\text{ncode}] \rightarrow \text{double} \quad \text{Output}

On exit: $f[i-1]$ must contain the $i$th component of $f$, for $i = 1,2,\ldots,\text{ncode}$, where $f$ is defined as

\[ F = -B\dot{V} - CU_t^*, \]

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

\[ F = A - B\dot{V} - CU_t^*, \]

i.e., all terms in equation (4). The definition of $f$ is determined by the input value of $\text{ires}$.

12: \texttt{ires} \rightarrow \text{Integer} \quad \text{Input/Output}

On entry: the form of $f$ that must be returned in the array $f$. If $\text{ires} = -1$, then equation (13) above must be used. If $\text{ires} = 1$, then equation (14) above must be used.

On exit: should usually remain unchanged. However, the user 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 \texttt{fail.code} = \texttt{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. The user may wish to set $\text{ires} = 3$ when a physically meaningless input or output value has been generated. If the user consecutively sets $\text{ires} = 3$, then nag_pde_parab_1d_keller_ode (d03pkc) returns to the calling function with the error indicator set to \texttt{fail.code} = \texttt{NE_FAILED_DERIV}.

13: \texttt{comm} \rightarrow \text{NAG_Comm} \quad \text{Input/Output}

The NAG communication parameter (see the Essential Introduction).

12: \texttt{nxi} \rightarrow \text{Integer} \quad \text{Input}

Constraints:

- if $\text{ncode} = 0$, $\text{nxi} = 0$;
- if $\text{ncode} > 0$, $\text{nxi} \geq 0$. 
13: \( \text{xi[dim]} \) – const double  
\text{Input}

\text{Note:}\ the\ dimension,\ \text{dim},\ of\ the\ array\ \text{xi}\ must\ be\ at\ least\ \max(1,\ \text{nxi}).

\text{On\ entry:}\ \text{xi}[i - 1],\ i = 1, 2, \ldots, \text{nxi},\ must\ be\ set\ to\ the\ ODE/PDE\ coupling\ points,\ \xi_i.  
\text{Constraint:}\ 0 \leq \text{xi}[0] < \text{xi}[1] < \cdots < \text{xi}[\text{nxi} - 1] \leq \text{x[npts} - 1].

14: \text{n}e\text{qn} – Integer  
\text{Input}

\text{On\ entry:}\ \text{the\ number\ of\ ODEs\ in\ the\ time\ direction.}  
\text{Constraint:}\ \text{n}e\text{qn} = \text{n}p\text{de} \times \text{n}pts + \text{nc}ode.

15: \text{rtol[dim]} – const double  
\text{Input}

\text{Note:}\ the\ dimension,\ \text{dim},\ of\ the\ array\ \text{rtol}\ must\ be\ at\ least\ 1\ when\ \text{itol} = 1\ or\ 2\ and\ at\ least\ \text{neqn}\ when\ \text{itol} = 3\ or\ 4.

\text{On\ entry:}\ \text{the\ relative\ local\ error\ tolerance.}  
\text{Constraint:}\ \text{rtol}[i - 1] \geq 0\ for\ all\ relevant\ i.

16: \text{atol[dim]} – const double  
\text{Input}

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

\text{On\ entry:}\ \text{the\ absolute\ local\ error\ tolerance.}  
\text{Constraint:}\ \text{atol}[i - 1] \geq 0\ for\ all\ relevant\ i.

17: \text{itol} – Integer  
\text{Input}

\text{On\ entry:}\ a\ value\ to\ indicate\ the\ form\ of\ the\ local\ error\ test.\ \text{itol}\ indicates\ to\ \text{nag_pde_parab_1d_keller_ode} (d03pkc)\ 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:}  

\[
\begin{array}{ccc}
\text{itol} & \text{rtol} & \text{atol} & \text{w}_i \\
1 & \text{scalar} & \text{scalar} & \text{rtol}[0] \times |u[i - 1]| + \text{atol}[0] \\
2 & \text{scalar} & \text{vector} & \text{rtol}[0] \times |u[i - 1]| + \text{atol}[i - 1] \\
3 & \text{vector} & \text{scalar} & \text{rtol}[i - 1] \times |u[i - 1]| + \text{atol}[0] \\
4 & \text{vector} & \text{vector} & \text{rtol}[i - 1] \times |u[i - 1]| + \text{atol}[i - 1]
\end{array}
\]

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

\text{The\ choice\ of\ norm\ used\ is\ defined\ by\ the\ parameter\ \text{norm},\ see\ below.}  
\text{Constraint:}\ 1 \leq \text{itol} \leq 4.

18: \text{norm} – Nag_NormType  
\text{Input}

\text{On\ entry:}\ the\ type\ of\ norm\ to\ be\ used.\ Two\ options\ are\ available:}  
\text{norm = Nag_MaxNorm}  
\text{Maximum\ norm.}  
\text{norm = Nag_TwoNorm}  
\text{Averaged\ L}_2\ \text{norm.}

\text{If}\ u_{\text{norm}}\ denotes\ the\ norm\ of\ the\ vector}\ u\ \text{of\ length}\ \text{neqn},\ \text{then\ for\ the\ averaged\ L}_2\ \text{norm}\ 

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

\text{while\ for\ the\ maximum\ norm}
\[ u_{norm} = \max_i |u[i - 1]/w_i|. \]

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

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

19: \( \text{laopt} = \text{Nag\_LinAlgOption} \)  

*Input*

*On entry:* the type of matrix algebra required. The possible choices are:

- \( \text{laopt} = \text{Nag\_LinAlgFull} \)
  
  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:** the user is recommended to use the banded option when no coupled ODEs are present (i.e., \( \text{ncode} = 0 \)).

20: \( \text{algopt}[30] = \text{const double} \)  

*Input*

*On entry:* \( \text{algopt} \) may be set to control various options available in the integrator. If the user wishes 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 \( \text{algopt}[0] = 1.0 \).

- If \( \text{algopt}[0] = 2.0 \), then \( \text{algopt}[i] \), for \( i = 1, 2, 3 \) are not used.

- \( \text{algopt}[1] \) 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 \( \dot{P}_{i,j} = 0.0 \), for \( j = 1, 2, \ldots, \text{npde} \) for some \( i \) or when there is no \( \dot{V}_i(t) \) 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 \( \text{algopt}[3] = 1.0 \).

- If \( \text{algopt}[0] = 1.0 \), then \( \text{algopt}[i] \), for \( i = 4, 5, 6 \) are not used.

- \( \text{algopt}[4] \) 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 \( \text{algopt}[6] = 1.0 \).
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 parameter itask. If \( \text{algopt}[0] \neq 0.0 \), a value of 0.0 for \( \text{algopt}[10] \), 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, \( \text{algopt}[11] \) 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, \( \text{algopt}[12] \) should be set to 0.0.

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.

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.

algopt[22] specifies what method is to be used to solve the nonlinear equations at the initial point to initialise 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 \( \text{algopt}[22] = 1.0 \).

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

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 \( \text{algopt}[28] \) 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 \( \text{algopt}[28] = 0.1 \).

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

21: \( \text{rsave}[\text{lrsave}] \) – double 

\( \text{Input/Output} \)

On entry: if \( \text{ind} = 0 \), \( \text{rsave} \) need not be set. If \( \text{ind} = 1 \) then it must be unchanged from the previous call to the function.

On exit: contains information about the iteration required for subsequent calls.

22: \( \text{lrsave} \) – Integer 

\( \text{Input} \)

On entry: the dimension of the array \( \text{rsave} \) as declared in the function from which \( \text{nag\_pde\_parab\_1d\_keller\_ode} \) (d03pkc) is called. Its size depends on the type of matrix algebra selected:

\[
\text{if } \text{laopt} = \text{Nag\_LinAlgFull}, \text{lrsave} \geq \text{neqn} \times \text{neqn} + \text{neqn} + \text{nwkres} + \text{lenode};
\]

\[
\text{if } \text{laopt} = \text{Nag\_LinAlgBand}, \text{lrsave} \geq (2 \times ml + mu + 2) \times \text{neqn} + \text{nwkres} + \text{lenode};
\]

\[
\text{if } \text{laopt} = \text{Nag\_LinAlgSparse}, \text{lrsave} \geq 4 \times \text{neqn} + 11 \times \text{neqn}/2 + 1 + \text{nwkres} + \text{lenode},
\]

where \( ml \) and \( mu \) are the lower and upper half bandwidths given by \( \text{npt} + \text{nleft} - 1 \), and \( mu = 2 \times \text{npde} - \text{nleft} - 1 \), for problems involving PDEs only, and \( ml = mu = \text{neqn} - 1 \), for coupled PDE/ODE problems.

\( \text{nwkres} = \text{npde} \times (2 \times \text{npnts} + 6 \times \text{nx} + 3 \times \text{npde} + 26) + \text{nx} + \text{ncode} + 7 \times \text{npnts} + 2 \),

when \( \text{nco} > 0 \) and \( \text{nx} > 0 \), and

\( \text{nwkres} = \text{npde} \times (2 \times \text{npnts} + 3 \times \text{npde} + 32) + \text{nco} + 7 \times \text{npnts} + 3 \), when \( \text{nco} > 0 \) and \( \text{nx} = 0 \), and

\( \text{nwkres} = \text{npde} \times (2 \times \text{npnts} + 3 \times \text{npde} + 32) + 7 \times \text{npnts} + 4 \), when \( \text{nco} = 0 \).

[NP2645/7]
\[ \text{lenode} = (6 + \text{int}(\text{algopt}[1])) \times \text{neqn} + 50, \text{ when the BDF method is used, and} \]
\[ \text{lenode} = 9 \times \text{neqn} + 50, \text{ when the Theta method is used.} \]

**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.

23: \text{isave} [\text{lisave}] \rightarrow \text{Integer} \quad \text{Input}/\text{Output}

\text{On entry:} On entry: if \text{ind} = 0, \text{isave} need not be set. If \text{ind} = 1 then it must be unchanged from the previous call to the function.

\text{On exit:} 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.

\text{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.

\text{isave}[2] contains the number of Jacobian evaluations performed by the time integrator.

\text{isave}[3] contains the order of the ODE method last used in the time integration.

\text{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 \text{LU} decomposition of the Jacobian matrix.

24: \text{lisave} \rightarrow \text{Integer} \quad \text{Input}

\text{On entry:} the dimension of the array \text{isave} as declared in the function from which \text{nag_pde_parab_1d_keller_ode (d03pkc)} is called. Its size depends on the type of matrix algebra selected:

- if \text{laopt} = \text{Nag_LinAlgFull}, \text{lisave} \geq 24
- if \text{laopt} = \text{Nag_LinAlgBand}, \text{lisave} \geq \text{neqn} + 24
- if \text{laopt} = \text{Nag_LinAlgSparse}, \text{lisave} \geq 25 \times \text{neqn} + 24

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

25: \text{itask} \rightarrow \text{Integer} \quad \text{Input}

\text{On entry:} the task to be performed by the ODE integrator. The permitted values of \text{itask} and their meanings are detailed below:

- \text{itask} = 1
  normal computation of output values \text{u} at \(t = \text{tout}\) (by overshooting and interpolating).

- \text{itask} = 2
  take one step in the time direction and return.

- \text{itask} = 3
  stop at first internal integration point at or beyond \(t = \text{tout}\).

- \text{itask} = 4
  normal computation of output values \text{u} at \(t = \text{tout}\) but without overshooting \(t = t_{\text{crit}}\), where \(t_{\text{crit}}\) is described under the parameter \text{algopt}.
itask = 5

take one step in the time direction and return, without passing $t_{crit}$, where $t_{crit}$ is described under the parameter algopt.

Constraint: $1 \leq \text{itask} \leq 5$.

26: itrace – Integer

On entry: the level of trace information required from nag_pde_parab_1d_keller_ode (d03pkc) and the underlying ODE solver as follows:

If $\text{itrace} \leq -1$, no output is generated.

If $\text{itrace} = 0$, only warning messages from the PDE solver are printed.

If $\text{itrace} = 1$, then 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.

If $\text{itrace} = 2$, then the output from the underlying ODE solver is similar to that produced when $\text{itrace} = 1$, except that the advisory messages are given in greater detail.

If $\text{itrace} \geq 3$, then the output from the underlying ODE solver is similar to that produced when $\text{itrace} = 2$, except that the advisory messages are given in greater detail.

27: outfile – char *

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.

28: ind – Integer *

On entry: ind 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 parameters tout and fail should be reset between calls to nag_pde_parab_1d_keller_ode (d03pkc).

Constraint: $0 \leq \text{ind} \leq 1$.

On exit: $\text{ind} = 1$.

29: comm – NAG_Comm *

The NAG communication parameter (see the Essential Introduction).

30: saved – Nag_D03_Save *

Note: saved is a NAG defined structure. See Section 2.2.1.1 of the Essential Introduction.

On entry: if the current call to nag_pde_parab_1d_keller_ode (d03pkc) follows a previous call to a Chapter d03 function then saved must contain the unchanged value output from that previous call.

On exit: data to be passed unchanged to any subsequent call to a Chapter d03 function.

31: fail – NagError *

The NAG error parameter (see the Essential Introduction).
6 Error Indicators and Warnings

NE_INT

On entry, npde = (value).
Constraint: npde ≥ 1.

On entry, npts = (value).
Constraint: npts ≥ 3.

On entry, ncode = (value).
Constraint: ncode ≥ 0.

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

On entry, nleft = (value).
Constraint: nleft ≥ 0.

On entry, itol is not equal to 1, 2, 3, or 4: itol = (value).

On entry, ind is not equal to 0 or 1: ind = (value).

On entry, itask is not equal to 1, 2, 3, 4 or 5: itask = (value).

On entry, ncode = 0, but nxi is not equal to 0: nxi = (value).

On entry, nxi = (value).
Constraint: nxi ≥ 0.

NE_INT_2

On entry, nleft = (value), npde = (value).
Constraint: 0 ≤ nleft ≤ npde.

On entry, ncode = (value), nxi = (value).
Constraint: if ncode = 0, nxi = 0;
if ncode > 0, nxi ≥ 0.

On entry, corresponding elements atol[i−1] and rtol[j−1] are both zero. i = (value), j = (value).

When using the sparse option lisave or lrsave is too small: lisave = (value), lrsave = (value).

On entry, lrsave is too small: lrsave = (value). Minimum possible dimension: (value).

On entry, lisave is too small: lisave = (value). Minimum possible dimension: (value).

On entry, nleft > npde: nleft = (value), npde = (value).

NE_INT_4

On entry, npde = (value), npts = (value), ncode = (value), neqn = (value).
Constraint: neqn = npde × npts + ncode.

On entry, neqn is not equal to npde × npts + ncode: neqn = (value), npde = (value),
     npts = (value), ncode = (value).

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

NE_FAILED_DERIV
In setting up the ODE system an internal auxiliary was unable to initialize the derivative. This could be due to user setting ires = 3 in pdedef or bndary.

NE_FAILED_START
atol and rtol were too small to start integration.
NE_FAILED_STEP
Repeated errors in an attempted step of underlying ODE solver. Integration was successful as far as ts: ts = ⟨value⟩.
Error during Jacobian formulation for ODE system. Increase itrace for further details.
Underlying ODE solver cannot make further progress from the point ts with the supplied values of atol and rtol. ts = ⟨value⟩.

NE_INTERNAL_ERROR
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] = ⟨value⟩.

NE_NOT STRICTLY_INCREASING
On entry xi[i] ≤ xi[i − 1]: i = ⟨value⟩, xi[i] = ⟨value⟩, xi[i − 1] = ⟨value⟩.
On entry, mesh points x badly ordered: i = ⟨value⟩, x[i − 1] = ⟨value⟩, j = ⟨value⟩, x[j − 1] = ⟨value⟩.

NE_REAL
On entry, at least one point in xi lies outside [x[0], x[npts − 1]]: x[0] = ⟨value⟩, x[npts − 1] = ⟨value⟩.
On entry, tout − ts is too small: tout = ⟨value⟩, ts = ⟨value⟩.
On entry, tout ≤ ts: tout = ⟨value⟩, ts = ⟨value⟩.

NE_REAL_ARRAY
On entry, rtol[i − 1] < 0.0: i = ⟨value⟩, rtol[i − 1] = ⟨value⟩.
On entry, atol[i − 1] < 0.0: i = ⟨value⟩, atol[i − 1] = ⟨value⟩.

NE_SING_JAC
Singular Jacobian of ODE system. Check problem formulation.

NE_USER_STOP
In evaluating residual of ODE system, ires = 2 has been set in pdedef, bndary, or odedef. Integration is successful as far as ts: ts = ⟨value⟩.

NE_ZERO_WTS
Zero error weights encountered during time integration.

NE_ALLOC_FAIL
Memory allocation failed.

NE_BAD_PARAM
On entry, parameter ⟨value⟩ had an illegal value.

NE_NOT_WRITE_FILE
Cannot open file ⟨value⟩ for writing.
7 Accuracy
The function 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. The user should therefore test the effect of varying
the accuracy parameters, $atol$ and $rtol$.

8 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 9 below). In general, a second-order
problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite-difference scheme (see nag_pde_parab_1d_fd (d03pcc) or nag_pde_parab_1d_fd_ode (d03phc) 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 discretisation scheme with upwind weighting (nag_pde_parab_1d_cd_ode (d03plc) for example), or the addition of a second-order artificial dissipation term.

The time taken depends on the complexity of the system and on the accuracy requested. For a given
system and a fixed accuracy it is approximately proportional to $neqn$.

9 Example
This problem provides a simple coupled system of two PDEs and one ODE.

\[
(V_1)^2 \frac{\partial U_1}{\partial t} - x V_1 \frac{\partial U_2}{\partial x} = 0, \\
U_2 - \frac{\partial U_1}{\partial x} = 0, \\
\dot{V}_1 - V_1 U_2 - U_1 - t = 0,
\]

for $t \in [10^{-4}, 0.1 \times 2^i]$, for $i = 1, 2, \ldots, 5$, $x \in [0, 1]$. The left boundary condition at $x = 0$ is

\[U_2 = -V_1 \exp t,\]

and the right boundary condition at $x = 1$ is

\[U_2 = -V_1 \dot{V}_1.\]

The initial conditions at $t = 10^{-4}$ are defined by the exact solution:

\[V_1 = t, \quad U_1(x, t) = \exp\{t(1 - x)\} - 1.0 \quad \text{and} \quad U_2(x, t) = -t \exp\{t(1 - x)\}, \quad x \in [0, 1],\]

and the coupling point is at $\xi_1 = 1.0$.

This problem is exactly the same as the nag_pde_parab_1d_fd_ode (d03phc) example problem, but
reduced to first-order by the introduction of a second PDE variable (as mentioned in Section 8).
9.1 Program Text

/* nag_pde_parab_1d_keller_ode (d03pkc) Example Program. */
* Copyright 2001 Numerical Algorithms Group.
*/

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

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

static void bndary(Integer npde, double t, Integer ibnd, Integer nobc,
                    const double u[], const double ut[], Integer ncode,
                    const double v[], const double vdot[], double res[],
                    Integer *ires, Nag_Comm *comm);

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

static void uvinit(Integer npde, Integer npts, double *x, double *u,
                    Integer ncode, Integer neqn, double ts);

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

#define UCP(I,J) ucp[npde*((J)-1)+(I)-1]

int main(void)
{
    const Integer npde=2, npts=21, ncode=1, nxi=1, nleft=1,
    neqn=npde*npts+ncode, lisave=24,
    nwkres=npde*(npts+6*nxi+3*ncode+15)+ncode+nxi+7*npts+2,
    lenode=11*neqn+50, lrsave=neqn*neqn+neqn+nwkres+lenode;
    double tout, ts;
    Integer exit_status, i, ind, it, itask, itol, itrace;
    Boolean theta;
    double *algopt=0, *atol=0, *exy=0, *rsave=0, *rtol=0,
    *u=0, *x=0, *xi=0;
    Integer *isave=0;
    NagError fail;
    Nag_Comm comm;
    Nag_D03_Save saved;

    /* Allocate memory */

    if ( !(algopt = NAG_ALLOC(30, double)) ||
        !(atol = NAG_ALLOC(1, double)) ||
        !(exy = NAG_ALLOC(neqn, double)) ||
        !(rsave = NAG_ALLOC(lrsave, double)) ||
        !(rtol = NAG_ALLOC(1, double)) ||
        !(u = NAG_ALLOC(neqn, double)) ||
        !(x = NAG_ALLOC(npts, double)) ||
        !(xi = NAG_ALLOC(nxi, double)) ||
        !(isave = NAG_ALLOC(lisave, Integer)) )
    {
        Vprintf("Allocation failure\n");
        exit_status = 1;
        goto END;
    }

    Vprintf("d03pkc Example Program Results\n\n");
    INIT_FAIL(fail);
    exit_status = 0;

    [NP3645/7] d03pkc.17
itrace = 0;
itol = 1;
atol[0] = 1e-4;
rtol[0] = atol[0];

Vprintf(" Accuracy requirement =%10.3e", atol[0]);
Vprintf(" Number of points = %ld\n\n", npts);

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

xi[0] = 1.0;
ind = 0;
itask = 1;

/* Set THETA to TRUE if the Theta integrator is required */
theta = FALSE;
for (i = 0; i < 30; ++i) algopt[i] = 0.0;
if (theta)
  { algopt[0] = 2.0; }
else {
  algopt[0] = 0.0;
}
algopt[0] = 1.0;
algopt[12] = 0.005;

/* Loop over output value of t */
ts = 1e-4;
tout = 0.0;
Vprintf(" x %9.3f%9.3f%9.3f%9.3f%9.3f\n\n", x[0], x[4], x[8], x[12], x[20]);

uvinit(npde, npts, x, u, ncode, neqn, ts);
for (it = 0; it < 5; ++it)
  {
    tout = 0.1*pow(2.0, (it+1));
d03pkc(npde, &ts, tout, pdedef, bndary, u, npts, x, nleft, ncode, odedef, nxi, xi, neqn, rtol, atol, itol, Nag_TwoNorm, Nag_LinAlgFull, algopt, rsave, lrsave, isave, lisave, itask, itrace, 0, &ind, &comm, &saved, &fail);
    if (fail.code != NE_NOERROR)
      {
        Vprintf("Error from d03pkc.\n%s\n", fail.message);
        exit_status = 1;
goto END;
      }
  /* Check against the exact solution */
  exact(tout, neqn, npts, x, exy);
  Vprintf(" t = %6.3f\n", ts);
  Vprintf(" App. sol. %7.3f%9.3f%9.3f%9.3f%9.3f", u[0], u[8], u[16], u[24], u[40]);
  Vprintf(" ODE sol. =%8.3f\n", u[42]);
  Vprintf(" Exact sol. %7.3f%9.3f%9.3f%9.3f%9.3f", exy[0], exy[8], exy[16], exy[24], exy[40]);
  Vprintf(" ODE sol. =%8.3f\n", ts);
  }
Vprintf(" Number of integration steps in time = %6ld\n", isave[0]);
Vprintf(" Number of function evaluations = %6ld\n", isave[1]);
Vprintf(" Number of Jacobian evaluations =%6ld\n", isave[2]);
Vprintf(" Number of iterations = %6ld\n\n", isave[4]);

END:
if (algopt) NAG_FREE(algopt);
if (atol) NAG_FREE(atol);
if (exy) NAG_FREE(exy);
if (rsave) NAG_FREE(rsave);
if (rtol) NAG_FREE(rtol);
if (u) NAG_FREE(u);
if (x) NAG_FREE(x);
if (xi) NAG_FREE(xi);
if (isave) NAG_FREE(isave);
return exit_status;
}

static void uvinit(Integer npde, Integer npts, double *x,
    double *u, Integer ncode, Integer neqn,
    double ts)
{
    Integer i, k;
    /* Routine for PDE initial values */
    k = 0;
    for (i = 0; i < npts; ++i)
    {
        u[k] = exp(ts*(1.0-x[i])) - 1.0;
        u[k+1] = -ts*exp(ts*(1.0-x[i]));
        k += 2;
    }
    u[neqn-1] = ts;
    return;
}

static 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 f[], Integer *ires,
    Nag_Comm *comm)
{
    if (*ires == -1) {
        f[0] = vdot[0];
    } else {
        f[0] = vdot[0] - v[0]*UCP(1, 1) - UCP(2, 1) - 1.0 - t;
    }
    return;
}

static void pdedef(Integer npde, double t, double x, const double u[],
    const double ut[], const double ux[], Integer ncode,
    const double v[], const double vdot[], double res[],
    Integer *ires, Nag_Comm *comm)
{
    if (*ires == -1)
    {
        res[0] = v[0]*v[0]*ut[0] - x*ux[1]*v[0]*vdot[0];
        res[1] = 0.0;
    } else {
        res[0] = v[0]*v[0]*ut[0] - x*ux[1]*v[0]*vdot[0] - ux[1];
        res[1] = u[1] - ux[0];
    }
    return;
}

static void bndary(Integer npde, double t, Integer ibnd,
    Integer nobc, const double u[], const double ut[],
    Integer ncode, const double v[], const double vdot[],
    double res[], Integer *ires, Nag_Comm *comm)
{
    if (ibnd == 0) {  

}
if (*ires == -1) {
    res[0] = 0.0;
} else {
    res[0] = u[1] + v[0]*exp(t);
} else {
    if (*ires == -1) {
        res[0] = v[0]*vdot[0];
    } else {
        res[0] = u[1] + v[0]*vdot[0];
    }
}
return;

static void exact(double time, Integer neqn, Integer npts, 
                   double *x, double *u) 
{
    /* Exact solution (for comparison purposes) */
    Integer i, k;
    k=0 ;
    for (i = 0; i < npts; ++i) {
        u[k] = exp(time*(1.0-x[i])) - 1.0;
        k += 2;
    }
    return;
}

9.2 Program Data

None.

9.3 Program Results

d03pkc Example Program Results

Accuracy requirement = 1.000e-04 Number of points = 21

<table>
<thead>
<tr>
<th>t</th>
<th>App. sol.</th>
<th>Exact sol.</th>
<th>ODE sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.200</td>
<td>0.222</td>
<td>0.221</td>
<td>0.200</td>
</tr>
<tr>
<td>0.400</td>
<td>0.492</td>
<td>0.492</td>
<td>0.400</td>
</tr>
<tr>
<td>0.800</td>
<td>1.226</td>
<td>1.226</td>
<td>0.800</td>
</tr>
<tr>
<td>1.600</td>
<td>3.952</td>
<td>3.953</td>
<td>1.600</td>
</tr>
<tr>
<td>3.200</td>
<td>23.522</td>
<td>23.533</td>
<td>3.197</td>
</tr>
</tbody>
</table>

Number of integration steps in time = 642
Number of function evaluations = 3022
Number of Jacobian evaluations = 39
Number of iterations = 1328