NAG Library Routine Document

D03PPF/D03PPA

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

D03PPF/D03PPA integrates a system of linear or nonlinear parabolic partial differential equations (PDEs) in one space variable, with scope for coupled ordinary differential equations (ODEs), and automatic adaptive spatial remeshing. The spatial discretization is performed using finite differences, and the method of lines is employed to reduce the PDEs to a system of ODEs. The resulting system is solved using a Backward Differentiation Formula (BDF) method or a Theta method (switching between Newton's method and functional iteration).

D03PPA is a version of D03PPF that has additional arguments in order to make it safe for use in multithreaded applications (see Section 5).

2 Specification

2.1 Specification for D03PPF

```
SUBROUTINE D03PPF (NPDE, M, TS, TOUT, PDEDEF, BNDARY, UVINIT, U, NPTS,     &
                   X, NCODE, ODEDEF, NXI, XI, NEQN, RTOL, ATOL, ITOL,      &
                   NORM, LAOPT, ALGOPT, REMESH, NXFIX, XFIX, NRMESH,       &
                   DXMESH, TRMESH, IPMINF, XRATIO, CON, MONITF, RSAVE,     &
                   LRSAVE, ISAVE, LISAVE, ITASK, ITRACE, IND, IFAIL)

INTEGER            NPDE, M, NPTS, NCODE, NXI, NEQN, ITOL, NXFIX,           &
                   NRMESH, IPMINF, LRSAVE, ISAVE(LISAVE), LISAVE,          &
                   ITASK, ITRACE, IND, IFAIL
REAL (KIND=nag_wp) TS, TOUT, U(NEQN), X(NPTS), XI(NXI), RTOL(*),           &
                   ATOL(*), ALGOPT(30), XFIX(*), DXMESH, TRMESH,           &
                   XRATIO, CON, RSAVE(LRSAVE)
LOGICAL            REMESH
CHARACTER(1)       NORM, LAOPT
EXTERNAL           PDEDEF, BNDARY, UVINIT, ODEDEF, MONITF
```

2.2 Specification for D03PPA

```
SUBROUTINE D03PPA (NPDE, M, TS, TOUT, PDEDEF, BNDARY, UVINIT, U, NPTS,     &
                   X, NCODE, ODEDEF, NXI, XI, NEQN, RTOL, ATOL, ITOL,      &
                   NORM, LAOPT, ALGOPT, REMESH, NXFIX, XFIX, NRMESH,       &
                   DXMESH, TRMESH, IPMINF, XRATIO, CON, MONITF, RSAVE,     &
                   LRSAVE, ISAVE, LISAVE, ITASK, ITRACE, IND, IUSER,       &
                   RUSER, CWSAV, LWSAV, IWSAV, RWSAV, IFAIL)

INTEGER            NPDE, M, NPTS, NCODE, NXI, NEQN, ITOL, NXFIX,           &
                   NRMESH, IPMINF, LRSAVE, ISAVE(LISAVE), LISAVE,          &
                   ITASK, ITRACE, IND, IUSER(*), IWSAV(505), IFAIL
REAL (KIND=nag_wp) TS, TOUT, U(NEQN), X(NPTS), XI(NXI), RTOL(*),           &
                   ATOL(*), ALGOPT(30), XFIX(*), DXMESH, TRMESH,           &
                   XRATIO, CON, RSAVE(LRSAVE), RUSER(*), RWSAV(1100)
LOGICAL            REMESH, LWSAV(100)
CHARACTER(1)       NORM, LAOPT
CHARACTER(80)      CWSAV(10)
EXTERNAL           PDEDEF, BNDARY, UVINIT, ODEDEF, MONITF
```

3 Description

D03PPF/D03PPA integrates the system of parabolic-elliptic equations and coupled ODEs

$$\sum_{j=1}^{\text{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + Q_i = x^{-m}\frac{\partial}{\partial x}(x^m R_i), \quad i = 1,2,\ldots,\text{NPDE}, \quad a \le x \le b, t \ge t_0, \tag{1}$$

$$F_i\left(t, V, \dot{V}, \xi, U^*, U_x^*, R^*, U_t^*, U_{xt}^*\right) = 0, \quad i = 1,2,\ldots,\text{NCODE}, \tag{2}$$

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

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

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

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

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

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

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

$$F = G - A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}, \tag{3}$$

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

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

The PDE system which is defined by the functions $P_{i,j}$, Q_i and R_i must be specified in PDEDEF.

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

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

$$\beta_i(x,t)R_i(x,t,U,U_x,V) = \gamma_i\left(x,t,U,U_x,V,\dot{V}\right), \quad i = 1,2,\ldots,\text{NPDE}, \tag{4}$$

where $x = a$ or $x = b$.

The boundary conditions must be specified in BNDARY. The function γ_i may depend **linearly** on $\dot{V}$.

The problem is subject to the following restrictions:

(i) In (1), $\dot{V}_j(t)$, for $j = 1,2,\ldots,\text{NCODE}$, may only appear **linearly** in the functions Q_i, for $i = 1,2,\ldots,\text{NPDE}$, with a similar restriction for γ;

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

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

(iv) The evaluation of the terms $P_{i,j}$, Q_i and R_i is done approximately at the mid-points of the mesh $X(i)$, for $i = 1, 2, \ldots,$ NPTS, by calling the PDEDEF for each mid-point in turn. Any discontinuities in these functions **must** therefore be at one or more of the fixed mesh points specified by XFIX;

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

(vi) If $m > 0$ and $x_1 = 0.0$, which is the left boundary point, then it must be ensured that the PDE solution is bounded at this point. This can be done by either specifying the solution at $x = 0.0$ or by specifying a zero flux there, that is $\beta_i = 1.0$ and $\gamma_i = 0.0$. See also Section 9.

The algebraic-differential equation system which is defined by the functions F_i must be specified in ODEDEF. You must also specify the coupling points ξ 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. For simple problems in Cartesian coordinates, this system is obtained by replacing the space derivatives by the usual central, three-point finite difference formula. However, for polar and spherical problems, or problems with nonlinear coefficients, the space derivatives are replaced by a modified three-point formula which maintains second order accuracy. In total there are 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.

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

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

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

Note that unless the monitor function for the initial solution values is zero at all user-specified initial mesh points, a new initial mesh is calculated and adopted according to the user-specified remeshing criteria. 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

Berzins M (1990) Developments in the NAG Library software for parabolic equations *Scientific Software Systems* (eds J C Mason and M G Cox) 59–72 Chapman and Hall

Berzins M, Dew P M and Furzeland R M (1989) Developing software for time-dependent problems using the method of lines and differential-algebraic integrators *Appl. Numer. Math.* **5** 375–397

Berzins M and Furzeland R M (1992) An adaptive theta method for the solution of stiff and nonstiff differential equations *Appl. Numer. Math.* **9** 1–19

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

Skeel R D and Berzins M (1990) A method for the spatial discretization of parabolic equations in one space variable *SIAM J. Sci. Statist. Comput.* **11(1)** 1–32

5 Arguments

1: NPDE – INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE ≥ 1.

2: M – INTEGER *Input*

On entry: the coordinate system used:

M $= 0$
Indicates Cartesian coordinates.

M $= 1$
Indicates cylindrical polar coordinates.

M $= 2$
Indicates spherical polar coordinates.

Constraint: M $= 0$, 1 or 2.

3: TS – REAL (KIND=nag_wp) *Input/Output*

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

On exit: the value of t corresponding to the solution values in U. Normally TS $=$ TOUT.

Constraint: TS $<$ TOUT.

4: TOUT – REAL (KIND=nag_wp) *Input*

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

5: PDEDEF – SUBROUTINE, supplied by the user. *External Procedure*

PDEDEF must evaluate the functions $P_{i,j}$, Q_i and R_i which define the system of PDEs. The functions may depend on x, t, U, U_x and V. Q_i may depend linearly on $\dot{V}$. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PPF/D03PPA.

The specification of PDEDEF for D03PPF is:

```
SUBROUTINE PDEDEF (NPDE, T, X, U, UX, NCODE, V, VDOT, P, Q, R,      &
                   IRES)

INTEGER            NPDE, NCODE, IRES
REAL (KIND=nag_wp) T, X, U(NPDE), UX(NPDE), V(NCODE),               &
                   VDOT(NCODE), P(NPDE,NPDE), Q(NPDE), R(NPDE)
```

The specification of PDEDEF for D03PPA is:

```
SUBROUTINE PDEDEF (NPDE, T, X, U, UX, NCODE, V, VDOT, P, Q, R,      &
                   IRES, IUSER, RUSER)

INTEGER            NPDE, NCODE, IRES, IUSER(*)
REAL (KIND=nag_wp) T, X, U(NPDE), UX(NPDE), V(NCODE),               &
                   VDOT(NCODE), P(NPDE,NPDE), Q(NPDE), R(NPDE),     &
                   RUSER(*)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – REAL (KIND=nag_wp) *Input*

On entry: the current value of the independent variable t.

3: X – REAL (KIND=nag_wp) *Input*

On entry: the current value of the space variable x.

4: U(NPDE) – REAL (KIND=nag_wp) array *Input*

On entry: U(i) contains the value of the component $U_i(x,t)$, for $i = 1,2,\ldots,$NPDE.

5: UX(NPDE) – REAL (KIND=nag_wp) array *Input*

On entry: UX(i) contains the value of the component $\dfrac{\partial U_i(x,t)}{\partial x}$, for $i = 1,2,\ldots,$NPDE.

6: NCODE – INTEGER *Input*

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

7: V(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if NCODE > 0, V(i) contains the value of the component $V_i(t)$, for $i = 1,2,\ldots,$NCODE.

8: VDOT(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if NCODE > 0, VDOT(i) contains the value of component $\dot{V}_i(t)$, for $i = 1,2,\ldots,$NCODE.

Note: $\dot{V}_i(t)$, for $i = 1,2,\ldots,$NCODE, may only appear linearly in Q_j, for $j = 1,2,\ldots,$NPDE.

9: P(NPDE, NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: P(i,j) must be set to the value of $P_{i,j}(x,t,U,U_x,V)$, for $i = 1,2,\ldots,$NPDE and $j = 1,2,\ldots,$NPDE.

10: Q(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: Q(i) must be set to the value of $Q_i\left(x,t,U,U_x,V,\dot{V}\right)$, for $i = 1,2,\ldots,$NPDE.

11: R(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: R(i) must be set to the value of $R_i(x,t,U,U_x,V)$, for $i = 1,2,\ldots,$NPDE.

12: IRES – INTEGER *Input/Output*

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, you may set IRES to force the integration routine to take certain actions as described below:

IRES $= 2$
: Indicates to the integrator that control should be passed back immediately to the calling (sub)routine with the error indicator set to IFAIL $= 6$.

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 D03PPF/D03PPA returns to the calling subroutine with the error indicator set to IFAIL $= 4$.

Note: *the following are additional arguments for specific use with D03PPA. Users of D03PPF therefore need not read the remainder of this description*.

13: IUSER(∗) – INTEGER array *User Workspace*

14: RUSER(∗) – REAL (KIND=nag_wp) array *User Workspace*

PDEDEF is called with the arguments IUSER and RUSER as supplied to D03PPF/D03PPA. You should use the arrays IUSER and RUSER to supply information to PDEDEF.

PDEDEF must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D03PPF/D03PPA is called. Arguments denoted as *Input* must **not** be changed by this procedure.

6: BNDARY – SUBROUTINE, supplied by the user. *External Procedure*

BNDARY must evaluate the functions β_i and γ_i which describe the boundary conditions, as given in (4).

The specification of BNDARY for D03PPF is:

```
SUBROUTINE BNDARY (NPDE, T, U, UX, NCODE, V, VDOT, IBND, BETA,      &
                   GAMMA, IRES)

INTEGER            NPDE, NCODE, IBND, IRES
REAL (KIND=nag_wp) T, U(NPDE), UX(NPDE), V(NCODE), VDOT(NCODE),     &
                   BETA(NPDE), GAMMA(NPDE)
```

The specification of BNDARY for D03PPA is:

```
SUBROUTINE BNDARY (NPDE, T, U, UX, NCODE, V, VDOT, IBND, BETA,      &
                   GAMMA, IRES, IUSER, RUSER)

INTEGER            NPDE, NCODE, IBND, IRES, IUSER(*)
REAL (KIND=nag_wp) T, U(NPDE), UX(NPDE), V(NCODE), VDOT(NCODE),     &
                   BETA(NPDE), GAMMA(NPDE), RUSER(*)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – REAL (KIND=nag_wp) *Input*

On entry: the current value of the independent variable t.

3: U(NPDE) – REAL (KIND=nag_wp) array *Input*

On entry: U(i) contains the value of the component $U_i(x,t)$ at the boundary specified by IBND, for $i = 1,2,\ldots,\text{NPDE}$.

4: UX(NPDE) – REAL (KIND=nag_wp) array *Input*

On entry: UX(i) contains the value of the component $\dfrac{\partial U_i(x,t)}{\partial x}$ at the boundary specified by IBND, for $i = 1,2,\ldots,\text{NPDE}$.

5: NCODE – INTEGER *Input*

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

6: V(NCODE) – REAL (KIND=nag_wp) array *Input*

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

7: VDOT(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: VDOT(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

Note: $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ NCODE, may only appear linearly in γ_j, for $j = 1, 2, \ldots,$ NPDE.

8: IBND – INTEGER *Input*

On entry: specifies which boundary conditions are to be evaluated.

IBND $= 0$
BNDARY must set up the coefficients of the left-hand boundary, $x = a$.

IBND $\neq 0$
BNDARY must set up the coefficients of the right-hand boundary, $x = b$.

9: BETA(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: BETA(i) must be set to the value of $\beta_i(x, t)$ at the boundary specified by IBND, for $i = 1, 2, \ldots,$ NPDE.

10: GAMMA(NPDE) – REAL (KIND=nag_wp) array *Output*

On exit: GAMMA(i) must be set to the value of $\gamma_i\left(x, t, U, U_x, V, \dot{V}\right)$ at the boundary specified by IBND, for $i = 1, 2, \ldots,$ NPDE.

11: IRES – INTEGER *Input/Output*

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, you may set IRES to force the integration routine to take certain actions as described below:

IRES $= 2$
Indicates to the integrator that control should be passed back immediately to the calling (sub)routine with the error indicator set to IFAIL $= 6$.

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 D03PPF/D03PPA returns to the calling subroutine with the error indicator set to IFAIL $= 4$.

Note: *the following are additional arguments for specific use with D03PPA. Users of D03PPF therefore need not read the remainder of this description*.

12: IUSER($*$) – INTEGER array *User Workspace*
13: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

BNDARY is called with the arguments IUSER and RUSER as supplied to D03PPF/D03PPA. You should use the arrays IUSER and RUSER to supply information to BNDARY.

BNDARY must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03PPF/D03PPA is called. Arguments denoted as *Input* must **not** be changed by this procedure.

7: UVINIT – SUBROUTINE, supplied by the user. *External Procedure*

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

The specification of UVINIT for D03PPF is:

```
SUBROUTINE UVINIT (NPDE, NPTS, NXI, X, XI, U, NCODE, V)
INTEGER             NPDE, NPTS, NXI, NCODE
REAL (KIND=nag_wp) X(NPTS), XI(NXI), U(NPDE,NPTS), V(NCODE)
```

The specification of UVINIT for D03PPA is:

```
SUBROUTINE UVINIT (NPDE, NPTS, NXI, X, XI, U, NCODE, V, IUSER,      &
                   RUSER)
INTEGER             NPDE, NPTS, NXI, NCODE, IUSER(*)
REAL (KIND=nag_wp) X(NPTS), XI(NXI), U(NPDE,NPTS), V(NCODE),        &
                   RUSER(*)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: NPTS – INTEGER *Input*

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

3: NXI – INTEGER *Input*

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

4: X(NPTS) – REAL (KIND=nag_wp) array *Input*

On entry: the current mesh. X(i) contains the value of x_i, for $i = 1,2,\ldots,\text{NPTS}$.

5: XI(NXI) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, XI(i) contains the value of the ODE/PDE coupling point, ξ_i, for $i = 1,2,\ldots,\text{NXI}$.

6: U(NPDE, NPTS) – REAL (KIND=nag_wp) array *Output*

On exit: if NXI > 0, U(i,j) contains the value of the component $U_i\left(x_j,t_0\right)$, for $i = 1,2,\ldots,\text{NPDE}$ and $j = 1,2,\ldots,\text{NPTS}$.

7: NCODE – INTEGER *Input*

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

8: V(NCODE) – REAL (KIND=nag_wp) array *Output*

On exit: V(i) contains the value of component $V_i(t_0)$, for $i = 1,2,\ldots,\text{NCODE}$.

Note: *the following are additional arguments for specific use with D03PPA. Users of D03PPF therefore need not read the remainder of this description*.

9: IUSER($*$) – INTEGER array *User Workspace*

10: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

UVINIT is called with the arguments IUSER and RUSER as supplied to D03PPF/ D03PPA. You should use the arrays IUSER and RUSER to supply information to UVINIT.

UVINIT must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub) program from which D03PPF/D03PPA is called. Arguments denoted as *Input* must **not** be changed by this procedure.

8: U(NEQN) – REAL (KIND=nag_wp) array *Input/Output*

On entry: if IND = 1, the value of U must be unchanged from the previous call.

On exit: U(NPDE × (j − 1) + i) contains the computed solution $U_i(x_j, t)$, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NPTS, and U(NPTS × NPDE + k) contains $V_k(t)$, for $k = 1, 2, \ldots,$ NCODE, evaluated at $t =$ TS.

9: NPTS – INTEGER *Input*

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

Constraint: NPTS ≥ 3.

10: X(NPTS) – REAL (KIND=nag_wp) array *Input/Output*

On entry: the initial mesh points in the space direction. X(1) must specify the left-hand boundary, a, and X(NPTS) must specify the right-hand boundary, b.

Constraint: X(1) < X(2) < ⋯ < X(NPTS).

On exit: the final values of the mesh points.

11: NCODE – INTEGER *Input*

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

Constraint: NCODE ≥ 0.

12: ODEDEF – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

ODEDEF must evaluate the functions F, which define the system of ODEs, as given in (3).

If you wish to compute the solution of a system of PDEs only (NCODE = 0), ODEDEF must be the dummy routine D03PCK for D03PPF (or D53PCK for D03PPA). D03PCK and D53PCK are included in the NAG Library.

The specification of ODEDEF for D03PPF is:

```
SUBROUTINE ODEDEF (NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,      &
                   RCP, UCPT, UCPTX, F, IRES)

INTEGER             NPDE, NCODE, NXI, IRES
REAL (KIND=nag_wp) T, V(NCODE), VDOT(NCODE), XI(NXI),                &
                   UCP(NPDE,*), UCPX(NPDE,*), RCP(NPDE,*),           &
                   UCPT(NPDE,*), UCPTX(NPDE,*), F(NCODE)
```

The specification of ODEDEF for D03PPA is:

```
SUBROUTINE ODEDEF (NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,      &
                   RCP, UCPT, UCPTX, F, IRES, IUSER, RUSER)

INTEGER             NPDE, NCODE, NXI, IRES, IUSER(*)
REAL (KIND=nag_wp) T, V(NCODE), VDOT(NCODE), XI(NXI),                &
                   UCP(NPDE,*), UCPX(NPDE,*), RCP(NPDE,*),           &
                   UCPT(NPDE,*), UCPTX(NPDE,*), F(NCODE),            &
                   RUSER(*)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – REAL (KIND=nag_wp) *Input*

On entry: the current value of the independent variable t.

3: NCODE – INTEGER *Input*

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

4: V(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if NCODE > 0, V(i) contains the value of the component $V_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

5: VDOT(NCODE) – REAL (KIND=nag_wp) array *Input*

On entry: if NCODE > 0, VDOT(i) contains the value of component $\dot{V}_i(t)$, for $i = 1, 2, \ldots,$ NCODE.

6: NXI – INTEGER *Input*

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

7: XI(NXI) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, XI(i) contains the ODE/PDE coupling points, ξ_i, for $i = 1, 2, \ldots,$ NXI.

8: UCP(NPDE, $*$) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, UCP(i, j) contains the value of $U_i(x, t)$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NXI.

9: UCPX(NPDE, $*$) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, UCPX(i, j) contains the value of $\dfrac{\partial U_i(x,t)}{\partial x}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NXI.

10: RCP(NPDE, $*$) – REAL (KIND=nag_wp) array *Input*

On entry: RCP(i, j) contains the value of the flux R_i at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NXI.

11: UCPT(NPDE, $*$) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, UCPT(i, j) contains the value of $\dfrac{\partial U_i}{\partial t}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NXI.

12: UCPTX(NPDE, $*$) – REAL (KIND=nag_wp) array *Input*

On entry: UCPTX(i, j) contains the value of $\dfrac{\partial^2 U_i}{\partial x \partial t}$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NXI.

13: F(NCODE) – REAL (KIND=nag_wp) array *Output*

On exit: F(i) must contain the ith component of F, for $i = 1, 2, \ldots,$ NCODE, where F is defined as

$$F = G - A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}, \tag{5}$$

or

$$F = -A\dot{V} - B\begin{pmatrix} U_t^* \\ U_{xt}^* \end{pmatrix}. \tag{6}$$

The definition of F is determined by the input value of IRES.

14: IRES – INTEGER *Input/Output*

On entry: the form of F that must be returned in the array F.

IRES $= 1$
Equation (5) must be used.

IRES $= -1$
Equation (6) must be used.

On exit: should usually remain unchanged. However, you may reset IRES to force the integration routine to take certain actions as described below:

IRES $= 2$
Indicates to the integrator that control should be passed back immediately to the calling (sub)routine with the error indicator set to IFAIL $= 6$.

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 D03PPF/D03PPA returns to the calling subroutine with the error indicator set to IFAIL $= 4$.

Note: *the following are additional arguments for specific use with D03PPA. Users of D03PPF therefore need not read the remainder of this description*.

15: IUSER$(*)$ – INTEGER array *User Workspace*
16: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

ODEDEF is called with the arguments IUSER and RUSER as supplied to D03PPF/D03PPA. You should use the arrays IUSER and RUSER to supply information to ODEDEF.

ODEDEF must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03PPF/D03PPA is called. Arguments denoted as *Input* must **not** be changed by this procedure.

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(NXI) – REAL (KIND=nag_wp) array *Input*

On entry: if NXI > 0, XI(i), for $i = 1, 2, \ldots,$ NXI, must be set to the ODE/PDE coupling points.

Constraint: X$(1) \leq$ XI$(1) <$ XI$(2) < \cdots <$ XI(NXI) $\leq$ X(NPTS).

15: NEQN – INTEGER *Input*

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

Constraint: NEQN $=$ NPDE $\times$ NPTS $+$ NCODE.

16: RTOL$(*)$ – REAL (KIND=nag_wp) array *Input*

Note: the dimension of the array RTOL must be at least 1 if ITOL $= 1$ or 2 and at least NEQN if ITOL $= 3$ or 4.

On entry: the relative local error tolerance.

Constraint: RTOL$(i) \geq 0.0$ for all relevant i.

17: ATOL($*$) – REAL (KIND=nag_wp) array *Input*

Note: the dimension of the array ATOL must be at least 1 if ITOL $= 1$ or 3 and at least NEQN if ITOL $= 2$ or 4.

On entry: the absolute local error tolerance.

Constraints:

ATOL(i) ≥ 0.0 for all relevant i;
Corresponding elements of ATOL and RTOL cannot both be 0.0.

18: ITOL – INTEGER *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D03PPF/D03PPA whether to interpret either or both of RTOL or 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:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	RTOL(1) $\times \lvert U_i \rvert$ + ATOL(1)
2	scalar	vector	RTOL(1) $\times \lvert U_i \rvert$ + ATOL(i)
3	vector	scalar	RTOL(i) $\times \lvert U_i \rvert$ + ATOL(1)
4	vector	vector	RTOL(i) $\times \lvert U_i \rvert$ + ATOL(i)

In the above, e_i denotes the estimated local error for the ith component of the coupled PDE/ODE system in time, U(i), for $i = 1, 2, \ldots,$ NEQN.

The choice of norm used is defined by the argument NORM.

Constraint: $1 \leq$ ITOL ≤ 4.

19: NORM – CHARACTER(1) *Input*

On entry: the type of norm to be used.

NORM = 'M'
Maximum norm.

NORM = 'A'
Averaged L_2 norm.

If $\mathrm{U}_{\mathrm{norm}}$ denotes the norm of the vector U of length NEQN, then for the averaged L_2 norm

$$\mathrm{U}_{\mathrm{norm}} = \sqrt{\frac{1}{\mathrm{NEQN}} \sum_{i=1}^{\mathrm{NEQN}} \left(\mathrm{U}(i)/w_i\right)^2},$$

while for the maximum norm

$$\mathrm{U}_{\mathrm{norm}} = \max_i |\mathrm{U}(i)/w_i|.$$

See the description of ITOL for the formulation of the weight vector w.

Constraint: NORM = 'M' or 'A'.

20: LAOPT – CHARACTER(1) *Input*

On entry: the type of matrix algebra required.

LAOPT = 'F'
Full matrix methods to be used.

LAOPT = 'B'
Banded matrix methods to be used.

LAOPT = 'S'
Sparse matrix methods to be used.

Constraint: LAOPT = 'F', 'B' or 'S'.

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

21: ALGOPT(30) – REAL (KIND=nag_wp) array *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(1) 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(1)
Selects the ODE integration method to be used. If ALGOPT(1) = 1.0, a BDF method is used and if ALGOPT(1) = 2.0, a Theta method is used. The default value is ALGOPT(1) = 1.0.

If ALGOPT(1) = 2.0, then ALGOPT(i), for $i = 2, 3, 4$ are not used.

ALGOPT(2)
Specifies the maximum order of the BDF integration formula to be used. ALGOPT(2) may be 1.0, 2.0, 3.0, 4.0 or 5.0. The default value is ALGOPT(2) = 5.0.

ALGOPT(3)
Specifies what method is to be used to solve the system of nonlinear equations arising on each step of the BDF method. If ALGOPT(3) = 1.0 a modified Newton iteration is used and if ALGOPT(3) = 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(3) = 1.0.

ALGOPT(4)
Specifies whether or not the Petzold error test is to be employed. The Petzold error test results in extra overhead but is more suitable when algebraic equations are present, such as $P_{i,j} = 0.0$, for $j = 1, 2, \ldots,$ NPDE, for some i or when there is no $\dot{V}_i(t)$ dependence in the coupled ODE system. If ALGOPT(4) = 1.0, then the Petzold test is used. If ALGOPT(4) = 2.0, then the Petzold test is not used. The default value is ALGOPT(4) = 1.0.

If ALGOPT(1) = 1.0, then ALGOPT(i), for $i = 5, 6, 7$, are not used.

ALGOPT(5)
Specifies the value of Theta to be used in the Theta integration method. $0.51 \leq$ ALGOPT(5) ≤ 0.99. The default value is ALGOPT(5) = 0.55.

ALGOPT(6)
Specifies what method is to be used to solve the system of nonlinear equations arising on each step of the Theta method. If ALGOPT(6) = 1.0, a modified Newton iteration is used and if ALGOPT(6) = 2.0, a functional iteration method is used. The default value is ALGOPT(6) = 1.0.

ALGOPT(7)
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(7) = 1.0, then switching is allowed and if ALGOPT(7) = 2.0, then switching is not allowed. The default value is ALGOPT(7) = 1.0.

ALGOPT(11)
Specifies a point in the time direction, t_{crit}, beyond which integration must not be attempted. The use of t_{crit} is described under the argument ITASK. If ALGOPT(1) $\neq 0.0$, a value of 0.0 for ALGOPT(11), say, should be specified even if ITASK subsequently specifies that t_{crit} will not be used.

ALGOPT(12)
Specifies the minimum 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 maximum absolute step size to be allowed in the time integration. If this option is not required, ALGOPT(13) should be set to 0.0.

ALGOPT(14)
Specifies the initial step size to be attempted by the integrator. If ALGOPT(14) = 0.0, then the initial step size is calculated internally.

ALGOPT(15)
Specifies the maximum number of steps to be attempted by the integrator in any one call. If ALGOPT(15) = 0.0, then no limit is imposed.

ALGOPT(23)
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(23) = 1.0, a modified Newton iteration is used and if ALGOPT(23) = 2.0, functional iteration is used. The default value is ALGOPT(23) = 1.0.

ALGOPT(29) and ALGOPT(30) are used only for the sparse matrix algebra option, LAOPT = 'S'.

ALGOPT(29)
Governs the choice of pivots during the decomposition of the first Jacobian matrix. It should lie in the range $0.0 < \text{ALGOPT}(29) < 1.0$, with smaller values biasing the algorithm towards maintaining sparsity at the expense of numerical stability. If ALGOPT(29) lies outside this range then the default value is used. If the routines regard the Jacobian matrix as numerically singular then increasing ALGOPT(29) towards 1.0 may help, but at the cost of increased fill-in. The default value is ALGOPT(29) = 0.1.

ALGOPT(30)
Is used as a relative pivot threshold during subsequent Jacobian decompositions (see ALGOPT(29)) below which an internal error is invoked. If ALGOPT(30) 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(29)). The default value is ALGOPT(30) = 0.0001.

22: REMESH – LOGICAL *Input*

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

REMESH = .TRUE.
Indicates that spatial remeshing should be performed as specified.

REMESH = .FALSE.
Indicates that spatial remeshing should be suppressed.

Note: REMESH should **not** be changed between consecutive calls to D03PPF/D03PPA. Remeshing can be switched off or on at specified times by using appropriate values for the arguments NRMESH and TRMESH at each call.

23: NXFIX – INTEGER *Input*

On entry: the number of fixed mesh points.

Constraint: $0 \leq \text{NXFIX} \leq \text{NPTS} - 2$.

Note: the end points X(1) and X(NPTS) are fixed automatically and hence should not be specified as fixed points.

24: XFIX($*$) – REAL (KIND=nag_wp) array *Input*

Note: the dimension of the array XFIX must be at least $\max(1, \text{NXFIX})$.

On entry: XFIX(i), for $i = 1, 2, \ldots, \text{NXFIX}$, must contain the value of the x coordinate at the ith fixed mesh point.

Constraints:

XFIX(i) $<$ XFIX($i+1$), for $i = 1, 2, \ldots, \text{NXFIX} - 1$;
each fixed mesh point must coincide with a user-supplied initial mesh point, that is XFIX(i) = X(j) for some j, $2 \leq j \leq \text{NPTS} - 1$.

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

25: NRMESH – INTEGER *Input*

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

NRMESH < 0
Indicates that a new mesh is adopted according to the argument DXMESH. The mesh is tested every |NRMESH| timesteps.

NRMESH $= 0$
Indicates that remeshing should take place just once at the end of the first time step reached when $t >$ TRMESH.

NRMESH > 0
Indicates that remeshing will take place every NRMESH time steps, with no testing using DXMESH.

Note: NRMESH may be changed between consecutive calls to D03PPF/D03PPA to give greater flexibility over the times of remeshing.

26: DXMESH – REAL (KIND=nag_wp) *Input*

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

$$x_i^{(\text{new})} > x_i^{(\text{old})} + \text{DXMESH} \times \left(x_{i+1}^{(\text{old})} - x_i^{(\text{old})} \right)$$

or

$$x_i^{(\text{new})} < x_i^{(\text{old})} - \text{DXMESH} \times \left(x_i^{(\text{old})} - x_{i-1}^{(\text{old})} \right)$$

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

Constraint: DXMESH ≥ 0.0.

27: TRMESH – REAL (KIND=nag_wp) *Input*

On entry: specifies when remeshing will take place when NRMESH is set to zero. Remeshing will occur just once at the end of the first time step reached when t is greater than TRMESH.

Note: TRMESH may be changed between consecutive calls to D03PPF/D03PPA to force remeshing at several specified times.

28: IPMINF – INTEGER *Input*

On entry: the level of trace information regarding the adaptive remeshing. Details are directed to the current advisory message unit (see X04ABF).

IPMINF $= 0$
No trace information.

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 – REAL (KIND=nag_wp) *Input*

On entry: an input bound on the adjacent mesh ratio (greater than 1.0 and typically in the range 1.5 to 3.0). The remeshing routines 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 – REAL (KIND=nag_wp) *Input*

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

$$\int_{x_i}^{x_{i+1}} F^{\text{mon}}(x)\,dx \leq \text{CON} \int_{x_1}^{x_{\text{NPTS}}} F^{\text{mon}}(x)\,dx,$$

(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: $0.1/(\text{NPTS} - 1) \leq \text{CON} \leq 10.0/(\text{NPTS} - 1)$.

31: MONITF – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

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

If you specify REMESH = .FALSE., i.e., no remeshing, then MONITF will not be called and the dummy routine D03PCL for D03PPF (or D53PCL for D03PPA) may be used for MONITF. (D03PCL and D53PCL are included in the NAG Library.)

The specification of MONITF for D03PPF is:

```
SUBROUTINE MONITF (T, NPTS, NPDE, X, U, R, FMON)
INTEGER             NPTS, NPDE
REAL (KIND=nag_wp) T, X(NPTS), U(NPDE,NPTS), R(NPDE,NPTS),          &
                    FMON(NPTS)
```

The specification of MONITF for D03PPA is:

```
SUBROUTINE MONITF (T, NPTS, NPDE, X, U, R, FMON, IUSER, RUSER)
INTEGER             NPTS, NPDE, IUSER(*)
REAL (KIND=nag_wp) T, X(NPTS), U(NPDE,NPTS), R(NPDE,NPTS),          &
                    FMON(NPTS), RUSER(*)
```

1: T – REAL (KIND=nag_wp) *Input*

On entry: the current value of the independent variable t.

2: NPTS – INTEGER *Input*

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

3: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

4: X(NPTS) – REAL (KIND=nag_wp) array *Input*

On entry: the current mesh. X(i) contains the value of x_i, for $i = 1, 2, \ldots,$ NPTS.

5: U(NPDE, NPTS) – REAL (KIND=nag_wp) array *Input*

On entry: U(i, j) contains the value of $U_i(x, t)$ at $x =$ X(j) and time t, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NPTS.

6: R(NPDE, NPTS) – REAL (KIND=nag_wp) array *Input*

On entry: R(i, j) contains the value of $R_i(x, t, U, U_x, V)$ at $x =$ X(j) and time t, for $i = 1, 2, \ldots,$ NPDE and $j = 1, 2, \ldots,$ NPTS.

7: FMON(NPTS) – REAL (KIND=nag_wp) array *Output*

On exit: FMON(i) must contain the value of the monitor function $F^{\mathrm{mon}}(x)$ at mesh point $x =$ X(i).

Constraint: FMON(i) ≥ 0.0.

Note: *the following are additional arguments for specific use with D03PPA. Users of D03PPF therefore need not read the remainder of this description*.

8: IUSER($*$) – INTEGER array *User Workspace*

9: RUSER($*$) – REAL (KIND=nag_wp) array *User Workspace*

MONITF is called with the arguments IUSER and RUSER as supplied to D03PPF/D03PPA. You should use the arrays IUSER and RUSER to supply information to MONITF.

MONITF must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D03PPF/D03PPA is called. Arguments denoted as *Input* must **not** be changed by this procedure.

32: RSAVE(LRSAVE) – REAL (KIND=nag_wp) array *Communication Array*

If IND $= 0$, RSAVE need not be set on entry.

If IND $= 1$, RSAVE must be unchanged from the previous call to the routine because it contains required information about the iteration.

33: LRSAVE – INTEGER *Input*

On entry: the dimension of the array RSAVE as declared in the (sub)program from which D03PPF/D03PPA is called. Its size depends on the type of matrix algebra selected.

If LAOPT $=$ 'F', LRSAVE $\geq$ NEQN $\times$ NEQN + NEQN + $nwkres + lenode$.

If LAOPT $=$ 'B', LRSAVE $\geq (3 \times mlu + 1) \times$ NEQN + $nwkres + lenode$.

If LAOPT $=$ 'S', LRSAVE $\geq 4 \times$ NEQN + 11 $\times$ NEQN/2 + 1 + $nwkres + lenode$.

Where

mlu is the lower or upper half bandwidths such that
$mlu = 2 \times \text{NPDE} - 1$, for PDE problems only; or
$mlu = \text{NEQN} - 1$, for coupled PDE/ODE problems.

$$nwkres = \begin{cases} \text{NPDE} \times (3 \times \text{NPDE} + 6 \times \text{NXI} + \text{NPTS} + 15) + \text{NXI} + \text{NCODE} + 7 \times \text{NPTS} + \text{NXFIX} + 1, & \text{when NCODE} > 0 \text{ and NXI} > 0; \\ \text{NPDE} \times (3 \times \text{NPDE} + \text{NPTS} + 21) + \text{NCODE} + 7 \times \text{NPTS} + \text{NXFIX} + 2, & \text{when NCODE} > 0 \text{ and NXI} = 0; \text{ or} \\ \text{NPDE} \times (3 \times \text{NPDE} + \text{NPTS} + 21) + 7 \times \text{NPTS} + \text{NXFIX} + 3, & \text{when NCODE} = 0. \end{cases}$$

$$lenode = \begin{cases} (6 + \text{int}(\text{ALGOPT}(2))) \times \text{NEQN} + 50, & \text{when the BDF method is used;} \\ 9 \times \text{NEQN} + 50, & \text{when the Theta method is used.} \end{cases}$$

Note: when using the sparse option, the value of LRSAVE may be too small when supplied to the integrator. An estimate of the minimum size of LRSAVE is printed on the current error message unit if ITRACE > 0 and the routine returns with IFAIL $= 15$.

34: ISAVE(LISAVE) – INTEGER array *Communication Array*

If IND $= 0$, ISAVE need not be set on entry.

If IND $= 1$, ISAVE must be unchanged from the previous call to the routine because it contains required information about the iteration required for subsequent calls. In particular:

ISAVE(1)
Contains the number of steps taken in time.

ISAVE(2)
Contains the number of residual evaluations of the resulting ODE system used. One such evaluation involves computing the PDE functions at all the mesh points, as well as one evaluation of the functions in the boundary conditions.

ISAVE(3)
Contains the number of Jacobian evaluations performed by the time integrator.

ISAVE(4)
Contains the order of the ODE method last used in the time integration.

ISAVE(5)
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 *Input*

On entry: the dimension of the array ISAVE as declared in the (sub)program from which D03PPF/D03PPA is called.

Its size depends on the type of matrix algebra selected:

if LAOPT = 'B', LISAVE $\geq$ NEQN + 25 + NXFIX;

if LAOPT = 'F', LISAVE $\geq$ 25 + NXFIX;

if LAOPT = 'S', LISAVE $\geq 25 \times$ NEQN + 25 + NXFIX.

Note: when using the sparse option, the value of LISAVE may be too small when supplied to the integrator. An estimate of the minimum size of LISAVE is printed on the current error message unit if ITRACE > 0 and the routine returns with IFAIL $= 15$.

36: ITASK – INTEGER *Input*

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

ITASK $= 1$
Normal computation of output values U at $t =$ TOUT.

ITASK $= 2$
One step and return.

ITASK = 3
Stop at first internal integration point at or beyond $t =$ TOUT.

ITASK = 4
Normal computation of output values U at $t =$ TOUT but without overshooting $t = t_{\text{crit}}$ 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 *Input*

On entry: the level of trace information required from D03PPF/D03PPA and the underlying ODE solver:

ITRACE ≤ -1
No output is generated.

ITRACE = 0
Only warning messages from the PDE solver are printed on the current error message unit (see X04AAF).

ITRACE = 1
Output from the underlying ODE solver is printed on the current advisory message unit (see X04ABF). This output contains details of Jacobian entries, the nonlinear iteration and the time integration during the computation of the ODE system.

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

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

You are advised to set ITRACE = 0, unless you are experienced with Sub-chapter D02M–N.

38: IND – INTEGER *Input/Output*

On entry: must be set to 0 or 1.

IND = 0
Starts or restarts the integration in time.

IND = 1
Continues the integration after an earlier exit from the routine. In this case, only the arguments TOUT and IFAIL and the remeshing arguments NRMESH, DXMESH, TRMESH, XRATIO and CON may be reset between calls to D03PPF/D03PPA.

Constraint: $0 \leq$ IND ≤ 1.

On exit: IND = 1.

39: IFAIL – INTEGER *Input/Output*

Note: *for D03PPA, IFAIL does not occur in this position in the argument list. See the additional arguments described below.*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the

recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

Note: *the following are additional arguments for specific use with D03PPA. Users of D03PPF therefore need not read the remainder of this description.*

40: IUSER$(*)$ – INTEGER array *User Workspace*
41: RUSER$(*)$ – REAL (KIND=nag_wp) array *User Workspace*

IUSER and RUSER are not used by D03PPF/D03PPA, but are passed directly to PDEDEF, BNDARY, UVINIT, ODEDEF and MONITF and should be used to pass information to these routines.

42: CWSAV(10) – CHARACTER(80) array *Communication Array*

43: LWSAV(100) – LOGICAL array *Communication Array*

44: IWSAV(505) – INTEGER array *Communication Array*

45: RWSAV(1100) – REAL (KIND=nag_wp) array *Communication Array*

46: IFAIL – INTEGER *Input/Output*

Note: see the argument description for IFAIL above.

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, TOUT $-$ TS is too small,
or ITASK $\neq$ 1, 2, 3, 4 or 5,
or M $\neq$ 0, 1 or 2,
or at least one of the coupling points defined in array XI is outside the interval $[\mathrm{X}(1), \mathrm{X}(\mathrm{NPTS})]$,
or M > 0 and X$(1) <$ 0.0,
or NPTS < 3,
or NPDE < 1,
or NORM $\neq$ 'A' or 'M',
or LAOPT $\neq$ 'F', 'B' or 'S',
or ITOL $\neq$ 1, 2, 3 or 4,
or IND $\neq$ 0 or 1,
or mesh points X(i) are badly ordered,
or LRSAVE is too small,
or LISAVE is too small,
or NCODE and NXI are incorrectly defined,
or an element of RTOL or ATOL < 0.0,
or corresponding elements of RTOL and ATOL are both 0.0,
or NEQN $\neq$ NPDE $\times$ NPTS $+$ NCODE,
or NXFIX not in the range 0 to NPTS $- 2$,
or fixed mesh point(s) do not coincide with any of the user-supplied mesh points,
or DXMESH < 0.0,
or IPMINF $\neq$ 0, 1 or 2,
or XRATIO $\leq$ 1.0,

or CON not in the range $0.1/(\text{NPTS} - 1)$ to $10/(\text{NPTS} - 1)$.

IFAIL $= 2$

The underlying ODE solver cannot make any further progress, with the values of ATOL and RTOL, across the integration range from the current point $t =$ TS. The components of U contain the computed values at the current point $t =$ TS.

IFAIL $= 3$

In the underlying ODE solver, there were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as $t =$ TS. The problem may have a singularity, or the error requirement may be inappropriate.

IFAIL $= 4$

In setting up the ODE system, the internal initialization routine was unable to initialize the derivative of the ODE system. This could be due to the fact that IRES was repeatedly set to 3 in at least PDEDEF, BNDARY or ODEDEF, when the residual in the underlying ODE solver was being evaluated.

IFAIL $= 5$

In solving the ODE system, a singular Jacobian has been encountered. You should check your problem formulation.

IFAIL $= 6$

When evaluating the residual in solving the ODE system, IRES was set to 2 in at least PDEDEF, BNDARY or ODEDEF. Integration was successful as far as $t =$ TS.

IFAIL $= 7$

The values of ATOL and RTOL are so small that the routine is unable to start the integration in time.

IFAIL $= 8$

In one of PDEDEF, BNDARY or ODEDEF, IRES was set to an invalid value.

IFAIL $= 9$ (D02NNF)

A serious error has occurred in an internal call to the specified routine. Check the problem specification and all arguments and array dimensions. Setting ITRACE $= 1$ may provide more information. If the problem persists, contact NAG.

IFAIL $= 10$

The required task has been completed, but it is estimated that a small change in ATOL and RTOL is unlikely to produce any change in the computed solution. (Only applies when you are not operating in one step mode, that is when ITASK $\neq 2$ or 5.)

IFAIL $= 11$

An error occurred during Jacobian formulation of the ODE system (a more detailed error description may be directed to the current error message unit). If using the sparse matrix algebra option, the values of ALGOPT(29) and ALGOPT(30) may be inappropriate.

IFAIL $= 12$

In solving the ODE system, the maximum number of steps specified in ALGOPT(15) have been taken.

IFAIL $= 13$

Some error weights w_i became zero during the time integration (see the description of ITOL). Pure relative error control (ATOL$(i) = 0.0$) was requested on a variable (the ith) which has become zero. The integration was successful as far as $t =$ TS.

IFAIL $= 14$

The flux function R_i was detected as depending on time derivatives, which is not permissible.

IFAIL $= 15$

When using the sparse option, the value of LISAVE or LRSAVE was not sufficient (more detailed information may be directed to the current error message unit).

IFAIL $= 16$

REMESH has been changed between calls to D03PPF/D03PPA, which is not permissible.

IFAIL $= 17$

The remeshing process has produced zero or negative mesh spacing. You are advised to check MONITF and to try adjusting the values of DXMESH, XRATIO and CON. Setting IPMINF $= 1$ may provide more information.

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

D03PPF/D03PPA 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

D03PPF/D03PPA is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

D03PPF/D03PPA 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 routine. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The argument specification allows you to include equations with only first-order derivatives in the space direction but there is no guarantee that the method of integration will be satisfactory for such systems. The position and nature of the boundary conditions in particular are critical in defining a stable problem. It may be advisable in such cases to reduce the whole system to first-order and to use the Keller box scheme routine D03PRF.

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

10 Example

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

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

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

The initial and boundary conditions are given by the exact solution

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

where

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

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

$$C = \frac{500}{E}(x - 0.375).$$

10.1 Program Text

the following program illustrates the use of D03PPF. An equivalent program illustrating the use of D03PPA is available with the supplied Library and is also available from the NAG web site.

```
!   D03PPF Example Program Text
!   Mark 26 Release. NAG Copyright 2016.

    Module d03ppfe_mod

!     D03PPF Example Program Module:
!            Parameters and User-defined Routines

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Accessibility Statements ..
      Private
      Public                              :: bndary, exact, monitf, pdedef,       &
                                             uvinit
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter       :: four = 4.0_nag_wp
      Real (Kind=nag_wp), Parameter       :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter       :: ptone = 0.1_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: half = 0.5_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: two = 2.0_nag_wp
      Real (Kind=nag_wp), Parameter, Public :: zero = 0.0_nag_wp
      Integer, Parameter, Public          :: itrace = 0, m = 0, ncode = 0,        &
                                             nin = 5, nout = 6, npde = 1,         &
```

```
                                       nxfix = 0, nxi = 0
!     .. Local Scalars ..
      Real (Kind=nag_wp), Public, Save :: e
    Contains
      Subroutine uvinit(npde,npts,nxi,x,xi,u,ncode,v)

!       .. Scalar Arguments ..
        Integer, Intent (In)           :: ncode, npde, npts, nxi
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: u(npde,npts), v(ncode)
        Real (Kind=nag_wp), Intent (In) :: x(npts), xi(nxi)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: a, b, c, t
        Integer                        :: i
!       .. Intrinsic Procedures ..
        Intrinsic                      :: exp
!       .. Executable Statements ..
        t = zero
        Do i = 1, npts
          a = (x(i)-0.25_nag_wp-0.75_nag_wp*t)/(four*e)
          b = (0.9_nag_wp*x(i)-0.325_nag_wp-0.495_nag_wp*t)/(two*e)
          If (a>zero .And. a>b) Then
            a = exp(-a)
            c = (0.8_nag_wp*x(i)-0.4_nag_wp-0.24_nag_wp*t)/(four*e)
            c = exp(c)
            u(1,i) = (half+ptone*c+a)/(one+c+a)
          Else If (b>zero .And. b>=a) Then
            b = exp(-b)
            c = (-0.8_nag_wp*x(i)+0.4_nag_wp+0.24_nag_wp*t)/(four*e)
            c = exp(c)
            u(1,i) = (ptone+half*c+b)/(one+c+b)
          Else
            a = exp(a)
            b = exp(b)
            u(1,i) = (one+half*a+ptone*b)/(one+a+b)
          End If
        End Do
        Return
      End Subroutine uvinit
      Subroutine pdedef(npde,t,x,u,ux,ncode,v,vdot,p,q,r,ires)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t, x
        Integer, Intent (Inout)        :: ires
        Integer, Intent (In)           :: ncode, npde
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: p(npde,npde), q(npde), r(npde)
        Real (Kind=nag_wp), Intent (In) :: u(npde), ux(npde), v(ncode),        &
                                           vdot(ncode)
!       .. Executable Statements ..
        p(1,1) = one
        r(1) = e*ux(1)
        q(1) = u(1)*ux(1)
        Return
      End Subroutine pdedef
      Subroutine bndary(npde,t,u,ux,ncode,v,vdot,ibnd,beta,gamma,ires)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (In)           :: ibnd, ncode, npde
        Integer, Intent (Inout)        :: ires
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: beta(npde), gamma(npde)
        Real (Kind=nag_wp), Intent (In) :: u(npde), ux(npde), v(ncode),        &
                                           vdot(ncode)
!       .. Local Scalars ..
        Real (Kind=nag_wp)             :: a, b, c, ue, x
!       .. Intrinsic Procedures ..
        Intrinsic                      :: exp
!       .. Executable Statements ..
        beta(1) = zero
```

```
        If (ibnd==0) Then
          x = zero
          a = (x-0.25_nag_wp-0.75_nag_wp*t)/(four*e)
          b = (0.9_nag_wp*x-0.325_nag_wp-0.495_nag_wp*t)/(two*e)
          If (a>zero .And. a>b) Then
            a = exp(-a)
            c = (0.8_nag_wp*x-0.4_nag_wp-0.24_nag_wp*t)/(four*e)
            c = exp(c)
            ue = (half+ptone*c+a)/(one+c+a)
          Else If (b>zero .And. b>=a) Then
            b = exp(-b)
            c = (-0.8_nag_wp*x+0.4_nag_wp+0.24_nag_wp*t)/(four*e)
            c = exp(c)
            ue = (ptone+half*c+b)/(one+c+b)
          Else
            a = exp(a)
            b = exp(b)
            ue = (one+half*a+ptone*b)/(one+a+b)
          End If
        Else
          x = one
          a = (x-0.25_nag_wp-0.75_nag_wp*t)/(four*e)
          b = (0.9_nag_wp*x-0.325_nag_wp-0.495_nag_wp*t)/(two*e)
          If (a>zero .And. a>b) Then
            a = exp(-a)
            c = (0.8_nag_wp*x-0.4_nag_wp-0.24_nag_wp*t)/(four*e)
            c = exp(c)
            ue = (half+ptone*c+a)/(one+c+a)
          Else If (b>zero .And. b>=a) Then
            b = exp(-b)
            c = (-0.8_nag_wp*x+0.4_nag_wp+0.24_nag_wp*t)/(four*e)
            c = exp(c)
            ue = (ptone+half*c+b)/(one+c+b)
          Else
            a = exp(a)
            b = exp(b)
            ue = (one+half*a+ptone*b)/(one+a+b)
          End If
        End If
        gamma(1) = u(1) - ue
        Return
      End Subroutine bndary
      Subroutine monitf(t,npts,npde,x,u,r,fmon)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (In)            :: npde, npts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: fmon(npts)
        Real (Kind=nag_wp), Intent (In) :: r(npde,npts), u(npde,npts), x(npts)
!       .. Local Scalars ..
        Real (Kind=nag_wp)              :: drdx, h
        Integer                         :: i
!       .. Intrinsic Procedures ..
        Intrinsic                       :: abs
!       .. Executable Statements ..
        fmon(1) = abs((r(1,2)-r(1,1))/((x(2)-x(1))*half))
        Do i = 2, npts - 1
          h = (x(i+1)-x(i-1))*half
!         Second derivative ..
          drdx = (r(1,i+1)-r(1,i))/h
          fmon(i) = abs(drdx)
        End Do
        fmon(npts) = fmon(npts-1)
        Return
      End Subroutine monitf
      Subroutine exact(t,x,npts,u)

!       Exact solution (for comparison purposes)

!       .. Scalar Arguments ..
```

```
        Real (Kind=nag_wp), Intent (In) :: t
        Integer, Intent (In)            :: npts
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out) :: u(npts)
        Real (Kind=nag_wp), Intent (In) :: x(npts)
!       .. Local Scalars ..
        Real (Kind=nag_wp)              :: a, b, c
        Integer                         :: i
!       .. Intrinsic Procedures ..
        Intrinsic                       :: exp
!       .. Executable Statements ..
        Do i = 1, npts
          a = (x(i)-0.25_nag_wp-0.75_nag_wp*t)/(four*e)
          b = (0.9_nag_wp*x(i)-0.325_nag_wp-0.495_nag_wp*t)/(two*e)
          If (a>zero .And. a>b) Then
            a = exp(-a)
            c = (0.8_nag_wp*x(i)-0.4_nag_wp-0.24_nag_wp*t)/(four*e)
            c = exp(c)
            u(i) = (half+ptone*c+a)/(one+c+a)
          Else If (b>zero .And. b>=a) Then
            b = exp(-b)
            c = (-0.8_nag_wp*x(i)+0.4_nag_wp+0.24_nag_wp*t)/(four*e)
            c = exp(c)
            u(i) = (ptone+half*c+b)/(one+c+b)
          Else
            a = exp(a)
            b = exp(b)
            u(i) = (one+half*a+ptone*b)/(one+a+b)
          End If
        End Do
        Return
      End Subroutine exact
    End Module d03ppfe_mod
    Program d03ppfe

!     D03PPF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d03pck, d03ppf, d03pzf, nag_wp
      Use d03ppfe_mod, Only: bndary, e, exact, half, itrace, m, monitf, ncode, &
                             nin, nout, npde, nxfix, nxi, pdedef, two, uvinit, &
                             zero
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)              :: con, dx, dxmesh, tout, trmesh, ts,   &
                                         x0, xmid, xratio
      Integer                         :: i, ifail, ind, intpts, ipminf, it,   &
                                         itask, itol, itype, lenode, lisave,  &
                                         lrsave, neqn, npts, nrmesh, nwkres
      Logical                         :: remesh, theta
      Character (1)                   :: laopt, norm
!     .. Local Arrays ..
      Real (Kind=nag_wp)              :: algopt(30), atol(1), rtol(1),        &
                                         xfix(1), xi(1)
      Real (Kind=nag_wp), Allocatable :: rsave(:), u(:), ue(:), uout(:,:,:),  &
                                         x(:), xout(:)
      Integer, Allocatable            :: isave(:)
!     .. Intrinsic Procedures ..
      Intrinsic                       :: min, real
!     .. Executable Statements ..
      Write (nout,*) 'D03PPF Example Program Results'
!     Skip heading in data file
      Read (nin,*)
      Read (nin,*) npts, intpts, itype
      lisave = 25 + nxfix
      neqn = npde*npts + ncode
      nwkres = npde*(npts+3*npde+21) + 7*npts + nxfix + 3
      lenode = 11*neqn + 50
      lrsave = neqn*neqn + neqn + nwkres + lenode
```

```
      Allocate (u(neqn),ue(intpts),uout(npde,intpts,itype),rsave(lrsave),      &
        x(npts),xout(intpts),isave(lisave))
      Read (nin,*) itol
      Read (nin,*) atol(1), rtol(1)
      Read (nin,*) e

!     Initialize mesh
      Do i = 1, npts
        x(i) = real(i-1,kind=nag_wp)/real(npts-1,kind=nag_wp)
      End Do

!     Set remesh parameters
      remesh = .True.
      nrmesh = 3
      dxmesh = half
      con = two/real(npts-1,kind=nag_wp)
      xratio = 1.5_nag_wp
      ipminf = 0

      xi(1) = zero
      norm = 'A'
      laopt = 'F'
      ind = 0
      itask = 1

!     Set theta to .TRUE. if the Theta integrator is required

      theta = .False.
      algopt(1:30) = zero
      If (theta) Then
        algopt(1) = two
      Else
        algopt(1) = zero
      End If

!     Loop over output value of t

      ts = zero
      tout = zero
      Do it = 1, 5
        xmid = half + half*tout
        tout = 0.2_nag_wp*real(it,kind=nag_wp)

!       ifail: behaviour on error exit
!              =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
        ifail = 0
        Call d03ppf(npde,m,ts,tout,pdedef,bndary,uvinit,u,npts,x,ncode,d03pck, &
          nxi,xi,neqn,rtol,atol,itol,norm,laopt,algopt,remesh,nxfix,xfix,      &
          nrmesh,dxmesh,trmesh,ipminf,xratio,con,monitf,rsave,lrsave,isave,    &
          lisave,itask,itrace,ind,ifail)

        If (it==1) Then
          Write (nout,99998) atol, npts
          Write (nout,99993) nrmesh
          Write (nout,99992) e
          Write (nout,*)
        End If

!       Set output points ..
        dx = 0.1_nag_wp
        If (tout>half) Then
          dx = 0.05_nag_wp
        End If

        x0 = xmid - half*real(intpts-1,kind=nag_wp)*dx
        Do i = 1, intpts
          xout(i) = x0
          x0 = x0 + dx
        End Do
        xout(intpts) = min(xout(intpts),x(npts))
```

```
        Write (nout,99999) ts
        Write (nout,99996) xout(1:intpts)

!       Interpolate at output points ..
        ifail = 0
        Call d03pzf(npde,m,u,npts,x,xout,intpts,itype,uout,ifail)

!       Check against exact solution ..
        Call exact(ts,xout,intpts,ue)

        Write (nout,99995) uout(1,1:intpts,1)
        Write (nout,99994) ue(1:intpts)

      End Do
      Write (nout,99997) isave(1), isave(2), isave(3), isave(5)

99999 Format (' T = ',F6.3)
99998 Format (/,/,'  Accuracy requirement =',E10.3,' Number of points = ',I3,  &
        /)
99997 Format (' Number of integration steps in time = ',I6,/,' Number o',      &
        'f function evaluations = ',I6,/,' Number of Jacobian eval',           &
        'uations =',I6,/,' Number of iterations = ',I6)
99996 Format (1X,'X           ',5F9.4)
99995 Format (1X,'Approx sol. ',5F9.4)
99994 Format (1X,'Exact  sol. ',5F9.4,/)
99993 Format (2X,'Remeshing every',I3,' time steps',/)
99992 Format (2X,'E =',F8.3)
    End Program d03ppfe
```

10.2 Program Data

```
D03PPF Example Program Data
 61  5  1                    : npts, intpts, itype
 1                           : itol
 0.5E-4  0.5E-4              : atol(1), rtol(1)
 0.5E-2                      : e
```

10.3 Program Results

```
 D03PPF Example Program Results


  Accuracy requirement = 0.500E-04 Number of points =  61

  Remeshing every  3 time steps

  E =   0.005

 T =  0.200
 X             0.3000   0.4000   0.5000   0.6000   0.7000
 Approx sol.   0.9968   0.7448   0.4700   0.1667   0.1018
 Exact  sol.   0.9967   0.7495   0.4700   0.1672   0.1015

 T =  0.400
 X             0.4000   0.5000   0.6000   0.7000   0.8000
 Approx sol.   1.0003   0.9601   0.4088   0.1154   0.1005
 Exact  sol.   0.9997   0.9615   0.4094   0.1157   0.1003

 T =  0.600
 X             0.6000   0.6500   0.7000   0.7500   0.8000
 Approx sol.   0.9966   0.9390   0.3978   0.1264   0.1037
 Exact  sol.   0.9964   0.9428   0.4077   0.1270   0.1033

 T =  0.800
 X             0.7000   0.7500   0.8000   0.8500   0.9000
 Approx sol.   1.0003   0.9872   0.5450   0.1151   0.1010
 Exact  sol.   0.9996   0.9878   0.5695   0.1156   0.1008

 T =  1.000
 X             0.8000   0.8500   0.9000   0.9500   1.0000
```

```
Approx sol.    1.0001   0.9961   0.7324   0.1245   0.1004
Exact  sol.    0.9999   0.9961   0.7567   0.1273   0.1004

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

Example Program
Solution of Burgers Equation using Moving Mesh