NAG Fortran Library Routine Document

D03PFF

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

D03PFF integrates a system of linear or nonlinear convection-diffusion equations in one space dimension, with optional source terms. The system must be posed in conservative form. Convection terms are discretised using a sophisticated upwind scheme involving a user-supplied numerical flux function based on the solution of a Riemann problem at each mesh point. The method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs), and the resulting system is solved using a backward differentiation formula (BDF) method.

2 Specification

```
 SUBROUTINE D03PFF(NPDE, TS, TOUT, PDEDEF, NUMFLX, BNDARY, U, NPTS, X,
1                  ACC, TSMAX, W, NW, IW, NIW, ITASK, ITRACE, IND, IFAIL)
 INTEGER           NPDE, NPTS, NW, IW(NIW), NIW, ITASK, ITRACE, IND,
1                  IFAIL
 real              TS, TOUT, U(NPDE,NPTS), X(NPTS), ACC(2), TSMAX, W(NW)
 EXTERNAL          PDEDEF, NUMFLX, BNDARY
```

3 Description

D03PFF integrates the system of convection-diffusion equations in conservative form:

$$\sum_{j=1}^{\text{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + \frac{\partial F_i}{\partial x} = C_i\frac{\partial D_i}{\partial x} + S_i, \tag{1}$$

or the hyperbolic convection-only system:

$$\frac{\partial U_i}{\partial t} + \frac{\partial F_i}{\partial x} = 0, \tag{2}$$

for $i = 1, 2, \ldots, \text{NPDE}$, $a \le x \le b$, $t \ge t_0$, where the vector U is the set of solution values

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

The functions $P_{i,j}$, F_i, C_i and S_i depend on x, t and U; and D_i depends on x, t, U and U_x, where U_x is the spatial derivative of U. Note that $P_{i,j}$, F_i, C_i and S_i must not depend on any space derivatives; and none of the functions may depend on time derivatives. In terms of conservation laws, F_i, $C_i\partial D_i/\partial x$ and S_i are the convective flux, diffusion and source terms respectively.

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 user-defined mesh $x_1, x_2, \ldots, x_{\text{NPTS}}$. The initial values of the functions $U(x,t)$ must be given at $t = t_0$.

The PDEs are approximated by a system of ODEs in time for the values of U_i at mesh points using a spatial discretisation method similar to the central-difference scheme used in D03PCF/D03PCA, D03PHF/D03PHA and D03PPF/D03PPA, but with the flux F_i replaced by a *numerical flux*, which is a representation of the flux taking into account the direction of the flow of information at that point (i.e., the direction of the characteristics). Simple central differencing of the numerical flux then becomes a sophisticated upwind scheme in which the correct direction of upwinding is automatically achieved.

The numerical flux vector, $\hat{F}_i$ say, must be calculated by the user in terms of the *left* and *right* values of the solution vector U (denoted by U_L and U_R respectively), at each mid-point of the mesh $x_{j-1/2} = (x_{j-1} + x_j)/2$ for $j = 2, 3, \ldots, \text{NPTS}$. The left and right values are calculated by D03PFF

from two adjacent mesh points using a standard upwind technique combined with a Van Leer slope-limiter (see LeVeque (1990)). The physically correct value for $\hat{F}_i$ is derived from the solution of the Riemann problem given by

$$\frac{\partial U_i}{\partial t} + \frac{\partial F_i}{\partial y} = 0, \tag{3}$$

where $y = x - x_{j-1/2}$, i.e., $y = 0$ corresponds to $x = x_{j-1/2}$, with discontinuous initial values $U = U_L$ for $y < 0$ and $U = U_R$ for $y > 0$, using an *approximate Riemann solver*. This applies for either of the systems (1) or (2); the numerical flux is independent of the functions $P_{i,j}$, C_i, D_i and S_i. A description of several approximate Riemann solvers can be found in LeVeque (1990) and Berzins *et al.* (1989). Roe's scheme (Roe (1981)) is perhaps the easiest to understand and use, and a brief summary follows. Consider the system of PDEs $U_t + F_x = 0$ or equivalently $U_t + AU_x = 0$. Provided the system is linear in U, i.e., the Jacobian matrix A does not depend on U, the numerical flux $\hat{F}$ is given by

$$\hat{F} = \tfrac{1}{2}(F_L + F_R) - \tfrac{1}{2}\sum_{k=1}^{\text{NPDE}} \alpha_k |\lambda_k| e_k, \tag{4}$$

where F_L (F_R) is the flux F calculated at the left (right) value of U, denoted by U_L (U_R); the λ_k are the eigenvalues of A; the e_k are the right eigenvectors of A; and the α_k are defined by

$$U_R - U_L = \sum_{k=1}^{\text{NPDE}} \alpha_k e_k. \tag{5}$$

An example is given in Section 9.

If the system is nonlinear, Roe's scheme requires that a linearized Jacobian is found (see Roe (1981)).

The functions $P_{i,j}$, C_i, D_i and S_i (but **not** F_i) must be specified in a subroutine PDEDEF supplied by the user. The numerical flux $\hat{F}_i$ must be supplied in a separate user-supplied subroutine NUMFLX. For problems in the form (2), the actual argument D03PFP may be used for PDEDEF (D03PFP is included in the NAG Fortran Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details). D03PFP sets the matrix with entries $P_{i,j}$ to the identity matrix, and the functions C_i, D_i and S_i to zero.

The boundary condition specification has sufficient flexibility to allow for different types of problems. For second-order problems i.e., D_i depending on U_x, a boundary condition is required for each PDE at both boundaries for the problem to be well-posed. If there are no second-order terms present, then the continuous PDE problem generally requires exactly one boundary condition for each PDE, that is NPDE boundary conditions in total. However, in common with most discretisation schemes for first-order problems, a *numerical boundary condition* is required at the other boundary for each PDE. In order to be consistent with the characteristic directions of the PDE system, the numerical boundary conditions must be derived from the solution inside the domain in some manner (see below). Both types of boundary conditions must be supplied by the user, i.e., a total of NPDE conditions at each boundary point.

The position of each boundary condition should be chosen with care. In simple terms, if information is flowing into the domain then a physical boundary condition is required at that boundary, and a numerical boundary condition is required at the other boundary. In many cases the boundary conditions are simple, e.g., for the linear advection equation. In general the user should calculate the characteristics of the PDE system and specify a physical boundary condition for each of the characteristic variables associated with incoming characteristics, and a numerical boundary condition for each outgoing characteristic.

A common way of providing numerical boundary conditions is to extrapolate the characteristic variables from the inside of the domain. Note that only linear extrapolation is allowed in this routine (for greater flexibility the routine D03PLF should be used). For problems in which the solution is known to be uniform (in space) towards a boundary during the period of integration then extrapolation is unneccesary; the numerical boundary condition can be supplied as the known solution at the boundary. Examples can be found in Section 9.

The boundary conditions must be specified in a subroutine BNDARY (provided by the user) in the form

$$G_i^L(x, t, U) = 0 \quad \text{at} \quad x = a, \quad i = 1, 2, ..., \text{NPDE}, \tag{6}$$

at the left-hand boundary, and

$$G_i^R(x,t,U) = 0 \quad \text{at} \quad x = b, \quad i = 1,2,...,\text{NPDE}, \tag{7}$$

at the right-hand boundary.

Note that spatial derivatives at the boundary are not passed explicitly to the subroutine BNDARY, but they can be calculated using values of U at and adjacent to the boundaries if required. However, it should be noted that instabilities may occur if such one-sided differencing opposes the characteristic direction at the boundary.

The problem is subject to the following restrictions:

(i) $P_{i,j}$, F_i, C_i and S_i must not depend on any space derivatives;

(ii) $P_{i,j}$, F_i, C_i, D_i and S_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 $P_{i,j}$, C_i, D_i and S_i is done by calling the routine PDEDEF at a point approximately midway between each pair of mesh points in turn. Any discontinuities in these functions **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\text{NPTS}}$;

(v) 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;

In total there are NPDE $\times$ NPTS ODEs in the time direction. This system is then integrated forwards in time using a BDF method.

For further details of the algorithm, see Pennington and Berzins (1994) and the references therein.

4 References

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

Hirsch C (1990) *Numerical Computation of Internal and External Flows, Volume 2: Computational Methods for Inviscid and Viscous Flows* John Wiley

LeVeque R J (1990) *Numerical Methods for Conservation Laws* Birkhäuser Verlag

Pennington S V and Berzins M (1994) New NAG Library software for first-order partial differential equations *ACM Trans. Math. Softw.* **20** 63–99

Roe P L (1981) Approximate Riemann solvers, parameter vectors, and difference schemes *J. Comput. Phys.* **43** 357–372

5 Parameters

1: NPDE – INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE ≥ 1.

2: TS – ***real*** *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.

3: TOUT – ***real*** *Input*

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

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

PDEDEF must evaluate the functions $P_{i,j}$, C_i, D_i and S_i which partially define the system of PDEs. $P_{i,j}$, C_i and S_i may depend on x, t and U; D_i may depend on x, t, U and U_x. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PFF. The actual argument D03PFP may be used for PDEDEF for problems in the form (2) (D03PFP is included in the NAG Fortran Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details).

Its specification is:

```
      SUBROUTINE PDEDEF(NPDE, T, X, U, UX, P, C, D, S, IRES)
      INTEGER           NPDE, IRES
      real              T, X, U(NPDE), UX(NPDE), P(NPDE,NPDE), C(NPDE),
     1                  D(NPDE), S(NPDE)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – ***real*** *Input*

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

3: X – ***real*** *Input*

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

4: U(NPDE) – ***real*** 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*** array *Input*

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

6: P(NPDE,NPDE) – ***real*** array *Output*

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

7: C(NPDE) – ***real*** array *Output*

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

8: D(NPDE) – ***real*** array *Output*

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

9: S(NPDE) – ***real*** array *Output*

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

10: IRES – INTEGER *Input/Output*

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, the user 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)program 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. The user may wish to set IRES = 3 when a physically meaningless input or output value has been generated. If the user consecutively sets IRES = 3, then D03PFF returns to the calling (sub)program with the error indicator set to IFAIL = 4.

PDEDEF must be declared as EXTERNAL in the (sub)program from which D03PFF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

NUMFLX must supply the numerical flux for each PDE given the *left* and *right* values of the solution vector U. NUMFLX is called approximately midway between each pair of mesh points in turn by D03PFF.

Its specification is:

```
SUBROUTINE NUMFLX(NPDE, T, X, ULEFT, URIGHT, FLUX, IRES)
INTEGER           NPDE, IRES
real              T, X, ULEFT(NPDE), URIGHT(NPDE), FLUX(NPDE)
```

1: NPDE – INTEGER *Input*

On entry: the number of PDEs in the system.

2: T – ***real*** *Input*

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

3: X – ***real*** *Input*

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

4: ULEFT(NPDE) – ***real*** array *Input*

On entry: ULEFT(i) contains the *left* value of the component $U_i(x)$, for $i = 1, 2, \ldots, \text{NPDE}$.

5: URIGHT(NPDE) – ***real*** array *Input*

On entry: URIGHT(i) contains the *right* value of the component $U_i(x)$, for $i = 1, 2, \ldots, \text{NPDE}$.

6: FLUX(NPDE) – ***real*** array *Output*

On exit: FLUX(i) must be set to the numerical flux $\hat{F}_i$, for $i = 1, 2, \ldots, \text{NPDE}$.

7: IRES – INTEGER *Input/Output*

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, the user 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)program 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. The user may wish to set IRES = 3 when a physically meaningless input or output value has been generated. If the user

consecutively sets IRES $= 3$, then D03PFF returns to the calling (sub)program with the error indicator set to IFAIL $= 4$.

NUMFLX must be declared as EXTERNAL in the (sub)program from which D03PFF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

BNDARY must evaluate the functions G_i^L and G_i^R which describe the physical and numerical boundary conditions, as given by (6) and (7).

Its specification is:

```
SUBROUTINE BNDARY(NPDE, NPTS, T, X, U, IBND, G, IRES)
INTEGER           NPDE, NPTS, IBND, IRES
real              T, X(NPTS), U(NPDE,3), G(NPDE)
```

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: T – ***real*** *Input*

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

4: X(NPTS) – ***real*** array *Input*

On entry: the mesh points in the spatial direction. X(1) corresponds to the left-hand boundary, a, and X(NPTS) corresponds to the right-hand boundary, b.

5: U(NPDE,3) – ***real*** array *Input*

On entry: contains the value of solution components in the boundary region. If IBND $= 0$, then U(i, j) contains the value of the component $U_i(x, t)$ at $x =$ X(j), for $i = 1, 2, \ldots,$ NPDE; $j = 1, 2, 3$. If IBND $\neq 0$, then U(i, j) contains the value of the component $U_i(x, t)$ at $x =$ X(NPTS $- j + 1$), for $i = 1, 2, \ldots,$ NPDE; $j = 1, 2, 3$.

6: IBND – INTEGER *Input*

On entry: specifies which boundary conditions are to be evaluated. If IBND $= 0$, then BNDARY must evaluate the left-hand boundary condition at $x = a$. If IBND $\neq 0$, then BNDARY must evaluate the right-hand boundary condition at $x = b$.

7: G(NPDE) – ***real*** array *Output*

On exit: G(i) must contain the ith component of either G^L or G^R in (6) and (7), depending on the value of IBND, for $i = 1, 2, \ldots,$ NPDE.

8: IRES – INTEGER *Input/Output*

On entry: set to -1 or 1.

On exit: should usually remain unchanged. However, the user 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)program 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. The user may wish to set IRES = 3 when a physically meaningless input or output value has been generated. If the user consecutively sets IRES = 3, then D03PFF returns to the calling (sub)program with the error indicator set to IFAIL = 4.

BNDARY must be declared as EXTERNAL in the (sub)program from which D03PFF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: U(NPDE,NPTS) – ***real*** array *Input/Output*

On entry: U(i,j) must contain the initial value of $U_i(x,t)$ at $x = \mathrm{X}(j)$ and $t = \mathrm{TS}$; for $i = 1,2,\ldots,\mathrm{NPDE}$; $j = 1,2,\ldots,\mathrm{NPTS}$.

On exit: U(i,j) will contain the computed solution $U_i(x,t)$ at $x = \mathrm{X}(j)$ and $t = \mathrm{TS}$; for $i = 1,2,\ldots,\mathrm{NPDE}$; $j = 1,2,\ldots,\mathrm{NPTS}$.

8: NPTS – INTEGER *Input*

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

Constraint: NPTS ≥ 3.

9: X(NPTS) – ***real*** array *Input*

On entry: the 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: $\mathrm{X}(1) < \mathrm{X}(2) < \ldots < \mathrm{X}(\mathrm{NPTS})$.

10: ACC(2) – ***real*** array *Input*

On entry: the components of ACC contain the relative and absolute error tolerances used in the local error test in the time integration.

If E(i,j) is the estimated error for U_i at the jth mesh point, the error test is

$$\mathrm{E}(i,j) = \mathrm{ACC}(1) \times \mathrm{U}(i,j) + \mathrm{ACC}(2).$$

Constraint: ACC(1) and ACC$(2) \geq 0.0$ (but not both zero).

11: TSMAX – ***real*** *Input*

On entry: the maximum absolute step size to be allowed in the time integration. If TSMAX = 0.0 then no maximum is imposed.

Constraint: TSMAX ≥ 0.0.

12: W(NW) – ***real*** array *Workspace*

13: NW – INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PFF is called.

Constraint:

$$\mathrm{NW} \geq (11 + 9 \times \mathrm{NPDE}) \times \mathrm{NPDE} \times \mathrm{NPTS} + (32 + 3 \times \mathrm{NPDE}) \times \mathrm{NPDE} + 7 \times \mathrm{NPTS} + 54.$$

14: IW(NIW) – INTEGER array *Output*

On exit: the following components of the array IW concern the efficiency of the integration.

IW(1) contains the number of steps taken in time.

IW(2) 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.

IW(3) contains the number of Jacobian evaluations performed by the time integrator.

IW(4) contains the order of the ODE method last used in the time integration.

IW(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.

15: NIW – INTEGER *Input*

On entry: the dimension of the array IW.

Constraint: NIW $\geq$ NPDE $\times$ NPTS + 24.

16: ITASK – INTEGER *Input*

On entry: the task to be performed by the ODE integrator. The permitted values of ITASK and their meanings are detailed below:

ITASK = 1

normal computation of output values U at $t =$ TOUT (by overshooting and interpolating).

ITASK = 2

take one step in the time direction and return.

ITASK = 3

stop at first internal integration point at or beyond $t =$ TOUT.

Constraint: $1 \leq$ ITASK ≤ 3.

17: ITRACE – INTEGER *Input*

On entry: the level of trace information required from D03PFF and the underlying ODE solver. ITRACE may take the value -1, 0, 1, 2, or 3. If ITRACE < -1, then -1 is assumed and similarly if ITRACE > 3, then 3 is assumed. If ITRACE $= -1$, no output is generated. If ITRACE $= 0$, only warning messages from the PDE solver are printed on the current error message unit (see X04AAF). If ITRACE > 0, then 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. The advisory messages are given in greater detail as ITRACE increases. Users are advised to set ITRACE $= 0$, unless they are experienced with the sub-chapter D02M–N of the NAG Fortran Library.

18: IND – INTEGER *Input/Output*

On entry: IND 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 parameters TOUT and IFAIL should be reset between calls to D03PFF.

Constraint: $0 \leq$ IND ≤ 1.

On exit: IND = 1.

19: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. Users who are unfamiliar with this parameter should refer to Chapter P01 for details.

On exit: IFAIL $= 0$ unless the routine detects an error (see Section 6).

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, for users not familiar with this parameter the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

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, TS $\geq$ TOUT,
or TOUT $-$ TS is too small,
or ITASK $\neq 1$, 2, or 3,
or NPTS < 3,
or NPDE < 1,
or IND $\neq 0$ or 1,
or incorrectly defined user mesh, i.e., X$(i) \geq$ X$(i+1)$ for some $i = 1, 2, \ldots,$ NPTS $- 1$,
or NW or NIW are too small,
or IND $= 1$ on initial entry to D03PFF,
or ACC(1) or ACC$(2) < 0.0$,
or ACC(1) or ACC(2) are both zero,
or TSMAX < 0.0.

IFAIL $= 2$

The underlying ODE solver cannot make any further progress, with the values of ACC, 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. Incorrect specification of boundary conditions may also result in this error.

IFAIL $= 4$

In setting up the ODE system, the internal initialisation routine was unable to initialise the derivative of the ODE system. This could be due to the fact that IRES was repeatedly set to 3 in one of the user-supplied subroutines PDEDEF, NUMFLX or BNDARY when the residual in the underlying ODE solver was being evaluated. Incorrect specification of boundary conditions may also result in this error.

IFAIL $= 5$

In solving the ODE system, a singular Jacobian has been encountered. Check the problem formulation.

IFAIL $= 6$

When evaluating the residual in solving the ODE system, IRES was set to 2 in at least one of the user-supplied subroutines PDEDEF, NUMFLX or BNDARY. Integration was successful as far as $t =$ TS.

IFAIL $= 7$

The values of ACC(1) and ACC(2) are so small that the routine is unable to start the integration in time.

IFAIL $= 8$

In one of the user-supplied routines, PDEDEF, NUMFLX or BNDARY, IRES was set to an invalid value.

IFAIL $= 9$

A serious error has occurred in an internal call to D02NNF. Check problem specification and all parameters 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 the values of ACC is unlikely to produce any change in the computed solution. (Only applies when the user is not operating in one step mode, that is when ITASK $\neq 2$.)

IFAIL $= 11$

An error occurred during Jacobian formulation of the ODE system (a more detailed error description may be directed to the current advisory message unit when ITRACE ≥ 1).

IFAIL $= 12$

Not applicable.

IFAIL $= 13$

Not applicable.

IFAIL $= 14$

One or more of the functions $P_{i,j}$, D_i or C_i was detected as depending on time derivatives, which is not permissible.

7 Accuracy

The routine 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 components of the accuracy parameter, ACC.

8 Further Comments

The routine is designed to solve systems of PDEs in conservative form, with optional source terms which are independent of space derivatives, and optional second-order diffusion terms. The use of the routine to solve systems which are not naturally in this form is discouraged, and users are advised to use one of the central-difference scheme routines for such problems.

Users should be aware of the stability limitations for hyperbolic PDEs. For most problems with small error tolerances the ODE integrator does not attempt unstable time steps, but in some cases a maximum time step should be imposed using TSMAX. It is worth experimenting with this parameter, particularly if the integration appears to progress unrealistically fast (with large time steps). Setting the maximum time step to the minimum mesh size is a safe measure, although in some cases this may be too restrictive.

Problems with source terms should be treated with caution, as it is known that for large source terms stable and reasonable looking solutions can be obtained which are in fact incorrect, exhibiting non-physical

speeds of propagation of discontinuities (typically one spatial mesh point per time step). It is essential to employ a very fine mesh for problems with source terms and discontinuities, and to check for non-physical propagation speeds by comparing results for different mesh sizes. Further details and an example can be found in Pennington and Berzins (1994).

The time taken by the routine depends on the complexity of the system and on the accuracy requested.

9 Example

For this routine two examples are presented, Section 9.1 of the documents for D03PFF and D03PFF. In the example programs distributed to sites, there is a single example program for D03PFF, with a main program:

```
*     D03PFF Example Program Text
*     Mark 17 Release. NAG Copyright 1995.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PFF Example Program Results'
      CALL EX1
      CALL EX2
      STOP
      END
```

The code to solve the two example problems is given in the subroutines EX1 and EX2, in D03PFF and D03PFF respectively.

9.1 Example 1

This example is a simple first-order system which illustrates the calculation of the numerical flux using Roe's approximate Riemann solver, and the specification of numerical boundary conditions using extrapolated characteristic variables. The PDEs are

$$\frac{\partial U_1}{\partial t} + \frac{\partial U_1}{\partial x} + \frac{\partial U_2}{\partial x} = 0,$$

$$\frac{\partial U_2}{\partial t} + 4\frac{\partial U_1}{\partial x} + \frac{\partial U_2}{\partial x} = 0,$$

for $x \in [0,1]$ and $t \geq 0$. The PDEs have an exact solution given by

$$U_1(x,t) = \frac{1}{2}\{\exp(x+t) + \exp(x-3t)\} + \frac{1}{4}\{\sin(2\pi(x-3t)^2) - \sin(2\pi(x+t)^2)\} + 2t^2 - 2xt,$$

$$U_2(x,t) = \exp(x-3t) - \exp(x+t) + \frac{1}{2}\{\sin(2\pi(x-3t)^2) + \sin(2\pi(x-3t)^2)\} + x^2 + 5t^2 - 2xt.$$

The initial conditions are given by the exact solution. The characteristic variables are $2U_1 + U_2$ and $2U_1 - U_2$ corresponding to the characteristics given by $dx/dt = 3$ and $dx/dt = -1$ respectively. Hence a physical boundary condition is required for $2U_1 + U_2$ at the left-hand boundary, and for $2U_1 - U_2$ at the right-hand boundary (corresponding to the incoming characteristics); and a numerical boundary condition is required for $2U_1 - U_2$ at the left-hand boundary, and for $2U_1 + U_2$ at the right-hand boundary (outgoing characteristics). The physical boundary conditions are obtained from the exact solution, and the numerical boundary conditions are calculated by linear extrapolation of the appropriate characteristic variable. The numerical flux is calculated using Roe's approximate Riemann solver: Using the notation in Section 3, the flux vector F and the Jacobian matrix A are

$$F = \begin{bmatrix} U_1 + U_2 \\ 4U_1 + U_2 \end{bmatrix} \quad \text{and} \quad A = \begin{bmatrix} 1 & 1 \\ 4 & 1 \end{bmatrix},$$

and the eigenvalues of A are 3 and -1 with right eigenvectors $[1 \quad 2]^{\mathrm{T}}$ and $[-1 \quad 2]^{\mathrm{T}}$ respectively. Using equation (5) the α_k are given by

$$\begin{bmatrix} U_{1R}-U_{1L} \\ U_{2R}-U_{2L} \end{bmatrix} = \alpha_1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} + \alpha_2 \begin{bmatrix} -1 \\ 2 \end{bmatrix},$$

that is

$$\alpha_1 = \tfrac{1}{4}(2U_{1R} - 2U_{1L} + U_{2R} - U_{2L}) \quad \text{and} \quad \alpha_2 = \tfrac{1}{4}(-2U_{1R} + 2U_{1L} + U_{2R} - U_{2L}).$$

F_L is given by

$$F_L = \begin{bmatrix} U_{1L} + U_{2L} \\ 4U_{1L} + U_{2L} \end{bmatrix},$$

and similarly for F_R. From equation (4), the numerical flux vector is

$$\hat{F} = \tfrac{1}{2}\begin{bmatrix} U_{1L} + U_{2L} + U_{1R} + U_{2R} \\ 4U_{1L} + U_{2L} + 4U_{1R} + U_{2R} \end{bmatrix} - \tfrac{1}{2}\alpha_1|3|\begin{bmatrix} 1 \\ 2 \end{bmatrix} - \tfrac{1}{2}\alpha_2|-1|\begin{bmatrix} -1 \\ 2 \end{bmatrix},$$

that is

$$\hat{F} = \tfrac{1}{2}\begin{bmatrix} 3U_{1L} - U_{1R} + \frac{3}{2}U_{2L} + \frac{1}{2}U_{2R} \\ 6U_{1L} + 2U_{1R} + 3U_{2L} - U_{2R} \end{bmatrix}.$$

9.1.1 Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
      SUBROUTINE EX1
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, NIW, NW, INT, OUTPTS
      PARAMETER        (NPDE=2,NPTS=101,NIW=24+NPDE*NPTS,NW=(11+9*NPDE)
     +                 *NPDE*NPTS+(32+3*NPDE)*NPDE+7*NPTS+54,INT=20,
     +                 OUTPTS=7)
*     .. Scalars in Common ..
      real             P
*     .. Local Scalars ..
      real             TOUT, TS, TSMAX, XX
      INTEGER          I, IFAIL, IND, IT, ITASK, ITRACE, J, NOP
*     .. Local Arrays ..
      real             ACC(2), U(NPDE,NPTS), UE(NPDE,OUTPTS), W(NW),
     +                 X(NPTS), XOUT(OUTPTS)
      INTEGER          IW(NIW)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         BNDRY1, D03PFF, D03PFP, EXACT, NMFLX1
*     .. Common blocks ..
      COMMON           /PI/P
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Example 1'
      WRITE (NOUT,*)
*
      XX = 0.0e0
      P = X01AAF(XX)
      ITRACE = 0
      ACC(1) = 0.1e-3
      ACC(2) = 0.1e-4
      TSMAX = 0.0e0
      WRITE (NOUT,99996) NPTS, ACC(1), ACC(2)
      WRITE (NOUT,99999)
*
*     Initialise mesh ..
```

```
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
*     Set initial values ..
      TS = 0.0e0
      CALL EXACT(TS,U,NPDE,X,NPTS)
*
      IND = 0
      ITASK = 1
*
      DO 80 IT = 1, 2
         TOUT = 0.1e0*IT
         IFAIL = 0
*
         CALL D03PFF(NPDE,TS,TOUT,D03PFP,NMFLX1,BNDRY1,U,NPTS,X,ACC,
     +              TSMAX,W,NW,IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
*       Set output points ..
         NOP = 0
         DO 40 I = 1, NPTS, INT
            NOP = NOP + 1
            XOUT(NOP) = X(I)
   40    CONTINUE
*
         WRITE (NOUT,99995) TS
*
*       Check against exact solution ..
         CALL EXACT(TOUT,UE,NPDE,XOUT,NOP)
         DO 60 I = 1, NOP
            J = 1 + INT*(I-1)
            WRITE (NOUT,99998) XOUT(I), U(1,J), UE(1,I), U(2,J), UE(2,I)
   60    CONTINUE
   80 CONTINUE
*
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      RETURN
*
99999 FORMAT (8X,'X',8X,'Approx U',4X,'Exact U',5X,'Approx V',4X,'Exac',
     +       't V')
99998 FORMAT (5(3X,F9.4))
99997 FORMAT (/' Number of integration steps in time = ',I6,/' Number ',
     +       'of function evaluations = ',I6,/' Number of Jacobian ',
     +       'evaluations =',I6,/' Number of iterations = ',I6,/)
99996 FORMAT (/' NPTS = ',I4,' ACC(1) = ',e10.3,' ACC(2) = ',e10.3,/)
99995 FORMAT (/' T = ',F6.3,/)
      END
*
      SUBROUTINE BNDRY1(NPDE,NPTS,T,X,U,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,3), X(NPTS)
*     .. Local Scalars ..
      real              C, EXU1, EXU2
*     .. Local Arrays ..
      real              UE(2,1)
*     .. External Subroutines ..
      EXTERNAL          EXACT
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         CALL EXACT(T,UE,NPDE,X(1),1)
         C = (X(2)-X(1))/(X(3)-X(2))
         EXU1 = (1.0e0+C)*U(1,2) - C*U(1,3)
         EXU2 = (1.0e0+C)*U(2,2) - C*U(2,3)
         G(1) = 2.0e0*U(1,1) + U(2,1) - 2.0e0*UE(1,1) - UE(2,1)
         G(2) = 2.0e0*U(1,1) - U(2,1) - 2.0e0*EXU1 + EXU2
      ELSE
         CALL EXACT(T,UE,NPDE,X(NPTS),1)
```

```
         C = (X(NPTS)-X(NPTS-1))/(X(NPTS-1)-X(NPTS-2))
         EXU1 = (1.0e0+C)*U(1,2) - C*U(1,3)
         EXU2 = (1.0e0+C)*U(2,2) - C*U(2,3)
         G(1) = 2.0e0*U(1,1) - U(2,1) - 2.0e0*UE(1,1) + UE(2,1)
         G(2) = 2.0e0*U(1,1) + U(2,1) - 2.0e0*EXU1 - EXU2
      END IF
      RETURN
      END
*
      SUBROUTINE NMFLX1(NPDE,T,X,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real               T, X
      INTEGER            IRES, NPDE
*     .. Array Arguments ..
      real               FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE)
*     .. Executable Statements ..
      FLUX(1) = 0.5e0*(-URIGHT(1)+3.0e0*ULEFT(1)+0.5e0*URIGHT(2)
     +          +1.5e0*ULEFT(2))
      FLUX(2) = 0.5e0*(2.0e0*URIGHT(1)+6.0e0*ULEFT(1)-URIGHT(2)
     +          +3.0e0*ULEFT(2))
      RETURN
      END
*
      SUBROUTINE EXACT(T,U,NPDE,X,NPTS)
*     Exact solution (for comparison and b.c. purposes)
*     .. Scalar Arguments ..
      real               T
      INTEGER            NPDE, NPTS
*     .. Array Arguments ..
      real               U(NPDE,*), X(*)
*     .. Scalars in Common ..
      real               P
*     .. Local Scalars ..
      real               X1, X2
      INTEGER            I
*     .. Intrinsic Functions ..
      INTRINSIC          EXP, SIN
*     .. Common blocks ..
      COMMON             /PI/P
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         X1 = X(I) + T
         X2 = X(I) - 3.0e0*T
         U(1,I) = 0.5e0*(EXP(X1)+EXP(X2)) + 0.25e0*(SIN(2.0e0*P*X2**2)
     +            -SIN(2.0e0*P*X1**2)) + 2.0e0*T**2 - 2.0e0*X(I)*T
         U(2,I) = EXP(X2) - EXP(X1) + 0.5e0*(SIN(2.0e0*P*X2**2)
     +            +SIN(2.0e0*P*X1**2)) + X(I)**2 + 5.0e0*T**2 -
     +            2.0e0*X(I)*T
   20 CONTINUE
      RETURN
      END
```

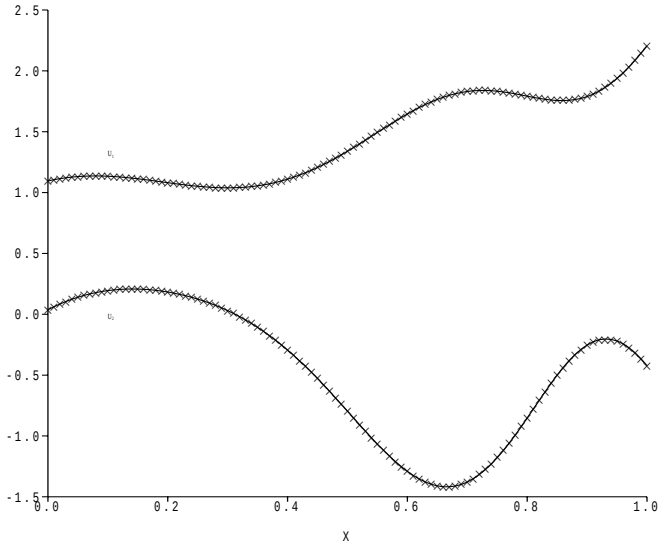
9.1.2 Program Data

None.

9.1.3 Program Results

```
 D03PFF Example Program Results


 Example 1


 NPTS =  101 ACC(1) =  0.100E-03 ACC(2) =  0.100E-04

        X        Approx U    Exact U     Approx V    Exact V

 T =  0.100
```

```
      0.0000      1.0615      1.0613     -0.0155     -0.0150
      0.2000      0.9892      0.9891     -0.0953     -0.0957
      0.4000      1.0826      1.0826      0.1180      0.1178
      0.6000      1.7001      1.7001     -0.0751     -0.0746
      0.8000      2.3959      2.3966     -0.2453     -0.2458
      1.0000      2.1029      2.1025      0.3760      0.3753

 T =  0.200

      0.0000      1.0957      1.0956      0.0368      0.0370
      0.2000      1.0808      1.0811      0.1826      0.1828
      0.4000      1.1102      1.1100     -0.2935     -0.2938
      0.6000      1.6461      1.6454     -1.2921     -1.2908
      0.8000      1.7913      1.7920     -0.8510     -0.8525
      1.0000      2.2050      2.2050     -0.4222     -0.4221

 Number of integration steps in time =     56
 Number of function evaluations =    229
 Number of Jacobian evaluations =     7
 Number of iterations =    143
```



9.2 Example 2

This example is an advection-diffusion equation in which the flux term depends explicitly on x:

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

for $x \in [-1, 1]$ and $0 \leq t \leq 10$. The parameter ϵ is taken to be 0.01. The two physical boundary conditions are $\mathrm{U}(-1, t) = 3.0$ and $\mathrm{U}(1, t) = 5.0$ and the initial condition is $\mathrm{U}(x, 0) = x + 4$. The integration is run to steady state at which the solution is known to be $\mathrm{U} = 4$ across the domain with a narrow boundary layer at both boundaries. In order to write the PDE in conservative form, a source term must be introduced, i.e.

$$\frac{\partial U}{\partial t} + \frac{\partial (xU)}{\partial x} = \epsilon\frac{\partial^2 U}{\partial x^2} + U.$$

As in Example 1, the numerical flux is calculated using the Roe approximate Riemann solver. The Riemann problem to solve locally is

$$\frac{\partial U}{\partial t} + \frac{\partial (xU)}{\partial x} = 0.$$

The x in the flux term is assumed to be constant at a local level, and so using the notation in Section 3, $F = xU$ and $A = x$. The eigenvalue is x and the eigenvector (a scalar in this case) is 1. The numerical flux is therefore

$$\hat{F} = \begin{cases} xU_L & \text{if} \quad x \geq 0, \\ xU_R & \text{if} \quad x < 0. \end{cases}$$

9.2.1 Program Text

Note: the listing of the example program presented below uses ***bold italicised*** terms to denote precision-dependent details. Please read the Users' Note for your implementation to check the interpretation of these terms. As explained in the Essential Introduction to this manual, the results produced may not be identical for all implementations.

```
      SUBROUTINE EX2
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, NIW, NW, OUTPTS
      PARAMETER        (NPDE=1,NPTS=151,NIW=24+NPDE*NPTS,NW=(11+9*NPDE)
     +                 *NPDE*NPTS+(32+3*NPDE)*NPDE+7*NPTS+54,OUTPTS=7)
*     .. Local Scalars ..
      real             TOUT, TS, TSMAX
      INTEGER          I, IFAIL, IND, IT, ITASK, ITRACE
*     .. Local Arrays ..
      real             ACC(2), U(NPDE,NPTS), W(NW), X(NPTS),
     +                 XOUT(OUTPTS)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDRY2, D03PFF, NMFLX2, PDEDEF
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Example 2'
      WRITE (NOUT,*)
*
      ITRACE = 0
      ACC(1) = 0.1e-4
      ACC(2) = 0.1e-4
      WRITE (NOUT,99998) NPTS, ACC(1), ACC(2)
*
*     Initialise mesh ..
*
      DO 20 I = 1, NPTS
         X(I) = -1.0e0 + 2.0e0*(I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
*     Set initial values ..
      DO 40 I = 1, NPTS
         U(1,I) = X(I) + 4.0e0
   40 CONTINUE
*
      IND = 0
      ITASK = 1
      TSMAX = 0.2e-1
*
*     Set output points ..
      XOUT(1) = X(1)
      XOUT(2) = X(4)
      XOUT(3) = X(37)
      XOUT(4) = X(76)
      XOUT(5) = X(112)
      XOUT(6) = X(148)
      XOUT(7) = X(151)
*
      WRITE (NOUT,99996) (XOUT(I),I=1,OUTPTS)
*
*     Loop over output value of t
*
      TS = 0.0e0
      TOUT = 1.0e0
      DO 60 IT = 1, 2
         IF (IT.EQ.2) TOUT = 10.0e0
```

```
         IFAIL = 0
*
         CALL D03PFF(NPDE,TS,TOUT,PDEDEF,NMFLX2,BNDRY2,U,NPTS,X,ACC,
     +              TSMAX,W,NW,IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
         WRITE (NOUT,99999) TS
         WRITE (NOUT,99995) U(1,1), U(1,4), U(1,37), U(1,76), U(1,112),
     +     U(1,148), U(1,151)
   60 CONTINUE
*
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      RETURN
*
99999 FORMAT (' T = ',F6.3)
99998 FORMAT (/' NPTS = ',I4,'  ACC(1) = ',e10.3,' ACC(2) = ',e10.3,/)
99997 FORMAT (' Number of integration steps in time = ',I6,/' Number ',
     +       'of function evaluations = ',I6,/' Number of Jacobian ',
     +       'evaluations =',I6,/' Number of iterations = ',I6,/)
99996 FORMAT (1X,'X    ',7F9.4,/)
99995 FORMAT (1X,'U    ',7F9.4,/)
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,UX,P,C,D,S,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NPDE
*     .. Array Arguments ..
      real              C(NPDE), D(NPDE), P(NPDE,NPDE), S(NPDE),
     +                  U(NPDE), UX(NPDE)
*     .. Executable Statements ..
      P(1,1) = 1.0e0
      C(1) = 0.1e-1
      D(1) = UX(1)
      S(1) = U(1)
      RETURN
      END
*
      SUBROUTINE BNDRY2(NPDE,NPTS,T,X,U,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,3), X(NPTS)
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         G(1) = U(1,1) - 3.0e0
      ELSE
         G(1) = U(1,1) - 5.0e0
      END IF
      RETURN
      END
*
      SUBROUTINE NMFLX2(NPDE,T,X,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NPDE
*     .. Array Arguments ..
      real              FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE)
*     .. Executable Statements ..
      IF (X.GE.0) THEN
         FLUX(1) = X*ULEFT(1)
      ELSE
         FLUX(1) = X*URIGHT(1)
      END IF
      RETURN
      END
```

9.2.2 Program Data

None.

9.2.3 Program Results

```
 D03PFF Example Program Results


 Example 2


 NPTS =  151  ACC(1) =  0.100E-04 ACC(2) =  0.100E-04

 X      -1.0000  -0.9600  -0.5200   0.0000   0.4800   0.9600   1.0000

 T =  1.000
 U       3.0000   3.6221   3.8087   4.0000   4.1766   4.3779   5.0000

 T = 10.000
 U       3.0000   3.9592   4.0000   4.0000   4.0000   4.0408   5.0000

 Number of integration steps in time =    503
 Number of function evaluations =   1190
 Number of Jacobian evaluations =    28
 Number of iterations =   1035
```