NAG Fortran Library Manual
Mark 19

Volume 3

D02P – D03

D02P – Ordinary Differential Equations (cont'd from Volume 2)
D03 – Partial Differential Equations

NAG Fortran Library Manual, Mark 19

September 1999 ISBN 1-85206-169-3

NAG Ltd
Wilkinson House
Jordan Hill Road
Oxford
OX2 8DR
United Kingdom

Tel: +44 (0)1865 511245
Fax: +44 (0)1865 310139

Nihon NAG KK
Nagashima Building 2F
2-24-3 Higashi
Shibuya-ku
Tokyo
Japan

Tel: +81 (0)3 5485 2901
Fax: +81 (0)3 5485 2903

NAG GmbH
Schleißheimerstraße 5
85748 Garching
Deutschland

Tel: +49 (0)89 3207395
Fax: +49 (0)89 3207396

NAG Inc
1400 Opus Place, Suite 200
Downers Grove, IL 60515-5702
USA

Tel: +1 630 971 2337
Fax: +1 630 971 2706

NAG also has a number of distributors throughout the world. Please contact NAG for further details.

D02PCF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02PCF solves the initial value problem for a first order system of ordinary differential equations using Runge-Kutta methods.

2. Specification

```
SUBROUTINE D02PCF (F, TWANT, TGOT, YGOT, YPGOT, YMAX, WORK, IFAIL)
INTEGER          IFAIL
real             TWANT, TGOT, YGOT(*), YPGOT(*), YMAX(*), WORK(*)
EXTERNAL
```

3. Description

D02PCF and its associated routines (D02PVF, D02PYF, D02PZF) solve the initial value problem for a first order system of ordinary differential equations. The routines, based on Runge-Kutta methods and derived from RKSUITE [1], integrate

$$y' = f(t,y) \text{ given } y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

D02PCF is designed for the usual task, namely to compute an approximate solution at a sequence of points. You must first call D02PVF to specify the problem and how it is to be solved. Thereafter you call D02PCF repeatedly with successive values of TWANT, the points at which you require the solution, in the range from TSTART to TEND (as specified in D02PVF). In this manner D02PCF returns the point at which it has computed a solution TGOT (usually TWANT), the solution there (YGOT) and its derivative (YPGOT). If D02PCF encounters some difficulty in taking a step toward TWANT, then it returns the point of difficulty (TGOT) and the solution and derivative computed there (YGOT and YPGOT, respectively).

In the call to D02PVF you can specify either the first step size for D02PCF to attempt or that it compute automatically an appropriate value. Thereafter D02PCF estimates an appropriate step size for its next step. This value and other details of the integration can be obtained after any call to D02PCF by a call to D02PYF. The local error is controlled at every step as specified in D02PVF. If you wish to assess the true error, you must set ERRASS = .TRUE. in the call to D02PVF. This assessment can be obtained after any call to D02PCF by a call to D02PZF.

For more complicated tasks, you are referred to routines D02PDF, D02PXF and D02PWF, all of which are used by D02PCF.

4. References

[1] BRANKIN, R.W., GLADWELL, I. and SHAMPINE, L.F.
RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs.
SoftReport 91-S1, Department of Mathematics, Southern Methodist University, Dallas, TX 75275, U.S.A, 1991.

5. Parameters

1: F – SUBROUTINE, supplied by the user. *External Procedure*

F must evaluate the functions f_i (that is the first derivatives y_i') for given values of the arguments t, y_i.

Its specification is:

```
SUBROUTINE F (T, Y, YP)
real          T, Y(*), YP(*)
```

1: T – ***real.*** *Input*

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

2: Y(*) – ***real*** array. *Input*

On entry: the current values of the dependent variables, y_i for $i = 1,2,...,n$.

3: YP(*) – ***real*** array. *Output*

On exit: the values of f_i for $i = 1,2,...,n$.

F must be declared as EXTERNAL in the (sub)program from which D02PCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: TWANT – ***real.*** *Input*

On entry: the next value of the independent variable, t, where a solution is desired.

Constraints: TWANT must be closer to TEND than the previous value of TGOT (or TSTART on the first call to D02PCF); see D02PVF for a description of TSTART and TEND.
TWANT must not lie beyond TEND in the direction of integration.

3: TGOT – ***real.*** *Output*

On exit: the value of the independent variable t at which a solution has been computed. On successful exit with IFAIL = 0, TGOT will equal TWANT. On exit with IFAIL > 1, a solution has still been computed at the value of TGOT but in general TGOT will not equal TWANT.

4: YGOT(*) – ***real*** array. *Input/Output*

Note: the dimension of YGOT must be at least n.

On entry: on the first call to D02PCF, YGOT need not be set. On all subsequent calls YGOT must remain unchanged.

On exit: an approximation to the true solution at the value of TGOT. At each step of the integration to TGOT, the local error has been controlled as specified in D02PVF. The local error has still been controlled even when TGOT ≠ TWANT, that is after a return with IFAIL > 1.

5: YPGOT(*) – ***real*** array. *Output*

Note: the dimension of YPGOT must be at least n.

On exit: an approximation to the first derivative of the true solution at TGOT.

6: YMAX(*) – ***real*** array. *Input/Output*

Note: the dimension of YMAX must be at least n.

On entry: on the first call to D02PCF, YMAX need not be set. On all subsequent calls YMAX must remain unchanged.

On exit: YMAX(i) contains the largest value of $|y_i|$ computed at any step in the integration so far.

7: WORK(*) – ***real*** array. *Input/Output*

On entry: this **must** be the same array as supplied to D02PVF. It **must** remain unchanged between calls.

On exit: information about the integration for use on subsequent calls to D02PCF or other associated routines.

8: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

On entry, an invalid input value for TWANT was detected or an invalid call to D02PCF was made, for example without a previous call to the setup routine D02PVF. If on entry IFAIL = 0 or –1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF). You cannot continue integrating the problem.

IFAIL = 2

This return is possible only when METHOD = 3 has been selected in the preceding call of D02PVF. D02PCF is being used inefficiently because the step size has been reduced drastically many times to get answers at many values of TWANT. If you really need the solution at this many points, you should change to METHOD = 2 because it is (much) more efficient in this situation. To change METHOD, restart the integration from TGOT, YGOT by a call to D02PVF. If you wish to continue with METHOD = 3, just call D02PCF again without altering any of the arguments other than IFAIL. The monitor of this kind of inefficiency will be reset automatically so that the integration can proceed.

IFAIL = 3

A considerable amount of work has been expended in the (primary) integration. This is measured by counting the number of calls to the subroutine F. At least 5000 calls have been made since the last time this counter was reset. Calls to F in a secondary integration for global error assessment (when ERRASS = .TRUE. in the call to D02PVF) are not counted in this total. The integration was interrupted, so TGOT is not equal to TWANT. If you wish to continue on towards TWANT, just call D02PCF again without altering any of the arguments other than IFAIL. The counter measuring work will be reset to zero automatically.

IFAIL = 4

It appears that this problem is stiff. The methods implemented in D02PCF can solve such problems, but they are inefficient. You should change to another code based on methods appropriate for stiff problems. The integration was interrupted so TGOT is not equal to TWANT. If you want to continue on towards TWANT, just call D02PCF again without altering any of the arguments other than IFAIL. The stiffness monitor will be reset automatically.

IFAIL = 5

It does not appear possible to achieve the accuracy specified by TOL and THRES in the call to D02PVF with the precision available on the computer being used and with this value of METHOD. You cannot continue integrating this problem. A larger value for METHOD, if possible, will permit greater accuracy with this precision. To increase METHOD and/or continue with larger values of TOL and/or THRES, restart the integration from TGOT, YGOT by a call to D02PVF.

IFAIL = 6

(This error exit can only occur if ERRASS = .TRUE. in the call to D02PVF.) The global error assessment may not be reliable beyond the current integration point TGOT. This may occur because either too little or too much accuracy has been requested or because $f(t,y)$ is not smooth enough for values of t just past TGOT and current values of the solution y. The integration cannot be continued. This return does not mean that you cannot integrate past TGOT, rather that you cannot do it with ERRASS = .TRUE.. However, it may also indicate problems with the primary integration.

7. Accuracy

The accuracy of integration is determined by the parameters TOL and THRES in a prior call to D02PVF (see the routine document for further details and advice). Note that only the local error at each step is controlled by these parameters. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the properties of the differential system.

8. Further Comments

If D02PCF returns with IFAIL = 5 and the accuracy specified by TOL and THRES is really required then you should consider whether there is a more fundamental difficulty. For example, the solution may contain a singularity. In such a region the solution components will usually be large in magnitude. Successive output values of YGOT and YMAX should be monitored (or D02PDF should be used since this takes one integration step at a time) with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary.

Performance statistics are available after any return from D02PCF by a call to D02PYF. If ERRASS = .TRUE. in the call to D02PVF, global error assessment is available after any return from D02PCF (except when IFAIL = 1) by a call to D02PZF.

After a failure with IFAIL = 5 or 6 the diagnostic routines D02PYF and D02PZF may be called only once.

If D02PCF returns with IFAIL = 4 then it is advisable to change to another code more suited to the solution of stiff problems. D02PCF will not return with IFAIL = 4 if the problem is actually stiff but it is estimated that integration can be completed using less function evaluations than already computed.

9. Example

We solve the equation

$$y'' = -y, \quad y(0) = 0, \; y'(0) = 1$$

reposed as

$$y_1' = y_2$$
$$y_2' = -y_1$$

over the range $[0,2\pi]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$. We use relative error control with threshold values of 1.0E−8 for each solution component and compute the solution at intervals of length $\pi/4$ across the range. We use a low order Runge-Kutta method (METHOD = 1) with tolerances TOL = 1.0E−3 and TOL = 1.0E−4 in turn so that we may compare the solutions. The value of π is obtained by using X01AAF.

Note that the length of WORK is large enough for any valid combination of input arguments to D02PVF.

See also the example program for D02PZF.

9.1. 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.

```
*     D02PCF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, LENWRK, METHOD
      PARAMETER        (NEQ=2,LENWRK=32*NEQ,METHOD=1)
      real             ZERO, ONE, TWO
      PARAMETER        (ZERO=0.0e0,ONE=1.0e0,TWO=2.0e0)
*     .. Local Scalars ..
      real             HNEXT, HSTART, PI, TEND, TGOT, TINC, TOL, TSTART,
     +                 TWANT, WASTE
      INTEGER          I, IFAIL, J, L, NPTS, STPCST, STPSOK, TOTF
      LOGICAL          ERRASS
*     .. Local Arrays ..
      real             THRES(NEQ), WORK(LENWRK), YGOT(NEQ), YMAX(NEQ),
     +                 YPGOT(NEQ), YSTART(NEQ)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02PCF, D02PVF, D02PYF, F
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02PCF Example Program Results'
*
*     Set initial conditions and input for D02PVF
*
      PI = X01AAF(ZERO)
      TSTART = ZERO
      YSTART(1) = ZERO
      YSTART(2) = ONE
      TEND = TWO*PI
      DO 20 L = 1, NEQ
         THRES(L) = 1.0e-8
   20 CONTINUE
      ERRASS = .FALSE.
      HSTART = ZERO
*
*     Set control for output
*
      NPTS = 8
      TINC = (TEND-TSTART)/NPTS
*
      DO 60 I = 1, 2
         IF (I.EQ.1) TOL = 1.0e-3
         IF (I.EQ.2) TOL = 1.0e-4
         IFAIL = 0
         CALL D02PVF(NEQ,TSTART,YSTART,TEND,TOL,THRES,METHOD,
     +               'Usual Task',ERRASS,HSTART,WORK,LENWRK,IFAIL)
*
         WRITE (NOUT,'(/A,D8.1)') 'Calculation with TOL = ', TOL
         WRITE (NOUT,'(/A/)') '    t         y1        y2'
         WRITE (NOUT,'(1X,F6.3,2(3X,F7.3))') TSTART, (YSTART(L),L=1,NEQ)
         DO 40 J = NPTS - 1, 0, -1
            TWANT = TEND - J*TINC
            IFAIL = 1
            CALL D02PCF(F,TWANT,TGOT,YGOT,YPGOT,YMAX,WORK,IFAIL)
*
```

```
            WRITE (NOUT,'(1X,F6.3,2(3X,F7.3))') TGOT, (YGOT(L),L=1,NEQ)
   40    CONTINUE
*
         IFAIL = 0
         CALL D02PYF(TOTF,STPCST,WASTE,STPSOK,HNEXT,IFAIL)
         WRITE (NOUT,'(/A,I6)')
     +     ' Cost of the integration in evaluations of F is', TOTF
*
   60 CONTINUE
*
      STOP
      END
      SUBROUTINE F(T,Y,YP)
*     .. Scalar Arguments ..
      real              T
*     .. Array Arguments ..
      real              Y(*), YP(*)
*     .. Executable Statements ..
      YP(1) = Y(2)
      YP(2) = -Y(1)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02PCF Example Program Results

Calculation with TOL =  0.1E-02

    t          y1         y2

  0.000      0.000      1.000
  0.785      0.707      0.707
  1.571      0.999      0.000
  2.356      0.706     -0.706
  3.142      0.000     -0.999
  3.927     -0.706     -0.706
  4.712     -0.998      0.000
  5.498     -0.705      0.706
  6.283      0.001      0.997

 Cost of the integration in evaluations of F is   124

Calculation with TOL =  0.1E-03

    t          y1         y2

  0.000      0.000      1.000
  0.785      0.707      0.707
  1.571      1.000      0.000
  2.356      0.707     -0.707
  3.142      0.000     -1.000
  3.927     -0.707     -0.707
  4.712     -1.000      0.000
  5.498     -0.707      0.707
  6.283      0.000      1.000

 Cost of the integration in evaluations of F is   235
```

D02PDF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02PDF is a one-step routine for solving the initial value problem for a first order system of ordinary differential equations using Runge-Kutta methods.

2. Specification

```
SUBROUTINE D02PDF (F, TNOW, YNOW, YPNOW, WORK, IFAIL)
INTEGER        IFAIL
real           TNOW, YNOW(*), YPNOW(*), WORK(*)
EXTERNAL       F
```

3. Description

D02PDF and its associated routines (D02PVF, D02PWF, D02PXF, D02PYF, D02PZF) solve the initial value problem for a first order system of ordinary differential equations. The routines, based on Runge-Kutta methods and derived from RKSUITE [1], integrate

$$y' = f(t,y) \text{ given } y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

D02PDF is designed to be used in complicated tasks when solving systems of ordinary differential equations. You must first call D02PVF to specify the problem and how it is to be solved. Thereafter you (repeatedly) call D02PDF to take one integration step at a time from TSTART in the direction of TEND (as specified in D02PVF). In this manner D02PDF returns an approximation to the solution YNOW and its derivative YPNOW at successive points TNOW. If D02PDF encounters some difficulty in taking a step, the integration is not advanced and the routine returns with the same values of TNOW, YNOW and YPNOW as returned on the previous successful step. D02PDF tries to advance the integration as far as possible subject to passing the test on the local error and not going past TEND. In the call to D02PVF you can specify either the first step size for D02PDF to attempt or that it compute automatically an appropriate value. Thereafter D02PDF estimates an appropriate step size for its next step. This value and other details of the integration can be obtained after any call to D02PDF by a call to D02PYF. The local error is controlled at every step as specified in D02PVF. If you wish to assess the true error, you must set ERRASS = .TRUE. in the call to D02PVF. This assessment can be obtained after any call to D02PDF by a call to D02PZF.

If you want answers at specific points there are two ways to proceed:

(1) The more efficient way is to step past the point where a solution is desired, and then call D02PXF to get an answer there. Within the span of the current step, you can get all the answers you want at very little cost by repeated calls to D02PXF. This is very valuable when you want to find where something happens, e.g., where a particular solution component vanishes. You cannot proceed in this way with METHOD = 3.

(2) The other way to get an answer at a specific point is to set TEND to this value and integrate to TEND. D02PDF will not step past TEND, so when a step would carry it past, it will reduce the step size so as to produce an answer at TEND exactly. After getting an answer there (TNOW = TEND), you can reset TEND to the next point where you want an answer, and repeat. TEND could be reset by a call to D02PVF, but you should not do this. You should use D02PWF instead because it is both easier to use and much more efficient. This way of getting answers at specific points can be used with any of the available methods, but it is the only way with METHOD = 3. It can be inefficient. Should this be the case, the code will bring the matter to your attention.

4. References

[1] BRANKIN, R.W., GLADWELL, I. and SHAMPINE, L.F.
RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs.
SoftReport 91-S1, Department of Mathematics, Southern Methodist University, Dallas, TX 75275, U.S.A, 1991.

5. Parameters

1: F – SUBROUTINE, supplied by the user. *External Procedure*

F must evaluate the functions f_i (that is the first derivatives y_i') for given values of the arguments t, y_i.

Its specification is:

```
SUBROUTINE F (T, Y, YP)
real          T, Y(*), YP(*)
```

1: T – ***real***. *Input*

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

2: Y(*) – ***real*** array. *Input*

On entry: the current values of the dependent variables, y_i for $i = 1,2,...,n$.

3: YP(*) – ***real*** array. *Output*

On exit: the values of f_i for $i = 1,2,...,n$.

F must be declared as EXTERNAL in the (sub)program from which D02PDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: TNOW – ***real***. *Output*

On exit: the value of the independent variable t at which a solution has been computed.

3: YNOW(*) – ***real*** array. *Output*

Note: the dimension of YNOW must be at least n.

On exit: an approximation to the solution at TNOW. The local error of the step to TNOW was no greater than permitted by the specified tolerances (see D02PVF).

4: YPNOW(*) – ***real*** array. *Output*

Note: the dimension of YPNOW must be at least n.

On exit: an approximation to the derivative of the solution at TNOW.

5: WORK(*) – ***real*** array. *Input/Output*

On entry: this **must** be the same array as supplied to D02PVF. It **must** remain unchanged between calls.

On exit: information about the integration for use on subsequent calls to D02PDF or other associated routines.

6: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

On entry, an invalid call to D02PDF was made, for example without a previous call to the setup routine D02PVF. If on entry IFAIL = 0 or −1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF). You cannot continue integrating the problem.

IFAIL = 2

D02PDF is being used inefficiently because the step size has been reduced drastically many times to obtain answers at many points TEND. If you really need the solution at this many points, you should use D02PXF to obtain the answers inexpensively. If you need to change from METHOD = 3 to do this, restart the integration from TNOW, YNOW by a call to D02PVF. If you wish to continue as before, call D02PDF again. The monitor of this kind of inefficiency will be reset automatically so that the integration can proceed.

IFAIL = 3

A considerable amount of work has been expended in the (primary) integration. This is measured by counting the number of calls to the subroutine F. At least 5000 calls have been made since the last time this counter was reset. Calls to F in a secondary integration for global error assessment (when ERRASS = .TRUE. in the call to D02PVF) are not counted in this total. The integration was interrupted. If you wish to continue on towards TEND, just call D02PDF again. The counter measuring work will be reset to zero automatically.

IFAIL = 4

It appears that this problem is stiff. The methods implemented in D02PDF can solve such problems, but they are inefficient. You should change to another code based on methods appropriate for stiff problems. The integration was interrupted. If you want to continue on towards TEND, just call D02PDF again. The stiffness monitor will be reset automatically.

IFAIL = 5

It does not appear possible to achieve the accuracy specified by TOL and THRES in the call to D02PVF with the precision available on the computer being used and with this value of METHOD. You cannot continue integrating this problem. A larger value for METHOD, if possible, will permit greater accuracy with this precision. To increase METHOD and/or continue with larger values of TOL and/or THRES, restart the integration from TNOW, YNOW by a call to D02PVF.

IFAIL = 6

(This error exit can only occur if ERRASS = .TRUE. in the call to D02PVF.) The global error assessment may not be reliable beyond the current integration point TNOW. This may occur because either too little or too much accuracy has been requested or because $f(t,y)$ is not smooth enough for values of t just beyond TNOW and current values of the solution y. The integration cannot be continued. This return does not mean that you cannot integrate past TGOT, rather that you cannot do it with ERRASS = .TRUE.. However, it may also indicate problems with the primary integration.

7. Accuracy

The accuracy of integration is determined by the parameters TOL and THRES in a prior call to D02PVF (see the routine document for further details and advice). Note that only the local error at each step is controlled by these parameters. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the properties of the differential system.

8. Further Comments

If D02PDF returns with IFAIL = 5 and the accuracy specified by TOL and THRES is really required then you should consider whether there is a more fundamental difficulty. For example, the solution may contain a singularity. In such a region the solution components will usually be large in magnitude. Successive output values of YNOW should be monitored with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary.

Performance statistics are available after any return from D02PDF (except when IFAIL = 1) by a call to D02PYF. If ERRASS = .TRUE. in the call to D02PVF, global error assessment is available after any return from D02PDF (except when IFAIL = 1) by a call to D02PZF.

After a failure with IFAIL = 5 or 6 the diagnostic routines D02PYF and D02PZF may be called only once.

If D02PDF returns with IFAIL = 4 then it is advisable to change to another code more suited to the solution of stiff problems. D02PDF will not return with IFAIL = 4 if the problem is actually stiff but it is estimated that integration can be completed using less function evaluations than already computed.

9. Example

We solve the equation

$$y'' = -y, \quad y(0) = 0,\ y'(0) = 1$$

reposed as

$$y_1' = y_2$$
$$y_2' = -y_1$$

over the range $[0,2\pi]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$. We use relative error control with threshold values of 1.0E−8 for each solution component and print the solution at each integration step across the range. We use a medium order Runge-Kutta method (METHOD = 2) with tolerances TOL = 1.0E−4 and TOL = 1.0E−5 in turn so that we may compare the solutions. The value of π is obtained by using X01AAF.

Note that the length of WORK is large enough for any valid combination of input arguments to D02PVF.

See also the example programs for D02PWF and D02PXF.

9.1. 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.

```
*     D02PDF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, LENWRK, METHOD
      PARAMETER        (NEQ=2,LENWRK=32*NEQ,METHOD=2)
      real             ZERO, ONE, TWO
      PARAMETER        (ZERO=0.0e0,ONE=1.0e0,TWO=2.0e0)
*     .. Local Scalars ..
      real             HNEXT, HSTART, PI, TEND, TNOW, TOL, TSTART, WASTE
      INTEGER          I, IFAIL, L, STPCST, STPSOK, TOTF
      LOGICAL          ERRASS
```

```
*     .. Local Arrays ..
      real             THRES(NEQ), WORK(LENWRK), YNOW(NEQ), YPNOW(NEQ),
     +                 YSTART(NEQ)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02PDF, D02PVF, D02PYF, F
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02PDF Example Program Results'
*
*     Set initial conditions and input for D02PVF
*
      PI = X01AAF(ZERO)
      TSTART = ZERO
      YSTART(1) = ZERO
      YSTART(2) = ONE
      TEND = TWO*PI
      DO 20 L = 1, NEQ
         THRES(L) = 1.0e-8
   20 CONTINUE
      ERRASS = .FALSE.
      HSTART = ZERO
*
      DO 60 I = 1, 2
         IF (I.EQ.1) TOL = 1.0e-4
         IF (I.EQ.2) TOL = 1.0e-5
         IFAIL = 0
         CALL D02PVF(NEQ,TSTART,YSTART,TEND,TOL,THRES,METHOD,
     +               'Complex Task',ERRASS,HSTART,WORK,LENWRK,IFAIL)
*
         WRITE (NOUT,'(/A,D8.1)') 'Calculation with TOL = ', TOL
         WRITE (NOUT,'(/A/)') '    t          y1         y2'
         WRITE (NOUT,'(1X,F6.3,2(3X,F8.4))') TSTART, (YSTART(L),L=1,NEQ)
*
   40    CONTINUE
         IFAIL = -1
         CALL D02PDF(F,TNOW,YNOW,YPNOW,WORK,IFAIL)
*
         IF (IFAIL.EQ.0) THEN
            WRITE (NOUT,'(1X,F6.3,2(3X,F8.4))') TNOW, (YNOW(L),L=1,NEQ)
            IF (TNOW.LT.TEND) GO TO 40
         END IF
*
         IFAIL = 0
         CALL D02PYF(TOTF,STPCST,WASTE,STPSOK,HNEXT,IFAIL)
         WRITE (NOUT,'(/A,I6)')
     +     ' Cost of the integration in evaluations of F is', TOTF
*
   60 CONTINUE
*
      STOP
      END
      SUBROUTINE F(T,Y,YP)
*     .. Scalar Arguments ..
      real             T
*     .. Array Arguments ..
      real             Y(*), YP(*)
*     .. Executable Statements ..
      YP(1) = Y(2)
      YP(2) = -Y(1)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02PDF Example Program Results

Calculation with TOL =  0.1E-03

    t          y1          y2

  0.000      0.0000      1.0000
  0.785      0.7071      0.7071
  1.519      0.9987      0.0513
  2.282      0.7573     -0.6531
  2.911      0.2285     -0.9735
  3.706     -0.5348     -0.8450
  4.364     -0.9399     -0.3414
  5.320     -0.8209      0.5710
  5.802     -0.4631      0.8863
  6.283      0.0000      1.0000

 Cost of the integration in evaluations of F is     78

Calculation with TOL =  0.1E-04

    t          y1          y2

  0.000      0.0000      1.0000
  0.393      0.3827      0.9239
  0.785      0.7071      0.7071
  1.416      0.9881      0.1538
  1.870      0.9557     -0.2943
  2.204      0.8062     -0.5916
  2.761      0.3711     -0.9286
  3.230     -0.0880     -0.9961
  3.587     -0.4304     -0.9026
  4.022     -0.7710     -0.6368
  4.641     -0.9974     -0.0717
  5.152     -0.9049      0.4256
  5.521     -0.6903      0.7235
  5.902     -0.3718      0.9283
  6.283      0.0000      1.0000

 Cost of the integration in evaluations of F is    118
```

D02PVF – NAG Fortran Library Routine Document

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

D02PVF is a setup routine which must be called prior to the first call of either of the integration routines D02PCF and D02PDF.

2 Specification

```
      SUBROUTINE D02PVF(NEQ, TSTART, YSTART, TEND, TOL, THRES, METHOD,
     1                  TASK, ERRASS, HSTART, WORK, LENWRK, IFAIL)
      INTEGER           NEQ, METHOD, LENWRK, IFAIL
      real              TSTART, YSTART(NEQ), TEND, TOL, THRES(NEQ),
     1                  HSTART, WORK(LENWRK)
      LOGICAL           ERRASS
      CHARACTER*1       TASK
```

3 Description

D02PVF and its associated routines (D02PCF, D02PDF, D02PWF, D02PXF, D02PYF, D02PZF) solve the initial value problem for a first order system of ordinary differential equations. The routines, based on Runge–Kutta methods and derived from RKSUITE [1], integrate

$$y' = f(t, y) \text{ given } y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

The integration proceeds by steps from the initial point t_0 towards the final point t_f. An approximate solution y is computed at each step. For each component $y_i, i = 1, 2, \ldots, n$ the error made in the step, i.e., the local error, is estimated. The step size is chosen automatically so that the integration will proceed efficiently while keeping this local error estimate smaller than a tolerance that you specify by means of parameters TOL and THRES.

D02PCF can be used to solve the 'usual task', namely integrating the system of differential equations to obtain answers at points you specify. D02PDF is used for all more 'complicated tasks'.

You should consider carefully how you want the local error to be controlled. Essentially the code uses relative local error control, with TOL being the desired relative accuracy. For reliable computation, the code must work with approximate solutions that have some correct digits, so there is an upper bound on the value you can specify for TOL. It is impossible to compute a numerical solution that is more accurate than the correctly rounded value of the true solution, so you are not allowed to specify TOL too small for the precision you are using. The magnitude of the local error in y_i on any step will not be greater than TOL $\times$ max(μ_i, THRES(i)) where μ_i is an average magnitude of y_i over the step. If THRES(i) is smaller than the current value of μ_i, this is a relative error test and TOL indicates how many significant digits you want in y_i. If THRES(i) is larger than the current value of μ_i, this is an absolute error test with tolerance TOL $\times$ THRES(i). Relative error control is the recommended mode of operation, but pure relative error control, THRES(i) = 0.0, is not permitted. See Section 8 for further information about error control.

D02PCF and D02PDF control local error rather than the true (global) error, the difference between the numerical and true solution. Control of the local error controls the true error indirectly. Roughly speaking, the code produces a solution that satisfies the differential equation with a discrepancy bounded in magnitude by the error tolerance. What this implies about how close the numerical solution is to the true solution depends on the stability of the problem. Most practical problems are at least moderately stable, and the true error is then comparable to the error tolerance. To judge the accuracy of the numerical solution, you could reduce TOL substantially, e.g. use 0.1 $\times$ TOL, and solve the problem again. This will usually result in a rather more accurate solution, and the true error of the first integration can be estimated by comparison. Alternatively, a global error assessment can be computed automatically using the parameter ERRASS. Because indirect control of the true error by controlling the local error is generally satisfactory and because both ways of assessing true errors cost twice, or more, the cost of the integration itself, such assessments are used mostly for spot checks, selecting appropriate tolerances for local error control, and exploratory computations.

D02PCF and D02PDF each implement three Runge–Kutta formula pairs, and you must select one for the integration. The best choice for METHOD depends on the problem. The order of accuracy is 3,5,8, respectively. As a rule, the smaller TOL is, the larger you should take the value of METHOD. If the components THRES are small enough that you are effectively specifying relative error control, experience suggests

TOL	efficient METHOD
$10^{-2} - 10^{-4}$	1
$10^{-3} - 10^{-6}$	2
$10^{-5} -$	3

The overlap in the ranges of tolerances appropriate for a given METHOD merely reflects the dependence of efficiency on the problem being solved. Making TOL smaller will normally make the integration more expensive. However, in the range of tolerances appropriate to a METHOD, the increase in cost is modest. There are situations for which one METHOD, or even this kind of code, is a poor choice. You should not specify a very small value for THRES(i), when the ith solution component might vanish. In particular, you should not do this when $y_i = 0.0$. If you do, the code will have to work hard with any value for METHOD to compute significant digits, but METHOD = 1 is a particularly poor choice in this situation. All three methods are inefficient when the problem is 'stiff'. If it is only mildly stiff, you can solve it with acceptable efficiency with METHOD = 1, but if it is moderately or very stiff, a code designed specifically for such problems will be much more efficient. The higher the order, i.e., the larger the value of METHOD, the more smoothness is required of the solution in order for the method to be efficient.

When assessment of the true (global) error is requested, this error assessment is updated at each step. Its value can be obtained at any time by a call to D02PZF. The code monitors the computation of the global error assessment and reports any doubts it has about the reliability of the results. The assessment scheme requires some smoothness of $f(t, y)$, and it can be deceived if f is insufficiently smooth. At very crude tolerances the numerical solution can become so inaccurate that it is impossible to continue assessing the accuracy reliably. At very stringent tolerances the effects of finite precision arithmetic can make it impossible to assess the accuracy reliably. The cost of this is roughly twice the cost of the integration itself with METHOD = 2,3, and three times with METHOD = 1.

The first step of the integration is critical because it sets the scale of the problem. The integrator will find a starting step size automatically if you set the parameter HSTART to 0.0. Automatic selection of the first step is so effective that you should normally use it. Nevertheless, you might want to specify a trial value for the first step to be certain that the code recognizes the scale on which phenomena occur near the initial point. Also, automatic computation of the first step size involves some cost, so supplying a good value for this step size will result in a less expensive start. If you are confident that you have a good value, provide it via the parameter HSTART.

4 References

[1] Brankin R W, Gladwell I and Shampine L F (1991) RKSUITE: A suite of Runge–Kutta codes for the initial value problems for ODEs *SoftReport 91-S1* Southern Methodist University

5 Parameters

1: NEQ — INTEGER *Input*

On entry: the number of ordinary differential equations in the system to be solved by the integration routine, n.

Constraint: NEQ ≥ 1.

2: TSTART — ***real*** *Input*

On entry: the initial value of the independent variable, t_0.

3: YSTART(NEQ) — ***real*** array *Input*

On entry: y_0, the initial values of the solution, y_i for $i = 1, 2,,$ at t_0.

4: TEND — ***real*** *Input*

On entry: the final value of the independent variable, t_f, at which the solution is required. TSTART and TEND together determine the direction of integration.

Constraint: TEND must be distinguishable from TSTART for the method and the precision of the machine being used.

5: TOL — ***real*** *Input*

On entry: a relative error tolerance.

Constraint: $10.0 \times$ ***machine precision*** $\leq$ TOL ≤ 0.01.

6: THRES(NEQ) — ***real*** array *Input*

On entry: a vector of thresholds.

Constraint: THRES(i) $\geq \sqrt{\sigma}$, where σ is approximately the smallest possible machine number that can be reciprocated without overflow (see X02AMF).

7: METHOD — INTEGER *Input*

On entry: the Runge–Kutta method to be used.

If METHOD = 1 then a 2(3) pair is used;

if METHOD = 2 then a 4(5) pair is used;

if METHOD = 3 then a 7(8) pair is used.

Constraint: $1 \leq$ METHOD ≤ 3.

8: TASK — CHARACTER*1 *Input*

On entry: determines whether the usual integration task is to be performed using D02PCF or a more complicated task is to be performed using D02PDF.

If TASK = 'U' then D02PCF is to be used for the integration.

If TASK = 'C' then D02PDF is to be used for the integration.

Constraint: TASK = 'U' or 'C'.

9: ERRASS — LOGICAL *Input*

On entry: specifies whether a global error assessment is to be computed with the main integration. ERRASS = .TRUE. specifies that it is.

10: HSTART — ***real*** *Input*

On entry: a value for the size of the first step in the integration to be attempted. The absolute value of HSTART is used with the direction being determined by TSTART and TEND. The actual first step taken by the integrator may be different to HSTART if the underlying algorithm determines that HSTART is unsuitable. If HSTART = 0.0 then the size of the first step is computed automatically.

Suggested value: HSTART = 0.0.

11: WORK(LENWRK) — ***real*** array *Output*

On exit: contains information for use by D02PCF or D02PDF. This **must** be the same array as supplied to D02PCF or D02PDF. The contents of this array must remain unchanged between calls.

12: LENWRK — INTEGER *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which D02PVF is called. (LENWRK $\geq$ 32 $\times$ NEQ is always sufficient.)

Constraints:

if TASK = 'U' and ERRASS = .FALSE. and
METHOD = 1, LENWRK $\geq$ 10 $\times$ NEQ;
METHOD = 2, LENWRK $\geq$ 20 $\times$ NEQ;
METHOD = 3, LENWRK $\geq$ 16 $\times$ NEQ;
if TASK = 'U' and ERRASS = .TRUE. and
METHOD = 1, LENWRK $\geq$ 17 $\times$ NEQ;
METHOD = 2, LENWRK $\geq$ 32 $\times$ NEQ};
METHOD = 3, LENWRK $\geq$ 21 $\times$ NEQ};
if TASK = 'C' and ERRASS = .FALSE. and
METHOD = 1, LENWRK $\geq$ 10 $\times$ NEQ;
METHOD = 2, LENWRK $\geq$ 14 $\times$ NEQ;
METHOD = 3, LENWRK $\geq$ 16 $\times$ NEQ;
if TASK = 'C' and ERRASS = .TRUE. and
METHOD = 1, LENWRK $\geq$ 15 $\times$ NEQ;
METHOD = 2, LENWRK $\geq$ 26 $\times$ NEQ;
METHOD = 3, LENWRK $\geq$ 21 $\times$ NEQ.

13: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

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 detected by the routine:

IFAIL = 1

On entry, NEQ < 1,
or TEND is too close to TSTART,
or TOL > 0.01 or TOL < 10.0 $\times$ ***machine precision***,
or THRES(i) < $\sqrt{\sigma}$, where σ is approximately the smallest possible machine number that can be reciprocated without overflow (see X02AMF),
or METHOD $\neq$ 1,2 or 3,
or TASK $\neq$ 'U' or 'C',
or LENWRK is too small.

7 Accuracy

Not applicable.

8 Further Comments

If TASK = 'C' then the value of the parameter TEND may be reset during the integration without the overhead associated with a complete restart; this can be achieved by a call to D02PWF.

It is often the case that a solution component y_i is of no interest when it is smaller in magnitude than a certain threshold. You can inform the code of this by setting THRES(i) to this threshold. In this way you avoid the cost of computing significant digits in y_i when only the fact that i) to this threshold. In this way you avoid the cost of comit is smaller than the threshold is of interest. This matter is important when y_i vanishes, and in particular, when the initial value YSTART(i) vanishes. An appropriate threshold depends on the general size of y_i in the course of the integration. Physical reasoning may help you select suitable threshold values. If you do not know what to expect of y, you can find out by a preliminary integration using D02PCF with nominal values of THRES. As D02PCF steps from t_0 towards t_f for each $i = 1, 2, \ldots, n$ it forms YMAX(i), the largest magnitude of y_i computed at any step in the integration so far. Using this you can determine more appropriate values for THRES for an accurate integration. You might, for example, take THRES(i) to be 10.0× ***machine precision*** times the final value of YMAX(i).

9 Example

See Section 9 of the document for D02PCF, Section 9 of the document for D02PDF, Section 9 of the document for D02PXF, Section 9 of the document for D02PWF and Section 9 of the document for D02PZF.

D02PWF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02PWF resets the end-point in an integration performed by D02PDF.

2. Specification

```
SUBROUTINE D02PWF (TENDNU, IFAIL)
INTEGER        IFAIL
real           TENDNU
```

3. Description

D02PWF and its associated routines (D02PVF, D02PDF, D02PXF, D02PYF, D02PZF) solve the initial value problem for a first order system of ordinary differential equations. The routines, based on Runge-Kutta methods and derived from RKSUITE [1], integrate

$$y' = f(t,y) \text{ given } y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

D02PWF is used to reset the the final value of the independent variable, t_f when the integration is already underway. It can be used to extend or reduce the range of integration. The new value must be beyond the current value of the independent variable (as returned in TNOW by D02PDF) in the current direction of integration. It is much more efficient to use D02PWF for this purpose than to use D02PVF which involves the overhead of a complete restart of the integration.

If you want to change the direction of integration then you must restart by a call to D02PVF.

4. References

[1] BRANKIN, R.W., GLADWELL, I. and SHAMPINE, L.F.
RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs.
SoftReport 91-S1, Department of Mathematics, Southern Methodist University, Dallas, TX 75275, U.S.A, 1991.

5. Parameters

1: TENDNU – ***real***. *Input*

On entry: the new value for t_f.

Constraints: sign(TENDNU – TNOW) = sign(TEND – TSTART), where TSTART and TEND are as supplied in the previous call to D02PVF and TNOW is returned by the preceding call to D02PDF.
TENDNU must be distinguishable from TNOW for the method and the precision of the machine being used.

2: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, an invalid input value for TENDNU was detected or an invalid call to D02PWF was made, for example without a previous call to the integration routine D02PDF. If on entry IFAIL = 0 or −1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF). You cannot continue integrating the problem.

7. Accuracy

Not applicable.

8. Further Comments

None.

9. Example

We integrate a two body problem. The equations for the coordinates $(x(t),y(t))$ of one body as functions of time t in a suitable frame of reference are

$$x'' = -\frac{x}{r^3}$$

$$y'' = -\frac{y}{r^3}, \quad r = \sqrt{x^2+y^2}.$$

The initial conditions

$$x(0) = 1-\varepsilon, \quad x'(0) = 0$$

$$y(0) = 0, \quad y'(0) = \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}$$

lead to elliptic motion with $0 < \varepsilon < 1$. We select $\varepsilon = 0.7$ and repose as

$$y_1' = y_3$$

$$y_2' = y_4$$

$$y_3' = -\frac{y_1}{r^3}$$

$$y_4' = -\frac{y_2}{r^3}$$

over the range $[0,6\pi]$. We use relative error control with threshold values of 1.0E−10 for each solution component and compute the solution at intervals of length π across the range using D02PWF to reset the end of the integration range. We use a high order Runge-Kutta method (METHOD = 3) with tolerances TOL = 1.0E−4 and TOL = 1.0E−5 in turn so that we may compare the solutions. The value of π is obtained by using X01AAF.

Note that the length of WORK is large enough for any valid combination of input arguments to D02PVF.

9.1. 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.

```
*     D02PWF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, LENWRK, METHOD
      PARAMETER        (NEQ=4,LENWRK=32*NEQ,METHOD=3)
      real             ZERO, ONE, SIX, ECC
      PARAMETER        (ZERO=0.0e0,ONE=1.0e0,SIX=6.0e0, ECC=0.7e0)
```

```
*     .. Local Scalars ..
      real             HNEXT, HSTART, PI, TEND, TFINAL, TINC, TNOW, TOL,
     +                 TSTART, WASTE
      INTEGER          I, IFAIL, J, L, NPTS, STPCST, STPSOK, TOTF
      LOGICAL          ERRASS
*     .. Local Arrays ..
      real             THRES(NEQ), WORK(LENWRK), YNOW(NEQ), YPNOW(NEQ),
     +                 YSTART(NEQ)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02PDF, D02PVF, D02PWF, D02PYF, F
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02PWF Example Program Results'
*
*     Set initial conditions and input for D02PVF
*
      PI = X01AAF(ZERO)
      TSTART = ZERO
      YSTART(1) = ONE - ECC
      YSTART(2) = ZERO
      YSTART(3) = ZERO
      YSTART(4) = SQRT((ONE+ECC)/(ONE-ECC))
      TFINAL = SIX*PI
      DO 20 L = 1, NEQ
         THRES(L) = 1.0e-10
   20 CONTINUE
      ERRASS = .FALSE.
      HSTART = ZERO
*
*     Set output control
*
      NPTS = 6
      TINC = TFINAL/NPTS
*
      DO 60 I = 1, 2
         IF (I.EQ.1) TOL = 1.0e-4
         IF (I.EQ.2) TOL = 1.0e-5
         J = NPTS - 1
         TEND = TFINAL - J*TINC
         IFAIL = 0
         CALL D02PVF(NEQ,TSTART,YSTART,TEND,TOL,THRES,METHOD,
     +               'Complex Task',ERRASS,HSTART,WORK,LENWRK,IFAIL)
*
         WRITE (NOUT,'(/A,D8.1)') 'Calculation with TOL = ', TOL
         WRITE (NOUT,'(/A/)') '    t         y1         y2'//
     +     '         y3         y4'
         WRITE (NOUT,'(1X,F6.3,4(3X,F8.4))') TSTART, (YSTART(L),L=1,NEQ)
*
   40    CONTINUE
         IFAIL = -1
         CALL D02PDF(F,TNOW,YNOW,YPNOW,WORK,IFAIL)
*
         IF (IFAIL.EQ.0) THEN
            IF (TNOW.LT.TEND) GO TO 40
            WRITE (NOUT,'(1X,F6.3,4(3X,F8.4))') TNOW, (YNOW(L),L=1,NEQ)
            IF (TNOW.LT.TFINAL) THEN
               J = J - 1
               TEND = TFINAL - J*TINC
               CALL D02PWF(TEND,IFAIL)
               GO TO 40
            END IF
         END IF
*
         IFAIL = 0
         CALL D02PYF(TOTF,STPCST,WASTE,STPSOK,HNEXT,IFAIL)
```

```
          WRITE (NOUT,'(/A,I6)')
     +      ' Cost of the integration in evaluations of F is', TOTF
*
   60 CONTINUE
*
      STOP
      END
      SUBROUTINE F(T,Y,YP)
*     .. Scalar Arguments ..
      real              T
*     .. Array Arguments ..
      real              Y(*), YP(*)
*     .. Local Scalars ..
      real              R
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      R = SQRT(Y(1)**2+Y(2)**2)
      YP(1) = Y(3)
      YP(2) = Y(4)
      YP(3) = -Y(1)/R**3
      YP(4) = -Y(2)/R**3
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02PWF Example Program Results

Calculation with TOL =  0.1E-03

    t         y1         y2         y3         y4

  0.000     0.3000     0.0000     0.0000     2.3805
  3.142    -1.7000     0.0000     0.0000    -0.4201
  6.283     0.3000     0.0000     0.0001     2.3805
  9.425    -1.7000     0.0000     0.0000    -0.4201
 12.566     0.3000    -0.0003     0.0016     2.3805
 15.708    -1.7001     0.0001    -0.0001    -0.4201
 18.850     0.3000    -0.0010     0.0045     2.3805

 Cost of the integration in evaluations of F is   571

Calculation with TOL =  0.1E-04

    t         y1         y2         y3         y4

  0.000     0.3000     0.0000     0.0000     2.3805
  3.142    -1.7000     0.0000     0.0000    -0.4201
  6.283     0.3000     0.0000     0.0000     2.3805
  9.425    -1.7000     0.0000     0.0000    -0.4201
 12.566     0.3000    -0.0001     0.0004     2.3805
 15.708    -1.7000     0.0000     0.0000    -0.4201
 18.850     0.3000    -0.0003     0.0012     2.3805

 Cost of the integration in evaluations of F is   748
```

D02PXF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02PXF computes the solution of a system of ordinary differential equations using interpolation anywhere on an integration step taken by D02PDF.

2. Specification

```
      SUBROUTINE D02PXF (TWANT, REQEST, NWANT, YWANT, YPWANT, F, WORK,
     1                   WRKINT, LENINT, IFAIL)
      INTEGER          NWANT, LENINT, IFAIL
      real             TWANT, YWANT(*), YPWANT(*), WORK(*), WRKINT(LENINT)
      CHARACTER*1      REQEST
      EXTERNAL         F
```

3. Description

D02PXF and its associated routines (D02PVF, D02PDF, D02PWF, D02PYF, D02PZF) solve the initial value problem for a first order system of ordinary differential equations. The routines, based on Runge-Kutta methods and derived from RKSUITE [1], integrate

$$y' = f(t,y) \text{ given } y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

D02PDF computes the solution at the end of an integration step. Using the information computed on that step D02PXF computes the solution by interpolation at any point on that step. It cannot be used if METHOD = 3 was specified in the call to set-up routine D02PVF.

4. References

[1] BRANKIN, R.W., GLADWELL, I. and SHAMPINE, L.F.
RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs.
SoftReport 91-S1, Department of Mathematics, Southern Methodist University, Dallas, TX 75275, U.S.A, 1991.

5. Parameters

1: TWANT – ***real***. *Input*

On entry: the value of the independent variable, t, where a solution is desired.

2: REQEST – CHARACTER*1. *Input*

On entry: determines whether the solution and/or its first derivative are to be computed as follows:

REQEST = 'S' - compute the approximate solution only;
REQEST = 'D' - compute the approximate first derivative of the solution only;
REQEST = 'B' - compute both the approximate solution and its first derivative.

Constraint: REQEST = 'S', 'D' or 'B'.

3: NWANT – INTEGER. *Input*

On entry: the number of components of the solution to be computed. The first NWANT components are evaluated

Constraint: $1 \leq$ NWANT $\leq n$, where n is specified by NEQ the prior call to D02PVF.

4: YWANT(*) – ***real*** array. *Output*

Note: when REQEST = 'S' or 'B', the dimension of the array YWANT must be at least NWANT and at least 1 otherwise.

On exit: an approximation to the first NWANT components of the solution at TWANT if REQEST = 'S' or 'B'. Otherwise YWANT is not defined.

5: YPWANT(*) – ***real*** array. *Output*

Note: when REQEST = 'D' or 'B', the dimension of the array YPWANT must be at least NWANT and at least 1 otherwise.

On exit: an approximation to the first NWANT components of the the first derivative at TWANT if REQEST = 'D' or 'B'. Otherwise YPWANT is not defined.

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

F must evaluate the functions f_i (that is the first derivatives y_i') for given values of the arguments t,y_i. It must be the same procedure as supplied to D02PDF.

Its specification is:

```
SUBROUTINE F (T, Y, YP)
real          T, Y(*), YP(*)
```

1: T – ***real***. *Input*

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

2: Y(*) – ***real*** array. *Input*

On entry: the current values of the dependent variables, y_i for $i = 1,2,...,n$.

3: YP(*) – ***real*** array. *Output*

On exit: the values of f_i for $i = 1,2,...,n$.

F must be declared as EXTERNAL in the (sub)program from which D02PXF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: WORK(*) – ***real*** array. *Input/Output*

On entry: this **must** be the same array as supplied to D02PDF and **must** remain unchanged between calls.

On exit: contains information about the integration for use on subsequent calls to D02PDF or other associated routines.

8: WRKINT(LENINT) – ***real*** array. *Input/Output*

On entry: must be the same array as supplied in previous calls, if any, and must remain unchanged between calls to D02PXF.

On exit: the contents are modified.

9: LENINT – INTEGER. *Input*

On entry: the dimension of the array WRKINT as declared in the (sub)program from which D02PXF is called.

Constraints: LENINT ≥ 1 if METHOD = 1 in the prior call to D02PVF.
LENINT ≥ n + 5×NWANT if METHOD = 2 and n is specified by NEQ in the prior call of D02PVF.

10: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

On entry, an invalid input value for NWANT or LENINT was detected or an invalid call to D02PXF was made, for example without a previous call to the integration routine D02PDF, or after an error return from D02PDF, or if D02PDF was being used with METHOD = 3. If on entry IFAIL = 0 or –1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF). You cannot continue integrating the problem.

7. Accuracy

The computed values will be of a similar accuracy to that computed by D02PDF.

8. Further Comments

None.

9. Example

We solve the equation

$$y'' = -y, \quad y(0) = 0,\ y'(0) = 1$$

reposed as

$$y_1' = y_2$$
$$y_2' = -y_1$$

over the range $[0,2\pi]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$. We use relative error control with threshold values of 1.0E–8 for each solution component. D02PDF is used to integrate the problem one step at a time and D02PXF is used to compute the first component of the solution and its derivative at intervals of length $\pi/8$ across the range whenever these points lie in one of those integration steps. We use a moderate order Runge-Kutta method (METHOD = 2) with tolerances TOL = 1.0E–3 and TOL = 1.0E–4 in turn so that we may compare the solutions. The value of π is obtained by using X01AAF.

Note that the length of WORK is large enough for any valid combination of input arguments to D02PVF and the length of WRKINT is large enough for any valid value of the argument NWANT.

9.1. 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.

```
*     D02PXF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, NWANT, LENINT, LENWRK, METHOD
      PARAMETER        (NEQ=2,NWANT=1,LENINT=NEQ+5*NWANT,LENWRK=32*NEQ,
     +                 METHOD=2)
      real             ZERO, ONE, TWO
      PARAMETER        (ZERO=0.0e0,ONE=1.0e0,TWO=2.0e0)
```

```
*     .. Local Scalars ..
      real              HNEXT, HSTART, PI, TEND, TINC, TNOW, TOL, TSTART,
     +                  TWANT, WASTE
      INTEGER           I, IFAIL, J, L, NPTS, STPCST, STPSOK, TOTF
      LOGICAL           ERRASS
*     .. Local Arrays ..
      real              THRES(NEQ), WORK(LENWRK), WRKINT(LENINT),
     +                  YNOW(NEQ), YPNOW(NEQ), YPWANT(NWANT),
     +                  YSTART(NEQ), YWANT(NWANT)
*     .. External Functions ..
      real              X01AAF
      EXTERNAL          X01AAF
*     .. External Subroutines ..
      EXTERNAL          D02PDF, D02PVF, D02PXF, D02PYF, F
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02PXF Example Program Results'
*
*     Set initial conditions and input for D02PVF
*
      PI = X01AAF(ZERO)
      TSTART = ZERO
      YSTART(1) = ZERO
      YSTART(2) = ONE
      TEND = TWO*PI
      DO 20 L = 1, NEQ
         THRES(L) = 1.0e-8
   20 CONTINUE
      ERRASS = .FALSE.
      HSTART = ZERO
*
*     Set output control
*
      NPTS = 16
      TINC = TEND/NPTS
*
      DO 80 I = 1, 2
         IF (I.EQ.1) TOL = 1.0e-3
         IF (I.EQ.2) TOL = 1.0e-4
*
         IFAIL = 0
         CALL D02PVF(NEQ,TSTART,YSTART,TEND,TOL,THRES,METHOD,
     +               'Complex Task',ERRASS,HSTART,WORK,LENWRK,IFAIL)
*
         WRITE (NOUT,'(/A,D8.1)') 'Calculation with TOL = ', TOL
         WRITE (NOUT,'(/A/)') '    t         y1        y1'''
         WRITE (NOUT,'(1X,F6.3,2(3X,F8.4))') TSTART, (YSTART(L),L=1,NEQ)
*
         J = NPTS - 1
         TWANT = TEND - J*TINC
*
   40    CONTINUE
         IFAIL = -1
         CALL D02PDF(F,TNOW,YNOW,YPNOW,WORK,IFAIL)
*
         IF (IFAIL.EQ.0) THEN
   60       CONTINUE
            IF (TWANT.LE.TNOW) THEN
               IFAIL = 0
               CALL D02PXF(TWANT,'Both',NWANT,YWANT,YPWANT,F,WORK,
     +                     WRKINT,LENINT,IFAIL)
               WRITE (NOUT,'(1X,F6.3,2(3X,F8.4))') TWANT, YWANT(1),
     +           YPWANT(1)
               J = J - 1
               TWANT = TEND - J*TINC
               GO TO 60
            END IF
            IF (TNOW.LT.TEND) GO TO 40
         END IF
*
```

```
            IFAIL = 0
            CALL D02PYF(TOTF,STPCST,WASTE,STPSOK,HNEXT,IFAIL)
            WRITE (NOUT,'(/A,I6)')
     +        ' Cost of the integration in evaluations of F is', TOTF
*
   80 CONTINUE
*
      STOP
      END
      SUBROUTINE F(T,Y,YP)
*     .. Scalar Arguments ..
      real              T
*     .. Array Arguments ..
      real              Y(*), YP(*)
*     .. Executable Statements ..
      YP(1) = Y(2)
      YP(2) = -Y(1)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02PXF Example Program Results

Calculation with TOL =  0.1E-02

    t          y1          y1'

  0.000      0.0000      1.0000
  0.393      0.3827      0.9239
  0.785      0.7071      0.7071
  1.178      0.9239      0.3826
  1.571      1.0000     -0.0001
  1.963      0.9238     -0.3828
  2.356      0.7070     -0.7073
  2.749      0.3825     -0.9240
  3.142     -0.0002     -0.9999
  3.534     -0.3829     -0.9238
  3.927     -0.7072     -0.7069
  4.320     -0.9239     -0.3823
  4.712     -0.9999      0.0004
  5.105     -0.9236      0.3830
  5.498     -0.7068      0.7073
  5.890     -0.3823      0.9239
  6.283      0.0004      0.9998

 Cost of the integration in evaluations of F is    68

Calculation with TOL =  0.1E-03

    t          y1          y1'

  0.000      0.0000      1.0000
  0.393      0.3827      0.9239
  0.785      0.7071      0.7071
  1.178      0.9239      0.3827
  1.571      1.0000      0.0000
  1.963      0.9239     -0.3827
  2.356      0.7071     -0.7071
  2.749      0.3827     -0.9239
  3.142      0.0000     -1.0000
  3.534     -0.3827     -0.9239
  3.927     -0.7071     -0.7071
  4.320     -0.9239     -0.3827
  4.712     -1.0000      0.0000
  5.105     -0.9238      0.3827
```

```
  5.498    -0.7071     0.7071
  5.890    -0.3826     0.9239
  6.283     0.0000     1.0000

 Cost of the integration in evaluations of F is   105
```

D02PYF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02PYF provides details about an integration performed by either D02PCF or D02PDF.

2. Specification

```
SUBROUTINE D02PYF (TOTFCN, STPCST, WASTE, STPSOK, HNEXT, IFAIL)
INTEGER        TOTFCN, STPCST, STPSOK, IFAIL
real           WASTE, HNEXT
```

3. Description

D02PYF and its associated routines (D02PCF, D02PDF, D02PVF, D02PWF, D02PXF, D02PZF) solve the initial value problem for a first order system of ordinary differential equations. The routines, based on Runge-Kutta methods and derived from RKSUITE [1], integrate

$$y' = f(t,y) \text{ given } y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

After a call to D02PCF or D02PDF, D02PYF can be called to obtain information about the cost of the integration and the size of the next step.

4. References

[1] BRANKIN, R.W., GLADWELL, I. and SHAMPINE, L.F.
RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs.
SoftReport 91-S1, Department of Mathematics, Southern Methodist University, Dallas, TX 75275, U.S.A, 1991.

5. Parameters

1: TOTFCN – INTEGER. *Output*

On exit: the total number of evaluations of f used in the primary integration so far; this does not include evaluations of f for the secondary integration specified by a prior call to D02PVF with ERRASS = .TRUE..

2: STPCST – INTEGER. *Output*

On exit: the cost in terms of number of evaluations of f of a typical step with the method being used for the integration. The method is specified by the parameter METHOD in a prior call to D02PVF.

3: WASTE – ***real***. *Output*

On exit: the number of attempted steps that failed to meet the local error requirement divided by the total number of steps attempted so far in the integration. A "large" fraction indicates that the integrator is having trouble with the problem being solved. This can happen when the problem is "stiff" and also when the solution has discontinuities in a low order derivative.

4: STPSOK – INTEGER. *Output*

On exit: the number of accepted steps.

5: HNEXT – ***real***. *Output*

On exit: the step size the integrator will attempt to use for the next step.

6: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

An invalid call to D02PYF has been made, for example without a previous call to D02PCF or D02PDF. If on entry IFAIL = 0 or –1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF). You cannot continue integrating the problem.

7. Accuracy

Not applicable.

8. Further Comments

When a secondary integration has taken place, that is when global error assessment has been specified using ERRASS = .TRUE. in a prior call to D02PVF, then the approximate extra number of evaluations of f used is given by 2×STPSOK×STPCST for METHOD = 2 or 3 and 3×STPSOK×STPCST for METHOD = 1.

9. Example

See the example programs for D02PCF, D02PDF, D02PWF, D02PXF and D02PZF.

D02PZF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02PZF provides details about global error assessment computed during an integration with either D02PCF or D02PDF.

2. Specification

```
SUBROUTINE D02PZF (RMSERR, ERRMAX, TERRMX, WORK, IFAIL)
INTEGER        IFAIL
real           RMSERR(*), ERRMAX, TERRMX, WORK(*)
```

3. Description

D02PZF and its associated routines (D02PCF, D02PDF, D02PVF, D02PWF, D02PXF, D02PYF) solve the initial value problem for a first order system of ordinary differential equations. The routines, based on Runge-Kutta methods and derived from RKSUITE [1], integrate

$$y' = f(t,y) \text{ given } y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

After a call to D02PCF or D02PDF, D02PZF can be called for information about error assessment, if this assessment was specified in the setup routine D02PVF. A more accurate "true" solution $\hat{y}$ is computed in a secondary integration. The error is measured as specified in D02PVF for local error control. At each step in the primary integration, an average magnitude μ_i of component y_i is computed, and the error in the component is

$$\frac{|y_i - \hat{y}_i|}{\max(\mu_i, \mathrm{THRES}(i))}.$$

It is difficult to estimate reliably the true error at a single point. For this reason the RMS (root-mean-square) average of the estimated global error in each solution component is computed. This average is taken over all steps from the beginning of the integration through to the current integration point. If all has gone well, the average errors reported will be comparable to TOL (see D02PVF). The maximum error seen in any component in the integration so far and the point where the maximum error first occurred are also reported.

4. References

[1] BRANKIN, R.W., GLADWELL, I. and SHAMPINE, L.F.
RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs.
SoftReport 91-S1, Department of Mathematics, Southern Methodist University, Dallas, TX 75275, U.S.A, 1991.

5. Parameters

1: RMSERR(*) – ***real*** array. *Output*

Note: the dimension of the array RMSERR must be at least n.

On exit: RMSERR(i) approximates the RMS average of the true error of the numerical solution for the ith solution component, for $i = 1,2,...,n$. The average is taken over all steps from the beginning of the integration to the current integration point.

2: ERRMAX – ***real***. *Output*

On exit: the maximum weighted approximate true error taken over all solution components and all steps.

3: TERRMX – ***real.*** *Output*

On exit: the first value of the independent variable where an approximate true error attains the maximum value, ERRMAX.

4: WORK(*) – ***real*** array. *Input*

On entry: this **must** be the same array as supplied to D02PCF or D02PDF and **must** remain unchanged between calls.

5: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

An invalid call to D02PZF has been made, for example without a previous call to D02PCF or D02PDF, or without error assessment having been specified in a call to D02PVF. If on entry IFAIL = 0 or –1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF). You cannot continue integrating the problem.

7. Accuracy

Not applicable.

8. Further Comments

If the integration has proceeded "well" and the problem is smooth enough, stable and not too difficult then the values returned in the arguments RMSERR and ERRMAX should be comparable to the value of TOL specified in the prior call to D02PVF.

9. Example

We integrate a two body problem. The equations for the coordinates $(x(t),y(t))$ of one body as functions of time t in a suitable frame of reference are

$$x'' = -\frac{x}{r^3}$$

$$y'' = -\frac{y}{r^3}, \quad r = \sqrt{x^2+y^2}.$$

The initial conditions

$$x(0) = 1-\varepsilon, \quad x'(0) = 0$$

$$y(0) = 0, \quad y'(0) = \sqrt{\frac{1+\varepsilon}{1-\varepsilon}}$$

lead to elliptic motion with $0 < \varepsilon < 1$. We select $\varepsilon = 0.7$ and repose as

$$y_1' = y_3$$

$$y_2' = y_4$$

$$y_3' = -\frac{y_1}{r^3}$$

$$y_4' = -\frac{y_2}{r^3}$$

over the range $[0,3\pi]$. We use relative error control with threshold values of 1.0E−10 for each solution component and a high order Runge-Kutta method (METHOD = 3) with tolerance TOL = 1.0E−6. The value of π is obtained by using X01AAF.

Note that the length of WORK is large enough for any valid combination of input arguments to D02PVF. Note also, for illustration purposes since it is not necessary for this problem, we choose to integrate the to the end of the range regardless of efficiency concerns (i.e. returns from D02PCF with IFAIL = 2,3,4).

9.1. 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.

```
*     D02PZF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, LENWRK, METHOD
      PARAMETER        (NEQ=4,LENWRK=32*NEQ,METHOD=3)
      real             ZERO, ONE, THREE, ECC
      PARAMETER        (ZERO=0.0e0,ONE=1.0e0,THREE=3.0e0, ECC=0.7e0)
*     .. Local Scalars ..
      real             ERRMAX, HNEXT, HSTART, PI, TEND, TERRMX, TGOT,
     +                 TOL, TSTART, TWANT, WASTE
      INTEGER          IFAIL, L, STPCST, STPSOK, TOTF
      LOGICAL          ERRASS
*     .. Local Arrays ..
      real             RMSERR(NEQ), THRES(NEQ), WORK(LENWRK), YGOT(NEQ),
     +                 YMAX(NEQ), YPGOT(NEQ), YSTART(NEQ)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02PCF, D02PVF, D02PYF, D02PZF, F
*     .. Intrinsic Functions ..
      INTRINSIC        SQRT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02PZF Example Program Results'
*
*     Set initial conditions and input for D02PVF
*
      PI = X01AAF(ZERO)
      TSTART = ZERO
      YSTART(1) = ONE - ECC
      YSTART(2) = ZERO
      YSTART(3) = ZERO
      YSTART(4) = SQRT((ONE+ECC)/(ONE-ECC))
      TEND = THREE*PI
      DO 20 L = 1, NEQ
         THRES(L) = 1.0e-10
   20 CONTINUE
      ERRASS = .TRUE.
      HSTART = ZERO
      TOL = 1.0e-6
      IFAIL = 0
      CALL D02PVF(NEQ,TSTART,YSTART,TEND,TOL,THRES,METHOD,'Usual Task',
     +            ERRASS,HSTART,WORK,LENWRK,IFAIL)
*
      WRITE (NOUT,'(/A,D8.1)') 'Calculation with TOL = ', TOL
      WRITE (NOUT,'(/A/)') '    t         y1         y2'//
     + '         y3         y4'
      WRITE (NOUT,'(1X,F6.3,4(3X,F8.4))') TSTART, (YSTART(L),L=1,NEQ)
*
```

```
      TWANT = TEND
*
   40 CONTINUE
      IFAIL = 1
      CALL D02PCF(F,TWANT,TGOT,YGOT,YPGOT,YMAX,WORK,IFAIL)
*
      IF (IFAIL.GE.2 .AND. IFAIL.LE.4) THEN
         GO TO 40
      ELSE IF (IFAIL.NE.0) THEN
         WRITE (NOUT,'(A,I2)') ' D02PCF returned with IFAIL set to',
     +     IFAIL
      ELSE
         WRITE (NOUT,'(1X,F6.3,4(3X,F8.4))') TGOT, (YGOT(L),L=1,NEQ)
*
         IFAIL = 0
         CALL D02PZF(RMSERR,ERRMAX,TERRMX,WORK,IFAIL)
         WRITE (NOUT,'(/A/9X,4(2X,E9.2))')
     +     ' Componentwise error '//'assessment', (RMSERR(L),L=1,NEQ)
         WRITE (NOUT,'(/A,E9.2,A,F6.3)')
     +     ' Worst global error observed '//'was ', ERRMAX,
     +     ' - it occurred at T = ', TERRMX
*
         IFAIL = 0
         CALL D02PYF(TOTF,STPCST,WASTE,STPSOK,HNEXT,IFAIL)
         WRITE (NOUT,'(/A,I6)')
     +     ' Cost of the integration in evaluations of F is', TOTF
      END IF
*
      STOP
      END
      SUBROUTINE F(T,Y,YP)
*     .. Scalar Arguments ..
      real              T
*     .. Array Arguments ..
      real              Y(*), YP(*)
*     .. Local Scalars ..
      real              R
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      R = SQRT(Y(1)**2+Y(2)**2)
      YP(1) = Y(3)
      YP(2) = Y(4)
      YP(3) = -Y(1)/R**3
      YP(4) = -Y(2)/R**3
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02PZF Example Program Results

Calculation with TOL =  0.1E-05

   t         y1          y2          y3          y4

 0.000     0.3000      0.0000      0.0000      2.3805
 9.425    -1.7000      0.0000      0.0000     -0.4201

 Componentwise error assessment
           0.38E-05    0.71E-05    0.69E-05    0.21E-05

 Worst global error observed was  0.34E-04 - it occurred at T =  6.302

 Cost of the integration in evaluations of F is  1361
```

D02QFF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02QFF is a routine for integrating a non-stiff system of first order ordinary differential equations using a variable-order variable-step Adams method. A root-finding facility is provided.

2. Specification

```
      SUBROUTINE D02QFF (FCN, NEQF, T, Y, TOUT, G, NEQG, ROOT, RWORK,
     1                   LRWORK, IWORK, LIWORK, IFAIL)
      INTEGER          NEQF, NEQG, LRWORK, IWORK(LIWORK), LIWORK, IFAIL
      real             T, Y(NEQF), TOUT, G, RWORK(LRWORK)
      LOGICAL          ROOT
      EXTERNAL         FCN, G
```

3. Description

Given the initial values $x, y_1, y_2, \ldots, y_{\mathrm{NEQF}}$ the routine integrates a non-stiff system of first order differential equations of the type, $y_i' = f_i(x, y_1, y_2, \ldots, y_{\mathrm{NEQF}})$, for $i = 1,2,\ldots,\mathrm{NEQF}$, from $x = \mathrm{T}$ to $x = \mathrm{TOUT}$ using a variable-order variable-step Adams method. The system is defined by a subroutine FCN supplied by the user, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_{\mathrm{NEQF}}$, and $y_1, y_2, \ldots, y_{\mathrm{NEQF}}$ are supplied at $x = \mathrm{T}$. The routine is capable of finding roots (values of x) of prescribed event functions of the form

$$g_j(x,y,y') = 0, \qquad j = 1,2,\ldots,\mathrm{NEQG}.$$

Each g_j is considered to be independent of the others so that roots are sought of each g_j individually. The root reported by the routine will be the first root encountered by any g_j. Two techniques for determining the presence of a root in an integration step are available: the sophisticated method described in Watts [3] and a simplified method whereby sign changes in each g_j are looked for at the ends of each integration step. The event functions are defined by a real function G supplied by the user which evaluates g_j in terms of $x, y_1, \ldots, y_{\mathrm{NEQF}}$ and $y_1', \ldots, y_{\mathrm{NEQF}}'$. In one-step mode the routine returns an approximation to the solution at each integration point. In interval mode this value is returned at the end of the integration range. If a root is detected this approximation is given at the root. The user selects the mode of operation, the error control, the root-finding technique and various optional inputs by a prior call of the setup routine D02QWF.

For a description of the practical implementation of an Adams formula see Shampine and Gordon [1] and Shampine and Watts [2].

4. References

[1] SHAMPINE, L.F. and GORDON, M.K.
Computer Solution of Ordinary Differential Equations – The Initial Value Problem.
WH Freeman & Co., San Fransisco, 1975.

[2] SHAMPINE, L.F. and WATTS, H.A.
DEPAC – Design of a user oriented package of ODE solvers.
Sandia National Laboratory Report SAND79-2374, 1979.

[3] WATTS, H.A.
RDEAM – An Adams ODE Code with Root Solving Capability.
Sandia National Laboratory Report SAND85-1595, 1985.

5. Parameters

1: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (that is the first derivatives y_i') for given values of its arguments x, y_1, y_2,...,y_{NEQF}.

Its specification is:

```
SUBROUTINE FCN(NEQF, X, Y, F)
INTEGER    NEQF
real       X, Y(NEQF), F(NEQF)
```

1: NEQF – INTEGER. *Input*

On entry: the number of differential equations.

2: X – ***real***. *Input*

On entry: the current value of the argument x.

3: Y(NEQF) – ***real*** array. *Input*

On entry: the current value of the argument y_i, for i = 1,2,...,NEQF.

4: F(NEQF) – ***real*** array. *Output*

On exit: the value of f_i, for i = 1,2,...,NEQF.

FCN must be declared as EXTERNAL in the (sub)program from which D02QFF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: NEQF – INTEGER. *Input*

On entry: the number of first order ordinary differential equations to be solved by D02QFF. It must contain the same value as the parameter NEQF used in a prior call of D02QWF.

Constraint: NEQF $\geq$ 1.

3: T – ***real***. *Input/Output*

On entry: after a call to D02QWF with STATEF = 'S' (i.e. an initial entry), T must be set to the initial value of the independent variable x.

On exit: the value of x at which y has been computed. This may be an intermediate output point, a root, TOUT or a point at which an error has occurred. If the integration is to be continued, possibly with a new value for TOUT, T must not be changed.

4: Y(NEQF) – ***real*** array. *Input/Output*

On entry: the initial values of the solution y_1,y_2,..., y_{NEQF}.

On exit: the computed values of the solution at the exit value of T. If the integration is to be continued, possibly with a new value for TOUT, these values must not be changed.

5: TOUT – ***real***. *Input*

On entry: the next value of x at which a computed solution is required. For the initial T, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction. If TOUT = T on exit, TOUT must be reset beyond T **in the direction of integration**, before any continuation call.

6: G – ***real*** FUNCTION, supplied by the user. *External Procedure*

G must evaluate a given component of $g(x,y,y')$ at a specifed point.

If root-finding is not required the actual argument for G must be the dummy routine D02QFZ. (D02QFZ is included in the NAG Fortran Library and so need not be supplied by the user. Its name may be implemention dependent: see the Users' Note for your implementation for details.)

Its specification is:

```
real FUNCTION G(NEQF, X, Y, YP, K)
INTEGER       NEQF, K
real          X, Y(NEQF), YP(NEQF)
```

1: NEQF – INTEGER. *Input*

On entry: the number of differential equations being solved.

2: X – ***real***. *Input*

On entry: the current value of the independent variable.

3: Y(NEQF) – ***real*** array. *Input*

On entry: the current values of the dependent variables.

4: YP(NEQF) – ***real*** array. *Input*

On entry: the current values of the derivatives of the dependent variables.

5: K – INTEGER. *Input*

On entry: the component of g which must be evaluated.

G must be declared as EXTERNAL in the (sub)program from which D02QFF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: NEQG – INTEGER. *Input*

On entry: the number of event functions which the user is defining for root-finding. If root-finding is not required the value for NEQG must be ≤ 0. Otherwise it must be the same parameter NEQG used in the prior call to D02QWF.

8: ROOT – LOGICAL. *Output*

On exit: if root-finding was required (NEQG > 0 on entry), then ROOT specifies whether or not the output value of the parameter T is a root of one of the event functions. If ROOT = .FALSE., then no root was detected, whereas ROOT = .TRUE. indicates a root and the user should make a call to D02QYF for further information.

If root-finding was not required (NEQG = 0 on entry) then on exit ROOT = .FALSE..

9: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK as supplied to D02QWF. It is used to pass information from D02QWF to D02QFF, and from D02QFF to D02QXF, D02QYF and D02QZF. Therefore the contents of this array **must not** be changed before the call to D02QFF or calling any of the routines D02QXF, D02QYF and D02QZF.

10: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02QFF is called.

This must be the same parameter LRWORK as supplied to D02QWF.

11: IWORK(LIWORK) – INTEGER array. *Workspace*

This **must** be the same parameter IWORK as supplied to D02QWF. It is used to pass information from D02QWF to D02QFF, and from D02QFF to D02QXF, D02QYF and D02QZF. Therefore the contents of this array **must not** be changed before the call to D02QFF or calling any of the routines D02QXF, D02QYF and D02QZF.

12: LIWORK – INTEGER. *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02QFF is called.

This must be the same parameter LIWORK as supplied to D02QWF.

13: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

On entry the integrator detected an illegal input, or D02QWF has not been called prior to the call to the integrator. If on entry IFAIL = 0 or –1, the form of the error will be detailed on the current error message unit (as defined by X04AAF).

This error may be caused by overwriting elements of RWORK and IWORK.

IFAIL = 2

The maximum number of steps has been attempted (at a cost of about 2 calls to FCN per step). (See parameter MAXSTP in D02QWF.) If integration is to be continued then the user need only reset IFAIL and call the routine again and a further MAXSTP steps will be attempted.

IFAIL = 3

The step size needed to satisfy the error requirements is too small for the ***machine precision*** being used. (See parameter TOLFAC in D02QXF.)

IFAIL = 4

Some error weight w_i became zero during the integration (see parameters VECTOL, RTOL and ATOL in D02QWF.) Pure relative error control (ATOL = 0.0) was requested on a variable (the ith) which has now become zero. (See parameter BADCMP in D02QXF.) The integration was successful as far as T.

IFAIL = 5

The problem appears to be stiff (see the Chapter Introduction for a discussion of the term 'stiff'). Although it is inefficient to use this integrator to solve stiff problems, integration may be continued by resetting IFAIL and calling the routine again.

IFAIL = 6

A change in sign of an event function has been detected but the root-finding process appears to have converged to a singular point T rather than a root. Integration may be continued by resetting IFAIL and calling the routine again.

IFAIL = 7

The code has detected two successive error exits at the current value of T and cannot proceed. Check all input variables.

7. Accuracy

The accuracy of integration is determined by the parameters VECTOL, RTOL and ATOL in a prior call to D02QWF. Note that only the local error at each step is controlled by these parameters. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the properties of the differential equation system. The code is designed so that a reduction in the tolerances should lead to an approximately proportional reduction in the error. The user is strongly recommended to call D02QFF with more than one set of tolerances and to compare the results obtained to estimate their accuracy.

The accuracy obtained depends on the type of error test used. If the solution oscillates around zero a relative error test should be avoided, whereas if the solution is exponentially increasing an absolute error test should not be used. If different accuracies are required for different components of the solution then a component-wise error test should be used. For a description of the error test see the specifications of the parameters VECTOL, ATOL and RTOL in the routine document for D02QWF.

The accuracy of any roots located will depend on the accuracy of integration and may also be restricted by the numerical properties of $g(x,y,y')$. When evaluating g the user should try to write the code so that unnecessary cancellation errors will be avoided.

8. Further Comments

If the routine fails with IFAIL = 3 then the combination of ATOL and RTOL may be so small that a solution cannot be obtained, in which case the routine should be called again with larger values for RTOL and/or ATOL. If the accuracy requested is really needed then the user should consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity the solution components will usually be of a large magnitude. The routine could be used in one-step mode to monitor the size of the solution with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary;

(b) for 'stiff' equations, where the solution contains rapidly decaying components, the routine will require a very small stepsize to preserve stability. This will usually be exhibited by excessive computing time and sometimes an error exit with IFAIL = 3, but usually an error exit with IFAIL = 2 or 5. The Adams methods are not efficient in such cases and the user should consider using a routine from the D02M-D02N subchapter. A high proportion of failed steps (see parameter NFAIL in D02QXF) may indicate stiffness but there may be other reasons for this phenomenon.

D02QFF can be used for producing results at short intervals (for example, for graph plotting); the user should set CRIT = .TRUE. and TCRIT to the last output point required in a prior call to D02QWF and then set TOUT appropriately for each output point in turn in the call to D02QFF.

9. Example

We solve the equation

$$y'' = -y, \quad y(0) = 0, \ y'(0) = 1$$

reposed as

$$y_1' = y_2$$
$$y_2' = -y_1$$

over the range [0,10.0] with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$ using vector error control (VECTOL = .TRUE.) and computation of the solution at TOUT = 10.0 with TCRIT = 10.0 (CRIT = .TRUE.). Also, we use D02QFF to locate the positions where $y_1 = 0.0$ or where the first component has a turning point, that is $y_1' = 0.0$.

9.1. 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.

```
*     D02QFF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQF, NEQG, LATOL, LRTOL, LRWORK, LIWORK
      PARAMETER        (NEQF=2,NEQG=2,LATOL=NEQF,LRTOL=NEQF,
     +                 LRWORK=23+23*NEQF+14*NEQG,LIWORK=21+4*NEQG)
      real             TSTART, HMAX
      PARAMETER        (TSTART=0.0e0,HMAX=0.0e0)
*     .. Local Scalars ..
      real             HLAST, HNEXT, T, TCRIT, TCURR, TOLFAC, TOUT
      INTEGER          BADCMP, I, IFAIL, INDEX, MAXSTP, NFAIL, NSUCC,
     +                 ODLAST, ODNEXT, TYPE
      LOGICAL          ALTERG, CRIT, ONESTP, ROOT, SOPHST, VECTOL
      CHARACTER*1      STATEF
*     .. Local Arrays ..
      real             ATOL(LATOL), RESIDS(NEQG), RTOL(LRTOL),
     +                 RWORK(LRWORK), Y(NEQF), YP(NEQF)
      INTEGER          EVENTS(NEQG), IWORK(LIWORK)
*     .. External Functions ..
      real             GTRY02
      EXTERNAL         GTRY02
*     .. External Subroutines ..
      EXTERNAL         D02QFF, D02QWF, D02QXF, D02QYF, FTRY02
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02QFF Example Program Results'
      TCRIT = 10.0e0
      STATEF = 'S'
      VECTOL = .TRUE.
      ONESTP = .FALSE.
      CRIT = .TRUE.
      MAXSTP = 0
      SOPHST = .TRUE.
      DO 20 I = 1, NEQF
         RTOL(I) = 1.0e-4
         ATOL(I) = 1.0e-6
   20 CONTINUE
      IFAIL = 0
*
      CALL D02QWF(STATEF,NEQF,VECTOL,ATOL,LATOL,RTOL,LRTOL,ONESTP,CRIT,
     +            TCRIT,HMAX,MAXSTP,NEQG,ALTERG,SOPHST,RWORK,LRWORK,
     +            IWORK,LIWORK,IFAIL)
*
      T = TSTART
      TOUT = TCRIT
      Y(1) = 0.0e0
      Y(2) = 1.0e0
*
   40 IFAIL = -1
*
      CALL D02QFF(FTRY02,NEQF,T,Y,TOUT,GTRY02,NEQG,ROOT,RWORK,LRWORK,
     +            IWORK,LIWORK,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
*
         CALL D02QXF(NEQF,YP,TCURR,HLAST,HNEXT,ODLAST,ODNEXT,NSUCC,
     +               NFAIL,TOLFAC,BADCMP,RWORK,LRWORK,IWORK,LIWORK,
     +               IFAIL)
*
         IF (ROOT) THEN
*
```

```
            CALL D02QYF(NEQG,INDEX,TYPE,EVENTS,RESIDS,RWORK,LRWORK,
     +                  IWORK,LIWORK,IFAIL)
*
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Root at ', T
            WRITE (NOUT,99998) 'for event equation ', INDEX,
     +        ' with type', TYPE, ' and residual ', RESIDS(INDEX)
            WRITE (NOUT,99999) ' Y(1) = ', Y(1), '   Y''(1) = ', YP(1)
            DO 60 I = 1, NEQG
               IF (I.NE.INDEX) THEN
                  IF (EVENTS(I).NE.0) THEN
                     WRITE (NOUT,99998) 'and also for event equation ',
     +                 I, ' with type', EVENTS(I), ' and residual ',
     +                 RESIDS(I)
                  END IF
               END IF
   60       CONTINUE
            IF (TCURR.LT.TOUT) GO TO 40
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,1P,e13.5,A,1P,e13.5)
99998 FORMAT (1X,A,I2,A,I3,A,1P,e13.5)
      END
*
      SUBROUTINE FTRY02(NEQF,T,Y,YP)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NEQF
*     .. Array Arguments ..
      real              Y(NEQF), YP(NEQF)
*     .. Executable Statements ..
      YP(1) = Y(2)
      YP(2) = -Y(1)
      RETURN
      END
*
      real FUNCTION GTRY02(NEQF,T,Y,YP,K)
*     .. Scalar Arguments ..
      real                 T
      INTEGER              K, NEQF
*     .. Array Arguments ..
      real                 Y(NEQF), YP(NEQF)
*     .. Executable Statements ..
      IF (K.EQ.1) THEN
         GTRY02 = YP(1)
      ELSE
         GTRY02 = Y(1)
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02QFF Example Program Results

 Root at   0.00000E+00
 for event equation  2 with type  1 and residual   0.00000E+00
  Y(1) =   0.00000E+00   Y'(1) =   1.00000E+00

 Root at   1.57076E+00
 for event equation  1 with type  1 and residual  -5.90965E-16
  Y(1) =   1.00003E+00   Y'(1) =  -5.90965E-16
```

```
Root at    3.14151E+00
for event equation  2 with type  1 and residual  -1.24023E-16
 Y(1) =  -1.24023E-16   Y'(1) =  -1.00012E+00

Root at    4.71228E+00
for event equation  1 with type  1 and residual   3.61473E-16
 Y(1) =  -1.00010E+00   Y'(1) =   3.61473E-16

Root at    6.28306E+00
for event equation  2 with type  1 and residual   2.43942E-15
 Y(1) =   2.43942E-15   Y'(1) =   9.99979E-01

Root at    7.85379E+00
for event equation  1 with type  1 and residual  -2.49722E-16
 Y(1) =   9.99970E-01   Y'(1) =  -2.49722E-16

Root at    9.42469E+00
for event equation  2 with type  1 and residual  -2.72748E-15
 Y(1) =  -2.72748E-15   Y'(1) =  -9.99854E-01
```

D02QGF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02QGF is a reverse communication routine for integrating a non-stiff system of first order ordinary differential equations using a variable-order variable-step Adams method. A root-finding facility is provided.

2. Specification

```
      SUBROUTINE D02QGF (NEQF, T, Y, TOUT, NEQG, ROOT, IREVCM, TRVCM,
     1                   YRVCM, YPRVCM, GRVCM, KGRVCM, RWORK, LRWORK,
     2                   IWORK, LIWORK, IFAIL)
      INTEGER            NEQF, NEQG, IREVCM, YRVCM, YPRVCM, KGRVCM, LRWORK,
     1                   IWORK(LIWORK), LIWORK, IFAIL
      real               T, Y(NEQF), TOUT, TRVCM, GRVCM, RWORK(LRWORK)
      LOGICAL            ROOT
```

3. Description

Given the initial values $x,y_1,y_2,...,y_{\mathrm{NEQF}}$ the routine integrates a non-stiff system of first order differential equations of the type, $y_i' = f_i(x,y_1,y_2,...,y_{\mathrm{NEQF}})$, for $i = 1,2,...,\mathrm{NEQF}$, from $x = \mathrm{T}$ to $x = \mathrm{TOUT}$ using a variable-order variable-step Adams method. The user defines the system by reverse communication, evaluating f_i in terms of x and $y_1,y_2,...,y_{\mathrm{NEQF}}$, and $y_1,y_2,...,y_{\mathrm{NEQF}}$ are supplied at $x = \mathrm{T}$ by D02QGF. The routine is capable of finding roots (values of x) of prescribed event functions of the form

$$g_j(x,y,y') = 0, \qquad j = 1,2,...,\mathrm{NEQG}.$$

Each g_j is considered to be independent of the others so that roots are sought of each g_j individually. The root reported by the routine will be the first root encountered by any g_j. Two techniques for determining the presence of a root in an integration step are available: the sophisticated method described in Watts [3] and a simplified method whereby sign changes in each g_j are looked for at the ends of each integration step. The user also defines each g_j by reverse communication. In one-step mode the routine returns an approximation to the solution at each integration point. In interval mode this value is returned at the end of the integration range. If a root is detected this approximation is given at the root. The user selects the mode of operation, the error control, the root-finding technique and various optional inputs by a prior call of the setup routine D02QWF.

For a description of the practical implementation of an Adams formula see Shampine and Gordon [1].

4. References

[1] SHAMPINE, L.F. and GORDON, M.K.
Computer Solution of Ordinary Differential Equations – The Initial Value Problem.
W.H. Freeman & Co., San Francisco, 1975.

[2] SHAMPINE, L.F. and WATTS, H.A.
DEPAC – Design of a user oriented package of ODE solvers.
Sandia National Laboratory Report SAND79-2374, 1979.

[3] WATTS, H.A.
RDEAM – An Adams ODE Code with Root Solving Capability.
Sandia National Laboratory Report SAND85-1595, 1985.

5. Parameters

Note: this routine uses **reverse communication.** Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IREVCM.** Between intermediate exits and re-entries, **all parameters other than RWORK and GRVCM must remain unchanged.**

1: NEQF – INTEGER. *Input*

On initial entry: the number of first order ordinary differential equations to be solved by D02QGF. It must contain the same value as the parameter NEQF used in the prior call to D02QWF.

Constraint: NEQF $\geq$ 1.

2: T – ***real.*** *Input/Output*

On initial entry: that is after a call to D02QWF with STATEF = 'S', T must be set to the initial value of the independent variable x.

On final exit: the value of x at which y has been computed. This may be an intermediate output point, a root, TOUT or a point at which an error has occurred. If the integration is to be continued, possibly with a new value for TOUT, T must not be changed.

3: Y(NEQF) – ***real*** array. *Input/Output*

On initial entry: the initial values of the solution $y_1, y_2, \ldots, y_{\mathrm{NEQF}}$.

On final exit: the computed values of the solution at the exit value of T. If the integration is to be continued, possibly with a new value for TOUT, these values must not be changed.

4: TOUT – ***real.*** *Input*

On initial entry: the next value of x at which a computed solution is required. For the initial T, the input value of TOUT is used to determine the direction of integration. Integration is permitted in either direction. If TOUT = T on exit, TOUT must be reset beyond T **in the direction of integration**, before any continuation call.

5: NEQG – INTEGER. *Input*

On initial entry: the number of event functions which the user is defining for root-finding. If root-finding is not required the value for NEQG must be $\leq$ 0. Otherwise it must be the same value as the parameter NEQG used in the prior call to D02QWF.

6: ROOT – LOGICAL. *Output*

On final exit: if root-finding was required (NEQG > 0 on entry), then ROOT specifies whether or not the output value of the parameter T is a root of one of the event functions. If ROOT = .FALSE., then no root was detected, whereas ROOT = .TRUE. indicates a root and the user should make a call to D02QYF for further information.

If root-finding was not required (NEQG = 0 on entry), then ROOT = .FALSE..

7: IREVCM – INTEGER. *Input/Output*

On initial entry: IREVCM must have the value 0.

On intermediate exit: IREVCM specifies what action the user must take before re-entering D02QGF **with IREVCM unchanged**. The possible values of IREVCM on exit from D02QGF are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 which should be interpreted as follows:

IREVCM = 1, 2, 3, 4, 5, 6 or 7

indicates that the user must supply $y' = f(x,y)$, where x is given by TRVCM and y_i is returned in Y(i), for i = 1,2,...,NEQF when YRVCM = 0 and RWORK(YRVCM+i−1), for i = 1,2,...,NEQF when YRVCM $\neq$ 0. y_i' should be placed in location RWORK(YPRVCM+i−1), for i = 1,2,...,NEQF.

IREVCM = 8

indicates that the current step was not succesful due to error test failure. The only information supplied to the user on this return is the current value of the independent variable T, as given by TRVCM. No values must be changed before re-entering D02QGF. This facility enables the user to determine the number of unsuccessful steps.

IREVCM = 9, 10, 11, or 12

indicates that the user must supply $g_k(x,y,y')$, where k is given by KGRVCM, x is given by TRVCM, y_i is given by Y(i) and y'_i is given by RWORK(YPRVCM−1+i). The result g_k should be placed in the variable GRVCM.

On final exit: IREVCM has the value 0, which indicates that an output point or root has been reached or an error has occurred (see IFAIL).

8: TRVCM – ***real***. *Output*

On intermediate exit: the current value of the independent variable.

9: YRVCM – INTEGER. *Output*

On intermediate exit: with IREVCM = 1, 2, 3, 4, 5, 6, 7, 9, 10, 11 or 12, YRVCM specifies the locations of the dependent variables y for use in evaluating the differential system or the event functions. If YRVCM = 0 then y_i is given by Y(i), for i = 1,2,...,NEQF. If YRVCM ≠ 0 then y_i is given by RWORK(YRVCM+i−1), for i = 1,2,...,NEQF.

10: YPRVCM – INTEGER. *Output*

On intermediate exit: with IREVCM = 1, 2, 3, 4, 5, 6, or 7, YPRVCM specifies the positions in RWORK at which the user should place the derivatives y'. y'_i should be placed in location RWORK(YPRVCM+i−1), for i = 1,2,...,NEQF.

With IREVCM = 9, 10, 11 or 12, YPRVCM specifies the locations of the derivatives y' for use in evaluating the event functions. y'_i is given by RWORK(YPRVCM+i−1), for i = 1,2,...,NEQF. YPRVCM must not be changed before re-entering D02QGF.

11: GRVCM – ***real***. *Input*

On intermediate re-entry: with IREVCM = 9, 10, 11 or 12, GRVCM must contain the value of $g_k(x,y,y')$, where k is given by KGRVCM.

12: KGRVCM – INTEGER. *Output*

On intermediate exit: with IREVCM = 9, 10, 11 or 12, KGRVCM specifies which event function $g_k(x,y,y')$ the user must evaluate.

13: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK as supplied to D02QWF. It is used to pass information from D02QWF to D02QGF, and from D02QGF to the D02QXF, D02QYF and D02QZF. Therefore the contents of this array **must not** be changed before the call to D02QGF or calling any of the routines D02QXF, D02QYF and D02QZF.

14: LRWORK – INTEGER. *Input*

On initial entry: the dimension of the array RWORK as declared in the (sub)program from which D02QGF is called.

This must be the same parameter LRWORK as supplied to D02QWF.

15: IWORK(LIWORK) – INTEGER array. *Workspace*

This **must** be the same parameter IWORK as supplied to D02QWF. It is used to pass information from D02QWF to D02QGF, and from D02QGF to D02QXF, D02QYF and D02QZF. Therefore the contents of this array **must not** be changed before the call to D02QGF or calling any of the routines D02QXF, D02QYF and D02QZF.

16: LIWORK – INTEGER. *Input*

On initial entry: the dimension of the array IWORK as declared in the (sub)program from which D02QGF is called.

This must be the same parameter LIWORK as supplied to D02QWF.

17: IFAIL – INTEGER. *Input/Output*

On initial 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 final exit: IFAIL = 0 unless the routine detects an error or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

On entry, the integrator detected an illegal input, or D02QWF has not been called prior to the call to the integrator. If on entry IFAIL = 0 or –1, the form of the error will be detailed on the current error message unit (as defined by X04AAF).

This error may be caused by overwriting elements of RWORK and IWORK.

IFAIL = 2

The maximum number of steps has been attempted (at a cost of about 2 derivative evaluations per step). (See parameter MAXSTP in D02QWF.) If integration is to be continued then the user need only reset IFAIL and call the routine again and a further MAXSTP steps will be attempted.

IFAIL = 3

The step size needed to satisfy the error requirements is too small for the ***machine precision*** being used. (See parameter TOLFAC in D02QXF.)

IFAIL = 4

Some error weight w_i became zero during the integration (see parameters VECTOL, RTOL and ATOL in D02QWF.) Pure relative error control (ATOL = 0.0) was requested on a variable (the ith) which has now become zero. (See parameter BADCMP in D02QXF.) The integration was successful as far as T.

IFAIL = 5

The problem appears to be stiff (see the Chapter Introduction for a discussion of the term 'stiff'). Although it is inefficient to use this integrator to solve stiff problems, integration may be continued by resetting IFAIL and calling the routine again.

IFAIL = 6

A change in sign of an event function has been detected but the root-finding process appears to have converged to a singular point T rather than a root. Integration may be continued by resetting IFAIL and calling the routine again.

IFAIL = 7

The code has detected two successive error exits at the current value of T and cannot proceed. Check all input variables.

7. Accuracy

The accuracy of integration is determined by the parameters VECTOL, RTOL and ATOL in a prior call to D02QWF. Note that only the local error at each step is controlled by these parameters. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the property of the differential equation system. The code is designed so that a reduction in the tolerances should lead to an approximately proportional reduction in the error. The user is strongly recommended to call D02QGF with more than one set of tolerances and to compare the results obtained to estimate their accuracy.

The accuracy obtained depends on the type of error test used. If the solution oscillates around zero a relative error test should be avoided, whereas if the solution is exponentially increasing an absolute error test should not be used. If different accuracies are required for different components of the solution then a component-wise error test should be used. For a description of the error test see the specifications of the parameters VECTOL, ATOL and RTOL in the routine document for D02QWF.

The accuracy of any roots located will depend on the accuracy of integration and may also be restricted by the numerical properties of $g(x,y,y')$. When evaluating g the user should try to write the code so that unnecessary cancellation errors will be avoided.

8. Further Comments

If the routine fails with IFAIL = 3 then the combination of ATOL and RTOL may be so small that a solution cannot be obtained, in which case the routine should be called again with larger values for RTOL and/or ATOL. If the accuracy requested is really needed then the user should consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity the solution components will usually be of a large magnitude. The routine could be used in one-step mode to monitor the size of the solution with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary;

(b) for 'stiff' equations, where the solution contains rapidly decaying components, the routine will require a very small step size to preserve stability. This will usually be exhibited by excessive computing time and sometimes an error exit with IFAIL = 3, but usually an error exit with IFAIL = 2 or 5. The Adams methods are not efficient in such cases and the user should consider using a routine from the subchapter D02M-D02N. A high proportion of failed steps (see parameter NFAIL in D02QXF) may indicate stiffness but there may be other reasons for this phenomenon.

D02QGF can be used for producing results at short intervals (for example, for graph plotting); the user should set CRIT = .TRUE. and TCRIT to the last output point required in a prior call to D02QWF and then set TOUT appropriately for each output point in turn in the call to D02QGF.

9. Example

We solve the following system (for a projectile)

$$y' = \tan \phi$$

$$v' = \frac{-0.032\tan \phi}{v} - \frac{0.02v}{\cos \phi}$$

$$\phi' = \frac{-0.032}{v^2}$$

over an interval [0.0,10.0] starting with values $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$ using scalar error control (VECTOL = .FALSE.) until the first point where $y = 0.0$ is encountered.

Also, we use D02QGF to produce output at intervals of 2.0.

9.1. 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.

```
*     D02QGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQF, NEQG, LATOL, LRTOL, LRWORK, LIWORK
      PARAMETER        (NEQF=3,NEQG=1,LATOL=1,LRTOL=1,
     +                 LRWORK=23+23*NEQF+14*NEQG,LIWORK=21+4*NEQG)
      real             TSTART, HMAX
      PARAMETER        (TSTART=0.0e0,HMAX=2.0e0)
*     .. Local Scalars ..
      real             GRVCM, PI, T, TCRIT, TINC, TOUT, TRVCM
      INTEGER          I, IFAIL, IREVCM, J, KGRVCM, MAXSTP, YPRVCM,
     +                 YRVCM
      LOGICAL          ALTERG, CRIT, ONESTP, ROOT, SOPHST, VECTOL
      CHARACTER*1      STATEF
*     .. Local Arrays ..
      real             ATOL(LATOL), RTOL(LRTOL), RWORK(LRWORK), Y(NEQF)
      INTEGER          IWORK(LIWORK)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02QGF, D02QWF
*     .. Intrinsic Functions ..
      INTRINSIC        COS, real, TAN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02QGF Example Program Results'
      TCRIT = 10.0e0
      STATEF = 'S'
      VECTOL = .FALSE.
      RTOL(1) = 1.0e-4
      ATOL(1) = 1.0e-7
      ONESTP = .FALSE.
      SOPHST = .TRUE.
      CRIT = .TRUE.
      TINC = 2.0e0
      MAXSTP = 500
      PI = X01AAF(0.0e0)
      T = TSTART
      Y(1) = 0.5e0
      Y(2) = 0.5e0
      Y(3) = 0.2e0*PI
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  T         Y(1)      Y(2)      Y(3)'
      WRITE (NOUT,99999) T, (Y(I),I=1,NEQF)
      IFAIL = 0
*
```

```
      CALL D02QWF(STATEF,NEQF,VECTOL,ATOL,LATOL,RTOL,LRTOL,ONESTP,CRIT,
     +            TCRIT,HMAX,MAXSTP,NEQG,ALTERG,SOPHST,RWORK,LRWORK,
     +            IWORK,LIWORK,IFAIL)
*
      J = 1
      TOUT = real(J)*TINC
      IREVCM = 0
*
   20 IFAIL = -1
*
      CALL D02QGF(NEQF,T,Y,TOUT,NEQG,ROOT,IREVCM,TRVCM,YRVCM,YPRVCM,
     +            GRVCM,KGRVCM,RWORK,LRWORK,IWORK,LIWORK,IFAIL)
*
      IF (IREVCM.GT.0) THEN
         IF (IREVCM.LT.8) THEN
            IF (YRVCM.EQ.0) THEN
               RWORK(YPRVCM) = TAN(Y(3))
               RWORK(YPRVCM+1) = -0.032e0*TAN(Y(3))/Y(2) - 0.02e0*Y(2)
     +                           /COS(Y(3))
               RWORK(YPRVCM+2) = -0.032e0/Y(2)**2
            ELSE
               RWORK(YPRVCM) = TAN(RWORK(YRVCM+2))
               RWORK(YPRVCM+1) = -0.032e0*TAN(RWORK(YRVCM+2))
     +                           /RWORK(YRVCM+1) - 0.02e0*RWORK(YRVCM+1)
     +                           /COS(RWORK(YRVCM+2))
               RWORK(YPRVCM+2) = -0.032e0/RWORK(YRVCM+1)**2
            END IF
         ELSE IF (IREVCM.GT.8) THEN
            GRVCM = Y(1)
         END IF
         GO TO 20
      ELSE IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,99999) T, (Y(I),I=1,NEQF)
         IF (T.EQ.TOUT .AND. J.LT.5) THEN
            J = J + 1
            TOUT = real(J)*TINC
            GO TO 20
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,F6.4,3X,3(F7.4,2X))
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02QGF Example Program Results

    T         Y(1)      Y(2)      Y(3)
 0.0000     0.5000    0.5000    0.6283
 2.0000     1.5493    0.4055    0.3066
 4.0000     1.7423    0.3743   -0.1289
 6.0000     1.0055    0.4173   -0.5507
 7.2883     0.0000    0.4749   -0.7601
```

D02QWF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02QWF is a setup routine which must be called by the user prior to the first call of either of the integration routines D02QFF and D02QGF, and may be called prior to any subsequent continuation call to these routines.

2. Specification

```
      SUBROUTINE D02QWF (STATEF, NEQF, VECTOL, ATOL, LATOL, RTOL, LRTOL,
     1                   ONESTP, CRIT, TCRIT, HMAX, MAXSTP, NEQG, ALTERG,
     2                   SOPHST, RWORK, LRWORK, IWORK, LIWORK, IFAIL)
      INTEGER          NEQF, LATOL, LRTOL, MAXSTP, NEQG, LRWORK,
     1                 IWORK(LIWORK), LIWORK, IFAIL
      real             ATOL(LATOL), RTOL(LRTOL), TCRIT, HMAX,
     1                 RWORK(LRWORK)
      LOGICAL          VECTOL, ONESTP, CRIT, ALTERG, SOPHST
      CHARACTER*1      STATEF
```

3. Description

This routine permits initialisation of the integration method and setting of optional inputs prior to any call of D02QFF or D02QGF.

It must be called before the first call of either of the routines D02QFF or D02QGF and it may be called before any continuation call of either of the routines D02QFF or D02QGF.

4. References

None.

5. Parameters

1: STATEF – CHARACTER*1. *Input/Output*

On entry: specifies whether that the integration routine (D02QFF or D02QGF) is to start a new system of ordinary differential equations, restart a system or continue with a system. STATEF is interpreted as follows:

STATEF = 'S' start integration with a new differential system;
= 'R' restart integration with the current differential system;
= 'C' continue integration with the current differential system.

Constraint: STATEF = 'S', 's', 'R', 'r', 'C' or 'c'.

On exit: STATEF is set to 'C', except that if an error is detected, STATEF is unchanged.

2: NEQF – INTEGER. *Input*

On entry: the number of ordinary differential equations to be solved by the integration routine. NEQF must remain unchanged on subsequent calls to D02QWF with STATEF = 'C' or 'R'.

Constraint: NEQF $\geq$ 1.

3: VECTOL – LOGICAL. *Input*

On entry: specifies whether vector or scalar error control is to be employed for the local error test in the integration.

If VECTOL = .TRUE., then vector error control will be used and the user must specify values of RTOL(i) and ATOL(i), for i = 1,2,...,NEQF.

Otherwise scalar error control will be used and the user must specify values of just RTOL(1) and ATOL(1).

The error test to be satisfied is of the form

$$\sqrt{\sum_{i=1}^{\mathrm{NEQF}} \left(\frac{e_i}{w_i}\right)^2} \leq 1.0,$$

where w_i is defined as follows:

VECTOL	w_i
.TRUE.	RTOL(i) $\times\lvert y_i\rvert$ + ATOL(i)
.FALSE.	RTOL(1)$\times\lvert y_i\rvert$ + ATOL(1)

and e_i is an estimate of the local error in y_i, computed internally. VECTOL must remain unchanged on subsequent calls to D02QWF with STATEF = 'C' or 'R'.

4: ATOL(LATOL) – ***real*** array. *Input*

On entry: the absolute local error tolerance (see VECTOL).

Constraint: ATOL(i) $\geq$ 0.0.

5: LATOL – INTEGER. *Input*

On entry: the dimension of the array ATOL as declared in the (sub)program from which D02QWF is called.

Constraints: LATOL $\geq$ NEQF if VECTOL = .TRUE.,
LATOL $\geq$ 1 if VECTOL = .FALSE..

6: RTOL(LRTOL) – ***real*** array. *Input*

On entry: the relative local error tolerance (see VECTOL).

Constraints: RTOL(i) $\geq$ 0.0,
RTOL(i) $\geq$ 4.0×***machine precision*** if ATOL(i) = 0.0.

7: LRTOL – INTEGER. *Input*

On entry: the dimension of the array RTOL as declared in the (sub)program from which D02QWF is called.

Constraints: LRTOL $\geq$ NEQF if VECTOL = .TRUE.,
LRTOL $\geq$ 1 if VECTOL = .FALSE..

8: ONESTP – LOGICAL. *Input*

On entry: the mode of operation of the integration routine. If ONESTP = .TRUE., the integration routine will operate in one-step mode, that is it will return after each successful step. Otherwise the integration routine will operate in interval mode, that is it will return at the end of the integration interval.

9: CRIT – LOGICAL. *Input*

On entry: specifies whether or not there is a value for the independent variable beyond which integration is not to be attempted. Setting CRIT = .TRUE. indicates that there is such a point, whereas CRIT = .FALSE. indicates that there is no such restriction.

10: TCRIT – ***real***. *Input*

On entry: with CRIT = .TRUE., TCRIT must be set to a value of the independent variable beyond which integration is not to be attempted. Otherwise TCRIT is not referenced.

11: HMAX – ***real***. *Input*

On entry: if HMAX ≠ 0.0 then a bound on the absolute step size during the integration is taken to be |HMAX|. If HMAX = 0.0 on entry, then no bound is assumed on the step size during the integration.

A bound may be required if there are features of the solution on very short ranges of integration which may be missed. The user should try HMAX = 0.0 first.

Note: this parameter only affects the step size if the option CRIT = .TRUE. is being used.

12: MAXSTP – INTEGER. *Input*

On entry: a bound on the number of attempted steps in any one call to the integration routine. If MAXSTP ≤ 0 on entry, a value of 1000 is used.

13: NEQG – INTEGER. *Input*

On entry: specifies whether or not root-finding is required in D02QFF or D02QGF. If NEQG ≤ 0 then **no** root-finding is attempted. If NEQG > 0 then root-finding is required and NEQG event functions will be specified for the integration routine.

14: ALTERG – LOGICAL. *Input/Output*

On entry: specifies whether or not the event functions have been redefined. ALTERG need not be set if STATEF = 'S'. On subsequent calls to D02QWF, if NEQG has been set positive, then ALTERG = .FALSE. specifies that the event functions remain unchanged, whereas ALTERG = .TRUE. specifies that the event functions have changed. Because of the expense in reinitialising the root searching procedure, ALTERG should be set to .TRUE. only if the event functions really have been altered. ALTERG need not be set if the root-finding option is not used.

On exit: ALTERG is set to .FALSE..

15: SOPHST – LOGICAL. *Input*

On entry: the type of search technique to be used in the root-finding. If SOPHST = .TRUE. then a sophisticated and reliable but expensive technique will be used, whereas for SOPHST = .FALSE. a simple but less reliable technique will be used. If NEQG ≤ 0 then SOPHST is not referenced.

16: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK supplied to the integration routine. It is used to pass information to the integration routine and therefore the contents of this array **must not** be changed before calling the integration routine.

17: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02QWF is called.

Constraint: LRWORK ≥ 21×(1+NEQF) + 2×J + K×NEQG + 2, where

$$J = \begin{cases} \text{NEQF} & \text{if VECTOL = .TRUE.} \\ 1 & \text{if VECTOL = .FALSE.} \end{cases}$$

and

$$K = \begin{cases} 14 & \text{if SOPHST = .TRUE.} \\ 5 & \text{if SOPHST = .FALSE.} \end{cases}$$

18: IWORK(LIWORK) – INTEGER array. *Workspace*

This **must** be the same parameter IWORK supplied to the integration routine. It is used to pass information to the integration routine and therefore the contents of this array **must not** be changed before calling the integration routine.

19: LIWORK – INTEGER. *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02QWF is called.

Constraints: LIWORK ≥ 21 + 4×NEQG if SOPHST = .TRUE.,
LIWORK ≥ 21 + NEQG if SOPHST = .FALSE..

20: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

Illegal input detected.

7. Accuracy

Not applicable.

8. Further Comments

Prior to a continuation call of the integration routine, the user may reset any of the optional parameters by calling D02QWF with STATEF = 'C'. The user may reset:

(a) HMAX – to alter the maximum step size selection;
(b) RTOL,ATOL – to change the error requirements;
(c) MAXSTP – to increase or decrease the number of attempted steps before an error exit is returned;
(d) ONESTP – to change the operation mode of the integration routine;
(e) CRIT,TCRIT – to alter the point beyond which integration must not be attempted; and
(f) NEQG,ALTERG,SOPHST – to alter the number and type of event functions, and also the search method.

If the behaviour of the system of differential equations has altered and the user wishes to restart the integration method from the value of T output from the integration routine, then STATEF should be set to 'R' and any of the optional parameters may be reset also. If the user wants to redefine the system of differential equations or start a new integration problem, then STATEF should be set to 'S'. Resetting STATEF to 'R' or 'S' on normal continuation calls causes a restart in the integration process, which is very inefficient when not needed.

9. Example

See example programs for D02QFF and D02QGF.

D02QXF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02QXF is a diagnostic routine which may be called after a call to either of the integration routines D02QFF and D02QGF.

2. Specification

```
      SUBROUTINE D02QXF (NEQF, YP, TCURR, HLAST, HNEXT, ODLAST, ODNEXT,
     1                   NSUCC, NFAIL, TOLFAC, BADCMP, RWORK, LRWORK,
     2                   IWORK, LIWORK, IFAIL)
      INTEGER            NEQF, ODLAST, ODNEXT, NSUCC, NFAIL, BADCMP,
     1                   LRWORK, IWORK(LIWORK), LIWORK, IFAIL
      real               YP(NEQF), TCURR, HLAST, HNEXT, TOLFAC,
     1                   RWORK(LRWORK)
```

3. Description

This routine permits the user to extract information about the performance of one of D02QFF or D02QGF. It may only be called after a call to D02QFF or D02QGF.

4. References

None.

5. Parameters

1: NEQF – INTEGER. *Input*

On entry: the number of first order ordinary differential equations solved by the integration routine. It must be the same parameter NEQF supplied to the setup routine D02QWF and the integration routines D02QFF or D02QGF.

2: YP(NEQF) – ***real*** array. *Output*

On exit: the approximate derivative of the solution component y_i, as supplied in y_i on output from the integration routine at the output value of T. These values are obtained by the evaluation of $y' = f(x,y)$ except when the output value of the parameter T in the call to the integration routine is TOUT and TCURR $\neq$ TOUT, in which case they are obtained by interpolation.

3: TCURR – ***real***. *Output*

On exit: the value of the independent variable which the integrator has actually reached. TCURR will always be at least as far as the output value of the argument T (from the integration routine) in the direction of integration, but may be further.

4: HLAST – ***real***. *Output*

On exit: the last successful step size used by the integrator.

5: HNEXT – ***real***. *Output*

On exit: the next step size which the integration routine would attempt.

6: ODLAST – INTEGER. *Output*

On exit: the order of the method last used (successfully) by the integration routine.

7: ODNEXT – INTEGER. *Output*

On exit: the order of the method which the integration routine would attempt on the next step.

8: NSUCC – INTEGER. *Output*

On exit: the number of steps attempted by the integration routine that have been successful since the start of the current problem.

9: NFAIL – INTEGER. *Output*

On exit: the number of steps attempted by the integration routine that have failed since the start of the current problem.

10: TOLFAC – ***real***. *Output*

On exit: a tolerance scale factor, TOLFAC ≥ 1.0, returned when the integration routine exits with IFAIL = 3. If RTOL and ATOL are uniformly scaled up by a factor of TOLFAC and D02QWF is called, the next call to the integration routine is deemed likely to succeed.

11: BADCMP – INTEGER. *Output*

On exit: if the integration routine returned with IFAIL = 4, then BADCMP specifies the index of the component which forced the error exit. Otherwise BADCMP is 0.

12: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK as supplied to D02QFF or D02QGF. It is used to pass information from the integration routine to D02QXF and therefore the contents of this array **must not** be changed before calling D02QXF.

13: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02QXF is called.

This must be the same parameter LRWORK as supplied to D02QWF.

14: IWORK(LIWORK) – INTEGER array. *Workspace*

This **must** be the same parameter IWORK as supplied to D02QFF or D02QGF. It is used to pass information from the integration routine to D02QXF and therefore the contents of this array **must not** be changed before calling D02QXF.

15: LIWORK – INTEGER. *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02QXF is called.

This must be the same parameter LIWORK as supplied to D02QWF.

16: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

An integration routine (D02QFF or D02QGF) has not been called or one or more of the parameters LRWORK, LIWORK and NEQF does not match the corresponding parameter supplied to D02QWF.

This error exit may be caused by overwriting elements of RWORK.

7. Accuracy

Not applicable.

8. Further Comments

The user should call D02QYF for information about any roots detected by D02QFF or D02QGF.

9. Example

See example program for D02QFF.

D02QYF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02QYF is a diagnostic routine which may be called after a call to the integrator routines D02QFF or D02QGF.

2. Specification

```
      SUBROUTINE D02QYF (NEQG, INDEX, TYPE, EVENTS, RESIDS, RWORK, LRWORK,
     1                   IWORK, LIWORK, IFAIL)
      INTEGER          NEQG, INDEX, TYPE, EVENTS(NEQG), LRWORK,
     1                 IWORK(LIWORK), LIWORK, IFAIL
      real             RESIDS(NEQG), RWORK(LRWORK)
```

3. Description

This routine should be called only after a call to one of routines D02QFF and D02QGF results in the output value ROOT = .TRUE., indicating that a root has been detected. D02QYF permits the user to examine information about the root detected, such as the indices of the event equations for which there is a root, the type of root (odd or even) and the residuals of the event equations.

4. References

None.

5. Parameters

1: NEQG – INTEGER. *Input*

On entry: the number of event functions defined for the integration routine. It must be the same parameter NEQG supplied to the setup routine D02QWF and to the integration routine (D02QFF or D02QGF).

2: INDEX – INTEGER. *Output*

On exit: the index k of the event equation $g_k(x,y,y') = 0$ for which the root has been detected.

3: TYPE – INTEGER. *Output*

On exit: information about the root detected for the event equation defined by INDEX. The possible values of TYPE with their interpretations are as follows:

TYPE = 1

a simple root, or lack of distinguishing information available;

TYPE = 2

a root of even multiplicity is believed to have been detected, that is no change in sign of the event function was found;

TYPE = 3

a high order root of odd multiplicity;

TYPE = 4

a possible root, but due to high multiplicity or a clustering of roots accurate evaluation of the event function was prohibited by roundoff error and/or cancellation.

In general, the accuracy of the root is less reliable for values of TYPE > 1.

4: EVENTS(NEQG) – INTEGER array. *Output*

On exit: information about the kth event function on a very small interval containing the root, T, as output from the integration routine. All roots lying in this interval are considered indistinguishable numerically and therefore should be regarded as defining a root at T. The possible values of EVENTS(k) with their interpretations are as follows:

EVENTS(k) = 0

the kth event function did not have a root;

EVENTS(k) = −1

the kth event function changed sign from positive to negative about a root, in the direction of integration;

EVENTS(k) = 1

the kth event function changed sign from negative to positive about a root, in the direction of integration;

EVENTS(k) = 2

a root was identified, but no change in sign was observed.

5: RESIDS(NEQG) – ***real*** array. *Output*

On exit: the value of the kth event function computed at the root, T.

6: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK as supplied to D02QFF or D02QGF. It is used to pass information from the integration routine to D02QYF and therefore the contents of this array **must not** be changed before calling D02QYF.

7: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02QYF is called.

This must be the same parameter LRWORK as supplied to D02QWF.

8: IWORK(LIWORK) – INTEGER array. *Workspace*

This **must** be the same parameter IWORK as supplied to D02QFF or D02QGF. It is used to pass information from the integration routine to D02QYF and therefore the contents of this array **must not** be changed before calling D02QYF.

9: LIWORK – INTEGER. *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02QYF is called.

This must be the same parameter LIWORK as supplied to D02QWF.

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

An integration routine (D02QFF or D02QGF) has not been called, no root was detected or one or more of the parameters LRWORK, LIWORK and NEQG does not match the corresponding values supplied to D02QWF. Values for the arguments INDEX, TYPE, EVENTS and RESIDS will not have been set.

This error exit may be caused by overwriting elements of IWORK.

7. Accuracy

Not applicable.

8. Further Comments

None.

9. Example

See example program for D02QFF.

D02QZF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02QZF interpolates components of the solution of a non-stiff system of first order differential equations from information provided by the integrator routines D02QFF or D02QGF.

2. Specification

```
      SUBROUTINE D02QZF (NEQF, TWANT, NWANT, YWANT, YPWANT, RWORK, LRWORK,
     1                   IWORK, LIWORK, IFAIL)
      INTEGER          NEQF, NWANT, LRWORK, IWORK(LIWORK), LIWORK, IFAIL
      real             TWANT, YWANT(NWANT), YPWANT(NWANT), RWORK(LRWORK)
```

3. Description

D02QZF evaluates the first NWANT components of the solution of a non-stiff system of first order ordinary differential equations at any point using the method of Watts and Shampine [1] and information generated by D02QFF or D02QGF. D02QZF should not normally be used to extrapolate outside the current range of the values produced by the integration routine.

4. References

[1] WATTS, H.A. and SHAMPINE, L.F.
Smoother Interpolants for Adams Codes.
SIAM J. Sci. Stat. Comput., 7, 334-345, 1986.

5. Parameters

1: NEQF – INTEGER. *Input*

On entry: the number of first order ordinary differential equations being solved by the integration routine. It must contain the same value as the parameter NEQF in a prior call to the setup routine D02QWF.

2: TWANT – ***real***. *Input*

On entry: the point at which components of the solution and derivative are to be evaluated. TWANT should not normally be an extrapolation point, that is TWANT should satisfy

$$\text{TOLD} \le \text{TWANT} \le \text{T},$$

or if integration is proceeding in the negative direction

$$\text{TOLD} \ge \text{TWANT} \ge \text{T},$$

where TOLD is the previous integration point and is, to within rounding, TCURR − HLAST (see D02QXF). Extrapolation is permitted but not recommended and an IFAIL value of 2 is returned whenever extrapolation is attempted.

3: NWANT – INTEGER. *Input*

On entry: the number of components of the solution and derivative whose values at TWANT are required. The first NWANT components are evaluated.

Constraint: $1 \le \text{NWANT} \le \text{NEQF}$.

4: YWANT(NWANT) – ***real*** array. *Output*

On exit: the calculated value of the ith component of the solution at TWANT, for $i = 1,2,...,\text{NWANT}$.

5: YPWANT(NWANT) – ***real*** array. *Output*

On exit: the calculated value of the ith component of the derivative at TWANT, for i = 1,2,...,NWANT.

6: RWORK(LRWORK) – ***real*** array. *Workspace*

This **must** be the same parameter RWORK as supplied to D02QWF and to D02QFF or D02QGF. It is used to pass information from these routines to D02QZF. Therefore its contents **must not** be changed prior to a call to D02QZF.

7: LRWORK – INTEGER. *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02QZF is called.

This must be the same parameter LRWORK as supplied to D02QWF.

8: IWORK(LIWORK) – INTEGER array. *Workspace*

This **must** be the same parameter IWORK as supplied to D02QWF and to D02QFF or D02QGF. It is used to pass information from these routines to D02QZF. Therefore its contents **must not** be changed prior to a call to D02QZF.

9: LIWORK – INTEGER. *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02QZF is called.

This must be the same parameter LIWORK as supplied to D02QWF.

10: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

An integration routine (D02QFF or D02QGF) has not been called, no integration steps have been taken since the last call to D02QWF with STATEF = 'S', one or more of the parameters LRWORK, LIWORK and NEQF does not match the same parameter supplied to D02QWF, or NWANT does not satisfy $1 \leq$ NWANT $\leq$ NEQF.

IFAIL = 2

D02QZF has been called for extrapolation. The values of the solution and its derivative at TWANT have been calculated and placed in YWANT and YPWANT before returning with this warning (see Section 7).

These error exits may be caused by overwriting elements of RWORK and IWORK.

7. Accuracy

The error in interpolation is of a similar order to the error arising from the integration. The same order of accuracy can be expected when extrapolating using D02QZF. However, the actual error in extrapolation will, in general, be much larger than for interpolation.

8. Further Comments

When interpolation for only a few components is required then it is more efficient to order the components of interest so that they are numbered first.

9. Example

We solve the equation

$$y'' = -y, \qquad y(0) = 0,\ y'(0) = 1$$

reposed as

$$y_1' = y_2$$
$$y_2' = -y_1$$

over the range $[0, \pi/2]$ with initial conditions $y_1 = 0$ and $y_2 = 1$ using vector error control (VECTOL = .TRUE.) and D02QFF in one-step mode (ONESTP = .TRUE.). D02QZF is used to provide solution values at intervals of $\pi/16$.

9.1. 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.

```
*     D02QZF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQF, NEQG, LATOL, LRTOL, LRWORK, LIWORK
      PARAMETER        (NEQF=2,NEQG=0,LATOL=NEQF,LRTOL=NEQF,
     +                 LRWORK=23+23*NEQF+14*NEQG,LIWORK=21+4*NEQG)
      real             TSTART, HMAX
      PARAMETER        (TSTART=0.0e0,HMAX=2.0e0)
*     .. Local Scalars ..
      real             PI, T, TCRIT, TINC, TOUT, TWANT
      INTEGER          I, IFAIL, J, MAXSTP, NWANT
      LOGICAL          ALTERG, CRIT, ONESTP, ROOT, SOPHST, VECTOL
      CHARACTER*1      STATEF
*     .. Local Arrays ..
      real             ATOL(LATOL), RTOL(LRTOL), RWORK(LRWORK), Y(NEQF),
     +                 YPWANT(NEQF), YWANT(NEQF)
      INTEGER          IWORK(LIWORK)
*     .. External Functions ..
      real             D02QFZ, X01AAF
      EXTERNAL         D02QFZ, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D02QFF, D02QWF, D02QZF, FTRY03
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02QZF Example Program Results'
      PI = X01AAF(0.0e0)
      STATEF = 'S'
      VECTOL = .TRUE.
      DO 20 I = 1, NEQF
         ATOL(I) = 1.0e-8
         RTOL(I) = 1.0e-4
   20 CONTINUE
      ONESTP = .TRUE.
      CRIT = .TRUE.
      TINC = 0.0625e0*PI
      TCRIT = 8.0e0*TINC
      TOUT = TCRIT
      MAXSTP = 500
      T = TSTART
      TWANT = TSTART + TINC
      NWANT = NEQF
      Y(1) = 0.0e0
      Y(2) = 1.0e0
      WRITE (NOUT,*)
      WRITE (NOUT,*) '  T          Y(1)       Y(2)'
      WRITE (NOUT,99999) T, Y(1), Y(2)
      IFAIL = -1
```

```
*
      CALL D02QWF(STATEF,NEQF,VECTOL,ATOL,LATOL,RTOL,LRTOL,ONESTP,CRIT,
     +            TCRIT,HMAX,MAXSTP,NEQG,ALTERG,SOPHST,RWORK,LRWORK,
     +            IWORK,LIWORK,IFAIL)
*
      J = 1
   40 IFAIL = -1
*
      CALL D02QFF(FTRY03,NEQF,T,Y,TOUT,D02QFZ,NEQG,ROOT,RWORK,LRWORK,
     +            IWORK,LIWORK,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
   60    IF (TWANT.LE.T) THEN
            IFAIL = 0
*
            CALL D02QZF(NEQF,TWANT,NWANT,YWANT,YPWANT,RWORK,LRWORK,
     +                  IWORK,LIWORK,IFAIL)
*
            WRITE (NOUT,99999) TWANT, YWANT(1), YWANT(2)
            J = J + 1
            TWANT = TSTART + real(J)*TINC
            GO TO 60
         END IF
         IF (T.LT.TOUT) GO TO 40
      END IF
      STOP
*
99999 FORMAT (1X,F6.4,3X,2(F7.4,2X))
      END
*
      SUBROUTINE FTRY03(NEQF,T,Y,YP)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NEQF
*     .. Array Arguments ..
      real              Y(NEQF), YP(NEQF)
*     .. Executable Statements ..
      YP(1) = Y(2)
      YP(2) = -Y(1)
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02QZF Example Program Results

   T         Y(1)      Y(2)
0.0000     0.0000    1.0000
0.1963     0.1951    0.9808
0.3927     0.3827    0.9239
0.5890     0.5556    0.8315
0.7854     0.7071    0.7071
0.9817     0.8315    0.5556
1.1781     0.9239    0.3827
1.3744     0.9808    0.1951
1.5708     1.0000    0.0000
```

D02RAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02RAF solves the two-point boundary-value problem with general boundary conditions for a system of ordinary differential equations, using a deferred correction technique and Newton iteration.

2. Specification

```
      SUBROUTINE D02RAF (N, MNP, NP, NUMBEG, NUMMIX, TOL, INIT, X, Y, IY,
     1                   ABT, FCN, G, IJAC, JACOBF, JACOBG, DELEPS,
     2                   JACEPS, JACGEP, WORK, LWORK, IWORK, LIWORK,
     3                   IFAIL)
      INTEGER            N, MNP, NP, NUMBEG, NUMMIX, INIT, IY, IJAC, LWORK,
     1                   IWORK(LIWORK), LIWORK, IFAIL
      real               TOL, X(MNP), Y(IY,MNP), ABT(N), DELEPS,
     1                   WORK(LWORK)
      EXTERNAL           FCN, G, JACOBF, JACOBG, JACEPS, JACGEP
```

3. Description

D02RAF solves a two-point boundary-value problem for a system of n ordinary differential equations in the interval (a,b) with $b > a$. The system is written in the form

$$y_i' = f_i(x,y_1,y_2,\ldots,y_n), \qquad i = 1,2,\ldots,n \tag{1}$$

and the derivatives f_i are evaluated by a subroutine FCN supplied by the user. With the differential equations (1) must be given a system of n (nonlinear) boundary conditions

$$g_i(y(a),y(b)) = 0, \qquad i = 1,2,\ldots,n$$

where

$$y(x) = [y_1(x),y_2(x),\ldots,y_n(x)]^T. \tag{2}$$

The functions g_i are evaluated by a subroutine G supplied by the user. The solution is computed using a finite-difference technique with deferred correction allied to a Newton iteration to solve the finite-difference equations. The technique used is described fully in Pereyra [1].

The user must supply an absolute error tolerance and may also supply an initial mesh for the finite-difference equations and an initial approximate solution (alternatively a default mesh and approximation are used). The approximate solution is corrected using Newton iteration and deferred correction. Then, additional points are added to the mesh and the solution is recomputed with the aim of making the error everywhere less than the user's tolerance and of approximately equidistributing the error on the final mesh. The solution is returned on this final mesh.

If the solution is required at a few specific points then these should be included in the initial mesh. If, on the other hand, the solution is required at several specific points then the user should use the interpolation routines provided in the E01 Chapter if these points do not themselves form a convenient mesh.

The Newton iteration requires Jacobian matrices

$$\left(\frac{\partial f_i}{\partial y_j}\right), \left(\frac{\partial g_i}{\partial y_j(a)}\right) \text{ and } \left(\frac{\partial g_i}{\partial y_j(b)}\right).$$

These may be supplied by the user through subroutines JACOBF for $\left(\frac{\partial f_i}{\partial y_j}\right)$ and JACOBG for the others. Alternatively the Jacobians may be calculated by numerical differentiation using the algorithm described in Curtis *et al.* [2].

For problems of the type (1) and (2) for which it is difficult to determine an initial approximation from which the Newton iteration will converge, a continuation facility is provided. The user must set up a family of problems

$$y' = f(x,y,\varepsilon), \qquad g(y(a),y(b),\varepsilon) = 0 \tag{3}$$

where $f = [f_1,f_2,\ldots,f_n]^T$ etc., and where ε is a continuation parameter. The choice $\varepsilon = 0$ must give a problem (3) which is easy to solve and $\varepsilon = 1$ must define the problem whose solution is actually required. The routine solves a sequence of problems with ε values

$$0 = \varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_p = 1 \tag{4}$$

The number p and the values ε_i are chosen by the routine so that each problem can be solved using the solution of its predecessor as a starting approximation. Jacobians $\frac{\partial f}{\partial \varepsilon}$ and $\frac{\partial g}{\partial \varepsilon}$ are required and they may be supplied by the user via routines JACEPS and JACGEP respectively or may be computed by numerical differentiation.

4. References

[1] PEREYRA, V.
PASVA3: An Adaptive Finite-Difference Fortran Program for First Order Nonlinear, Ordinary Boundary Problems.
In: 'Codes for Boundary Value Problems in Ordinary Differential Equations',
B. Childs, M. Scott, J.W. Daniel, E. Denman and P. Nelson. (eds.)
Springer-Verlag, Lecture Notes in Computer Science, 76, 1979.

[2] CURTIS, A.R., POWELL, M.J.D. and REID, J.K.
On the Estimation of Sparse Jacobian Matrices.
J. Inst. Maths. Applics, 13, pp. 117-119, 1974.

5. Parameters

1: N – INTEGER. *Input*

On entry: the number of differential equations, n.

Constraint: N > 0.

2: MNP – INTEGER. *Input*

On entry: MNP must be set to the maximum permitted number of points in the finite-difference mesh. If LWORK or LIWORK (see below) is too small then internally MNP will be replaced by the maximum permitted by these values. (A warning message will be output if on entry IFAIL is set to obtain monitoring information.)

Constraint: MNP ≥ 32.

3: NP – INTEGER. *Input/Output*

On entry: NP must be set to the number of points to be used in the initial mesh.

Constraint: 4 ≤ NP ≤ MNP.

On exit: the number of points in the final mesh.

4: NUMBEG – INTEGER. *Input*

On entry: the number of left-hand boundary conditions (that is the number involving $y(a)$ only).

Constraint: 0 ≤ NUMBEG < N.

5: NUMMIX – INTEGER. *Input*

On entry: the number of coupled boundary conditions (that is the number involving both $y(a)$ and $y(b)$).

Constraint: 0 ≤ NUMMIX ≤ N – NUMBEG.

6: TOL – ***real.*** *Input*

On entry: a positive absolute error tolerance. If

$$a = x_1 < x_2 < \ldots < x_{\mathrm{NP}} = b$$

is the final mesh, $z_j(x_i)$ is the jth component of the approximate solution at x_i, and $y_j(x)$ is the jth component of the true solution of (1) and (2), then, except in extreme circumstances, it is expected that

$$|z_j(x_i) - y_j(x_i)| \le \mathrm{TOL}, \qquad i = 1,2,\ldots,\mathrm{NP};\ j = 1,2,\ldots,n. \tag{5}$$

Constraint: TOL > 0.0.

7: INIT – INTEGER. *Input*

On entry: indicates whether the user wishes to supply an initial mesh and approximate solution (INIT ≠ 0) or whether default values are to be used, (INIT = 0).

8: X(MNP) – ***real*** array. *Input/Output*

On entry: the user must set X(1) = a and X(NP) = b. If INIT = 0 on entry a default equispaced mesh will be used, otherwise the user must specify a mesh by setting X(i) = x_i, for i = 2,3,...NP–1.

Constraints: X(1) < X(NP), if INIT = 0,
X(1) < X(2) < ... < X(NP), if INIT ≠ 0.

On exit: X(1),X(2),...,X(NP) define the final mesh (with the returned value of NP) and X(1) = a and X(NP) = b.

9: Y(IY,MNP) – ***real*** array. *Input/Output*

On entry: if INIT = 0, then Y need not be set.

If INIT ≠ 0, then the array Y must contain an initial approximation to the solution such that Y(j,i) contains an approximation to

$$y_j(x_i), \qquad i = 1,2,\ldots,\mathrm{NP};\ j = 1,2,\ldots,n.$$

On exit: the approximate solution $z_j(x_i)$ satisfying (5) on the final mesh, that is

$$\mathrm{Y}(j,i) = z_j(x_i), \qquad i = 1,2,\ldots,\mathrm{NP};\ j = 1,2,\ldots,n,$$

where NP is the number of points in the final mesh. If an error has occurred then Y contains the latest approximation to the solution. The remaining columns of Y are not used.

10: IY – INTEGER. *Input*

On entry: the first dimension of the array Y as declared in the (sub)program from which D02RAF is called.

Constraint: IY ≥ N.

11: ABT(N) – ***real*** array. *Output*

On exit: ABT(i), for i = 1,2,...,n, holds the largest estimated error (in magnitude) of the ith component of the solution over all mesh points.

12: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (i.e. the derivatives y_i') at a general point x for a given value of ϵ, the continuation parameter (see Section 3).

Its specification is:

```
SUBROUTINE FCN(X, EPS, Y, F, N)
INTEGER    N
real       X, EPS, Y(N), F(N)
```

1: X – ***real***. *Input*

On entry: the value of the argument x.

2: EPS – ***real***. *Input*

On entry: the value of the continuation parameter, ε. This is 1 if continuation is not being used.

3: Y(N) – ***real*** array. *Input*

On entry: the value of the argument y_i, for $i = 1,2,...,n$.

4: F(N) – ***real*** array. *Output*

On exit: the values of f_i, for $i = 1,2,...,n$.

5: N – INTEGER. *Input*

On entry: the number of equations.

FCN must be declared as EXTERNAL in the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: G – SUBROUTINE, supplied by the user. *External Procedure*

G must evaluate the boundary conditions in equation (3) and place them in the array BC.

Its specification is:

```
SUBROUTINE G(EPS, YA, YB, BC, N)
INTEGER    N
real       EPS, YA(N), YB(N), BC(N)
```

1: EPS – ***real***. *Input*

On entry: the value of the continuation parameter, ε. This is 1 if continuation is not being used.

2: YA(N) – ***real*** array. *Input*

On entry: the value $y_i(a)$, for $i = 1,2,...,n$.

3: YB(N) – ***real*** array. *Input*

On entry: the value $y_i(b)$, for $i = 1,2,...,n$.

4: BC(N) – ***real*** array. *Output*

On exit: the values $g_i(y(a),y(b),\varepsilon)$, for $i = 1,2,...,n$. These must be ordered as follows:

(i) first, the conditions involving only $y(a)$ (see NUMBEG description above);

(ii) next, the NUMMIX coupled conditions involving both $y(a)$ and $y(b)$ (see NUMMIX description above); and,

(iii) finally, the conditions involving only $y(b)$ (N–NUMBEG–NUMMIX).

5: N – INTEGER. *Input*

On entry: the number of equations, n.

G must be declared as EXTERNAL in the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

14: IJAC – INTEGER. *Input*

On entry: indicates whether or not the user is supplying Jacobian evaluation routines. If IJAC $\neq$ 0 then the user must supply routines JACOBF and JACOBG and also, when continuation is used, routines JACEPS and JACGEP. If IJAC = 0 numerical differentiation is used to calculate the Jacobian and the routines D02GAZ, D02GAY, D02GAZ and D02GAX respectively may be used as the dummy parameters.

15: JACOBF – SUBROUTINE, supplied by the user. *External Procedure*

JACOBF must evaluate the Jacobian $\left(\frac{\partial f_i}{\partial y_j}\right)$ for $i,j = 1,2,...,n$, given x and y_j, for $j = 1,2,...,n$.

Its specification is:

```
SUBROUTINE JACOBF(X, EPS, Y, F, N)
INTEGER    N
real       X, EPS, Y(N), F(N,N)
```

1: X – ***real***. *Input*

On entry: the value of the argument x.

2: EPS – ***real***. *Input*

On entry: the value of the continuation parameter ε. This is 1 if continuation is not being used.

3: Y(N) – ***real*** array. *Input*

On entry: the value of the argument y_i, for $i = 1,2,...,n$.

4: F(N,N) – ***real*** array. *Output*

On exit: F(i,j) must be set to the value of $\frac{\partial f_i}{\partial y_j}$, evaluated at the point (x,y), for $i,j = 1,2,...,n$.

5: N – INTEGER. *Input*

On entry: the number of equations, n.

JACOBF must be declared as EXTERNAL in the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: JACOBG – SUBROUTINE, supplied by the user. *External Procedure*

JACOBG must evaluate the Jacobians $\left(\frac{\partial g_i}{\partial y_j(a)}\right)$ and $\left(\frac{\partial g_i}{\partial y_j(b)}\right)$. The ordering of the rows of AJ and BJ must correspond to the ordering of the boundary conditions described in the specification of subroutine G above.

Its specification is:

```
SUBROUTINE JACOBG(EPS, YA, YB, AJ, BJ, N)
INTEGER    N
real       EPS, YA(N), YB(N), AJ(N,N), BJ(N,N)
```

1: EPS – ***real***. *Input*

On entry: the value of the continuation parameter, ε. This is 1 if continuation is not being used.

2: YA(N) – ***real*** array. *Input*

On entry: the value $y_i(a)$, for $i = 1,2,...,n$.

3: YB(N) – ***real*** array. *Input*

On entry: the value $y_i(b)$, for $i = 1,2,...,n$.

4: AJ(N,N) – ***real*** array. *Output*

On exit: AJ(i,j) must be set to the value $\frac{\partial g_i}{\partial y_j(a)}$, for $i,j = 1,2,...,n$.

5: BJ(N,N) – ***real*** array. *Output*

On exit: BJ(i,j) must be set to the value $\frac{\partial g_i}{\partial y_j(b)}$, for $i,j = 1,2...,n$.

6: N – INTEGER. *Input*

On entry: the number of equations, n.

JACOBG must be declared as EXTERNAL in the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

17: DELEPS – ***real.*** *Input/Output*

On entry: DELEPS must be given a value which specifies whether continuation is required. If DELEPS $\le$ 0.0 or DELEPS $\ge$ 1.0 then it is assumed that continuation is not required. If 0.0 < DELEPS < 1.0 then it is assumed that continuation is required unless DELEPS < $\sqrt{\textit{machine precision}}$ when an error exit is taken. DELEPS is used as the increment $\varepsilon_2 - \varepsilon_1$ (see (4)) and the choice DELEPS = 0.1 is recommended.

On exit: an overestimate of the increment $\varepsilon_p - \varepsilon_{p-1}$ (in fact the value of the increment which would have been tried if the restriction $\varepsilon_p = 1$ had not been imposed). If continuation was not requested then DELEPS = 0.0.

If continuation is not requested then the parameters JACEPS and JACGEP may be replaced by dummy actual parameters in the call to D02RAF. (D02GAZ and D02GAX respectively may be used as the dummy parameters.)

18: JACEPS – SUBROUTINE, supplied by the user. *External Procedure*

JACEPS must evaluate the derivative $\frac{\partial f_i}{\partial \varepsilon}$ given x and y if continuation is being used.

Its specification is:

```
SUBROUTINE  JACEPS(X, EPS, Y, F, N)
INTEGER     N
real        X, EPS, Y(N), F(N)
```

1: X – ***real.*** *Input*

On entry: the value of the argument x.

2: EPS – ***real.*** *Input*

On entry: the value of the continuation parameter, ε.

3: Y(N) – ***real*** array. *Input*

On entry: the solution values y_i at the point x, for $i = 1,2,...,n$.

4: F(N) – ***real*** array. *Output*

On exit: F(i) must contain the value $\frac{\partial f_i}{\partial \varepsilon}$ at the point (x,y), for $i = 1,2,...,n$.

5: N – INTEGER. *Input*

On entry: the number of equations, n.

JACEPS must be declared as EXTERNAL in the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

19: JACGEP – SUBROUTINE, supplied by the user. *External Procedure*

JACGEP must evaluate the derivatives $\frac{\partial g_i}{\partial \varepsilon}$ if continuation is being used.

Its specification is:

```
SUBROUTINE JACGEP(EPS, YA, YB, BCEP, N)
INTEGER    N
real       EPS, YA(N), YB(N), BCEP(N)
```

1: EPS – ***real***. *Input*

On entry: the value of the continuation parameter, ε.

2: YA(N) – ***real*** array. *Input*

On entry: the value of $y_i(a)$, for $i = 1,2,...,n$.

3: YB(N) – ***real*** array. *Input*

On entry: the value of $y_i(b)$, for $i = 1,2,...,n$.

4: BCEP(N) – ***real*** array. *Output*

On exit: BCEP(i) must contain the value of $\frac{\partial g_i}{\partial \varepsilon}$, for $i = 1,2,..,n$.

5: N – INTEGER. *Input*

On entry: the number of equations, n.

JACGEP must be declared as EXTERNAL in the (sub)program from which D02RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

20: WORK(LWORK) – ***real*** array. *Workspace*

21: LWORK – INTEGER. *Input*

On entry: the dimension of the array WORK as declared in the (sub)program from which D02RAF is called.

Constraint: LWORK $\geq$ MNP$\times(3N^2+6N+2) + 4N^2 + 3N$.

22: IWORK(LIWORK) – INTEGER array. *Workspace*

23: LIWORK – INTEGER. *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02RAF is called.

Constraints: LIWORK $\geq$ MNP$\times(2\times N+1) + N$, if IJAC $\neq 0$,
LIWORK $\geq$ MNP$\times(2\times N+1) + N^2 + 4\times N + 2$, if IJAC $= 0$.

24: IFAIL – INTEGER. *Input/Output*

For this routine, the normal use of IFAIL is extended to control the printing of error and warning messages as well as specifying hard or soft failure (see Chapter P01 for details).

Before entry, IFAIL must be set to a value with the decimal expansion cba, where each of the decimal digits c, b and a must have the value 0 or 1.

$a = 0$ specifies hard failure, otherwise soft failure;

$b = 0$ suppresses error messages, otherwise error messages will be printed (see Section 6);

$c = 0$ suppresses warning messages, otherwise warning messages will be printed (see Section 6).

The recommended value for inexperienced users is 110 (i.e. hard failure with all messages printed).

Unless the routine detects an error (see Section 6), IFAIL contains 0 on exit.

6. Error Indicators and Warnings

Errors detected by the routine:

For each error, an explanatory error message is output on the current error message unit (as defined by X04AAF), unless suppressed by the value of IFAIL on entry.

IFAIL = 1

One or more of the parameters N, MNP, NP, NUMBEG, NUMMIX, TOL, DELEPS, LWORK or LIWORK has been incorrectly set, or X(1) ≥ X(NP) or the mesh points X(i) are not in strictly ascending order.

IFAIL = 2

A finer mesh is required for the accuracy requested; that is MNP is not large enough. This error exit normally occurs when the problem being solved is difficult (for example, there is a boundary layer) and high accuracy is requested. A poor initial choice of mesh points will make this error exit more likely.

IFAIL = 3

The Newton iteration has failed to converge. There are several possible causes for this error:

(i) faulty coding in one of the Jacobian calculation routines;

(ii) if IJAC = 0 then inaccurate Jacobians may have been calculated numerically (this is a very unlikely cause); or,

(iii) a poor initial mesh or initial approximate solution has been selected either by the user or by default or there are not enough points in the initial mesh. Possibly, the user should try the continuation facility.

IFAIL = 4

The Newton iteration has reached roundoff error level. It could be however that the answer returned is satisfactory. The error is likely to occur if too high an accuracy is requested.

IFAIL = 5

The Jacobian calculated by JACOBG (or the equivalent matrix calculated by numerical differentiation) is singular. This may occur due to faulty coding of JACOBG or, in some circumstances, to a zero initial choice of approximate solution (such as is chosen when INIT = 0).

IFAIL = 6

There is no dependence on ε when continuation is being used. This can be due to faulty coding of JACEPS or JACGEP or, in some circumstances, to a zero initial choice of approximate solution (such as is chosen when INIT = 0).

IFAIL = 7

DELEPS is required to be less than ***machine precision*** for continuation to proceed. It is likely that either the problem (3) has no solution for some value near the current value of ε (see the advisory print out from D02RAF) or that the problem is so difficult that even with continuation it is unlikely to be solved using this routine. If the latter cause is suspected then using more mesh points initially may help.

IFAIL = 8
IFAIL = 9

Indicates that a serious error has occurred in a call to D02RAF or D02RAR respectively. Check all array subscripts and subroutine parameter lists in calls to D02RAF. Seek expert help.

7. Accuracy

The solution returned by the routine will be accurate to the user's tolerance as defined by the relation (5) except in extreme circumstances. The final error estimate over the whole mesh for each component is given in the array ABT. If too many points are specified in the initial mesh, the solution may be more accurate than requested and the error may not be approximately equidistributed.

8. Further Comments

There are too many factors present to quantify the timing. The time taken by the routine is negligible only on very simple problems.

The user is strongly recommended to set IFAIL to obtain self-explanatory error messages, and also monitoring information about the course of the computation.

In the case where the user wishes to solve a sequence of similar problems, the use of the final mesh and solution from one case as the initial mesh is strongly recommended for the next.

9. Example

We solve the differential equation

$$y''' = -yy'' - 2\varepsilon(1-y'^2)$$

with $\varepsilon = 1$ and boundary conditions

$$y(0) = y'(0) = 0, \quad y'(10) = 1$$

to an accuracy specified by TOL = 1.0E−4. The continuation facility is used with the continuation parameter ε introduced as in the differential equation above and with DELEPS = 0.1 initially. (The continuation facility is not needed for this problem and is used here for illustration.)

9.1. 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.

```
*     D02RAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, MNP, IY, LWORK, LIWORK
      PARAMETER        (N=3,MNP=40,IY=N,LWORK=MNP*(3*N*N+6*N+2)
     +                 +4*N*N+3*N,LIWORK=MNP*(2*N+1)+N)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             DELEPS, TOL
      INTEGER          I, IFAIL, IJAC, INIT, J, NP, NUMBEG, NUMMIX
*     .. Local Arrays ..
      real             ABT(N), WORK(LWORK), X(MNP), Y(IY,MNP)
      INTEGER          IWORK(LIWORK)
*     .. External Subroutines ..
      EXTERNAL         D02RAF, FCN, G, JACEPS, JACGEP, JACOBF, JACOBG,
     +                 X04ABF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02RAF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Calculation using analytic Jacobians'
      CALL X04ABF(1,NOUT)
      TOL = 1.0e-4
      NP = 17
      NUMBEG = 2
      NUMMIX = 0
      X(1) = 0.0e0
      X(NP) = 10.0e0
      INIT = 0
```

```
      DELEPS = 0.1e0
      IJAC = 1
*     * Set IFAIL to 111 to obtain monitoring information *
      IFAIL = 11
*
      CALL D02RAF(N,MNP,NP,NUMBEG,NUMMIX,TOL,INIT,X,Y,N,ABT,FCN,G,IJAC,
     +            JACOBF,JACOBG,DELEPS,JACEPS,JACGEP,WORK,LWORK,IWORK,
     +            LIWORK,IFAIL)
*
      IF (IFAIL.EQ.0 .OR. IFAIL.EQ.4) THEN
         IF (IFAIL.EQ.4) WRITE (NOUT,99996)
     +       'On exit from D02RAF IFAIL = ', IFAIL
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Solution on final mesh of ', NP, ' points'
         WRITE (NOUT,*)
     +     '      X(I)        Y1(I)        Y2(I)        Y3(I)'
         WRITE (NOUT,99998) (X(J),(Y(I,J),I=1,N),J=1,NP)
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Maximum estimated error by components'
         WRITE (NOUT,99997) (ABT(I),I=1,N)
      ELSE
         WRITE (NOUT,99996) 'On exit from D02RAF IFAIL = ', IFAIL
      END IF
   20 STOP
*
99999 FORMAT (1X,A,I2,A)
99998 FORMAT (1X,F10.3,3F13.4)
99997 FORMAT (11X,1P,3e13.2)
99996 FORMAT (1X,A,I3)
      END
*
      SUBROUTINE FCN(X,EPS,Y,F,M)
*     .. Scalar Arguments ..
      real              EPS, X
      INTEGER           M
*     .. Array Arguments ..
      real              F(M), Y(M)
*     .. Executable Statements ..
      F(1) = Y(2)
      F(2) = Y(3)
      F(3) = -Y(1)*Y(3) - 2.0e0*(1.0e0-Y(2)*Y(2))*EPS
      RETURN
      END
*
      SUBROUTINE G(EPS,Y,Z,AL,M)
*     .. Scalar Arguments ..
      real            EPS
      INTEGER         M
*     .. Array Arguments ..
      real            AL(M), Y(M), Z(M)
*     .. Executable Statements ..
      AL(1) = Y(1)
      AL(2) = Y(2)
      AL(3) = Z(2) - 1.0e0
      RETURN
      END
*
      SUBROUTINE JACEPS(X,EPS,Y,F,M)
*     .. Scalar Arguments ..
      real                 EPS, X
      INTEGER              M
*     .. Array Arguments ..
      real                 F(M), Y(M)
*     .. Executable Statements ..
      F(1) = 0.0e0
      F(2) = 0.0e0
      F(3) = -2.0e0*(1.0e0-Y(2)*Y(2))
      RETURN
      END
*
```

```
      SUBROUTINE JACGEP(EPS,Y,Z,AL,M)
*     .. Scalar Arguments ..
      real              EPS
      INTEGER           M
*     .. Array Arguments ..
      real              AL(M), Y(M), Z(M)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      DO 20 I = 1, M
         AL(I) = 0.0e0
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE JACOBF(X,EPS,Y,F,M)
*     .. Scalar Arguments ..
      real              EPS, X
      INTEGER           M
*     .. Array Arguments ..
      real              F(M,M), Y(M)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
      DO 40 I = 1, M
         DO 20 J = 1, M
            F(I,J) = 0.0e0
   20    CONTINUE
   40 CONTINUE
      F(1,2) = 1.0e0
      F(2,3) = 1.0e0
      F(3,1) = -Y(3)
      F(3,2) = 4.0e0*Y(2)*EPS
      F(3,3) = -Y(1)
      RETURN
      END
*
      SUBROUTINE JACOBG(EPS,Y,Z,A,B,M)
*     .. Scalar Arguments ..
      real              EPS
      INTEGER           M
*     .. Array Arguments ..
      real              A(M,M), B(M,M), Y(M), Z(M)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
      DO 40 I = 1, M
         DO 20 J = 1, M
            A(I,J) = 0.0e0
            B(I,J) = 0.0e0
   20    CONTINUE
   40 CONTINUE
      A(1,1) = 1.0e0
      A(2,2) = 1.0e0
      B(3,2) = 1.0e0
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D02RAF Example Program Results

Calculation using analytic Jacobians

Solution on final mesh of 33 points
      X(I)         Y1(I)         Y2(I)         Y3(I)
     0.000        0.0000        0.0000        1.6872
     0.063        0.0032        0.1016        1.5626
     0.125        0.0125        0.1954        1.4398
     0.188        0.0275        0.2816        1.3203
     0.250        0.0476        0.3605        1.2054
     0.375        0.1015        0.4976        0.9924
     0.500        0.1709        0.6097        0.8048
     0.625        0.2530        0.6999        0.6438
     0.703        0.3095        0.7467        0.5563
     0.781        0.3695        0.7871        0.4784
     0.938        0.4978        0.8513        0.3490
     1.094        0.6346        0.8977        0.2502
     1.250        0.7776        0.9308        0.1763
     1.458        0.9748        0.9598        0.1077
     1.667        1.1768        0.9773        0.0639
     1.875        1.3815        0.9876        0.0367
     2.031        1.5362        0.9922        0.0238
     2.188        1.6915        0.9952        0.0151
     2.500        2.0031        0.9983        0.0058
     2.656        2.1591        0.9990        0.0035
     2.813        2.3153        0.9994        0.0021
     3.125        2.6277        0.9998        0.0007
     3.750        3.2526        1.0000        0.0001
     4.375        3.8776        1.0000        0.0000
     5.000        4.5026        1.0000        0.0000
     5.625        5.1276        1.0000        0.0000
     6.250        5.7526        1.0000        0.0000
     6.875        6.3776        1.0000        0.0000
     7.500        7.0026        1.0000        0.0000
     8.125        7.6276        1.0000        0.0000
     8.750        8.2526        1.0000        0.0000
     9.375        8.8776        1.0000        0.0000
    10.000        9.5026        1.0000        0.0000

Maximum estimated error by components
               6.92E-05      1.81E-05      6.42E-05
```

With IFAIL set to 111 in the example program, monitoring information similar to that below is printed:

```
D02RAF MONITORING INFORMATION

 MONITORING NEWTON ITERATION
  NUMBER OF POINTS IN CURRENT MESH =   17
  CORRECTION NUMBER    0   RESIDUAL SHOULD BE .LE.  1.00E+00
   ITERATION NUMBER    0   RESIDUAL =  1.00E+00
    SQUARED NORM OF CORRECTION =  9.90E+01
    SQUARED NORM OF GRADIENT   =  1.00E+00
    SCALAR PRODUCT OF CORRECTION AND GRADIENT =  1.00E+00
   ITERATION NUMBER    1   RESIDUAL =  5.59E-01
 .
 .
```

intermediate results omitted

```
 .
 .
```

```
MESH SELECTION
 NUMBER OF NEW POINTS   5

MONITORING NEWTON ITERATION
 NUMBER OF POINTS IN CURRENT MESH =    33
 CORRECTION NUMBER    1   RESIDUAL SHOULD BE .LE.  1.22E-05
  ITERATION NUMBER    0   RESIDUAL =  3.58E-04
   SQUARED NORM OF CORRECTION =  1.70E-06
   SQUARED NORM OF GRADIENT   =  2.89E-07
   SCALAR PRODUCT OF CORRECTION AND GRADIENT =  1.28E-07
  ITERATION NUMBER    1   RESIDUAL =  2.70E-08

MESH SELECTION
 NUMBER OF NEW POINTS   0

CORRECTION NUMBER    1   ESTIMATED MAXIMUM ERROR =  6.92E-05
ESTIMATED ERROR BY COMPONENTS
  6.92E-05  1.81E-05  6.42E-05
```

D02SAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02SAF solves a two-point boundary-value problem for a system of first order ordinary differential equations with boundary conditions, combined with additional algebraic equations. It uses initial value techniques and a modified Newton iteration in a shooting and matching method.

2. Specification

```
      SUBROUTINE D02SAF (P, M, N, N1, PE, PF, E, DP, NPOINT, WP, IWP,
     1                   ICOUNT, RANGE, BC, FCN, EQN, CONSTR, YMAX, MONIT,
     2                   PRSOL, W, IW1, IW2, IFAIL)
      INTEGER          M, N, N1, NPOINT, IWP, ICOUNT, IW1, IW2, IFAIL
      real             P(M), PE(M), PF(M), E(N), DP(M), WP(IWP,6), YMAX,
     1                 W(IW1,IW2)
      LOGICAL          CONSTR
      EXTERNAL         RANGE, BC, FCN, EQN, CONSTR, MONIT, PRSOL
```

3. Description

D02SAF solves a two-point boundary-value problem for a system of n first-order ordinary differential equations with separated boundary conditions by determining certain unknown parameters $p_1,p_2,\ldots,p_m$. (There may also be additional algebraic equations to be solved in the determination of the parameters and, if so, these equations are defined by the routine EQN.) The parameters may be, but need not be, boundary values; they may include eigenvalues, parameters in the coefficients of the differential equations, coefficients in series expansions or asymptotic expansions for boundary values, the length of the range of definition of the system of differential equations etc.

It is assumed that we have a system of n differential equations of the form

$$y' = f(x,y,p) \qquad (1)$$

where $p = (p_1,p_2,\ldots,p_m)^T$ is the vector of parameters, and that the derivative f is evaluated by a routine FCN. Also, n_1 of the equations are assumed to depend on p. For $n_1 < n$ the $n - n_1$ equations of the system are not involved in the matching process. These are the driving equations; they should be independent of p and of the solution of the other n_1 equations. In numbering the equations in FCN and BC the driving equations must be put **first** (as they naturally occur in most applications). The range of definition $[a,b]$ of the differential equations is defined by the routine RANGE and may depend on the parameters $p_1,p_2,\ldots,p_m$ (that is, on p). RANGE must define the points $x_1,x_2,\ldots,x_{\mathrm{NPOINT}}$, NPOINT $\geq$ 2, which must satisfy

$$a = x_1 < x_2 < \ldots < x_{\mathrm{NPOINT}} = b \qquad (2)$$

(or a similar relationship with all the inequalities reversed).

If NPOINT > 2 the points $x_1,x_2,\ldots,x_{\mathrm{NPOINT}}$ can be used to break up the range of definition. Integration is restarted at each of these points. This means that the differential equations (1) can be defined differently in each subinterval $[x_i,x_{i+1}]$, for $i = 1,2,\ldots,\mathrm{NPOINT}-1$. Also, since initial and maximum integration step-sizes can be supplied on each subinterval (via the array WP), the user can indicate parts of the range $[a,b]$ where the solution $y(x)$ may be difficult to obtain accurately and can take appropriate action.

The boundary conditions may also depend on the parameters and are applied at $a = x_1$ and $b = x_{\mathrm{NPOINT}}$. They are defined (in the routine BC) in the form

$$y(a) = g_1(p), \quad y(b) = g_2(p). \qquad (3)$$

The boundary-value problem is solved by determining the unknown parameters p by a shooting and matching technique. The differential equations are always integrated from a to b with initial values $y(a) = g_1(p)$. The solution vector thus obtained at $x = b$ is subtracted from the vector $g_2(p)$ to give the n_1 residuals $r_1(p)$, ignoring the first $n - n_1$, driving equations. Because the direction of integration is always from a to b, it is unnecessary, in BC, to supply values for the first $n - n_1$ boundary values at b, that is the first $n - n_1$ components of g_2 in (3). For $n_1 < m$ then $r_1(p)$. Together with the $m - n_1$ equations defined by routine EQN,

$$r_2(p) = 0, \tag{4}$$

these give a vector of residuals r, which at the solution, p, must satisfy

$$r(p) = \begin{pmatrix} r_1(p) \\ r_2(p) \end{pmatrix} = 0. \tag{5}$$

These equations are solved by a pseudo-Newton iteration which uses a modified singular value decomposition of $J = \frac{\partial r}{\partial p}$ when solving the linear equations which arise. The Jacobian J used in Newton's method is obtained by numerical differentiation. The parameters at each Newton iteration are accepted only if the norm $\|D^{-1}\tilde{J}^{+}r\|_2$ is much reduced from its previous value. Here $\tilde{J}^{+}$ is the pseudo-inverse, calculated from the singular value decomposition, of a modified version of the Jacobian J ($\tilde{J}^{+}$ is actually the inverse of the Jacobian in well-conditioned cases). D is a diagonal matrix with

$$d_{ii} = \max(|p_i|, \text{PF}(i)), \tag{6}$$

where PF is an array of floor values.

See Deuflhard [3] for further details of the variants of Newton's method used, Gay [2] for the modification of the singular value decomposition and Gladwell [4] for an overview of the method used.

Two facilities are provided to prevent the pseudo-Newton iteration running into difficulty. First, the user is permitted to specify constraints on the values of the parameters p via a logical function CONSTR. These constraints are only used to prevent the Newton iteration using values for p which would violate them; that is, they are not used to determine the values of p. Secondly, the user is permitted to specify a maximum value y_{max} for $\|y(x)\|_\infty$ at all points in the range $[a,b]$. It is intended that this facility be used to prevent machine 'overflow' in the integrations of equation (1) due to poor choices of the parameters p which might arise during the Newton iteration. When using this facility, it is presumed that the user has an estimate of the likely size of $\|y(x)\|_\infty$ at all points $x \in [a,b]$. y_{max} should then be chosen rather larger (say by a factor of 10) than this estimate.

The user is strongly advised to supply a routine MONIT (or to call the 'default' routine D02HBX, see below) to monitor the progress of the pseudo-Newton iteration. The user can output the solution of the problem $y(x)$ by supplying a suitable routine PRSOL (an example is given in Section 9 of a routine designed to output the solution at equally spaced points).

D02SAF is designed to try all possible options before admitting failure and returning to the user. Provided the routine can start the Newton iteration from the initial point p it will exhaust all the options available to it (though the user can override this by specifying a maximum number of iterations to be taken). The fact that all its options have been exhausted is the only error exit from the iteration. Other error exits are possible, however, whilst setting up the Newton iteration and when computing the final solution.

The user who requires more background information about the solution of boundary value problems by shooting methods is recommended to read the appropriate chapters of Hall and Watt [1], and for a detailed description of D02SAF Gladwell [4] is recommended.

4. References

[1] HALL, G. and WATT, J.M.
Modern Numerical Methods in Ordinary Differential Equations.
Clarendon Press, Oxford, 1976.

[2] GAY, D.
On Modifying Singular Values to Solve Possibly Singular Systems of Nonlinear Equations. Working Paper 125, Computer Research Centre, National Bureau for Economics and Management Science, Cambridge, Mass, 1976.

[3] DEUFLHARD, P.
A Modified Newton Method for the Solution of Ill-conditioned Systems of Nonlinear Equations with Application to Multiple Shooting.
Num. Math. 22, pp. 289-315, 1974.

[4] GLADWELL, I.
The Development of the Boundary Value Codes in the Ordinary Differential Equation Chapter of the NAG Fortran Library.
In: 'Codes for Boundary Value Problems in Ordinary Differential Equations',
B. Child, M. Scott, J.W. Daniel, E. Denman and P. Nelson (eds).
Springer-Verlag Lecture Notes in Computer Science. 76, 1979.

5. Parameters

1: P(M) – ***real*** array. *Input/Output*

On entry: P(i) must be set to an estimate of the ith parameter, p_i, for $i = 1,2,...,m$.

On exit: the corrected value for the ith parameter, unless an error has occurred, when it contains the last calculated value of the parameter.

2: M – INTEGER. *Input*

On entry: the number of parameters, m.

Constraint: M > 0.

3: N – INTEGER. *Input*

On entry: the total number of differential equations, n.

Constraint: N > 0.

4: N1 – INTEGER. *Input*

On entry: the number of differential equations active in the matching process, n_1. The active equations must be placed last in the numbering in the routines FCN and BC (see below). The **first** N – N1 equations are used as the driving equations.

Constraint: N1 ≤ N, N1 ≤ M and N1 > 0.

5: PE(M) – ***real*** array. *Input*

On entry: PE(i), for $i = 1,2,...,m$, must be set to a positive value for use in the convergence test in the ith parameter p_i. See the specification of PF below for further details.

Constraint: PE(i) > 0, for $i = 1,2,...,m$.

6: PF(M) – ***real*** array. *Input/Output*

On entry: PF(i), for $i = 1,2,...,m$, should be set to a 'floor' value in the convergence test on the ith parameter p_i. If PF(i) ≤ 0.0 on entry then it is set to the small positive value $\sqrt{\varepsilon}$ (where ε may in most cases be considered to be ***machine precision***); otherwise it is used unchanged.

The Newton iteration is presumed to have converged if a full Newton step is taken (ISTATE = 1 in the specification of MONIT below), the singular values of the Jacobian are not being significantly perturbed (also see MONIT) and if the Newton correction C_i satisfies

$$|C_i| \leq \mathrm{PE}(i)\times\max(|p_i|,\mathrm{PF}(i)), \qquad i = 1,2,...,m,$$

where p_i is the current value of the ith parameter. The values PF(i) are also used in determining the Newton iterates as discussed in Section 3, see equation (6).

On exit: the values actually used.

7: E(N) – ***real*** array. *Input*

On entry: values for use in controlling the local error in the integration of the differential equations. If err_i is an estimate of the local error in y_i, for $i = 1,2,...,n$ then

$$|err_i| \leq \mathrm{E}(i) \times \max\{\sqrt{\varepsilon}, |y_i|\}$$

where ε may in most cases be considered to be ***machine precision***.

Suggested value: E(i) = 10^{-5}.

Constraint: E(i) > 0.0, for $i = 1,2,...,$N.

8: DP(M) – ***real*** array. *Input/Output*

On entry: a value to be used in perturbing the parameter p_i in the numerical differentiation to estimate the Jacobian used in Newton's method. If DP(i) = 0.0 on entry, an estimate is made internally by setting

$$\mathrm{DP}(i) = \sqrt{\varepsilon} \times \max(\mathrm{PF}(i), |p_i|) \qquad (7)$$

where p_i is the initial value of the parameter supplied by the user and ε may in most cases be considered to be ***machine precision***. The estimate of the Jacobian, J, is made using forward differences, that is for each i, for $i = 1,2,...,m$, p_i is perturbed to p_i + DP(i) and the ith column of J is estimated as

$$(r(p_i+\mathrm{DP}(i))-r(p_i))/\mathrm{DP}(i)$$

where the other components of p are unchanged (see equation (3) for the notation used). If this fails to produce a Jacobian with significant columns, backward differences are tried by perturbing p_i to p_i – DP(i) and if this also fails then central differences are used with p_i perturbed to p_i + 10.0×DP(i). If this also fails then the calculation of the Jacobian is abandoned. If the Jacobian has not previously been calculated then an error exit is taken. If an earlier estimate of the Jacobian is available then the current parameter set, p_i, for $i = 1,2,...,$M, is abandoned in favour of the last parameter set from which useful progress was made and the singular values of the Jacobian used at the point are modified before proceeding with the Newton iteration. The user is recommended to use the default value DP(i) = 0.0 unless he has prior knowledge of a better choice. If any of the perturbations described above are likely to lead to an unfortunate set of parameter values then the user should use the LOGICAL FUNCTION CONSTR (see below) to prevent such perturbations (all changes of parameters are checked by a call to CONSTR).

On exit: the values actually used.

9: NPOINT – INTEGER. *Input*

On entry: 2 plus the number of breakpoints in the range of definition of the system of differential equations (1),

Constraint: NPOINT ≥ 2.

10: WP(IWP,6) – ***real*** array. *Input/Output*

On entry: WP(i,1) must contain an estimate for an initial stepsize for integration across the ith subinterval [X(i), X(i+1)], i = 1,2,...,NPOINT–1 (see RANGE below). WP(i,1) should have the same sign as X(i+1) – X(i) if it is non-zero. If WP(i,1) = 0.0, on entry, a default value for the initial stepsize is calculated internally. This is the recommended mode of entry.

WP(i,2) must contain an upper bound on the modulus of the stepsize to be used in the integration on [X(i), X(i+1)], i = 1,2,...,NPOINT–1. If WP(i,2) = 0.0 on entry no bound is assumed. This is the recommended mode of entry unless the solution is expected to have important features which might be 'missed' in the integration if the stepsize were permitted to be chosen freely.

WP(i,3) must contain a lower bound for the modulus of the step size on the ith sub-interval [X(i),X(i+1)], for i = 1,2,...,NPOINT−1. If WP(i,3) = 0.0 on entry, a very small default value is used. By setting WP(i,3) > 0.0 but smaller than the expected step sizes (assuming the user has some insight into the likely step sizes) expensive integrations with parameters p far from the solution can be avoided.

On exit: WP(i,1) contains the initial step size used on the last integration on [X(i), X(i+1)], for i = 1,2,...,NPOINT−1, (excluding integrations during the calculation of the Jacobian).

WP(i,2), for i = 1,2,...,NPOINT−1, is usually unchanged. If the maximum step size WP(i,2) is so small or the length of the range [X(i), X(i+1)] is so short that on the last integration the step size was not controlled in the main by the size of the error tolerances E(i) but by these other factors, then WP(NPOINT,2) is set to the floating-point value of i if the problem last occurred in [X(i),X(i+1)]. Any results obtained when this value is returned as non-zero should be viewed with caution.

WP(i,3), for i = 1,2,...,NPOINT−1 are unchanged.

If an error exit with IFAIL = 4, 5, or 6 (see Section 6) occurs on the integration made from X(i) to X(i+1) the floating-point value of i is returned in WP(NPOINT,1). The actual point $x \in$ [X(i),X(i+1)] where the error occurred is returned in WP(1,5) (see also the specification of W). The floating-point value of NPOINT is returned in WP(NPOINT,1) if the error exit is caused by a call to BC.

If an error exit occurs when estimating the Jacobian matrix (IFAIL = 7, 8, 9, 10, 11, 12, see Section 6) and if parameter p_i was the cause of the failure then on exit WP(NPOINT,1) contains the floating-point value of i.

WP(i,4) contains the point X(i), for i = 1,2,...,NPOINT, used at the solution p or at the final values of p if an error occurred.

WP is also partly used as workspace.

11: IWP – INTEGER. *Input*

On entry: the first dimension of the array WP as declared in the (sub)program from which D02SAF is called.

Constraint: IWP ≥ NPOINT.

12: ICOUNT – INTEGER. *Input*

On entry: an upper bound on the number of Newton iterations. If ICOUNT = 0 on entry, no check on the number of iterations is made (this is the recommended mode of entry).

Constraint: ICOUNT ≥ 0.

13: RANGE – SUBROUTINE, supplied by the user. *External Procedure*

RANGE must specify the break-points x_i, for i = 1,2,...,NPOINT, which may depend on the parameters p_j, for j = 1,2,...,M.

Its specification is:

```
SUBROUTINE RANGE(X, NPOINT, P, M)
INTEGER    NPOINT, M
real       X(NPOINT), P(M)
```

1: X(NPOINT) – ***real*** array. *Output*

On exit: the ith break-point, for i = 1,2,...,NPOINT. The sequence (X(i)) must be strictly monotonic, that is either

a = X(1) < X(2) < ... < X(NPOINT) = b
or a = X(1) > X(2) > ... > X(NPOINT) = b

2: NPOINT – INTEGER. *Input*

On entry: two plus the number of break-points in (a,b).

3: P(M) – ***real*** array. *Input*

On entry: the current estimate of the ith parameter, for $i = 1,2,...,m$.

4: M – INTEGER. *Input*

On entry: the number of parameters, m.

RANGE must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

14: BC – SUBROUTINE, supplied by the user. *External Procedure*

BC must place in G1 and G2 the boundary conditions at a and b respectively.

Its specification is:

```
SUBROUTINE BC(G1, G2, P, M, N)
INTEGER    M, N
real       G1(N), G2(N), P(M)
```

1: G1(N) – ***real*** array. *Output*

On exit: the value of $y_i(a)$, (where this may be a known value or a function of the parameters p_j, for $j = 1,2,...,m$), for $i = 1,2,...,n$.

2: G2(N) – ***real*** array. *Output*

On exit: the value of $y_i(b)$, for $i = 1,2,...,n$, (where these may be known values or functions of the parameters p_j, for $j = 1,2,...,m$). If $n > n_1$, so that there are some driving equations, then the first $n - n_1$ values of G2 need not be set since they are never used.

3: P(M) – ***real*** array. *Input*

On entry: an estimate of the ith parameter, p_i, for $i = 1,2,...,m$.

4: M – INTEGER. *Input*

On entry: the number of parameters, m.

5: N – INTEGER. *Input*

On entry: the number of differential equations, n.

BC must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

15: FCN – SUBROUTINE, supplied by the user. *External Procedure*

FCN must evaluate the functions f_i (i.e. the derivatives y_i'), for $i = 1,2,...,n$.

Its specification is:

```
SUBROUTINE FCN(X, Y, F, N, P, M, I)
INTEGER    N, M, I
real       X, Y(N), F(N), P(M)
```

1: X – ***real***. *Input*

On entry: the value of the argument x.

2: Y(N) – ***real*** array. *Input*

On entry: the value of the argument, y_i, for $i = 1,2,...,n$.

3: F(N) – ***real*** array. *Output*

On exit: the derivative of y_i evaluated at x, for $i = 1,2,...,n$. F(i) may depend upon the parameters p_j, for $j = 1,2,...,m$. If there are any driving equations (see Section 3) then these must be numbered first in the ordering of the components of F.

4: N – INTEGER. *Input*

On entry: the number of equations, n.

5: P(M) – ***real*** array. *Input*

On entry: the current estimate of the ith parameter, p_i, for $i = 1,2,...,m$.

6: M – INTEGER. *Input*

On entry: the number of parameters, m.

7: I – INTEGER. *Input*

On entry: specifies the sub-interval $[x_i,x_{i+1}]$ on which the derivatives are to be evaluated.

FCN must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

16: EQN – SUBROUTINE, supplied by the user. *External Procedure*

EQN is used to describe the additional algebraic equations to be solved in the determination of the parameters, p_i, for $i = 1,2,...,m$. If there are no additional algebraic equations (i.e. $m = n_1$) then EQN is never called and the dummy routine D02HBZ should be used as the actual argument.

Its specification is:

```
SUBROUTINE EQN(E, Q, P, M)
INTEGER    Q, M
real       E(Q), P(M)
```

1: E(Q) – ***real*** array. *Output*

On exit: the vector of residuals, $r_2(p)$, that is the amount by which the current estimates of the parameters fail to satisfy the algebraic equations.

2: Q – INTEGER. *Input*

On entry: the number of algebraic equations, $m - n_1$.

3: P(M) – ***real*** array. *Input*

On entry: the current estimate of the ith parameter p_i, for $i = 1,2,...,m$.

4: M – INTEGER. *Input*

On entry: the number of parameters, m.

EQN must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

17: CONSTR – LOGICAL FUNCTION, supplied by the user. *External Procedure*

CONSTR is used to prevent the pseudo-Newton iteration running into difficulty. CONSTR should return the value .TRUE. if the constraints are satisfied by the parameters $p_1,p_2,...,p_m$. Otherwise CONSTR should return the value .FALSE.. Usually the dummy functon D02HBY, which returns the value .TRUE. at all times, will suffice and in the first instance this is recommended as the actual parameter.

Its specification is:

```
LOGICAL FUNCTION  CONSTR(P, M)
INTEGER           M
real              P(M)
```

1: P(M) – ***real*** array. *Input*

On entry: an estimate of the ith parameter, p_i, for $i = 1,2,...,m$.

2: M – INTEGER. *Input*

On entry: the number of parameters, m.

CONSTR must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

18: YMAX – ***real***. *Input*

On entry: a non-negative value which is used as a bound on all values $\|y(x)\|_\infty$ where $y(x)$ is the solution at any point x between X(1) and X(NPOINT) for the current parameters $p_1,p_2,...,p_m$. If this bound is exceeded the integration is terminated and the current parameters are rejected. Such a rejection will result in an error exit if it prevents the initial residual or Jacobian, or the final solution, being calculated. If YMAX = 0 on entry, no bound on the solution y is used; that is the integrations proceed without any checking on the size of $\|y\|_\infty$.

19: MONIT – SUBROUTINE, supplied by the user. *External Procedure*

MONIT enables the user to monitor the values of various quantities during the calculation. It is called by D02SAF after every calculation of the norm $\|D^{-1}\tilde{J}_{+r}\|_2$ which determines the strategy of the Newton method, every time there is an internal error exit leading to a change of strategy, and before an error exit when calculating the initial Jacobian. Usually the routine D02HBX will be adequate and the user is advised to use this as the actual parameter for MONIT in the first instance. (In this case a call to X04ABF must be made prior to the call of D02SAF). If no monitoring is required, the dummy routine D02SAS may be used. (In some implementations of the Library the names D02HBX and D02SAS are changed to HBXD02 and SASD02: refer to the Users' Note for your implementation).

Its specification is:

```
SUBROUTINE MONIT(ISTATE, IFLAG, IFAIL1, P, M, F, PNORM, PNORM1, EPS, D)
INTEGER    ISTATE, IFLAG, IFAIL1, M
real       P(M), F(M), PNORM, PNORM1, EPS, D(M)
```

1: ISTATE – INTEGER. *Input*

On entry: the state of the Newton iteration:

ISTATE = 0

the calculation of the residual, Jacobian and $\|D^{-1}\tilde{J}^{+}r\|_2$ are taking place.

ISTATE = 1 to 5

during the Newton iteration a factor of $2^{(-\text{ISTATE}+1)}$ of the Newton step is being used to try to reduce the norm.

ISTATE = 6

the current Newton step has been rejected and the Jacobian is being re-calculated.

ISTATE = −6 to −1

an internal error exit has caused the rejection of the current set of parameter values, p. −ISTATE is the value which ISTATE would have taken if the error had not occurred.

ISTATE = −7

an internal error exit has occurred when calculating the initial Jacobian.

2: IFLAG – INTEGER. *Input*

On entry: whether or not the Jacobian being used has been calculated at the beginning of the current iteration. If the Jacobian has been updated then IFLAG = 1; otherwise IFLAG = 2. The Jacobian is only calculated when convergence to the current parameter values has been slow.

3: IFAIL1 – INTEGER. *Input*

On entry: if $-6 \leq$ ISTATE ≤ -1, IFAIL1 specifies the IFAIL error number that would be produced were control returned to the user. IFAIL1 is unspecified for values of ISTATE outside this range.

4: P(M) – ***real*** array. *Input*

On entry: the current estimate of the ith parameter, p_i, for $i = 1,2,...,m$.

5: M – INTEGER. *Input*

On entry: the number of parameters, m.

6: F(M) – ***real*** array. *Input*

On entry: the residual r corresponding to the current parameter values, provided $1 \leq$ ISTATE ≤ 5 or ISTATE $= -7$. F is unspecified for other values of ISTATE.

7: PNORM – ***real***. *Input*

On entry: a quantity against which all reductions in norm are currently measured.

8: PNORM1 – ***real***. *Input*

On entry: the norm of the current parameters, p. It is set for $1 \leq$ ISTATE ≤ 5 and is undefined for other values of ISTATE.

9: EPS – ***real***. *Input*

On entry: EPS gives some indication of the convergence rate. It is the current singular value modification factor (see Gay [2]). It is 0 initially and whenever convergence is proceeding steadily. EPS is $\varepsilon^{\frac{3}{4}}$ or greater (where ε may in most cases be considered ***machine precision***) when the singular values of J are approximately zero or when convergence is not being achieved. The larger the value of EPS the worse the convergence rate. When EPS becomes too large the Newton iteration is terminated.

10: D(M) – ***real*** array. *Input*

On entry: the singular values of the current modified Jacobian matrix, J. If D(m) is small relative to D(1) for a number of Jacobians corresponding to different parameter values then the computed results should be viewed with suspicion. It could be that the matching equations do not depend significantly on some parameter (which could be due to a programming error in FCN, BC, RANGE or EQN). Alternatively, the system of differential equations may be very ill-conditioned when viewed as an initial value problem, in which case this routine is unsuitable. This may also be indicated by some singular values being very large. These values of D(i), $i = 1,2,...,m$ should not be changed.

MONIT must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

20: PRSOL – SUBROUTINE, supplied by the user. *External Procedure*

PRSOL can be used to obtain values of the solution y at a selected point z by integration across the final range [X(1),X(NPOINT)]. If no output is required D02HBW can be used as the actual parameter.

Its specification is:

```
SUBROUTINE PRSOL(Z, Y, N)
INTEGER     N
real        Z, Y(N)
```

1: Z – ***real***. *Input/Output*

On entry: contains x_1 on the first call. On subsequent calls Z contains its previous output value.

On exit: the next point at which output is required. The new point must be nearer X(NPOINT) than the old.

If Z is set to a point outside [X(1),X(NPOINT)] the process stops and control returns from D02SAF to the (sub)program from which D02SAF is called. Otherwise the next call to PRSOL is made by D02SAF at the point Z, with solution values $y_1,y_2,...,y_n$ at Z contained in Y. If Z is set to X(NPOINT) exactly, the final call to PRSOL is made with $y_1,y_2,...,y_n$ as values of the solution at X(NPOINT) produced by the integration. In general the solution values obtained at X(NPOINT) from PRSOL will differ from the values obtained at this point by a call to routine BC. The difference between the two solutions is the residual r. The user is reminded that the points X(1),X(2),...,X(NPOINT) are available in the locations WP(1,4),WP(2,4),...,WP(NPOINT,4) at all times.

2: Y(N) – ***real*** array. *Input*

On entry: the solution value y_i at z, for $i = 1,2,...,n$.

3: N – INTEGER. *Input*

On entry: the total number of differential equations, n.

PRSOL must be declared as EXTERNAL in the (sub)program from which D02SAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

21: W(IW1,IW2) – ***real*** array. *Output*

On exit: in the case of an error exit of the type where the point of failure is returned in WP(1,5), the solution at this point of failure is returned in W(i,1), for $i = 1,2,...,n$.

Otherwise W is used for workspace.

22: IW1 – INTEGER. *Input*

On entry: the first dimension of the array W as declared in the (sub)program from which D02SAF is called.

Constraint: IW1 ≥ max(N,M).

23: IW2 – INTEGER. *Input*

On entry: the second dimension of the array W as declared in the (sub)program from which D02SAF is called.

Constraint: IW2 ≥ 3×M + 12 + max(11,M).

24: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

One or more of the parameters N, N1, M,IWP, NPOINT, ICOUNT, IW1, IW2, E, PE or YMAX has been incorrectly set.

IFAIL = 2

The constraints have been violated by the initial parameters.

IFAIL = 3

The condition X(1) < X(2) < ... < X(NPOINT) (or X(1) > X(2) > ... > X(NPOINT)) has been violated on a call to RANGE with the initial parameters.

IFAIL = 4

In the integration from X(1) to X(NPOINT) with the initial or the final parameters, the step size was reduced too far for the integration to proceed. Consider reversing the order of the points X(1),X(2),...,X(NPOINT). If this error exit still results, it is likely that D02SAF is not a suitable method for solving the problem, or the initial choice of parameters is very poor, or the accuracy requirement specified by E(i), for $i = 1,2,...,n$, is too stringent.

IFAIL = 5

In the integration from X(1) to X(NPOINT) with the initial or final parameters, an initial step could not be found to start the integration on one of the intervals X(i) to X(i+1). Consider reversing the order of the points. If this error exit still results it is likely that D02SAF is not a suitable routine for solving the problem, or the initial choice of parameters is very poor, or the accuracy requirement specified by E(i), for $i = 1,2,...,n$, is much too stringent.

IFAIL = 6

In the integration from X(1) to X(NPOINT) with the initial or final parameters, the solution exceeded YMAX in magnitude (when YMAX > 0). It is likely that the initial choice of parameters was very poor or YMAX was incorrectly set.

Note: on an error with IFAIL = 4, 5 or 6 with the initial parameters, the interval in which failure occurs is contained in WP(NPOINT,1). If a subroutine MONIT similar to the one in Section 9 is being used then it is a simple matter to distinguish between errors using the initial and final parameters. None of the error exits IFAIL = 4, 5 or 6 should occur on the **final** integration (when computing the solution) as this integration has already been performed previously with exactly the same parameters p_i, for $i = 1,2,...,m$. Seek expert help if this error occurs.

IFAIL = 7

On calculating the initial approximation to the Jacobian, the constraints were violated.

IFAIL = 8

On perturbing the parameters when calculating the initial approximation to the Jacobian, the condition X(1) < X(2) < ... < X(NPOINT) (or X(1) > X(2) > ... > X(NPOINT)) is violated.

IFAIL = 9

On calculating the initial approximation to the Jacobian, the integration step size was reduce[illegible] (see IFAIL = 4)

IFAIL = 10

On calculating the initial approximation to the Jacobian, the initial integration step size on some interval was too small (see IFAIL = 5).

IFAIL = 11

On calculating the initial approximation to the Jacobian, the solution of the system of differential equations exceeded YMAX in magnitude (when YMAX > 0).

Note: all the error exits IFAIL = 7, 8, 9, 10 and 11 can be treated by reducing the size of some or all the elements of DP.

IFAIL = 12

On calculating the initial approximation to the Jacobian, a column of the Jacobian is found to be insignificant. This could be due to an element DP(i) being too small (but non-zero) or the solution having no dependence on one of the parameters (a programming error).

Note: on an error exit with IFAIL = 7, 8, 9, 10, 11 or 12, if a perturbation of the parameter p_i is the cause of the error then WP(NPOINT,1) will contain the floating-point value of i.

IFAIL = 13

After calculating the initial approximation to the Jacobian, F02SZF failed to calculate its singular value decomposition (see the specification of F02SZF for further discussion). It is likely that the error will never occur as it is usually associated with the Jacobian having multiple singular values. To remedy the error it should only be necessary to change the initial parameters. If the error persists it is likely that the problem has not been correctly formulated.

IFAIL = 14

The Newton iteration has failed to converge after exercising all its options. The user is strongly recommended to monitor the progress of the iteration via the parameter MONIT. There are many possible reasons for the iteration not converging. Amongst the most likely are:

(a) there is no solution;

(b) the initial parameters are too far away from the correct parameters;

(c) the problem is too ill-conditioned as an initial value problem for Newton's method to choose suitable corrections;

(d) the accuracy requirements for convergence are too restrictive, that is some of the components of PE (and maybe PF) are too small – in this case the final value of this norm output via MONIT will usually be very small; or

(e) the initial parameters are so close to the solution parameters p that the Newton iteration cannot find improved parameters. The norm output by MONIT should be very small.

IFAIL = 15

The number of iterations permitted by ICOUNT has been exceeded (in the case when ICOUNT > 0 on entry).

IFAIL = 16, 17, 18 and 19

These indicate that there has been a serious error in one of the auxiliary routines D02SAZ, D02SAW, D02SAX or D02SAU respectively. Check all subroutine calls and array dimensions. Seek expert help.

7. Accuracy

If the iteration converges, the accuracy to which the unknown parameters are determined is usually close to that specified by the user. The accuracy of the solution (output via PRSOL) depends on the error tolerances E(i), for $i = 1,2,...,n$. The user is strongly recommended to vary all tolerances to check the accuracy of the parameters p and the solution y.

8. Further Comments

The time taken by the routine depends on the complexity of the system of differential equations and on the number of iterations required. In practice, the integration of the differential system (1) is usually by far the most costly process involved. The computing time for integrating the differential equations can sometimes depend critically on the quality of the initial estimates for the parameters p. If it seems that too much computing time is required and, in particular, if the values of the residuals (output in MONIT) are much larger than expected given the user's knowledge of the expected solution, then the coding of the subroutines FCN, EQN, RANGE and BC should be checked for errors. If no errors can be found then an independent attempt should be made to improve the initial estimates p.

In the case of an error exit in the integration of the differential system indicated by IFAIL = 4, 5, 9 or 10 the user is strongly recommended to perform trial integrations with D02PDF to determine the effects of changes of the local error tolerances and of changes to the initial choice of the parameters p_i, for $i = 1,2...,m$ (that is the initial choice of p).

It is possible that by following the advice given in Section 6 an error exit with IFAIL = 7, 8, 9, 10 or 11 might be followed by one with IFAIL = 12 (or vice-versa) where the advice given is the opposite. If the user is unable to refine the choice of DP(i), for $i = 1,2,...,n$, such that both these types of exits are avoided then the problem should be rescaled if possible or the method must be abandoned.

The choice of the 'floor' values PF(i), for $i = 1,2,...,m$, may be critical in the convergence of the Newton iteration. For each value i, the initial choice of p_i and the choice of PF(i) should not both be very small unless it is expected that the final parameter p_i will be very small and that it should be determined accurately in a **relative** sense.

For many problems it is critical that a good initial estimate be found for the parameters p or the iteration will not converge or may even break down with an error exit. There are many mathematical techniques which obtain good initial estimates for p in simple cases but which may fail to produce useful estimates in harder cases. If no such technique is available it is recommended that the user try a continuation (homotopy) technique preferably based on a physical parameter (e.g. the Reynolds or Prandtl number is often a suitable continuation parameter). In a continuation method a sequence of problems is solved, one for each choice of the continuation parameter, starting with the problem of interest. At each stage the parameters p calculated at earlier stages are used to compute a good initial estimate for the parameters at the current stage (see Hall and Watt [1] for more details).

9. Example

The following example program is intended to illustrate the use of the break-point and equation solving facilities of D02SAF. Most of the facilities which are common to D02SAF and D02HBF are illustrated in the example in the specification of D02HBF (which should also be consulted).

The program solves a projectile problem in two media determining the position of change of media, p_3, and the gravity and viscosity in the second medium (p_2 represents gravity and p_4 represents viscosity).

9.1. 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.

```
*     D02SAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, M, NPOINT, IWP, NMMAX, IW1, IW2, N1
      PARAMETER        (N=3,M=4,NPOINT=3,IWP=NPOINT,NMMAX=M,IW1=NMMAX,
     +                 IW2=3*M+23,N1=N)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
```

```
*     .. Scalars in Common ..
      real              XEND
      INTEGER           ICAP
*     .. Local Scalars ..
      real              YMAX
      INTEGER           I, ICOUNT, IFAIL, J
*     .. Local Arrays ..
      real              DP(M), E(N), P(M), PE(M), PF(M), W(IW1,IW2),
     +                  WP(IWP,6)
*     .. External Functions ..
      LOGICAL           CONSTR
      EXTERNAL          CONSTR
*     .. External Subroutines ..
      EXTERNAL          BC, D02SAF, D02SAS, EQN, FCN, PRSOL, RANGE,
     +                  X04ABF
*     .. Common blocks ..
      COMMON            /END/XEND, ICAP
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02SAF Example Program Results'
      ICAP = 0
      ICOUNT = 0
      YMAX = 0.0e0
      XEND = 5.0e0
      DO 20 I = 1, M
         PE(I) = 1.0e-3
         PF(I) = 1.0e-6
         DP(I) = 0.0e0
   20 CONTINUE
      DO 40 I = 1, N
         E(I) = 1.0e-5
   40 CONTINUE
      CALL X04ABF(1,NOUT)
      DO 80 I = 1, NPOINT - 1
         DO 60 J = 1, 3
            WP(I,J) = 0.0e0
   60    CONTINUE
   80 CONTINUE
      P(1) = 1.2e0
      P(2) = 0.032e0
      P(3) = 2.5e0
      P(4) = 0.02e0
      IFAIL = 1
*
*     * To obtain monitoring information, replace the name D02SAS
*     by D02HBX in the next statement and declare D02HBX as external *
*
      CALL D02SAF(P,M,N,N1,PE,PF,E,DP,NPOINT,WP,IWP,ICOUNT,RANGE,BC,FCN,
     +            EQN,CONSTR,YMAX,D02SAS,PRSOL,W,IW1,IW2,IFAIL)
*
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,99999) 'IFAIL = ', IFAIL
         IF (IFAIL.GE.4) THEN
            IF (IFAIL.LE.12) WRITE (NOUT,99998) 'WP(NPOINT,1) = ',
     +          WP(NPOINT,1)
            IF (IFAIL.LE.6) THEN
               WRITE (NOUT,99998) 'WP(1,5) = ', WP(1,5)
               WRITE (NOUT,99997) 'W(.,1) ', (W(I,1),I=1,N)
            END IF
         END IF
      END IF
      STOP
*
99999 FORMAT (1X,A,I3)
99998 FORMAT (1X,A,F10.4)
99997 FORMAT (1X,A,10e10.3)
      END
*
```

```
      SUBROUTINE EQN(F,Q,P,M)
*     .. Scalar Arguments ..
      INTEGER        M, Q
*     .. Array Arguments ..
      real           F(Q), P(M)
*     .. Executable Statements ..
      F(1) = 0.02e0 - P(4) - 1.0e-5*P(3)
      RETURN
      END
*
      SUBROUTINE FCN(X,Y,F,N,P,M,I)
*     .. Scalar Arguments ..
      real           X
      INTEGER        I, M, N
*     .. Array Arguments ..
      real           F(N), P(M), Y(N)
*     .. Intrinsic Functions ..
      INTRINSIC      COS, TAN
*     .. Executable Statements ..
      F(1) = TAN(Y(3))
      IF (I.EQ.1) THEN
         F(2) = -0.032e0*TAN(Y(3))/Y(2) - 0.02e0*Y(2)/COS(Y(3))
         F(3) = -0.032e0/Y(2)**2
      ELSE
         F(2) = -P(2)*TAN(Y(3))/Y(2) - P(4)*Y(2)/COS(Y(3))
         F(3) = -P(2)/Y(2)**2
      END IF
      RETURN
      END
*
      SUBROUTINE BC(F,G,P,M,N)
*     .. Scalar Arguments ..
      INTEGER        M, N
*     .. Array Arguments ..
      real           F(N), G(N), P(M)
*     .. Executable Statements ..
      F(1) = 0.0e0
      F(2) = 0.5e0
      F(3) = P(1)
      G(1) = 0.0e0
      G(2) = 0.45e0
      G(3) = -1.2e0
      RETURN
      END
*
      SUBROUTINE RANGE(X,NPOINT,P,M)
*     .. Scalar Arguments ..
      INTEGER          M, NPOINT
*     .. Array Arguments ..
      real             P(M), X(NPOINT)
*     .. Scalars in Common ..
      real             XEND
      INTEGER          ICAP
*     .. Common blocks ..
      COMMON           /END/XEND, ICAP
*     .. Executable Statements ..
      X(1) = 0.0e0
      X(2) = P(3)
      X(3) = XEND
      RETURN
      END
*
```

```
      SUBROUTINE PRSOL(X,Y,N)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              X
      INTEGER           N
*     .. Array Arguments ..
      real              Y(N)
*     .. Scalars in Common ..
      real              XEND
      INTEGER           ICAP
*     .. Local Scalars ..
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         ABS
*     .. Common blocks ..
      COMMON            /END/XEND, ICAP
*     .. Executable Statements ..
      IF (ICAP.NE.1) THEN
         ICAP = 1
         WRITE (NOUT,*)
         WRITE (NOUT,*) '      X        Y(1)       Y(2)       Y(3)'
      END IF
      WRITE (NOUT,99999) X, (Y(I),I=1,N)
      X = X + 0.5e0
      IF (ABS(X-XEND).LT.0.25e0) X = XEND
      RETURN
*
99999 FORMAT (1X,F9.3,3F10.4)
      END
*
      LOGICAL FUNCTION CONSTR(P,M)
*     .. Scalar Arguments ..
      INTEGER                  M
*     .. Array Arguments ..
      real                     P(M)
*     .. Local Scalars ..
      INTEGER                  I
*     .. Executable Statements ..
      CONSTR = .TRUE.
      DO 20 I = 1, M
         IF (P(I).LT.0.0e0) CONSTR = .FALSE.
   20 CONTINUE
      IF (P(3).GT.5.0e0) CONSTR = .FALSE.
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02SAF Example Program Results

      X        Y(1)       Y(2)       Y(3)
    0.000     0.0000     0.5000     1.1753
    0.500     1.0881     0.4127     1.0977
    1.000     1.9501     0.3310     0.9802
    1.500     2.5768     0.2582     0.7918
    2.000     2.9606     0.2019     0.4796
    2.500     3.0958     0.1773     0.0245
    3.000     2.9861     0.1935    -0.4353
    3.500     2.6289     0.2409    -0.7679
    4.000     2.0181     0.3047    -0.9767
    4.500     1.1454     0.3759    -1.1099
    5.000     0.0000     0.4500    -1.2000
```

With D02HBX used instead of D02SAS as an argument to D02SAF in the example program, intermediate results similar to those below are obtained:

```
D02SAF MONITORING INFORMATION
 INITIAL PARAMETERS ARE
   1.200000E+00  3.200000E-02  2.500000E+00  2.000000E-02
 INITIAL NORM =   4.006420E+05
 INITIAL RESIDUALS ARE
  -9.521764E-01  6.328063E-02 -7.293026E-02 -2.500000E-05
 SINGULAR VALUES ARE
   1.271777E+02  2.783856E+00  9.940481E-01  6.357504E-06
   .
   .
```

intermediate results omitted

```
   .
   .
D02SAF MONITORING INFORMATION
 STEP WITH ISTATE =  1  AND IFLAG =  2
 CURRENT PARAMETERS ARE
   1.175331E+00  3.045430E-02  2.330241E+00  1.997670E-02
 BASIC NORM =   3.528809E-08  CURRENT NORM =   1.013181E-09
 CURRENT RESIDUALS ARE
  -7.859726E-06  3.327722E-07 -3.676384E-07 -3.896352E-19
 SINGULAR VALUES ARE
   1.155687E+02  2.019313E+00  9.563721E-01  3.445298E-03
 SINGULAR VALUE PERTURBATION FACTOR =   0.0000E+00
```

D02TGF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02TGF solves a system of linear ordinary differential equations by least-squares fitting of a series of Chebyshev polynomials using collocation.

2. Specification

```
      SUBROUTINE D02TGF (N, M, L, X0, X1, K1, KP, C, IC, COEFF, BDYC, W,
     1                   LW, IW, LIW, IFAIL)
      INTEGER            N, M(N), L(N), K1, KP, IC, LW, IW(LIW), LIW,
     1                   IFAIL
      real               X0, X1, C(IC,N), W(LW)
      EXTERNAL           COEFF, BDYC
```

3. Description

The routine calculates an approximate solution of a linear or linearised system of ordinary differential equations as a Chebyshev-series. Suppose there are n differential equations for n variables $y_1,y_2,\ldots,y_n$, over the range (x_0,x_1). Let the ith equation be

$$\sum_{j=1}^{m_i+1} \sum_{k=1}^{n} f_{kj}^i(x) y_k^{(j-1)}(x) = r^i(x)$$

where $y_k^{(j)}(x) = \dfrac{d^j y_k(x)}{dx^j}$. The routine COEFF provided by the user evaluates the coefficients $f_{kj}^i(x)$ and the right-hand side $r^i(x)$ for each i, $1 \leq i \leq n$, at any point x. The boundary conditions may be applied either at the end-points or at intermediate points; they are written in the same form as the differential equations, and specified by the routine BDYC. For example the jth boundary condition out of those associated with the ith differential equation takes the form

$$\sum_{j=1}^{l_i+1} \sum_{k=1}^{n} f_{kj}^{ij}(x^{ij}) y_k^{(j-1)}(x^{ij}) = r^{ij}(x^{ij}),$$

where x^{ij} lies between x_0 and x_1. It is assumed in this routine that certain of the boundary conditions are associated with each differential equation. This is for the user's convenience; the grouping does not affect the results.

The degree of the polynomial solution must be the same for all variables. The user specifies the degree required, k_1-1, and the number of collocation points, k_p, in the range. The routine sets up a system of linear equations for the Chebyshev coefficients, with n equations for each collocation point and one for each boundary condition. The collocation points are chosen at the extrema of a shifted Chebyshev polynomial of degree k_p-1. The boundary conditions are satisfied exactly, and the remaining equations are solved by a least-squares method. The result produced is a set of Chebyshev coefficients for the n functions $y_1,y_2,\ldots,y_n$, with the range normalised to $[-1,1]$.

E02AKF can be used to evaluate the components of the solution at any point on the range $[x_0,x_1]$ (see Section 9 for an example). E02AHF and E02AJF may be used to obtain Chebyshev-series representations of derivatives and integrals (respectively) of the components of the solution.

4. References

[1] PICKEN, S.M.
Algorithms for the Solution of Differential Equations in Chebyshev-series by the Selected Points Method.
Report Math., 94, National Physical Laboratory, Teddington, 1970.

5. Parameters

1: N – INTEGER. *Input*

On entry: the number of differential equations in the system, n.

Constraint: N $\geq$ 1.

2: M(N) – INTEGER array. *Input*

On entry: M(i) must be set to the highest order derivative occurring in the ith equation, for $i = 1,2,...,$N.

Constraint: M(i) $\geq$ 1, for $i = 1,2,...,n$.

3: L(N) – INTEGER array. *Input*

On entry: L(i) must be set to the number of boundary conditions associated with the i(th) equation, for $i = 1,2,...,n$.

Constraint: L(i) $\geq$ 0, for $i = 1,2,...,n$.

4: X0 – ***real***. *Input*

On entry: the left-hand boundary, x_0.

5: X1 – ***real***. *Input*

On entry: the right-hand boundary, x_1.

Constraint: X1 > X0.

6: K1 – INTEGER. *Input*

On entry: the number of coefficients, k_1, to be returned in the Chebyshev-series representation of the solution (hence, the degree of the polynomial approximation is K1−1).

Constraint: $\mathrm{K1} \geq 1 + \max_{1 \leq i \leq \mathrm{N}} \mathrm{M}(i)$.

7: KP – INTEGER. *Input*

On entry: the number of collocation points to be used, k_p.

Constraint: $\mathrm{N}\times\mathrm{KP} \geq \mathrm{N}\times\mathrm{K1} + \sum_{i=1}^{N} \mathrm{L}(i)$.

8: C(IC,N) – ***real*** array. *Output*

On exit: the kth column of C contains the computed Chebyshev coefficients of the kth component of the solution, y_k; that is, the computed solution is:

$$y_k = {\sum_{i=1}^{k_1}}' \mathrm{C}(i,k) T_{i-1}(x), \quad 1 \leq k \leq n,$$

where $T_i(x)$ is the Chebyshev polynomial of the first kind and Σ' denotes that the first coefficient, C(1,k), is halved.

9: IC – INTEGER. *Input*

On entry: the first dimension of the array C as declared in the (sub)program from which D02TGF is called.

Constraint: IC $\geq$ K1.

10: COEFF – SUBROUTINE, supplied by the user. *External Procedure*

COEFF defines the system of differential equations (see Section 3). It must evaluate the coefficient functions $f^i_{kj}(x)$ and the right-hand side function $r^i(x)$ of the i(th) equation at a given point. Only non-zero entries of the array A and RHS need be specifically assigned, since all elements are set to zero by D02TGF before calling COEFF.

Its specification is:

```
SUBROUTINE  COEFF(X, I, A, IA, IA1, RHS)
INTEGER     I, IA, IA1
real        X, A(IA,IA1), RHS
```

Important: the dimension declaration for A must contain the variable IA, not an integer constant.

1: X – ***real***. *Input*

On entry: the point x at which the functions must be evaluated.

2: I – INTEGER. *Input*

On entry: the equation for which the coefficients and right-hand side are to be evaluated.

3: A(IA,IA1) – ***real*** array. *Input/Output*

On entry: all elements of A are set to zero.

On exit: A(k,j) must contain the value $f^i_{kj}(x)$, for $1 \leq k \leq n$, $1 \leq j \leq m_i+1$.

4: IA – INTEGER. *Input*

5: IA1 – INTEGER. *Input*

On entry: the first and second dimensions of A, respectively.

6: RHS – ***real***. *Input/Output*

On entry: RHS is set to zero.

On exit: it must contain the value $r^i(x)$.

COEFF must be declared as EXTERNAL in the (sub)program from which D02TGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

11: BDYC – SUBROUTINE, supplied by the user. *External Procedure*

BDYC defines the boundary conditions (see Section 3). It must evaluate the coefficient functions f^{ij}_{kj} and right-hand side function r^{ij} in the jth boundary condition associated with the ith equation, at the point x^{ij} at which the boundary condition is applied. Only non-zero entries of the array A and RHS need be specifically assigned, since all elements are set to zero by D02TGF before calling BDYC.

Its specification is:

```
SUBROUTINE  BDYC(X, I, J, A, IA, IA1, RHS)
INTEGER     I, J, IA, IA1
real        X, A(IA,IA1), RHS
```

Important: the dimension declaration for A must contain the variable IA, not an integer constant.

1: X – ***real***. *Output*

On exit: the value x^{ij} at which the boundary condition is applied.

2: I – INTEGER. *Input*

On entry: the differential equation with which the condition is associated.

3: J – INTEGER. *Input*

On entry: the boundary condition for which the coefficients and right-hand side are to be evaluated.

4: A(IA,IA1) – ***real*** array. *Input/Output*

On entry: all elements of A are set to zero.

On exit: the value $f_{kj}^{ij}(x^{ij})$ for $1 \le k \le n$, $1 \le j \le m_i+1$.

5: IA – INTEGER. *Input*

6: IA1 – INTEGER. *Input*

On entry: the first and second dimensions of A, respectively.

7: RHS – ***real***. *Input/Output*

On entry: RHS is set to zero.

On exit: the value $r^{ij}(x^{ij})$.

BDYC must be declared as EXTERNAL in the (sub)program from which D02TGF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

12: W(LW) – ***real*** array. *Workspace*

13: LW – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D02TGF is called.

Constraint: LW $\ge$ 2×(N×KP+NL)×(N×K1+1) + 7×N×K1, where NL = $\sum_{i=1}^{n} \mathrm{L}(i)$.

14: IW(LIW) – INTEGER array. *Workspace*

15: LIW – INTEGER. *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D02TGF is called.

Constraint: LIW $\ge$ N×K1 + 1.

16: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, N < 1,
or M(i) < 1 for some i,
or L(i) < 0 for some i,
or X0 $\ge$ X1,
or K1 < 1 + M(i) for some i,
or N×KP < N×K1 + $\sum_{i=1}^{n} \mathrm{L}(i)$,
or IC < K1.

IFAIL = 2

On entry, LW is too small (see Section 5),
or LIW < N×K1.

IFAIL = 3

Either the boundary conditions are not linearly independent, or the rank of the matrix of equations for the coefficients is less than the number of unknowns. Increasing KP may overcome this latter problem.

IFAIL = 4

The least-squares routine F04AMF has failed to correct the first approximate solution (see NAG Fortran Library document F04AMF). Increasing KP may remove this difficulty.

7. Accuracy

Estimates of the accuracy of the solution may be obtained by using the checks described in Section 8. The Chebyshev coefficients are calculated by a stable numerical method.

8. Further Comments

The time taken by the routine depends on the complexity of the system of differential equations, the degree of the polynomial solution and the number of matching points.

If the number of matching points k_p is equal to the number of coefficients k_1 minus the average number of boundary conditions $\frac{1}{n}\sum_{i=1}^{n} l_i$, then the least-squares solution reduces to simple solution of linear equations and true collocation results. The accuracy of the solution may be checked by repeating the calculation with different values of k_1. If the Chebyshev coefficients decrease rapidly, the size of the last two or three gives an indication of the error. If they do not decrease rapidly, it may be desirable to use a different method. Note that the Chebyshev coefficients are calculated for the range normalised to $[-1,1]$.

Generally the number of boundary conditions required is equal to the sum of the orders of the n differential equations. However, in some cases fewer boundary conditions are needed, because the assumption of a polynomial solution is equivalent to one or more boundary conditions (since it excludes singular solutions).

A system of **nonlinear** differential equations must be linearised before using the routine. The calculation is repeated iteratively. On each iteration the linearised equation is used. In the example in Section 9, the y variables are to be determined at the current iteration whilst the z variables correspond to the solution determined at the previous iteration, (or the initial approximation on the first iteration). For a starting approximation, we may take, say, a linear function, and set up the appropriate Chebyshev coefficients before starting the iteration. For example, if $y_1 = ax + b$ in the range (x_0,x_1), we set B, the array of coefficients,

$\mathrm{B}(1,1) = a\times(x_0+x_1) + 2\times b,$
$\mathrm{B}(1,2) = a\times(x_1-x_0)/2,$
and the remainder of the entries to zero.

In some cases a better initial approximation may be needed and can be obtained by using E02ADF or E02AFF to obtain a Chebyshev-series for an approximate solution. The coefficients of the current iterate must be communicated to COEFF and BDYC, e.g. in COMMON. (See the example in Section 9). The convergence of the (Newton) iteration cannot be guaranteed in general, though it is usually satisfactory from a good starting approximation.

9. Example

To solve the nonlinear system

$$2y_1' + (y_2^2-1)y_1 + y_2 = 0,$$
$$2y_2'' - y_1' = 0,$$

in the range $(-1,1)$, with $y_1 = 0$, $y_2 = 3$, $y_2' = 0$ at $x = -1$.

Suppose an approximate solution is z_1, z_2 such that $y_1 \sim z_1$, $y_2 \sim z_2$: then the first equation gives, on linearising,

$$2y_1' + (z_2^2-1)y_1 + (2z_1z_2+1)y_2 = 2z_1z_2^2.$$

The starting approximation is taken to be $z_1 = 0$, $z_2 = 3$. In the program below, the array B is used to hold the coefficients of the previous iterate (or of the starting approximation). We iterate until the Chebyshev coefficients converge to 5 figures. E02AKF is used to calculate the solution from its Chebyshev coefficients.

9.1. 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.

```
*     D02TGF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, MIMAX, K1, IC, KP, LSUM, LW, LIW
      PARAMETER        (N=2,MIMAX=8,K1=MIMAX+1,IC=K1,KP=15,LSUM=3,
     +                 LW=2*(N*KP+LSUM)*(N*K1+1)+7*N*K1,LIW=N*K1)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      real             X0, X1
*     .. Arrays in Common ..
      real             B(K1,N)
*     .. Local Scalars ..
      real             EMAX, X
      INTEGER          I, IA1, IFAIL, ITER, J, K
*     .. Local Arrays ..
      real             C(IC,N), W(LW), Y(N)
      INTEGER          IW(LIW), L(N), M(N)
*     .. External Subroutines ..
      EXTERNAL         BDYC, COEFF, D02TGF, E02AKF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, MAX, real
*     .. Common blocks ..
      COMMON           /ABC/B, X0, X1
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02TGF Example Program Results'
      X0 = -1.0e0
      X1 = 1.0e0
      M(1) = 1
      M(2) = 2
      L(1) = 1
      L(2) = 2
      DO 40 J = 1, N
         DO 20 I = 1, K1
            B(I,J) = 0.0e0
   20    CONTINUE
   40 CONTINUE
      B(1,2) = 6.0e0
      ITER = 0
   60 ITER = ITER + 1
      WRITE (NOUT,*)
      WRITE (NOUT,99999) ' Iteration', ITER,
     +  ' Chebyshev coefficients are'
      IFAIL = 1
*
      CALL D02TGF(N,M,L,X0,X1,K1,KP,C,IC,COEFF,BDYC,W,LW,IW,LIW,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         DO 80 J = 1, N
            WRITE (NOUT,99998) (C(I,J),I=1,K1)
   80    CONTINUE
         EMAX = 0.0e0
         DO 120 J = 1, N
            DO 100 I = 1, K1
               EMAX = MAX(EMAX,ABS(C(I,J)-B(I,J)))
               B(I,J) = C(I,J)
  100       CONTINUE
  120    CONTINUE
         IF (EMAX.LT.1.0e-5) THEN
            K = 9
            IA1 = 1
            WRITE (NOUT,*)
            WRITE (NOUT,99999) 'Solution evaluated at', K,
     +        '  equally spaced points'
```

```
            WRITE (NOUT,*)
            WRITE (NOUT,99997) '       X ', (J,J=1,N)
            DO 160 I = 1, K
               X = (X0*real(K-I)+X1*real(I-1))/real(K-1)
               DO 140 J = 1, N
                  IFAIL = 0
*
                  CALL E02AKF(K1,X0,X1,C(1,J),IA1,K1,X,Y(J),IFAIL)
*
  140          CONTINUE
               WRITE (NOUT,99996) X, (Y(J),J=1,N)
  160       CONTINUE
         ELSE
            GO TO 60
         END IF
      ELSE
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'D02TGF fails with IFAIL =', IFAIL
      END IF
      STOP
*
99999 FORMAT (1X,A,I3,A)
99998 FORMAT (1X,9F8.4)
99997 FORMAT (1X,A,2('      Y(',I1,')'))
99996 FORMAT (1X,3F10.4)
      END
*
      SUBROUTINE COEFF(X,I,A,IA,IA1,RHS)
*     .. Parameters ..
      INTEGER          N, MIMAX, K1
      PARAMETER        (N=2,MIMAX=8,K1=MIMAX+1)
*     .. Scalar Arguments ..
      real             RHS, X
      INTEGER          I, IA, IA1
*     .. Array Arguments ..
      real             A(IA,IA1)
*     .. Scalars in Common ..
      real             X0, X1
*     .. Arrays in Common ..
      real             B(K1,N)
*     .. Local Scalars ..
      real             Z1, Z2
      INTEGER          IFAIL
*     .. External Subroutines ..
      EXTERNAL         E02AKF
*     .. Common blocks ..
      COMMON           /ABC/B, X0, X1
*     .. Executable Statements ..
      IF (I.LE.1) THEN
         IA1 = 1
         IFAIL = 0
*
         CALL E02AKF(K1,X0,X1,B(1,1),IA1,K1,X,Z1,IFAIL)
         CALL E02AKF(K1,X0,X1,B(1,2),IA1,K1,X,Z2,IFAIL)
*
         A(1,1) = Z2*Z2 - 1.0e0
         A(1,2) = 2.0e0
         A(2,1) = 2.0e0*Z1*Z2 + 1.0e0
         RHS = 2.0e0*Z1*Z2*Z2
      ELSE
         A(1,2) = -1.0e0
         A(2,3) = 2.0e0
      END IF
      RETURN
      END
*
      SUBROUTINE BDYC(X,I,J,A,IA,IA1,RHS)
*     .. Scalar Arguments ..
      real             RHS, X
      INTEGER          I, IA, IA1, J
```

```
*     .. Array Arguments ..
      real              A(IA,IA1)
*     .. Executable Statements ..
      X = -1.0e0
      A(I,J) = 1.0e0
      IF (I.EQ.2 .AND. J.EQ.1) RHS = 3.0e0
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D02TGF Example Program Results

  Iteration  1 Chebyshev coefficients are
  -0.5659 -0.1162  0.0906 -0.0468  0.0196 -0.0069  0.0021 -0.0006  0.0001
   5.7083 -0.1642 -0.0087  0.0059 -0.0025  0.0009 -0.0003  0.0001  0.0000

  Iteration  2 Chebyshev coefficients are
  -0.6338 -0.1599  0.0831 -0.0445  0.0193 -0.0071  0.0023 -0.0006  0.0001
   5.6881 -0.1792 -0.0144  0.0053 -0.0023  0.0008 -0.0003  0.0001  0.0000

  Iteration  3 Chebyshev coefficients are
  -0.6344 -0.1604  0.0828 -0.0446  0.0193 -0.0071  0.0023 -0.0006  0.0001
   5.6880 -0.1793 -0.0145  0.0053 -0.0023  0.0008 -0.0003  0.0001  0.0000

  Iteration  4 Chebyshev coefficients are
  -0.6344 -0.1604  0.0828 -0.0446  0.0193 -0.0071  0.0023 -0.0006  0.0001
   5.6880 -0.1793 -0.0145  0.0053 -0.0023  0.0008 -0.0003  0.0001  0.0000

 Solution evaluated at  9  equally spaced points

       X        Y(1)      Y(2)
    -1.0000    0.0000    3.0000
    -0.7500   -0.2372    2.9827
    -0.5000   -0.3266    2.9466
    -0.2500   -0.3640    2.9032
     0.0000   -0.3828    2.8564
     0.2500   -0.3951    2.8077
     0.5000   -0.4055    2.7577
     0.7500   -0.4154    2.7064
     1.0000   -0.4255    2.6538
```

D02TKF – NAG Fortran Library Routine Document

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

D02TKF solves a general two point boundary value problem for a nonlinear mixed order system of ordinary differential equations.

2 Specification

```
      SUBROUTINE D02TKF(FFUN, FJAC, GAFUN, GBFUN, GAJAC, GBJAC, GUESS,
     1                  WORK, IWORK, IFAIL)
      INTEGER           IWORK(*), IFAIL
      real              WORK(*)
      EXTERNAL          FFUN, FJAC, GAFUN, GBFUN, GAJAC, GBJAC, GUESS
```

3 Description

D02TKF and its associated routines (D02TVF, D02TXF, D02TYF and D02TZF) solve the two point boundary value problem for a nonlinear mixed order system of ordinary differential equations

$$\begin{array}{rcl} y_1^{(m_1)}(x) & = & f_1(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\ y_2^{(m_2)}(x) & = & f_2(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\ & \cdots & \\ y_n^{(m_n)}(x) & = & f_n(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \end{array}$$

over an interval $[a, b]$ subject to p (> 0) nonlinear boundary conditions at a and q (> 0) nonlinear boundary conditions at b, where $p + q = \sum_1^n m_i$. Note that $y_i^{(m)}(x)$ is the m-th derivative of the i-th solution component. Hence $y_i^{(0)}(x) = y_i(x)$. The left boundary conditions at a are defined as

$$g_i(z(y(a))) = 0, \quad i = 1, 2, \ldots, p,$$

and the right boundary conditions at b as

$$\bar{g}_j(z(y(b))) = 0, \quad j = 1, 2, \ldots, q,$$

where $y = (y_1, y_2, \ldots, y_n)$ and

$$z(y(x)) = (y_1(x), y_1^{(1)}(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)).$$

First, D02TVF must be called to specify the initial mesh, error requirements and other details. Note that the error requirements apply only to the solution components $y_1, y_2, \ldots, y_n$ and that no error control is applied to derivatives of solution components. (If error control is required on derivatives then the system must be reduced in order by introducing the derivatives whose error is to be controlled as new variables. See Section 8 of the document for D02TVF.) Then, D02TKF can be used to solve the boundary value problem. After successful computation, D02TZF can be used to ascertain details about the final mesh and other details of the solution procedure, and D02TYF can be used to compute the approximate solution anywhere on the interval $[a, b]$.

A description of the numerical technique used in D02TKF is given in Section 3 of the document for D02TVF.

D02TKF can also be used in the solution of a series of problems, for example in performing continuation, when the mesh used to compute the solution of one problem is to be used as the initial mesh for the solution of the next related problem. D02TXF should be used in between calls to D02TKF in this context.

See Section 8 of the document for D02TVF for details of how to solve boundary value problems of a more general nature.

The routines are based on modified versions of the codes COLSYS and COLNEW, [2] and [1] . A comprehensive treatment of the numerical solution of boundary value problems can be found in [3] and [4].

4 References

[1] Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

[2] Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

[3] Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice Hall, Englewood Cliffs, NJ

[4] Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

5 Parameters

1: FFUN — SUBROUTINE, supplied by the user. *External Procedure*

FFUN must evaluate the functions f_i for given values $x, z(y(x))$.

Its specification is:

```
SUBROUTINE FFUN(X, Y, NEQ, M, F)
INTEGER        NEQ, M(NEQ)
real           X, Y(NEQ,0:*), F(NEQ)
```

1: X — ***real*** *Input*

On entry: the independent variable, x.

2: Y(NEQ,0:*) — ***real*** array *Input*

On entry: Y(i,j) contains $y_i^{(j)}(x)$, for $i = 1,2,\ldots,$NEQ, $j = 0,1,\ldots,$M$(i) - 1$.

Note: $y_i^{(0)}(x) = y_i(x)$.

3: NEQ — INTEGER *Input*

On entry: the number of differential equations.

4: M(NEQ) — INTEGER array *Input*

On entry: the order, m_i, of the i-th differential equation, for $i = 1,2,\ldots,$NEQ.

5: F(NEQ) — ***real*** array *Output*

On exit: the values of f_i, for $i = 1,2,\ldots,$NEQ.

FFUN must be declared as EXTERNAL in the (sub)program from which D02TKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

2: FJAC — SUBROUTINE, supplied by the user. *External Procedure*

FJAC must evaluate the partial derivatives of f_i with respect to the elements of $z(y(x)) \; (= (y_1(x), y_1^1(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)))$.

Its specification is:

```
      SUBROUTINE FJAC(X, Y, NEQ, M, DFDY)
      INTEGER          NEQ, M(NEQ)
      real             X, Y(NEQ,0:*), DFDY(NEQ,NEQ,0:*)
```

1: X — ***real*** *Input*

On entry: the independent variable, x.

2: Y(NEQ,0:*) — ***real*** array *Input*

On entry: Y(i,j) contains $y_i^{(j)}(x)$, for $i = 1,2,\ldots,$NEQ, $j = 0,1,\ldots,$M(i) $- 1$.

Note: $y_i^{(0)}(x) = y_i(x)$.

3: NEQ — INTEGER *Input*

On entry: the number of differential equations.

4: M(NEQ) — INTEGER array *Input*

On entry: the order, m_i, of the i-th differential equation, for $i = 1,2,\ldots,$NEQ.

5: DFDY(NEQ,NEQ,0:*) — ***real*** array *Output*

On exit: DFDY(i,j,k) must contain the partial derivative of f_i with respect to $y_j^{(k)}$, for $i,j = 1,2,\ldots,$NEQ, $k = 0,1,\ldots,$M(j) $- 1$. Only non-zero partial derivatives need be set.

FJAC must be declared as EXTERNAL in the (sub)program from which D02TKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

3: GAFUN — SUBROUTINE, supplied by the user. *External Procedure*

GAFUN must evaluate the boundary conditions at the left hand end of the range, that is functions $g_i(z(y(a)))$ for given values of $z(y(a))$.

Its specification is:

```
      SUBROUTINE GAFUN(YA, NEQ, M, NLBC, GA)
      INTEGER          NEQ, M(NEQ), NLBC
      real             YA(NEQ,0:*), GA(NLBC)
```

1: YA(NEQ,0:*) — ***real*** array *Input*

On entry: YA(i,j) contains $y_i^{(j)}(a)$, for $i = 1,2,\ldots,$NEQ, $j = 0,1,\ldots,$M(i) $- 1$.

Note: $y_i^{(0)}(a) = y_i(a)$.

2: NEQ — INTEGER *Input*

On entry: the number of differential equations.

3: M(NEQ) — INTEGER array *Input*

On entry: the order, m_i, of the i-th differential equation, for $i = 1,2,\ldots,$NEQ.

4: NLBC — INTEGER *Input*

On entry: the number of boundary conditions at a.

5: GA(NLBC) — ***real*** array *Output*

On exit: the values of $g_i(z(y(a)))$, for $i = 1,2,\ldots,$NLBC.

GAFUN must be declared as EXTERNAL in the (sub)program from which D02TKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

4: GBFUN — SUBROUTINE, supplied by the user. *External Procedure*

GBFUN must evaluate the boundary conditions at the right hand end of the range, that is functions $\bar{g}_i(z(y(b)))$ for given values of $z(y(b))$.

Its specification is:

```
SUBROUTINE GBFUN(YB, NEQ, M, NRBC, GB)
INTEGER          NEQ, M(NEQ), NRBC
real             YB(NEQ,0:*), GB(NRBC)
```

1: YB(NEQ,0:*) — ***real*** array *Input*

On entry: YB(i,j) contains $y_i^{(j)}(b)$, for $i = 1,2,\ldots,\mathrm{NEQ}$, $j = 0,1,\ldots,\mathrm{M}(i)-1$.

Note: $y_i^{(0)}(b) = y_i(b)$.

2: NEQ — INTEGER *Input*

On entry: the number of differential equations.

3: M(NEQ) — INTEGER array *Input*

On entry: the order, m_i, of the i-th differential equation, for $i = 1,2,\ldots,\mathrm{NEQ}$.

4: NRBC — INTEGER *Input*

On entry: the number of boundary conditions at b.

5: GB(NRBC) — ***real*** array *Output*

On exit: the values of $\bar{g}_i(z(y(b)))$, for $i = 1,2,\ldots,\mathrm{NRBC}$.

GBFUN must be declared as EXTERNAL in the (sub)program from which D02TKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: GAJAC — SUBROUTINE, supplied by the user. *External Procedure*

GAJAC must evaluate the partial derivatives of $g_i(z(y(a)))$ with respect to the elements of $z(y(a))$ $(= (y_1(a), y_1^1(a), \ldots, y_1^{(m_1-1)}(a), y_2(a), \ldots, y_n^{(m_n-1)}(a)))$.

Its specification is:

```
SUBROUTINE GAJAC(YA, NEQ, M, NLBC, DGADY)
INTEGER          NEQ, M(NEQ), NLBC
real             YA(NEQ,0:*), DGADY(NLBC,NEQ,0:*)
```

1: YA(NEQ,0:*) — ***real*** array *Input*

On entry: YA(i,j) contains $y_i^{(j)}(a)$, for $i = 1,2,\ldots,\mathrm{NEQ}$, $j = 0,1,\ldots,\mathrm{M}(i)-1$.

Note: $y_i^{(0)}(a) = y_i(a)$.

2: NEQ — INTEGER *Input*

On entry: the number of differential equations.

3: M(NEQ) — INTEGER array *Input*

On entry: the order, m_i, of the i-th differential equation, for $i = 1,2,\ldots,\mathrm{NEQ}$.

4: NLBC — INTEGER *Input*

On entry: the number of boundary conditions at a.

5: DGADY(NLBC,NEQ,0:*) — ***real*** array *Output*

On exit: DGADY(i,j,k) must contain the partial derivative of $g_i(z(y(a)))$ with respect to $y_j^{(k)}(a)$, for $i = 1,2,\ldots,\mathrm{NLBC}$, $j = 1,2,\ldots,\mathrm{NEQ}$, $k = 0,1,\ldots,\mathrm{M}(j)-1$. Only non-zero partial derivatives need be set.

GAJAC must be declared as EXTERNAL in the (sub)program from which D02TKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: GBJAC — SUBROUTINE, supplied by the user. *External Procedure*

GBJAC must evaluate the partial derivatives of $\bar{g}_i(z(y(b)))$ with respect to the elements of $z(y(b)) \; (= (y_1(b), y_1^1(b), \ldots, y_1^{(m_1-1)}(b), y_2(b), \ldots, y_n^{(m_n-1)}(b)))$.

Its specification is:

```
SUBROUTINE GBJAC(YB, NEQ, M, NRBC, DGBDY)
INTEGER          NEQ, M(NEQ), NRBC
real             YB(NEQ,0:*), DGBDY(NRBC,NEQ,0:*)
```

1: YB(NEQ,0:*) — ***real*** array *Input*

On entry: YB(i,j) contains $y_i^{(j)}(b)$, for $i = 1,2,\ldots,$NEQ, $j = 0,1,\ldots,$M$(i)-1$.

Note: $y_i^{(0)}(b) = y_i(b)$.

2: NEQ — INTEGER *Input*

On entry: the number of differential equations.

3: M(NEQ) — INTEGER array *Input*

On entry: the order, m_i, of the i-th differential equation, for $i = 1,2,\ldots,$NEQ.

4: NRBC — INTEGER *Input*

On entry: the number of boundary conditions at a.

5: DGBDY(NRBC,NEQ,0:*) — ***real*** array *Output*

On exit: DGBDY(i,j,k) must contain the partial derivative of $\bar{g}_i(z(y(b)))$ with respect to $y_j^{(k)}(b)$, for $i = 1,2,\ldots,$NRBC, $j = 1,2,\ldots,$NEQ, $k = 0,1,\ldots,$M$(j)-1$. Only non-zero partial derivatives need be set.

GBJAC must be declared as EXTERNAL in the (sub)program from which D02TKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: GUESS — SUBROUTINE, supplied by the user. *External Procedure*

GUESS must return initial approximations for the solution compontents $y_i^{(j)}$ and the derivatives $y_i^{(m_i)}$, for $i = 1,2,\ldots,$NEQ, $j = 0,1,\ldots,$M$(i)-1$. Try to compute each derivative $y_i^{(m_i)}$ such that it corresponds to your approximations to $y_i^{(j)}$ for $j = 0,1,\ldots,$M$(i)-1$. You should **not** call FFUN to compute $y_i^{(m_i)}$.

If D02TKF is being used in conjunction with D02TXF as part of a continuation process, then GUESS is not called by D02TKF after the call to D02TXF.

Its specification is:

```
SUBROUTINE GUESS(X, NEQ, M, Y, DYM)
INTEGER          M(NEQ), NEQ
real             X, Y(NEQ,0:*), DYM(NEQ)
```

1: X — ***real*** *Input*

On entry: the independent variable, x; $x \in [a,b]$.

2: NEQ — INTEGER *Input*

On entry: the number of differential equations.

3: M(NEQ) — INTEGER array *Input*

On entry: the order, m_i, of the i-th differential equation, for $i = 1,2,\ldots,$NEQ.

4: Y(NEQ,0:*) — ***real*** array *Output*

On exit: Y(i,j) must contain $y_i^{(j)}(x)$, for $i = 1,2,\ldots,$NEQ, $j = 0,1,\ldots,$M(i) − 1.
Note: $y_i^{(0)}(x) = y_i(x)$.

5: DYM(NEQ) — ***real*** array *Output*

On exit: DYM(i) must contain $y_i^{(m_i)}(x)$, for $i = 1,2,\ldots,$NEQ.

GUESS must be declared as EXTERNAL in the (sub)program from which D02TKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: WORK(*) — ***real*** array *Input/Output*

On entry: this must be the same array as supplied to D02TVF and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated routines.

9: IWORK(*) — INTEGER array *Input/Output*

On entry: this must be the same array as supplied to D02TVF and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated routines.

10: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit.**

6 Errors 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 detected by the routine:

IFAIL = 1

On entry, an invalid call was made to D02TKF, for example, without a previous call to the setup routine D02TVF. If on entry IFAIL = 0 or −1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF).

IFAIL = 2

Numerical singularity has been detected in the Jacobian used in the underlying Newton iteration. No meaningful results have been computed. You should check carefully how you have coded procedures FJAC, GAJAC and GBJAC. If the procedures have been coded correctly then supplying a different initial approximation to the solution in GUESS might be appropriate. See also Section 8.

IFAIL = 3

The nonlinear iteration has failed to converge. At no time during the computation was convergence obtained and no meaningful results have been computed. You should check carefully how you have coded procedures FJAC, GAJAC and GBJAC. If the procedures have been coded correctly then supplying a better initial approximation to the solution in GUESS might be appropriate. See also Section 8.

IFAIL = 4

The nonlinear iteration has failed to converge. At some earlier time during the computation convergence was obtained and the corresponding results have been returned for diagnostic purposes and may be inspected by a call to D02TZF. Nothing can be said regarding the suitability of these results for use in any subsequent computation for the same problem. You should try to provide a better mesh and initial approximation to the solution in GUESS. See also Section 8.

IFAIL = 5

The expected number of sub-intervals required exceeds the maximum number specified by the argument MXMESH in the setup routine D02TVF. Results for the last mesh on which convergence was obtained have been returned. Nothing can be said regarding the suitability of these results for use in any subsequent computation for the same problem. An indication of the error in the solution on the last mesh where convergence was obtained can be obtained by calling D02TZF. The error requirements may need to be relaxed and/or the maximum number of mesh points may need to be increased. See also Section 8.

7 Accuracy

The accuracy of the solution is determined by the parameter TOLS in the prior call to D02TVF (see Sections 3 and 8 of the document for D02TVF for details and advice). Note that error control is applied only to solution components (variables) and not to any derivatives of the solution. An estimate of the maximum error in the computed solution is available by calling D02TZF.

8 Further Comments

If D02TKF returns with IFAIL = 2, 3, 4 or 5 and the call to D02TKF was a part of some continuation procedure for which successful calls to D02TKF have already been made, then it is possible that the adjustment(s) to the continuation parameter(s) between calls to D02TKF is (are) too large for the problem under consideration. More conservative adjustment(s) to the continuation parameter(s) might be appropriate.

9 Example

The following example is used to illustrate the treatment of a high order system, control of the error in a derivative of a component of the original system, and the use of continuation. See also D02TVF, D02TXF, D02TYF and D02TZF, for the illustration of other facilities.

Consider the steady flow of an incompressible viscous fluid between two infinite coaxial rotating discs. See [2] and the references therein. The governing equations are

$$\begin{aligned} \frac{1}{\sqrt{R}}f'''' + ff''' + gg' &= 0 \\ \frac{1}{\sqrt{R}}g'' + fg' - f'g &= 0 \end{aligned}$$

subject to the boundary conditions

$$f(0) = f'(0) = 0, \quad g(0) = \Omega_0, \quad f(1) = f'(1) = 0, \quad g(1) = \Omega_1,$$

where R is the Reynolds number and Ω_0, Ω_1 are the angular velocities of the disks.

We consider the case of counter-rotation and a symmetric solution, that is $\Omega_0 = 1, \Omega_1 = -1$. This problem is more difficult to solve, the larger the value of R. For illustration, we use simple continuation to compute the solution for three different values of R $(= 10^6, 10^8, 10^{10})$. However, this problem can be addressed directly for the largest value of R considered here. Instead of the values suggested in Section 5 of the document for D02TXF for NMESH, IPMESH and MESH in the call to D02TXF prior to a continuation call, we use every point of the final mesh for the solution of the first value of R, that is we must modify the contents of IPMESH. For illustrative purposes we wish to control the computed error in f' and so recast the equations as

$$\begin{aligned} y_1' &= y_2 \\ y_2''' &= -\sqrt{R}(y_1 y_2'' + y_3 y_3') \\ y_3'' &= \sqrt{R}(y_2 y_3 - y_1 y_3') \end{aligned}$$

subject to the boundary conditions

$$y_1(0) = y_2(0) = 0,\ \ y_3(0) = \Omega,\ \ y_1(1) = y_2(1) = 0,\ \ y_3(1) = -\Omega,\ \ \Omega = 1.$$

For the symmetric boundary conditions considered, there exists an odd solution about $x = 0.5$. Hence, to satisfy the boundary conditions, we use the following initial approximations to the solution in GUESS:

$$\begin{aligned} y_1(x) &= -x^2(x - \frac{1}{2})(x-1)^2 \\ y_2(x) &= -x(x-1)(5x^2 - 5x + 1) \\ y_3(x) &= -8\Omega(x - \frac{1}{2})^3. \end{aligned}$$

9.1 Example 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.

```
*     D02TKF Example Program Text
*     Mark 17 Release. NAG Copyright 1995.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, MMAX, NLBC, NRBC, NCOL, MXMESH
      PARAMETER        (NEQ=3,MMAX=3,NLBC=3,NRBC=3,NCOL=7,MXMESH=51)
      INTEGER          LRWORK, LIWORK
      PARAMETER        (LRWORK=MXMESH*(109*NEQ**2+78*NEQ+7),
     +                 LIWORK=MXMESH*(11*NEQ+6))
*     .. Scalars in Common ..
      real             OMEGA, SQROFR
*     .. Local Scalars ..
      real             ERMX, R
      INTEGER          I, IERMX, IFAIL, IJERMX, J, NCONT, NMESH
*     .. Local Arrays ..
      real             MESH(MXMESH), TOL(NEQ), WORK(LRWORK),
     +                 Y(NEQ,0:MMAX-1)
      INTEGER          IPMESH(MXMESH), IWORK(LIWORK), M(NEQ)
*     .. External Subroutines ..
      EXTERNAL         D02TKF, D02TVF, D02TXF, D02TYF, D02TZF, FFUN,
     +                 FJAC, GAFUN, GAJAC, GBFUN, GBJAC, GUESS
*     .. Intrinsic Functions ..
      INTRINSIC        real, SQRT
*     .. Common blocks ..
      COMMON           /PROBS/SQROFR, OMEGA
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02TKF Example Program Results'
      WRITE (NOUT,*)
      NMESH = 11
      MESH(1) = 0.0e0
      IPMESH(1) = 1
      DO 20 I = 2, NMESH - 1
         MESH(I) = (I-1)/real(NMESH-1)
         IPMESH(I) = 2
   20 CONTINUE
      MESH(NMESH) = 1.0e0
      IPMESH(NMESH) = 1
      M(1) = 1
      M(2) = 3
      M(3) = 2
      TOL(1) = 1.0e-4
      TOL(2) = TOL(1)
      TOL(3) = TOL(1)
      IFAIL = 0
      CALL D02TVF(NEQ,M,NLBC,NRBC,NCOL,TOL,MXMESH,NMESH,MESH,IPMESH,
     +            WORK,LRWORK,IWORK,LIWORK,IFAIL)
*     Initialize number of continuation steps
      NCONT = 3
*     Initialize problem dependent parameters
      OMEGA = 1.0e0
      R = 1.0e+6
      DO 80 J = 1, NCONT
         SQROFR = SQRT(R)
         WRITE (NOUT,99999) TOL(1), R
*     Solve
         CALL D02TKF(FFUN,FJAC,GAFUN,GBFUN,GAJAC,GBJAC,GUESS,WORK,IWORK,
     +               IFAIL)
*     Extract mesh
         CALL D02TZF(MXMESH,NMESH,MESH,IPMESH,ERMX,IERMX,IJERMX,WORK,
     +               IWORK,IFAIL)
         WRITE (NOUT,99998) NMESH, ERMX, IERMX, IJERMX,
     +     (I,IPMESH(I),MESH(I),I=1,NMESH)
*     Print solution components on mesh
         WRITE (NOUT,99997)
         DO 40 I = 1, NMESH
            CALL D02TYF(MESH(I),Y,NEQ,MMAX,WORK,IWORK,IFAIL)
            WRITE (NOUT,99996) MESH(I), Y(1,0), Y(2,0), Y(3,0)
   40    CONTINUE
*     Select mesh for continuation and modify problem dependent
*     parameters
         IF (J.LT.NCONT) THEN
            R = 1.0e+02*R
            DO 60 I = 2, NMESH - 1
               IPMESH(I) = 2
   60       CONTINUE
            CALL D02TXF(MXMESH,NMESH,MESH,IPMESH,WORK,IWORK,IFAIL)
         END IF
   80 CONTINUE
      STOP
*
99999 FORMAT (/' Tolerance = ',1P,e8.1,'  R = ',e10.3)
99998 FORMAT (/' Used a mesh of ',I4,' points',/' Maximum error = ',
     +       e10.2,'  in interval ',I4,' for component ',I4,//' Mesh p',
```

```
     +        'oints:',/4(I4,'(',I1,')',e11.4))
99997 FORMAT (/'      x         f         f''        g')
99996 FORMAT (' ',F8.3,1X,3F9.4)
      END
      SUBROUTINE FFUN(X,Y,NEQ,M,F)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              F(NEQ), Y(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Scalars in Common ..
      real              OMEGA, SQROFR
*     .. Common blocks ..
      COMMON            /PROBS/SQROFR, OMEGA
*     .. Executable Statements ..
      F(1) = Y(2,0)
      F(2) = -(Y(1,0)*Y(2,2)+Y(3,0)*Y(3,1))*SQROFR
      F(3) = (Y(2,0)*Y(3,0)-Y(1,0)*Y(3,1))*SQROFR
      RETURN
      END
      SUBROUTINE FJAC(X,Y,NEQ,M,DFDY)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              DFDY(NEQ,NEQ,0:*), Y(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Scalars in Common ..
      real              OMEGA, SQROFR
*     .. Common blocks ..
      COMMON            /PROBS/SQROFR, OMEGA
*     .. Executable Statements ..
      DFDY(1,2,0) = 1.0e0
      DFDY(2,1,0) = -Y(2,2)*SQROFR
      DFDY(2,2,2) = -Y(1,0)*SQROFR
      DFDY(2,3,0) = -Y(3,1)*SQROFR
      DFDY(2,3,1) = -Y(3,0)*SQROFR
      DFDY(3,1,0) = -Y(3,1)*SQROFR
      DFDY(3,2,0) = Y(3,0)*SQROFR
      DFDY(3,3,0) = Y(2,0)*SQROFR
      DFDY(3,3,1) = -Y(1,0)*SQROFR
      RETURN
      END
      SUBROUTINE GAFUN(YA,NEQ,M,NLBC,GA)
*     .. Scalar Arguments ..
      INTEGER           NEQ, NLBC
*     .. Array Arguments ..
      real              GA(NLBC), YA(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Scalars in Common ..
      real              OMEGA, SQROFR
*     .. Common blocks ..
      COMMON            /PROBS/SQROFR, OMEGA
*     .. Executable Statements ..
      GA(1) = YA(1,0)
      GA(2) = YA(2,0)
      GA(3) = YA(3,0) - OMEGA
```

```
      RETURN
      END
      SUBROUTINE GBFUN(YB,NEQ,M,NRBC,GB)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NRBC
*     .. Array Arguments ..
      real             GB(NRBC), YB(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Scalars in Common ..
      real             OMEGA, SQROFR
*     .. Common blocks ..
      COMMON           /PROBS/SQROFR, OMEGA
*     .. Executable Statements ..
      GB(1) = YB(1,0)
      GB(2) = YB(2,0)
      GB(3) = YB(3,0) + OMEGA
      RETURN
      END
      SUBROUTINE GAJAC(YA,NEQ,M,NLBC,DGADY)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NLBC
*     .. Array Arguments ..
      real             DGADY(NLBC,NEQ,0:*), YA(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      DGADY(1,1,0) = 1.0e0
      DGADY(2,2,0) = 1.0e0
      DGADY(3,3,0) = 1.0e0
      RETURN
      END
      SUBROUTINE GBJAC(YB,NEQ,M,NRBC,DGBDY)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NRBC
*     .. Array Arguments ..
      real             DGBDY(NRBC,NEQ,0:*), YB(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      DGBDY(1,1,0) = 1.0e0
      DGBDY(2,2,0) = 1.0e0
      DGBDY(3,3,0) = 1.0e0
      RETURN
      END
      SUBROUTINE GUESS(X,NEQ,M,Y,DYM)
*     .. Scalar Arguments ..
      real             X
      INTEGER          NEQ
*     .. Array Arguments ..
      real             DYM(NEQ), Y(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Scalars in Common ..
      real             OMEGA, SQROFR
*     .. Common blocks ..
      COMMON           /PROBS/SQROFR, OMEGA
*     .. Executable Statements ..
      Y(1,0) = -X**2*(X-0.5e0)*(X-1.0e0)**2
      Y(2,0) = -X*(X-1.0e0)*(5.0e0*X**2-5.0e0*X+1.0e0)
      Y(2,1) = -(20.0e0*X**3-30.0e0*X**2+12.0e0*X-1.0e0)
      Y(2,2) = -(60.0e0*X**2-60.0e0*X+12.0e0*X)
```

```
      Y(3,0) = -8.0e0*OMEGA*(X-0.5e0)**3
      Y(3,1) = -24.0e0*OMEGA*(X-0.5e0)**2
      DYM(1) = Y(2,0)
      DYM(2) = -(120.0e0*X-60.0e0)
      DYM(3) = -56.0e0*OMEGA*(X-0.5e0)
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
 D02TKF Example Program Results


 Tolerance =  1.0E-04  R =  1.000E+06

 Used a mesh of   21 points
 Maximum error =   0.62E-09  in interval   20 for component    3

 Mesh points:
   1(1) 0.0000E+00   2(3) 0.5000E-01   3(2) 0.1000E+00   4(3) 0.1500E+00
   5(2) 0.2000E+00   6(3) 0.2500E+00   7(2) 0.3000E+00   8(3) 0.3500E+00
   9(2) 0.4000E+00  10(3) 0.4500E+00  11(2) 0.5000E+00  12(3) 0.5500E+00
  13(2) 0.6000E+00  14(3) 0.6500E+00  15(2) 0.7000E+00  16(3) 0.7500E+00
  17(2) 0.8000E+00  18(3) 0.8500E+00  19(2) 0.9000E+00  20(3) 0.9500E+00
  21(1) 0.1000E+01

      x         f         f'        g
    0.000    0.0000    0.0000    1.0000
    0.050    0.0070    0.1805    0.4416
    0.100    0.0141    0.0977    0.1886
    0.150    0.0171    0.0252    0.0952
    0.200    0.0172   -0.0165    0.0595
    0.250    0.0157   -0.0400    0.0427
    0.300    0.0133   -0.0540    0.0322
    0.350    0.0104   -0.0628    0.0236
    0.400    0.0071   -0.0683    0.0156
    0.450    0.0036   -0.0714    0.0078
    0.500    0.0000   -0.0724    0.0000
    0.550   -0.0036   -0.0714   -0.0078
    0.600   -0.0071   -0.0683   -0.0156
    0.650   -0.0104   -0.0628   -0.0236
    0.700   -0.0133   -0.0540   -0.0322
    0.750   -0.0157   -0.0400   -0.0427
    0.800   -0.0172   -0.0165   -0.0595
    0.850   -0.0171    0.0252   -0.0952
    0.900   -0.0141    0.0977   -0.1886
    0.950   -0.0070    0.1805   -0.4416
    1.000    0.0000    0.0000   -1.0000

 Tolerance =  1.0E-04  R =  1.000E+08

 Used a mesh of   21 points
 Maximum error =   0.45E-08  in interval    6 for component    3
```

```
Mesh points:
  1(1) 0.0000E+00   2(3) 0.1757E-01   3(2) 0.3515E-01   4(3) 0.5203E-01
  5(2) 0.6891E-01   6(3) 0.8593E-01   7(2) 0.1030E+00   8(3) 0.1351E+00
  9(2) 0.1672E+00  10(3) 0.2306E+00  11(2) 0.2939E+00  12(3) 0.4713E+00
 13(2) 0.6486E+00  14(3) 0.7455E+00  15(2) 0.8423E+00  16(3) 0.8824E+00
 17(2) 0.9225E+00  18(3) 0.9449E+00  19(2) 0.9673E+00  20(3) 0.9836E+00
 21(1) 0.1000E+01

     x         f         f'        g
   0.000    0.0000    0.0000    1.0000
   0.018    0.0025    0.1713    0.3923
   0.035    0.0047    0.0824    0.1381
   0.052    0.0056    0.0267    0.0521
   0.069    0.0058    0.0025    0.0213
   0.086    0.0057   -0.0073    0.0097
   0.103    0.0056   -0.0113    0.0053
   0.135    0.0052   -0.0135    0.0027
   0.167    0.0047   -0.0140    0.0020
   0.231    0.0038   -0.0142    0.0015
   0.294    0.0029   -0.0142    0.0012
   0.471    0.0004   -0.0143    0.0002
   0.649   -0.0021   -0.0143   -0.0008
   0.745   -0.0035   -0.0142   -0.0014
   0.842   -0.0049   -0.0139   -0.0022
   0.882   -0.0054   -0.0127   -0.0036
   0.922   -0.0058   -0.0036   -0.0141
   0.945   -0.0057    0.0205   -0.0439
   0.967   -0.0045    0.0937   -0.1592
   0.984   -0.0023    0.1753   -0.4208
   1.000    0.0000    0.0000   -1.0000

Tolerance =  1.0E-04  R =  1.000E+10

Used a mesh of   21 points
Maximum error =   0.31E-05  in interval    7 for component    3

Mesh points:
  1(1) 0.0000E+00   2(3) 0.6256E-02   3(2) 0.1251E-01   4(3) 0.1851E-01
  5(2) 0.2450E-01   6(3) 0.3076E-01   7(2) 0.3702E-01   8(3) 0.4997E-01
  9(2) 0.6292E-01  10(3) 0.9424E-01  11(2) 0.1256E+00  12(3) 0.4190E+00
 13(2) 0.7125E+00  14(3) 0.8246E+00  15(2) 0.9368E+00  16(3) 0.9544E+00
 17(2) 0.9719E+00  18(3) 0.9803E+00  19(2) 0.9886E+00  20(3) 0.9943E+00
 21(1) 0.1000E+01

     x         f         f'        g
   0.000    0.0000    0.0000    1.0000
   0.006    0.0009    0.1623    0.3422
   0.013    0.0016    0.0665    0.1021
   0.019    0.0018    0.0204    0.0318
   0.025    0.0019    0.0041    0.0099
   0.031    0.0019   -0.0014    0.0028
   0.037    0.0019   -0.0031    0.0007
   0.050    0.0019   -0.0038   -0.0002
   0.063    0.0018   -0.0038   -0.0003
   0.094    0.0017   -0.0039   -0.0003
   0.126    0.0016   -0.0039   -0.0002
   0.419    0.0004   -0.0041   -0.0001
   0.712   -0.0008   -0.0042    0.0001
```

```
 0.825   -0.0013  -0.0043   0.0002
 0.937   -0.0018  -0.0043   0.0003
 0.954   -0.0019  -0.0042   0.0001
 0.972   -0.0019  -0.0003  -0.0049
 0.980   -0.0019   0.0152  -0.0252
 0.989   -0.0015   0.0809  -0.1279
 0.994   -0.0008   0.1699  -0.3814
 1.000    0.0000   0.0000  -1.0000
```

D02TVF – NAG Fortran Library Routine Document

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

D02TVF is a setup routine which must be called prior to the first call of the nonlinear two point boundary value solver D02TKF.

2 Specification

```
      SUBROUTINE D02TVF(NEQ, M, NLBC, NRBC, NCOL, TOLS, MXMESH, NMESH,
     1                  MESH, IPMESH, RWORK, LRWORK, IWORK, LIWORK,
     2                  IFAIL)
      INTEGER           NEQ, M(NEQ), NLBC, NRBC, NCOL, MXMESH, NMESH,
     1                  IPMESH(MXMESH), LRWORK, IWORK(LIWORK), LIWORK,
     2                  IFAIL
      real              TOLS(NEQ), MESH(MXMESH), RWORK(LRWORK)
```

3 Description

D02TVF and its associated routines (D02TKF, D02TXF, D02TYF and D02TZF) solve the two point boundary value problem for a nonlinear system of ordinary differential equations

$$\begin{aligned} y_1^{(m_1)} &= f_1(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\ y_2^{(m_2)} &= f_2(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\ &\cdots \\ y_n^{(m_n)} &= f_n(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \end{aligned}$$

over an interval $[a, b]$ subject to p (> 0) nonlinear boundary conditions at a and q (> 0) nonlinear boundary conditions at b, where $p + q = \sum_1^n m_i$. Note that $y_i^{(m)}(x)$ is the m-th derivative of the i-th solution component. Hence $y_i^{(0)}(x) = y_i(x)$. The left boundary conditions at a are defined as

$$g_i(z(y(a))) = 0, \quad i = 1, 2, \ldots, p,$$

and the right boundary conditions at b as

$$\bar{g}_j(z(y(b))) = 0, \quad j = 1, 2, \ldots, q,$$

where $y = (y_1, y_2, \ldots, y_n)$ and

$$z(y(x)) = (y_1(x), y_1^{(1)}(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)).$$

See Section 8 for information on how boundary value problems of a more general nature can be treated.

D02TVF is used to specify an initial mesh, error requirements and other details. D02TKF is then used to solve the boundary value problem.

The solution routine D02TKF proceeds as follows. A modified Newton method is applied to the equations

$$y_i^{(m_i)}(x) - f_i(x, z(y(x))) = 0, \quad i = 1, \ldots, n$$

and the boundary conditions. To solve these equations numerically the components y_i are approximated by piecewise polynomials v_{ij} using a monomial basis on the j-th mesh sub-interval. The coefficients of the polynomials v_{ij} form the unknowns to be computed. Collocation is applied at Gaussian points

$$v_{ij}^{(m_i)}(x_{jk}) - f_i(x_{jk}, z(v(x_{jk}))) = 0, \quad i = 1, \ldots, n,$$

where x_{jk} is the k-th collocation point in the j-th mesh sub-interval. Continuity at the mesh points is imposed, that is

$$v_{ij}(x_{j+1}) - v_{i,j+1}(x_{j+1}) = 0, \quad i = 1,2,\ldots,n,$$

where x_{j+1} is the right hand end of the j-th mesh sub-interval. The linearized collocation equations and boundary conditions, together with the continuity conditions form a system of linear algebraic equations which are solved using F01LHF and F04LHF. For use in the modified Newton method, an approximation to the solution on the initial mesh must be supplied via the procedure argument GUESS of D02TKF.

The solver attempts to satisfy the conditions

$$\frac{||y_i - v_i||}{(1.0 + ||v_i||)} \le \text{TOLS}(i), \quad i = 1,2,...,n, \tag{1}$$

where v_i is the approximate solution for the i-th solution component and TOLS is supplied by the user. The mesh is refined by trying to equidistribute the estimated error in the computed solution over all mesh sub-intervals, and an extrapolation-like test (doubling the number of mesh sub-intervals) is used to check for (1).

The routines are based on modified versions of the codes COLSYS and COLNEW, [2] and [1]. A comprehensive treatment of the numerical solution of boundary value problems can be found in [3] and [5].

4 References

[1] Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

[2] Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

[3] Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice Hall, Englewood Cliffs, NJ

[4] Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

[5] Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

[6] Schwartz I B (1983) Estimating regions of existence of unstable periodic orbits using computer-based techniques *SIAM J. Sci. Statist. Comput.* **20(1)** 106–120

5 Parameters

1: NEQ — INTEGER *Input*

On entry: the number of ordinary differential equations to be solved, n.

Constraint: NEQ ≥ 1.

2: M(NEQ) — INTEGER array *Input*

On entry: the order, m_i, of the i-th differential equation, for $i = 1,2,\ldots,n$.

Constraint: $1 \le \text{M}(i) \le 4, \quad i = 1,2,\ldots,n$.

3: NLBC — INTEGER *Input*

On entry: the number of left boundary conditions, p, defined at the left hand end, a (= MESH(1)).

Constraint: NLBC ≥ 1.

4: NRBC — INTEGER *Input*

On entry: the number of right boundary conditions, q, defined at the right hand end, b (= MESH(NMESH)).

Constraints:

$$\text{NRBC} \geq 1,$$

$$\text{NLBC} + \text{NRBC} = \sum_{1}^{n} \text{M}(i).$$

5: NCOL — INTEGER *Input*

On entry: the number of collocation points to be used in each mesh sub-interval.

Constraint: $m_{\text{max}} \leq \text{NCOL} \leq 7$, where $m_{\text{max}} = \max(\text{M}(i))$.

6: TOLS(NEQ) — ***real*** array *Input*

On entry: the error requirement for the ith solution component.

Constraint: $100 \times$ ***machine precision*** $< \text{TOLS}(i) < 1.0$, for $i = 1, 2, \ldots, n$.

7: MXMESH — INTEGER *Input*

On entry: the maximum number of mesh points to be used during the solution process.

Constraint: $\text{MXMESH} \geq 2 \times \text{NMESH} - 1$.

8: NMESH — INTEGER *Input*

On entry: the number of points to be used in the initial mesh of the solution process.

Constraint: $\text{NMESH} \geq 6$.

9: MESH(MXMESH) — ***real*** array *Input*

On entry: the positions of the initial NMESH mesh points. The remaining elements of MESH need not be set. You should try to place the mesh points in areas where you expect the solution to vary most rapidly. In the absence of any other information the points should be equally distributed on $[a,b]$.

MESH(1) must contain the left boundary point, a, and MESH(NMESH) must contain the right boundary point, b.

Constraint: $\text{MESH}(i) < \text{MESH}(i+1)$, for $i = 1, 2, \ldots, \text{NMESH} - 1$.

10: IPMESH(MXMESH) — INTEGER array *Input*

On entry: IPMESH(i) specifies whether or not the initial mesh point defined in MESH(i), $i = 1, \ldots, \text{NMESH}$, should be a fixed point in all meshes computed during the solution process. The remaining elements of IPMESH need not be set.

IPMESH(i) = 1 indicates that MESH(i) should be a fixed point in all meshes.

IPMESH(i) = 2 indicates that MESH(i) is not a fixed point.

Constraints:

IPMESH(1) = 1 and IPMESH(NMESH) = 1, (that is the left and right boundary points, a and b, must be fixed points, in all meshes)
IPMESH(i) = 1 or 2, $i = 2, 3, \ldots, \text{NMESH} - 1$.

11: RWORK(LRWORK) — ***real*** array *Output*

On exit: contains information for use by D02TKF. This **must** be the same array as will be supplied to D02TKF. The contents of this array **must** remain unchanged between calls.

12: LRWORK — INTEGER *Input*

On entry: the dimension of the array RWORK as declared in the (sub)program from which D02TVF is called.

Suggested value: LRWORK = MXMESH × (109 × N^2 + 78 × N + 7), which will permit MXMESH mesh points for a system of N differential equations regardless of their order or the number of collocation points used.

Constraint: LRWORK $\geq 50 + \mathrm{NEQ} \times (m_{\max} \times (1 + \mathrm{NEQ} + \max(\mathrm{NLBC}, \mathrm{NRBC})) + 6) - k_n \times (k_n + 6) - m^* \times (k_n + m^* - 2) + \mathrm{MXMESH} \times ((m^* + 3)(2m^* + 3) - 3 + k_n(k_n + m^* + 6)) + \mathrm{MXMESH}/2$, where $m^* = \sum_1^n \mathrm{M}(i)$ and $k_n = \mathrm{NCOL} \times \mathrm{NEQ}$.

13: IWORK(LIWORK) — INTEGER array *Output*

On exit: contains information for use by D02TKF. This **must** be the same array as will be supplied to D02TKF. The contents of this array **must** remain unchanged between calls.

14: LIWORK — INTEGER *Input*

On entry: the dimension of the array IWORK as declared in the (sub)program from which D02TVF is called.

Suggested value: LIWORK = MXMESH × (11 × N + 6), which will permit MXMESH mesh points for a system of N differential equations regardless of their order or the number of collocation points used.

Constraint: LIWORK $\geq 23 + 3 \times \mathrm{NEQ} - k_n + \mathrm{MXMESH} \times (m^* + k_n + 4)$.

15: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Errors 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 detected by the routine:

IFAIL = 1

On entry, NEQ < 1,
or M(i) < 1, for some i,
or M(i) > 4, for some i,
or NMESH < 6,
or the values of MESH are not strictly increasing,
or IPMESH(i) is invalid for some i,
or MXMESH < 2 × NMESH − 1,
or NCOL < $m_{\max}$, where $m_{\max} = \max(\mathrm{M}(i))$,
or NCOL > 7,
or NLBC < 1,
or NRBC < 1,
or a value of TOLS is invalid,
or $\mathrm{NLBC} + \mathrm{NRBC} \neq \sum_1^n \mathrm{M}(i)$,
or LRWORK or LIWORK is too small.

7 Accuracy

Not applicable.

8 Further Comments

For problems where sharp changes of behaviour are expected over short intervals it may be advisable to:

use a large value for NCOL;

cluster the initial mesh points where sharp changes in behaviour are expected;

maintain fixed points in the mesh using the argument IPMESH to ensure that the remeshing process does not inadvertently remove mesh points from areas of known interest before they are detected automatically by the algorithm.

8.1 Nonseparated boundary conditions

A boundary value problem with nonseparated boundary conditions can be treated by transformation to an equivalent problem with separated conditions. As a simple example consider the system

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2) \\ y_2' &= f_2(x, y_1, y_2) \end{aligned}$$

on $[a, b]$ subject to the boundary conditions

$$\begin{aligned} g_1(y_1(a)) &= 0 \\ g_2(y_2(a), y_2(b)) &= 0. \end{aligned}$$

By adjoining the trivial ordinary differential equation

$$r' = 0,$$

which implies $r(a) = r(b)$, and letting $r(b) = y_2(b)$, say, we have a new system

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2) \\ y_2' &= f_2(x, y_1, y_2) \\ r' &= 0, \end{aligned}$$

subject to the separated boundary conditions

$$\begin{aligned} g_1(y_1(a)) &= 0 \\ g_2(y_2(a), r(a)) &= 0 \\ y_2(b) - r(b) &= 0. \end{aligned}$$

There is an obvious overhead in adjoining an extra differential equation: the system to be solved is increased in size.

8.2 Multipoint boundary value problems

Multipoint boundary value problems, that is problems where conditions are specified at more than two points, can also be transformed to an equivalent problem with two boundary points. Each sub-interval defined by the multipoint conditions can be transformed onto the interval $[0, 1]$, say, leading to a larger set of differential equations. The boundary conditions of the transformed system consist of the original boundary conditions and the conditions imposed by the requirement that the solution components be continuous at the interior break points. For example, consider the equation

$$y^{(3)} = f(t, y, y^{(1)}, y^{(2)}) \quad \text{on} \quad [a, c]$$

subject to the conditions

$$\begin{aligned} y(a) &= A \\ y(b) &= B \\ y^{(1)}(c) &= C \end{aligned}$$

where $a < b < c$. This can be transformed to the system

$$\left.\begin{array}{rcl} y_1^{(3)} & = & f(t, y_1, y_1^{(1)}, y_1^{(2)}) \\ y_2^{(3)} & = & f(t, y_2, y_2^{(1)}, y_2^{(2)}) \end{array}\right\} \quad \text{on } [0,1]$$

where

$$\begin{array}{rcl} y_1 & \equiv & y \quad \text{on} \quad [a, b] \\ y_2 & \equiv & y \quad \text{on} \quad [b, c], \end{array}$$

subject to the boundary conditions

$$\begin{array}{rcl} y_1(0) & = & A \\ y_1(1) & = & B \\ y_2^{(1)}(1) & = & C \\ y_2(0) & = & B \qquad (\text{from } y_1(1) = y_2(0)) \\ y_1^{(1)}(1) & = & y_2^{(1)}(0) \\ y_1^{(2)}(1) & = & y_2^{(2)}(0). \end{array}$$

In this instance two of the resulting boundary conditions are nonseparated but they may next be treated as described above.

8.3 High order systems

Systems of ordinary differential equations containing derivatives of order greater than four can always be reduced to systems of order suitable for treatment by D02TVF and its related routines. For example suppose we have the sixth order equation

$$y^{(6)} = -y.$$

Writing the variables $y_1 = y$ and $y_2 = y^{(4)}$ we obtain the system

$$\begin{array}{rcl} y_1^{(4)} & = & y_2 \\ y_2^{(2)} & = & -y_1 \end{array}$$

which has maximal order four, or writing the variables $y_1 = y$ and $y_2 = y^{(3)}$ we obtain the system

$$\begin{array}{rcl} y_1^{(3)} & = & y_2 \\ y_2^{(3)} & = & -y_1 \end{array}$$

which has maximal order three. The best choice of reduction by choosing new variables will depend on the structure and physical meaning of the system. Note that you will control the error in each of the variables y_1 and y_2. Indeed, if you wish to control the error in certain derivatives of the solution of an equation of order greater than one, then you should make those derivatives new variables.

8.4 Fixed points and singularities

The solver routine D02TKF employs collocation at Gaussian points in each sub-interval of the mesh. Hence the coefficients of the differential equations are not evaluated at the mesh points. Thus, fixed points should be specified in the mesh where either the coefficients are singular, or the solution has less smoothness, or where the differential equations should not be evaluated. Singular coefficients at boundary points often arise when physical symmetry is used to reduce partial differential equations to ordinary differential equations. These do not pose a direct numerical problem for using this code but they can severely impact its convergence.

8.5 Numerical Jacobians

The solver routine D02TKF requires an external procedure FJAC to evaluate the partial derivatives of f_i with respect to the elements of $z(y)$ $(= (y_1, y_1^1, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}))$. In cases where the partial derivatives are difficult to evaluate, numerical approximations can be used. However, this approach might have a negative impact on the convergence of the modified Newton method. You could consider the use of symbolic mathematic packages and/or automatic differentiation packages if available to you.

See Section 9 of the document for D02TZF for an example using numerical approximations to the Jacobian. There central differences are used and each f_i is assumed to depend on all the components of z. This requires two evaluations of the system of differential equations for each component of z. The perturbation used depends on the size of each component of z and a minimum quantity dependent on the machine precision. The cost of this approach could be reduced by employing an alternative difference scheme and/or by only perturbing the components of z which appear in the definitions of the f_i. A discussion on the choice of perturbation factors for use in finite difference approximations to partial derivatives can be found in [4].

9 Example

The following example is used to illustrate the treatment of nonseparated boundary conditions. See also D02TKF, D02TXF, D02TYF and D02TZF, for the illustration of other facilities.

The following equations model of the spread of measles. See [6]. Under certain assumptions the dynamics of the model can be expressed as

$$\begin{aligned} y_1' &= \mu - \beta(x) y_1 y_3 \\ y_2' &= \beta(x) y_1 y_3 - y_2/\lambda \\ y_3' &= y_2/\lambda - y_3/\eta \end{aligned}$$

subject to the periodic boundary conditions

$$y_i(0) = y_i(1), \quad i = 1, 2, 3.$$

Here y_1, y_2 and y_3 are respectively the proportions of susceptibles, infectives and latents to the whole population. λ (= 0.0279 years) is the latent period, η (= 0.01 years) is the infectious period and μ (= 0.02) is the population birth rate. $\beta(x) = \beta_0(1.0 + \cos 2\pi x)$ is the contact rate where $\beta_0 = 1575.0$.

The nonseparated boundary conditions are treated as descibed in Section 8 by adjoining the trivial differential equations

$$\begin{aligned} y_4' &= 0 \\ y_5' &= 0 \\ y_6' &= 0 \end{aligned}$$

that is y_4, y_5 and y_6 are constants. The boundary conditions of the augmented system can then be posed in the separated form

$$\begin{aligned} y_1(0) - y_4(0) &= 0 \\ y_2(0) - y_5(0) &= 0 \\ y_3(0) - y_6(0) &= 0 \\ y_1(1) - y_4(1) &= 0 \\ y_2(1) - y_5(1) &= 0 \\ y_3(1) - y_6(1) &= 0. \end{aligned}$$

This is a relatively easy problem and an (arbitrary) initial guess of 1 for each component suffices, even though two components of the solution are much smaller than 1.

9.1 Example 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.

```
*     D02TVF Example Program Text
*     Mark 17 Release. NAG Copyright 1995.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, MMAX, NLBC, NRBC, NCOL, MXMESH
      PARAMETER        (NEQ=6,MMAX=1,NLBC=3,NRBC=3,NCOL=5,MXMESH=100)
      INTEGER          LRWORK, LIWORK
      PARAMETER        (LRWORK=MXMESH*(109*NEQ**2+78*NEQ+7),
     +                 LIWORK=MXMESH*(11*NEQ+6))
```

```
*     .. Scalars in Common ..
      real              BETA0, ETA, LAMBDA, MU, PI
*     .. Local Scalars ..
      real              ERMX
      INTEGER           I, IERMX, IFAIL, IJERMX, NMESH
*     .. Local Arrays ..
      real              MESH(MXMESH), RWORK(LRWORK), TOL(NEQ),
     +                  Y(NEQ,0:MMAX-1)
      INTEGER           IPMESH(MXMESH), IWORK(LIWORK), M(NEQ)
*     .. External Subroutines ..
      EXTERNAL          D02TKF, D02TVF, D02TYF, D02TZF, FFUN, FJAC,
     +                  GAFUN, GAJAC, GBFUN, GBJAC, GUESS
*     .. Intrinsic Functions ..
      INTRINSIC         ATAN
*     .. Common blocks ..
      COMMON            /PROB/ETA, MU, LAMBDA, BETA0, PI
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02TVF Example Program Results'
      WRITE (NOUT,*)
      NMESH = 11
      MESH(1) = 0.0e0
      IPMESH(1) = 1
      DO 20 I = 2, NMESH - 1
         MESH(I) = 0.1e0*(I-1)
         IPMESH(I) = 2
   20 CONTINUE
      IPMESH(NMESH) = 1
      MESH(NMESH) = 1.0e0
      DO 40 I = 1, NEQ
         TOL(I) = 1.0e-5
         M(I) = 1
   40 CONTINUE
      ETA = 0.01e0
      MU = 0.02e0
      LAMBDA = 0.0279e0
      BETA0 = 1575.0e0
      PI = 4.0e0*ATAN(1.0e0)
      IFAIL = 0
      CALL D02TVF(NEQ,M,NLBC,NRBC,NCOL,TOL,MXMESH,NMESH,MESH,IPMESH,
     +            RWORK,LRWORK,IWORK,LIWORK,IFAIL)
      IFAIL = -1
      CALL D02TKF(FFUN,FJAC,GAFUN,GBFUN,GAJAC,GBJAC,GUESS,RWORK,IWORK,
     +            IFAIL)
      CALL D02TZF(MXMESH,NMESH,MESH,IPMESH,ERMX,IERMX,IJERMX,RWORK,
     +            IWORK,IFAIL)
      WRITE (NOUT,99999) NMESH, ERMX, IERMX, IJERMX,
     +  (I,IPMESH(I),MESH(I),I=1,NMESH)
      WRITE (NOUT,99998)
      DO 60 I = 1, NMESH
         IFAIL = 1
         CALL D02TYF(MESH(I),Y,NEQ,MMAX,RWORK,IWORK,IFAIL)
         WRITE (NOUT,99997) MESH(I), Y(1,0), Y(2,0), Y(3,0)
   60 CONTINUE
      STOP
*
99999 FORMAT (/' Used a mesh of ',I4,' points',/' Maximum error = ',
     +       e10.2,'  in interval ',I4,' for component ',I4,//' Mesh p',
     +       'oints:',/4(I4,'(',I1,')',F7.4))
```

```
99998 FORMAT (/' Computed solution at mesh points',/'    x        y1    ',
     +       '        y2         y3')
99997 FORMAT (' ',F6.3,1X,3e11.3)
      END
      SUBROUTINE FFUN(X,Y,NEQ,M,F)
*     .. Scalar Arguments ..
      real             X
      INTEGER          NEQ
*     .. Array Arguments ..
      real             F(NEQ), Y(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Scalars in Common ..
      real             BETA0, ETA, LAMBDA, MU, PI
*     .. Local Scalars ..
      real             BETA
*     .. Intrinsic Functions ..
      INTRINSIC        COS
*     .. Common blocks ..
      COMMON           /PROB/ETA, MU, LAMBDA, BETA0, PI
*     .. Executable Statements ..
      BETA = BETA0*(1.0e0+COS(2.0e0*PI*X))
      F(1) = MU - BETA*Y(1,0)*Y(3,0)
      F(2) = BETA*Y(1,0)*Y(3,0) - Y(2,0)/LAMBDA
      F(3) = Y(2,0)/LAMBDA - Y(3,0)/ETA
      F(4) = 0.0e0
      F(5) = 0.0e0
      F(6) = 0.0e0
      RETURN
      END
      SUBROUTINE FJAC(X,Y,NEQ,M,DF)
*     .. Scalar Arguments ..
      real             X
      INTEGER          NEQ
*     .. Array Arguments ..
      real             DF(NEQ,NEQ,0:*), Y(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Scalars in Common ..
      real             BETA0, ETA, LAMBDA, MU, PI
*     .. Local Scalars ..
      real             BETA
*     .. Intrinsic Functions ..
      INTRINSIC        COS
*     .. Common blocks ..
      COMMON           /PROB/ETA, MU, LAMBDA, BETA0, PI
*     .. Executable Statements ..
      BETA = BETA0*(1.0e0+COS(2.0e0*PI*X))
      DF(1,1,0) = -BETA*Y(3,0)
      DF(1,3,0) = -BETA*Y(1,0)
      DF(2,1,0) = BETA*Y(3,0)
      DF(2,2,0) = -1.0e0/LAMBDA
      DF(2,3,0) = BETA*Y(1,0)
      DF(3,2,0) = 1.0e0/LAMBDA
      DF(3,3,0) = -1.0e0/ETA
      RETURN
      END
      SUBROUTINE GAFUN(YA,NEQ,M,NLBC,GA)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NLBC
```

```
*     .. Array Arguments ..
      real             GA(NLBC), YA(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      GA(1) = YA(1,0) - YA(4,0)
      GA(2) = YA(2,0) - YA(5,0)
      GA(3) = YA(3,0) - YA(6,0)
      RETURN
      END
      SUBROUTINE GBFUN(YB,NEQ,M,NRBC,GB)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NRBC
*     .. Array Arguments ..
      real             GB(NRBC), YB(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      GB(1) = YB(1,0) - YB(4,0)
      GB(2) = YB(2,0) - YB(5,0)
      GB(3) = YB(3,0) - YB(6,0)
      RETURN
      END
      SUBROUTINE GAJAC(YA,NEQ,M,NLBC,DGA)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NLBC
*     .. Array Arguments ..
      real             DGA(NLBC,NEQ,0:*), YA(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      DGA(1,1,0) = 1.0e0
      DGA(1,4,0) = -1.0e0
      DGA(2,2,0) = 1.0e0
      DGA(2,5,0) = -1.0e0
      DGA(3,3,0) = 1.0e0
      DGA(3,6,0) = -1.0e0
      RETURN
      END
      SUBROUTINE GBJAC(YB,NEQ,M,NRBC,DGB)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NRBC
*     .. Array Arguments ..
      real             DGB(NRBC,NEQ,0:*), YB(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      DGB(1,1,0) = 1.0e0
      DGB(1,4,0) = -1.0e0
      DGB(2,2,0) = 1.0e0
      DGB(2,5,0) = -1.0e0
      DGB(3,3,0) = 1.0e0
      DGB(3,6,0) = -1.0e0
      RETURN
      END
      SUBROUTINE GUESS(X,NEQ,M,Z,DMVAL)
*     .. Scalar Arguments ..
      real             X
      INTEGER          NEQ
*     .. Array Arguments ..
      real             DMVAL(NEQ), Z(NEQ,0:*)
      INTEGER          M(NEQ)
```

```
*     .. Local Scalars ..
      INTEGER          I
*     .. Executable Statements ..
      Z(1,0) = 1.0e0
      Z(2,0) = 1.0e0
      Z(3,0) = 1.0e0
      Z(4,0) = Z(1,0)
      Z(5,0) = Z(2,0)
      Z(6,0) = Z(3,0)
      DO 20 I = 1, NEQ
         DMVAL(I) = 0.0e0
   20 CONTINUE
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
D02TVF Example Program Results


Used a mesh of   21 points
Maximum error =   0.14E-07  in interval    5 for component    1

Mesh points:
  1(1) 0.0000   2(3) 0.0500   3(2) 0.1000   4(3) 0.1500
  5(2) 0.2000   6(3) 0.2500   7(2) 0.3000   8(3) 0.3500
  9(2) 0.4000  10(3) 0.4500  11(2) 0.5000  12(3) 0.5500
 13(2) 0.6000  14(3) 0.6500  15(2) 0.7000  16(3) 0.7500
 17(2) 0.8000  18(3) 0.8500  19(2) 0.9000  20(3) 0.9500
 21(1) 1.0000

Computed solution at mesh points
   x        y1          y2          y3
 0.000    0.752E-01  0.180E-04  0.498E-05
 0.050    0.761E-01  0.789E-04  0.219E-04
 0.100    0.766E-01  0.315E-03  0.892E-04
 0.150    0.758E-01  0.101E-02  0.298E-03
 0.200    0.726E-01  0.225E-02  0.713E-03
 0.250    0.678E-01  0.311E-02  0.108E-02
 0.300    0.641E-01  0.256E-02  0.984E-03
 0.350    0.629E-01  0.129E-02  0.550E-03
 0.400    0.633E-01  0.414E-03  0.197E-03
 0.450    0.643E-01  0.912E-04  0.478E-04
 0.500    0.653E-01  0.159E-04  0.881E-05
 0.550    0.663E-01  0.277E-05  0.151E-05
 0.600    0.673E-01  0.628E-06  0.313E-06
 0.650    0.683E-01  0.219E-06  0.964E-07
 0.700    0.693E-01  0.124E-06  0.487E-07
 0.750    0.703E-01  0.116E-06  0.409E-07
 0.800    0.713E-01  0.170E-06  0.551E-07
 0.850    0.723E-01  0.370E-06  0.113E-06
 0.900    0.733E-01  0.111E-05  0.322E-06
 0.950    0.743E-01  0.420E-05  0.118E-05
 1.000    0.752E-01  0.180E-04  0.498E-05
```

D02TXF – NAG Fortran Library Routine Document

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

D02TXF allows a solution to a nonlinear two point boundary value problem computed by D02TKF to be used as an initial approximation in the solution of a related nonlinear two point boundary value problem in a continuation call to D02TKF.

2 Specification

```
SUBROUTINE D02TXF(MXMESH, NMESH, MESH, IPMESH, RWORK, IWORK, IFAIL)
INTEGER          MXMESH, NMESH, IPMESH(MXMESH), IWORK(*), IFAIL
real             MESH(MXMESH), RWORK(*)
```

3 Description

D02TXF and its associated routines (D02TKF, D02TVF, D02TYF and D02TZF) solve the two point boundary value problem for a nonlinear system of ordinary differential equations

$$\begin{aligned} y_1^{(m_1)} &= f_1(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\ y_2^{(m_2)} &= f_2(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\ &\ldots \\ y_n^{(m_n)} &= f_n(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \end{aligned}$$

over an interval $[a,b]$ subject to p (> 0) nonlinear boundary conditions at a and q (> 0) nonlinear boundary conditions at b, where $p + q = \sum_1^n m_i$. Note that $y_i^{(m)}(x)$ is the m-th derivative of the i-th solution component. Hence $y_i^{(0)}(x) = y_i(x)$. The left boundary conditions at a are defined as

$$g_i(z(y(a))) = 0, \quad i = 1, 2, \ldots, p,$$

and the right boundary conditions at b as

$$\bar{g}_j(z(y(b))) = 0, \quad j = 1, 2, \ldots, q,$$

where $y = (y_1, y_2, \ldots, y_n)$ and

$$z(y(x)) = (y_1(x), y_1^{(1)}(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots y_n^{(m_n-1)}(x)).$$

First, D02TVF must be called to specify the initial mesh, error requirements and other details. Then, D02TKF can be used to solve the boundary value problem. After successful computation, D02TZF can be used to ascertain details about the final mesh. D02TYF can be used to compute the approximate solution anywhere on the interval $[a,b]$ using interpolation.

If the boundary value problem being solved is one of a sequence of related problems, for example as part of some continuation process, then D02TXF should be used between calls to D02TKF. This avoids the overhead of a complete initialization when the setup routine D02TVF is used. D02TXF allows the solution values computed in the previous call to D02TKF to be used as an initial approximation for the solution in the next call to D02TKF.

The new initial mesh must be specified by the user. The previous mesh can be obtained by a call to D02TZF. It may be used unchanged as the new mesh, in which case any fixed points in the previous mesh remain as fixed points in the new mesh. Fixed and other points may be added or subtracted from the mesh by manipulation of the contents of the array argument IPMESH. Initial values for the solution components on the new mesh are computed by interpolation on the values for the solution components on the previous mesh.

The routines are based on modified versions of the codes COLSYS and COLNEW, [2] and [1]. A comprehensive treatment of the numerical solution of boundary value problems can be found in [3] and [4].

4 References

[1] Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

[2] Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

[3] Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice Hall, Englewood Cliffs, NJ

[4] Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

5 Parameters

1: MXMESH — INTEGER *Input*

On entry: the maximum number of points allowed in the mesh.

Constraint: this must be identical to the value supplied for the argument MXMESH in the prior call to D02TVF.

2: NMESH — INTEGER *Input*

On entry: the number of points to be used in the new initial mesh.

Suggested value: $(n^* + 1)/2$, where n^* is the number of mesh points used in the previous mesh as returned in the argument NMESH of D02TZF.

Constraint: $6 \leq \text{NMESH} \leq (\text{MXMESH} + 1)/2$.

3: MESH(MXMESH) — ***real*** array *Input*

On entry: the NMESH points to be used in the new initial mesh as specified by IPMESH.

Suggested values: the argument MESH returned from a call to D02TZF.

Constraints:

$\text{MESH}(i_j) < \text{MESH}(i_{j+1})$, for $j = 1, 2, \ldots, \text{NMESH} - 1$; the values of $i_1, i_2, \ldots, i_{\text{NMESH}}$ are defined in IPMESH below.

$\text{MESH}(i_1)$ must contain the left boundary point, a, and $\text{MESH}(i_{\text{NMESH}})$ must contain the right boundary point, b, as specified in the previous call to D02TVF.

4: IPMESH(MXMESH) — INTEGER array *Input*

On entry: specifies the points in MESH to be used as the new initial mesh. Let $\{i_j : j = 1, 2, \ldots, \text{NMESH}\}$ be the set of array indices of IPMESH such that $\text{IPMESH}(i_j) = 1$ or 2 and $1 = i_1 < i_2 < \ldots < i_{\text{NMESH}}$. Then $\text{MESH}(i_j)$ will be included in the new initial mesh. If $\text{IPMESH}(i_j) = 1$, then $\text{MESH}(i_j)$ will be a fixed point in the new initial mesh. If $\text{IPMESH}(k) = 3$ for any k, then $\text{MESH}(k)$ will not be included in the new mesh.

Suggested values: the argument IPMESH returned in a call to D02TZF.

Constraints:

$\text{IPMESH}(k) = 1$, 2 or 3 for $k = 1, 2, \ldots, i_{\text{NMESH}}$

$\text{IPMESH}(1) = \text{IPMESH}(i_{\text{NMESH}}) = 1$.

5: RWORK(*) — ***real*** array *Input/Output*

On entry: this must be the same array as supplied to D02TKF and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated routines.

6: IWORK(*) — INTEGER array *Input/Output*

On entry: this must be the same array as supplied to D02TKF and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated routines.

7: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Errors 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 detected by the routine:

IFAIL = 1

An invalid call to D02TXF was made, for example without a previous successful call to the solver routine D02TKF, or, on entry, an invalid value for NMESH, MESH or IPMESH was detected. If on entry IFAIL = 0 or −1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF).

7 Accuracy

Not applicable.

8 Further Comments

For problems where sharp changes of behaviour are expected over short intervals it may be advisable to:

cluster the mesh points where sharp changes in behaviour are expected;

maintain fixed points in the mesh using the argument IPMESH to ensure that the remeshing process does not inadvertently remove mesh points from areas of known interest.

In the absence of any other information about the expected behaviour of the solution, using the values suggested in Section 5 for NMESH, IPMESH and MESH is strongly recommended.

9 Example

The following example is used to illustrate the use of continuation, solution on an infinite range, and solution of a system of two differential equations of orders 3 and 2. See also D02TKF, D02TVF, D02TYF and D02TZF, for the illustration of other facilities.

Consider the problem of swirling flow over an infinite stationary disk with a magnetic field along the axis of rotation. See [3] and the references therein. After transforming from a cylindrical coordinate system (r,θ,z), in which the θ component of the corresponding velocity field behaves like r^{-n}, the governing equations are

$$\begin{aligned} f''' + \frac{1}{2}(3-n)ff'' + n(f')^2 + g^2 - sf' &= \gamma^2 \\ g'' + \frac{1}{2}(3-n)fg' + (n-1)gf' - s(g-1) &= 0 \end{aligned}$$

with boundary conditions

$$f(0) = f'(0) = g(0) = 0, \quad f'(\infty) = 0, \quad g(\infty) = \gamma,$$

where s is the magnetic field strength, and γ is the Rossby number.

Some solutions of interest are for $\gamma = 1$, small n and $s \to 0$. An added complication is the infinite range, which we approximate by $[0, L]$. We choose $n = 0.2$ and first solve for $L = 60.0, s = 0.24$ using the initial approximations $f(x) = -x^2e^{-x}$ and $g(x) = 1.0 - e^{-x}$, which satisfy the boundary conditions, on a uniform mesh of 21 points. Simple continuation on the parameters L and s using the values $L = 120.0, s = 0.144$ and then $L = 240.0, s = 0.0864$ is used to compute further solutions. We use the suggested values for NMESH, IPMESH and MESH in the call to D02TXF prior to a continuation call, that is only every second point of the preceding mesh is used.

The equations are first mapped onto $[0, 1]$ to yield

$$\begin{aligned} f''' &= L^3(\gamma^2 - g^2) + L^2sg' - L(\frac{1}{2}(3-n)ff'' + n(g')^2) \\ g'' &= L^2s(g-1) - L(\frac{1}{2}(3-n)fg' + (n-1)f'g). \end{aligned}$$

9.1 Example 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.

```
*     D02TXF Example Program Text
*     Mark 17 Release. NAG Copyright 1995.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, MMAX, NLBC, NRBC, NCOL, MXMESH
      PARAMETER        (NEQ=2,MMAX=3,NLBC=3,NRBC=2,NCOL=6,MXMESH=250)
      INTEGER          LRWORK, LIWORK
      PARAMETER        (LRWORK=MXMESH*(109*NEQ**2+78*NEQ+7),
     +                 LIWORK=MXMESH*(11*NEQ+6))
*     .. Scalars in Common ..
      real             EL, EN, S
*     .. Local Scalars ..
      real             ERMX, XX
      INTEGER          I, IERMX, IFAIL, IJERMX, J, NCONT, NMESH
      LOGICAL          FAILED
*     .. Local Arrays ..
      real             MESH(MXMESH), TOL(NEQ), WORK(LRWORK),
     +                 Y(NEQ,0:MMAX-1)
      INTEGER          IPMESH(MXMESH), IWORK(LIWORK), M(NEQ)
*     .. External Subroutines ..
      EXTERNAL         D02TKF, D02TVF, D02TXF, D02TYF, D02TZF, FFUN,
     +                 FJAC, GAFUN, GAJAC, GBFUN, GBJAC, GUESS
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           /PROBS/EN, S, EL
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02TXF Example Program Results'
      WRITE (NOUT,*)
      NMESH = 21
      MESH(1) = 0.0e0
      IPMESH(1) = 1
      DO 20 I = 2, NMESH - 1
         MESH(I) = real(I-1)/real(NMESH-1)
```

```
         IPMESH(I) = 2
   20 CONTINUE
      IPMESH(NMESH) = 1
      MESH(NMESH) = 1.0e0
      M(1) = 3
      M(2) = 2
      TOL(1) = 1.0e-5
      TOL(2) = TOL(1)
      IFAIL = 0
      CALL D02TVF(NEQ,M,NLBC,NRBC,NCOL,TOL,MXMESH,NMESH,MESH,IPMESH,
     +            WORK,LRWORK,IWORK,LIWORK,IFAIL)
*     Initialize number of continuation steps
      NCONT = 3
*     Initialize problem dependent parameters
      EL = 6.0e1
      S = 0.24e0
      EN = 0.2e0
      DO 80 J = 1, NCONT
         WRITE (NOUT,99997) TOL(1), EL, S
         IFAIL = -1
*     Solve
         CALL D02TKF(FFUN,FJAC,GAFUN,GBFUN,GAJAC,GBJAC,GUESS,WORK,IWORK,
     +               IFAIL)
         FAILED = IFAIL .NE. 0
         IFAIL = 0
*     Extract mesh
         CALL D02TZF(MXMESH,NMESH,MESH,IPMESH,ERMX,IERMX,IJERMX,WORK,
     +               IWORK,IFAIL)
         WRITE (NOUT,99996) NMESH, ERMX, IERMX, IJERMX
         IF (FAILED) GO TO 100
*     Print solution components on mesh
         WRITE (NOUT,99999)
         DO 40 I = 1, 16
            XX = real(I-1)*2.0e0/EL
            CALL D02TYF(XX,Y,NEQ,MMAX,WORK,IWORK,IFAIL)
            WRITE (NOUT,99998) XX*EL, Y(1,0), Y(2,0)
   40    CONTINUE
         DO 60 I = 1, 10
            XX = (3.0e1+(EL-3.0e1)*real(I)/10.0e0)/EL
            CALL D02TYF(XX,Y,NEQ,MMAX,WORK,IWORK,IFAIL)
            WRITE (NOUT,99998) XX*EL, Y(1,0), Y(2,0)
   60    CONTINUE
*     Select mesh for continuation
         IF (J.LT.NCONT) THEN
            EL = 2.0e0*EL
            S = 0.6e0*S
            NMESH = (NMESH+1)/2
            CALL D02TXF(MXMESH,NMESH,MESH,IPMESH,WORK,IWORK,IFAIL)
         END IF
   80 CONTINUE
  100 CONTINUE
      STOP
*
99999 FORMAT (/' Solution on original interval:',/'    ',' x        f',
     +       '        g')
99998 FORMAT (' ',F8.2,2F11.4)
99997 FORMAT (//' Tolerance = ',e8.1,'  L = ',F8.3,'  S = ',F6.4)
99996 FORMAT (/' Used a mesh of ',I4,' points',/' Maximum error = ',
```

```
     +       e10.2,'  in interval ',I4,' for component ',I4)
      END
      SUBROUTINE FFUN(X,Y,NEQ,M,F)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              F(NEQ), Y(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Scalars in Common ..
      real              EL, EN, S
*     .. Common blocks ..
      COMMON            /PROBS/EN, S, EL
*     .. Executable Statements ..
      F(1) = EL**3*(1.0e0-Y(2,0)**2) + EL**2*S*Y(1,1) -
     +       EL*(0.5e0*(3.0e0-EN)*Y(1,0)*Y(1,2)+EN*Y(1,1)**2)
      F(2) = EL**2*S*(Y(2,0)-1.0e0) - EL*(0.5e0*(3.0e0-EN)*Y(1,0)*Y(2,1)
     +       +(EN-1.0e0)*Y(1,1)*Y(2,0))
      RETURN
      END
      SUBROUTINE FJAC(X,Y,NEQ,M,DF)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              DF(NEQ,NEQ,0:*), Y(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Scalars in Common ..
      real              EL, EN, S
*     .. Common blocks ..
      COMMON            /PROBS/EN, S, EL
*     .. Executable Statements ..
      DF(1,2,0) = -2.0e0*EL**3*Y(2,0)
      DF(1,1,0) = -EL*0.5e0*(3.0e0-EN)*Y(1,2)
      DF(1,1,1) = EL**2*S - EL*2.0e0*EN*Y(1,1)
      DF(1,1,2) = -EL*0.5e0*(3.0e0-EN)*Y(1,0)
      DF(2,2,0) = EL**2*S - EL*(EN-1.0e0)*Y(1,1)
      DF(2,2,1) = -EL*0.5e0*(3.0e0-EN)*Y(1,0)
      DF(2,1,0) = -EL*0.5e0*(3.0e0-EN)*Y(2,1)
      DF(2,1,1) = -EL*(EN-1.0e0)*Y(2,0)
      RETURN
      END
      SUBROUTINE GAFUN(YA,NEQ,M,NLBC,GA)
*     .. Scalar Arguments ..
      INTEGER           NEQ, NLBC
*     .. Array Arguments ..
      real              GA(NLBC), YA(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Executable Statements ..
      GA(1) = YA(1,0)
      GA(2) = YA(1,1)
      GA(3) = YA(2,0)
      RETURN
      END
      SUBROUTINE GBFUN(YB,NEQ,M,NRBC,GB)
*     .. Scalar Arguments ..
      INTEGER           NEQ, NRBC
```

```
*     .. Array Arguments ..
      real              GB(NRBC), YB(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Executable Statements ..
      GB(1) = YB(1,1)
      GB(2) = YB(2,0) - 1.0e0
      RETURN
      END
      SUBROUTINE GAJAC(YA,NEQ,M,NLBC,DGA)
*     .. Scalar Arguments ..
      INTEGER           NEQ, NLBC
*     .. Array Arguments ..
      real              DGA(NLBC,NEQ,0:*), YA(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Executable Statements ..
      DGA(1,1,0) = 1.0e0
      DGA(2,1,1) = 1.0e0
      DGA(3,2,0) = 1.0e0
      RETURN
      END
      SUBROUTINE GBJAC(YB,NEQ,M,NRBC,DGB)
*     .. Scalar Arguments ..
      INTEGER           NEQ, NRBC
*     .. Array Arguments ..
      real              DGB(NRBC,NEQ,0:*), YB(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Executable Statements ..
      DGB(1,1,1) = 1.0e0
      DGB(2,2,0) = 1.0e0
      RETURN
      END
      SUBROUTINE GUESS(X,NEQ,M,Z,DMVAL)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              DMVAL(NEQ), Z(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Scalars in Common ..
      real              EL, EN, S
*     .. Local Scalars ..
      real              EX, EXPMX
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Common blocks ..
      COMMON            /PROBS/EN, S, EL
*     .. Executable Statements ..
      EX = X*EL
      EXPMX = EXP(-EX)
      Z(1,0) = -EX**2*EXPMX
      Z(1,1) = (-2.0e0*EX+EX**2)*EXPMX
      Z(1,2) = (-2.0e0+4.0e0*EX-EX**2)*EXPMX
      Z(2,0) = 1.0e0 - EXPMX
      Z(2,1) = EXPMX
      DMVAL(1) = (6.0e0-6.0e0*EX+EX**2)*EXPMX
      DMVAL(2) = -EXPMX
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
D02TXF Example Program Results



Tolerance =  0.1E-04  L =   60.000  S = 0.2400

Used a mesh of   21 points
Maximum error =   0.27E-07  in interval    7 for component    1

Solution on original interval:
     x         f          g
    0.00     0.0000     0.0000
    2.00    -0.9769     0.8011
    4.00    -2.0900     1.1459
    6.00    -2.6093     1.2389
    8.00    -2.5498     1.1794
   10.00    -2.1397     1.0478
   12.00    -1.7176     0.9395
   14.00    -1.5465     0.9206
   16.00    -1.6127     0.9630
   18.00    -1.7466     1.0068
   20.00    -1.8286     1.0244
   22.00    -1.8338     1.0185
   24.00    -1.7956     1.0041
   26.00    -1.7582     0.9940
   28.00    -1.7445     0.9926
   30.00    -1.7515     0.9965
   33.00    -1.7695     1.0019
   36.00    -1.7730     1.0018
   39.00    -1.7673     0.9998
   42.00    -1.7645     0.9993
   45.00    -1.7659     0.9999
   48.00    -1.7672     1.0002
   51.00    -1.7671     1.0001
   54.00    -1.7666     0.9999
   57.00    -1.7665     0.9999
   60.00    -1.7666     1.0000


Tolerance =  0.1E-04  L =  120.000  S = 0.1440

Used a mesh of   21 points
Maximum error =   0.69E-05  in interval    7 for component    2

Solution on original interval:
     x         f          g
    0.00     0.0000     0.0000
    2.00    -1.1406     0.7317
    4.00    -2.6531     1.1315
    6.00    -3.6721     1.3250
    8.00    -4.0539     1.3707
   10.00    -3.8285     1.3003
   12.00    -3.1339     1.1407
```

```
   14.00     -2.2469      0.9424
   16.00     -1.6146      0.8201
   18.00     -1.5472      0.8549
   20.00     -1.8483      0.9623
   22.00     -2.1761      1.0471
   24.00     -2.3451      1.0778
   26.00     -2.3236      1.0600
   28.00     -2.1784      1.0165
   30.00     -2.0214      0.9775
   39.00     -2.1109      1.0155
   48.00     -2.0362      0.9931
   57.00     -2.0709      1.0023
   66.00     -2.0588      0.9995
   75.00     -2.0616      1.0000
   84.00     -2.0615      1.0001
   93.00     -2.0611      0.9999
  102.00     -2.0614      1.0000
  111.00     -2.0613      1.0000
  120.00     -2.0613      1.0000


Tolerance =  0.1E-04  L =  240.000  S = 0.0864

Used a mesh of   81 points
Maximum error =   0.33E-06  in interval   19 for component    2

Solution on original interval:
     x          f           g
    0.00      0.0000      0.0000
    2.00     -1.2756      0.6404
    4.00     -3.1604      1.0463
    6.00     -4.7459      1.3011
    8.00     -5.8265      1.4467
   10.00     -6.3412      1.5036
   12.00     -6.2862      1.4824
   14.00     -5.6976      1.3886
   16.00     -4.6568      1.2263
   18.00     -3.3226      1.0042
   20.00     -2.0328      0.7718
   22.00     -1.4035      0.6943
   24.00     -1.6603      0.8218
   26.00     -2.2975      0.9928
   28.00     -2.8661      1.1139
   30.00     -3.1641      1.1641
   51.00     -2.5307      1.0279
   72.00     -2.3520      0.9919
   93.00     -2.3674      0.9975
  114.00     -2.3799      1.0003
  135.00     -2.3800      1.0002
  156.00     -2.3792      1.0000
  177.00     -2.3791      1.0000
  198.00     -2.3792      1.0000
  219.00     -2.3792      1.0000
  240.00     -2.3792      1.0000
```

D02TYF – NAG Fortran Library Routine Document

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

D02TYF interpolates on the solution of a general two point boundary value problem computed by D02TKF.

2 Specification

```
SUBROUTINE D02TYF(X, Y, NEQ, MMAX, RWORK, IWORK, IFAIL)
INTEGER          NEQ, MMAX, IWORK(*), IFAIL
real             X, Y(NEQ,0:MMAX-1), RWORK(*)
```

3 Description

D02TYF and its associated routines (D02TVF, D02TKF, D02TXF and D02TZF) solve the two point boundary value problem for a nonlinear mixed order system of ordinary differential equations

$$\begin{aligned} y_1^{(m_1)}(x) &= f_1(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\ y_2^{(m_2)}(x) &= f_2(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\ &\cdots \\ y_n^{(m_n)}(x) &= f_n(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \end{aligned}$$

over an interval $[a,b]$ subject to p (>0) nonlinear boundary conditions at a and q (>0) nonlinear boundary conditions at b, where $p+q=\sum_1^n m_i$. Note that $y_i^{(m)}(x)$ is the m-th derivative of the i-th solution component. Hence $y_i^{(0)}(x)=y_i(x)$. The left boundary conditions at a are defined as

$$g_i(z(y(a))) = 0, \quad i = 1, 2, \ldots, p,$$

and the right boundary conditions at b as

$$\bar{g}_j(z(y(b))) = 0, \quad j = 1, 2, \ldots, q,$$

where $y = (y_1, y_2, \ldots, y_n)$ and

$$z(y(x)) = (y_1(x), y_1^{(1)}(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)).$$

First, D02TVF must be called to specify the initial mesh, error requirements and other details. Then, D02TKF can be used to solve the boundary value problem. After successful computation, D02TZF can be used to ascertain details about the final mesh and other details of the solution procedure, and D02TYF can be used to compute the approximate solution anywhere on the interval $[a,b]$ using interpolation.

The routines are based on modified versions of the codes COLSYS and COLNEW, [2] and [1]. A comprehensive treatment of the numerical solution of boundary value problems can be found in [3] and [5].

4 References

[1] Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

[2] Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

[3] Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice Hall, Englewood Cliffs, NJ

[4] Grossman C (1992) Enclosures of the solution of the Thomas-Fermi equation by monotone discretization *J. Comput. Phys.* **98** 26–32

[5] Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

5 Parameters

1: X — ***real*** *Input*

On entry: the independent variable, x.

Constraint: $a \leq \mathrm{X} \leq b$.

2: Y(NEQ,0:MMAX−1) — ***real*** array *Output*

On exit: Y(i,j) contains an approximation to $y_i^{(j)}(x)$, for $i = 1,2,\ldots,\mathrm{NEQ}$, $j = 0,1,\ldots,m_i - 1$. The remaining elements of Y (where $m_i < \mathrm{MMAX}$) are initialized to 0.0.

3: NEQ — INTEGER *Input*

On entry: the number of differential equations.

Constraint: NEQ must be the same value as supplied to D02TVF.

4: MMAX — INTEGER *Input*

On entry: the maximal order of the differential equations, $\max(m_i)$, for $i = 1,2,\ldots,\mathrm{NEQ}$.

Constraint: MMAX must contain the maximum value of the components of the argument M as supplied to D02TVF.

5: RWORK(∗) — ***real*** array *Input/Output*

On entry: this must be the same array as supplied to D02TKF and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated routines.

6: IWORK(∗) — INTEGER array *Input/Output*

On entry: this must be the same array as supplied to D02TKF and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated routines.

7: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit**. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Errors 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 specified by the routine:

IFAIL = 1

On entry, an invalid value for NEQ, MMAX ($\neq \max(m_i)$ for some i) or X (outside the range $[a, b]$) was detected, or an invalid call to D02TYF was made, for example without a previous call to the solver routine D02TKF. If on entry IFAIL = 0 or −1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF).

IFAIL = 2

The solver routine D02TKF did not converge to a solution or did not satisfy the error requirements. The last solution computed by D02TKF, for which convergence was obtained, has been used for interpolation by D02TYF. The results returned by D02TYF should be treated with extreme caution as regarding either their quality or accuracy. See Section 8.

7 Accuracy

If D02TYF returns the value IFAIL = 0, the computed values of the solution components y_i should be of similar accuracy to that specified by the argument TOLS of D02TVF. Note that during the solution process the error in the derivatives $y_i^{(j)}, j = 1, 2, \ldots, m_i - 1$ has not been controlled and that the derivative values returned by D02TYF are computed via differentiation of the piecewise polynomial approximation to y_i. See also Section 8.

8 Further Comments

If D02TYF returns the value IFAIL = 2, and the solver routine D02TKF returned IFAIL = 5, then the accuracy of the interpolated values may be proportional to the quantity ERMX as returned by D02TZF.

If D02TKF returned any other non-zero value for IFAIL, then nothing can be said regarding either the quality or accuracy of the values computed by D02TYF.

9 Example

The following example is used to illustrate that a system with singular coefficients can be treated without modification of the system definition. See also D02TKF, D02TVF, D02TXF and D02TZF, for the illustration of other facilities.

Consider the Thomas-Fermi equation used in the investigation of potentials and charge densities of ionized atoms. See [4], for example, and the references therein. The equation is

$$y'' = x^{-1/2} y^{3/2}$$

with boundary conditions

$$y(0) = 1, \quad y(a) = 0, \quad a > 0.$$

The coefficient $x^{-1/2}$ implies a singularity at the left hand boundary $x = 0$.

We use the initial approximation $y(x) = 1 - x/a$, which satisfies the boundary conditions, on a uniform mesh of six points. For illustration we choose $a = 1$, as in [4]. Note that in the subroutines FFUN and FJAC we have taken the precaution of setting the function value and Jacobian value to 0.0 in case a value of y becomes negative, although starting from our initial solution profile this proves unnecessary during the solution phase. Of course the true solution $y(x)$ is positive for all $x < a$.

9.1 Example 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.

```
*     D02TYF Example Program Text
*     Mark 17 Release. NAG Copyright 1995.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, MMAX, NLBC, NRBC, NCOL, MXMESH
      PARAMETER        (NEQ=1,MMAX=2,NLBC=1,NRBC=1,NCOL=4,MXMESH=100)
      INTEGER          LRWORK, LIWORK
      PARAMETER        (LRWORK=MXMESH*(109*NEQ**2+78*NEQ+7),
     +                 LIWORK=MXMESH*(11*NEQ+6))
*     .. Scalars in Common ..
      real             A
*     .. Local Scalars ..
      real             AINC, ERMX, XX
      INTEGER          I, IERMX, IFAIL, IJERMX, NMESH
      LOGICAL          FAILED
*     .. Local Arrays ..
      real             MESH(MXMESH), TOL(NEQ), WORK(LRWORK),
     +                 Y(NEQ,0:MMAX-1)
      INTEGER          IPMESH(MXMESH), IWORK(LIWORK), M(NEQ)
*     .. External Subroutines ..
      EXTERNAL         D02TKF, D02TVF, D02TYF, D02TZF, FFUN, FJAC,
     +                 GAFUN, GAJAC, GBFUN, GBJAC, GUESS
*     .. Intrinsic Functions ..
      INTRINSIC        real
*     .. Common blocks ..
      COMMON           /PROBS/A
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02TYF Example Program Results'
      WRITE (NOUT,*)
      A = 1.0e0
      NMESH = 6
      AINC = A/real(NMESH-1)
      MESH(1) = 0.0e0
      IPMESH(1) = 1
      DO 20 I = 2, NMESH - 1
         MESH(I) = real(I-1)*AINC
         IPMESH(I) = 2
   20 CONTINUE
      MESH(NMESH) = A
      IPMESH(NMESH) = 1
      TOL(1) = 1.0e-5
      M(1) = 2
      IFAIL = 0
      CALL D02TVF(NEQ,M,NLBC,NRBC,NCOL,TOL,MXMESH,NMESH,MESH,IPMESH,
     +            WORK,LRWORK,IWORK,LIWORK,IFAIL)
      WRITE (NOUT,99997) TOL(1), A
      IFAIL = -1
      CALL D02TKF(FFUN,FJAC,GAFUN,GBFUN,GAJAC,GBJAC,GUESS,WORK,IWORK,
     +            IFAIL)
      FAILED = IFAIL .NE. 0
      IFAIL = 0
      CALL D02TZF(MXMESH,NMESH,MESH,IPMESH,ERMX,IERMX,IJERMX,WORK,IWORK,
     +            IFAIL)
```

```
      WRITE (NOUT,99996) NMESH, ERMX, IERMX, IJERMX,
     +  (I,IPMESH(I),MESH(I),I=1,NMESH)
      IF ( .NOT. FAILED) THEN
         AINC = 0.1e0*A
         WRITE (NOUT,99999)
         DO 40 I = 1, 11
            XX = real(I-1)*AINC
            CALL D02TYF(XX,Y,NEQ,MMAX,WORK,IWORK,IFAIL)
            WRITE (NOUT,99998) XX, Y(1,0), Y(1,1)
   40    CONTINUE
      END IF
      STOP
*
99999 FORMAT (/' Computed solution',/'       x     solution   derivati',
     +       've')
99998 FORMAT (' ',F8.2,2F11.5)
99997 FORMAT (//' Tolerance = ',e8.1,' A = ',F8.2)
99996 FORMAT (/' Used a mesh of ',I4,' points',/' Maximum error = ',
     +       e10.2,'  in interval ',I4,' for component ',I4,//' Mesh p',
     +       'oints:',/4(I4,'(',I1,')',e11.4))
      END
      SUBROUTINE FFUN(X,Y,NEQ,M,F)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              F(NEQ), Y(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      IF (Y(1,0).LE.0.0e0) THEN
         F(1) = 0.0e0
         PRINT *, ' F'
      ELSE
         F(1) = (Y(1,0))**1.5e0/SQRT(X)
      END IF
      RETURN
      END
      SUBROUTINE FJAC(X,Y,NEQ,M,DF)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              DF(NEQ,NEQ,0:*), Y(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      IF (Y(1,0).LE.0.0e0) THEN
         DF(1,1,0) = 0.0e0
         PRINT *, ' JAC'
      ELSE
         DF(1,1,0) = 1.5e0*SQRT(Y(1,0))/SQRT(X)
      END IF
      RETURN
      END
      SUBROUTINE GAFUN(YA,NEQ,M,NLBC,GA)
```

```
*     .. Scalar Arguments ..
      INTEGER          NEQ, NLBC
*     .. Array Arguments ..
      real             GA(NLBC), YA(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      GA(1) = YA(1,0) - 1.0e0
      RETURN
      END
      SUBROUTINE GBFUN(YB,NEQ,M,NRBC,GB)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NRBC
*     .. Array Arguments ..
      real             GB(NRBC), YB(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      GB(1) = YB(1,0)
      RETURN
      END
      SUBROUTINE GAJAC(YA,NEQ,M,NLBC,DGA)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NLBC
*     .. Array Arguments ..
      real             DGA(NLBC,NEQ,0:*), YA(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      DGA(1,1,0) = 1.0e0
      RETURN
      END
      SUBROUTINE GBJAC(YB,NEQ,M,NRBC,DGB)
*     .. Scalar Arguments ..
      INTEGER          NEQ, NRBC
*     .. Array Arguments ..
      real             DGB(NRBC,NEQ,0:*), YB(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Executable Statements ..
      DGB(1,1,0) = 1.0e0
      RETURN
      END
      SUBROUTINE GUESS(X,NEQ,M,Z,DMVAL)
*     .. Scalar Arguments ..
      real             X
      INTEGER          NEQ
*     .. Array Arguments ..
      real             DMVAL(NEQ), Z(NEQ,0:*)
      INTEGER          M(NEQ)
*     .. Scalars in Common ..
      real             A
*     .. Common blocks ..
      COMMON           /PROBS/A
*     .. Executable Statements ..
      Z(1,0) = 1.0e0 - X/A
      Z(1,1) = -1.0e0/A
      DMVAL(1) = 0.0e0
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
D02TYF Example Program Results



Tolerance =  0.1E-04 A =     1.00

Used a mesh of   11 points
Maximum error =   0.31E-05  in interval    1 for component    1

Mesh points:
  1(1) 0.0000E+00   2(3) 0.1000E+00   3(2) 0.2000E+00   4(3) 0.3000E+00
  5(2) 0.4000E+00   6(3) 0.5000E+00   7(2) 0.6000E+00   8(3) 0.7000E+00
  9(2) 0.8000E+00  10(3) 0.9000E+00  11(1) 0.1000E+01

Computed solution
     x      solution   derivative
    0.00    1.00000   -1.84496
    0.10    0.84944   -1.32330
    0.20    0.72721   -1.13911
    0.30    0.61927   -1.02776
    0.40    0.52040   -0.95468
    0.50    0.42754   -0.90583
    0.60    0.33867   -0.87372
    0.70    0.25239   -0.85369
    0.80    0.16764   -0.84248
    0.90    0.08368   -0.83756
    1.00    0.00000   -0.83655
```

D02TZF – NAG Fortran Library Routine Document

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

D02TZF returns information about the solution of a general two point boundary value problem computed by D02TKF.

2 Specification

```
      SUBROUTINE D02TZF(MXMESH, NMESH, MESH, IPMESH, ERMX, IERMX,
     1                  IJERMX, RWORK, IWORK, IFAIL)
      INTEGER           MXMESH, NMESH, IPMESH(MXMESH), IERMX, IJERMX,
     1                  IWORK(*), IFAIL
      real              MESH(MXMESH), ERMX, RWORK(*)
```

3 Description

D02TZF and its associated routines (D02TVF, D02TKF, D02TXF and D02TYF) solve the two point boundary value problem for a nonlinear mixed order system of ordinary differential equations

$$\begin{aligned}
y_1^{(m_1)}(x) &= f_1(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\
y_2^{(m_2)}(x) &= f_2(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}) \\
&\ldots \\
y_n^{(m_n)}(x) &= f_n(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)})
\end{aligned}$$

over an interval $[a, b]$ subject to p (> 0) nonlinear boundary conditions at a and q (> 0) nonlinear boundary conditions at b, where $p + q = \sum_1^n m_i$. Note that $y_i^{(m)}(x)$ is the m-th derivative of the i-th solution component. Hence $y_i^{(0)}(x) = y_i(x)$. The left boundary conditions at a are defined as

$$g_i(z(y(a))) = 0, \quad i = 1, 2, \ldots, p,$$

and the right boundary conditions at b as

$$\bar{g}_j(z(y(b))) = 0, \quad j = 1, 2, \ldots, q,$$

where $y = (y_1, y_2, \ldots, y_n)$ and

$$z(y(x)) = (y_1(x), y_1^{(1)}(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots y_n^{(m_n-1)}(x)).$$

First, D02TVF must be called to specify the initial mesh, error requirements and other details. Then, D02TKF can be used to solve the boundary value problem. After successful computation, D02TZF can be used to ascertain details about the final mesh. D02TYF can be used to compute the approximate solution anywhere on the interval $[a, b]$ using interpolation.

The routines are based on modified versions of the codes COLSYS and COLNEW, [2] and [1] . A comprehensive treatment of the numerical solution of boundary value problems can be found in [3] and [5].

4 References

[1] Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

[2] Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

[3] Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice Hall, Englewood Cliffs, NJ

[4] Cole J D (1968) *Perturbation Methods in Applied Mathematics* Blaisdell, Waltham, Mass.

[5] Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

5 Parameters

1: MXMESH — INTEGER *Input*

On entry: the maximum number of points allowed in the mesh.

Constraint: this must be identical to the value supplied for the argument MXMESH in the prior call to D02TVF.

2: NMESH — INTEGER *Output*

On exit: the number of points in the mesh last used by D02TKF.

3: MESH(MXMESH) — ***real*** array *Output*

On exit: MESH(i) contains the i-th point of the mesh last used by D02TKF, for $i = 1, 2, \ldots,$ NMESH. MESH(1) will contain a and MESH(NMESH) will contain b. The remaining elements of MESH are not initialized.

4: IPMESH(MXMESH) — INTEGER array *Output*

On exit: IPMESH(i) specifies the nature of the point MESH(i), $i = 1, 2, \ldots,$ NMESH, in the final mesh computed by D02TKF.

IPMESH(i) = 1 indicates that the i-th point is a fixed point and was used by the solver prior to an extrapolation-like error test.

IPMESH(i) = 2 indicates that the i-th point was used by the solver prior to an extrapolation-like error test.

IPMESH(i) = 3 indicates that the i-th point was used by the solver only as part of an extrapolation-like error test.

The remaining elements of IPMESH are initialized to -1.

See Section 8 for advice on how these values may be used in conjunction with a continuation process.

5: ERMX — ***real*** *Output*

On exit: an estimate of the maximum error in the solution computed by D02TKF, that is

$$\mathrm{ERMX} = \max \frac{||y_i - v_i||}{(1.0 + ||v_i||)}$$

where v_i is the approximate solution for the i-th solution component. If D02TKF returned successfully with IFAIL = 0, then ERMX will be less than TOLS(IJERMX) where TOLS contains the error requirements as specified in Sections 3 and 5 of the document for D02TVF.

If D02TKF returned with IFAIL = 5, then ERMX will be greater than TOLS(IJERMX).

If D02TKF returned any other value for IFAIL then an error estimate is not available and ERMX is initialized to 0.0.

6: IERMX — INTEGER *Output*

On exit: indicates the mesh sub-interval where the value of ERMX has been computed, that is [MESH(IERMX),MESH(IERMX+1)].

If an estimate of the error is not available then IERMX is initialized to 0.

7: IJERMX — INTEGER *Output*

On exit: indicates the component i (= IJERMX) of the solution for which ERMX has been computed, that is the approximation of y_i on [MESH(IERMX),MESH(IERMX+1)] is estimated to have the largest error of all components y_i over mesh sub-intervals defined by MESH.

If an estimate of the error is not available then IJERMX is initialized to 0.

8: RWORK(*) — ***real*** array *Input/Output*

On entry: this must be the same array as supplied to D02TKF and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated routines.

9: IWORK(*) — INTEGER array *Input/Output*

On entry: this must be the same array as supplied to D02TKF and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated routines.

10: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq$ 0 on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit**. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Errors 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 detected by the routine:

IFAIL = 1

On entry, an illegal value for MXMESH was specified, or an invalid call to D02TZF was made, for example without a previous call to the solver routine D02TKF. If on entry IFAIL = 0 or -1, the precise form of the error will be detailed on the current error message unit (as defined by X04AAF).

IFAIL = 2

The solver routine D02TKF did not converge to a solution or did not satisfy the error requirements. The last mesh computed by D02TKF has been returned by D02TZF. This mesh should be treated with extreme caution as nothing can be said regarding its quality or suitability for any subsequent computation.

7 Accuracy

Not applicable.

8 Further Comments

Note that

if D02TKF returned IFAIL = 0, 4 or 5 then it will always be the case that IPMESH(1) = IPMESH(NMESH) = 1;

if D02TKF returned IFAIL = 0 or 5 then it will always be the case that IPMESH(i) = 3, $i = 2, 4, \ldots,$ NMESH − 1 and IPMESH(i) = 1 or 2, $i = 3, 5, \ldots,$ NMESH − 2.

if D02TKF returned IFAIL = 4 then it will always be the case that IPMESH(i) = 1 or 2, $i = 2, 3, \ldots,$ NMESH − 1.

If D02TZF returns the value IFAIL = 0, then examination of the mesh may provide assistance in determining a suitable starting mesh for D02TVF in any subsequent attempts to solve similar problems.

If the problem being treated by D02TKF is one of a series of related problems (for example, as part of a continuation process), then the values of IPMESH and MESH may be suitable as input parameters to D02TXF. Using the mesh points not involved in the extrapolation error test is usually appropriate. IPMESH and MESH should be passed unchanged to D02TXF but NMESH should be replaced by (NMESH+1)/2.

If D02TZF returns the value IFAIL = 2, nothing can be said regarding the quality of the mesh returned. However, it may be a useful starting mesh for D02TVF in any subsequent attempts to solve the same problem.

If D02TKF returns the value IFAIL = 5, this corresponds to the solver requiring more than MXMESH mesh points to satisfy the error requirements. If MXMESH can be increased and the preceding call to D02TKF was not part, or was the first part, of a continuation process then the values in MESH may provide a suitable mesh with which to initialize a subsequent attempt to solve the same problem. If it is not possible to provide more mesh points then relaxing the error requirements by setting TOL(IJERMX) to ERMX might lead to a successful solution. It may be necessary to reset the other components of TOL. Note that resetting the tolerances can lead to a different sequence of meshes being computed and hence to a different solution being computed.

9 Example

The following example is used to illustrate the use of fixed mesh points, simple continuation and numerical approximation of a Jacobian. See also D02TKF, D02TVF, D02TXF and D02TYF, for the illustration of other facilities.

Consider the Lagerstrom-Cole equation

$$y'' = (y - yy')/\epsilon$$

with the boundary conditions

$$y(0) = \alpha \quad y(1) = \beta,$$

where ϵ is small and positive. The nature of the solution depends markedly on the values of α, β. See [4].

We choose $\alpha = -\frac{1}{3}, \beta = \frac{1}{3}$ for which the solution is known to have corner layers at $x = \frac{1}{3}, \frac{2}{3}$. We choose an initial mesh of seven points $[0.0, 0.15, 0.3, 0.5, 0.7, 0.85, 1.0]$ and ensure that the points $x = 0.3, 0.7$ near the corner layers are fixed, that is the corresponding elements of the array IPMESH are set to 1. First we compute the solution for $\epsilon = 1.0\text{E}{-4}$ using in GUESS the initial approximation $y(x) = \alpha + (\beta - \alpha)x$ which satisifes the boundary conditions. Then we use simple continuation to compute the solution for $\epsilon = 1.0\text{E}{-5}$. We use the suggested values for NMESH, IPMESH and MESH in the call to D02TXF prior to the continuation call, that is only every second point of the preceding mesh is used and the fixed mesh points are retained.

Although the analytic Jacobian for this system is easy to evaluate, for illustration the procedure FJAC uses central differences and calls to FFUN to compute a numerical approximation to the Jacobian.

9.1 Example 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.

```
*     D02TZF Example Program Text
*     Mark 17 Release. NAG Copyright 1995.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NEQ, MMAX, NLBC, NRBC, NCOL, MXMESH
      PARAMETER        (NEQ=1,MMAX=2,NLBC=1,NRBC=1,NCOL=5,MXMESH=50)
      INTEGER          LRWORK, LIWORK
      PARAMETER        (LRWORK=MXMESH*(109*NEQ**2+78*NEQ+7),
     +                 LIWORK=MXMESH*(11*NEQ+6))
*     .. Scalars in Common ..
      real             ALPHA, BETA, EPS
*     .. Local Scalars ..
      real             ERMX
      INTEGER          I, IERMX, IFAIL, IJERMX, J, NMESH
      LOGICAL          FAILED
*     .. Local Arrays ..
      real             MESH(MXMESH), TOL(NEQ), WORK(LRWORK),
     +                 Y(NEQ,0:MMAX-1)
      INTEGER          IPMESH(MXMESH), IWORK(LIWORK), M(NEQ)
*     .. External Subroutines ..
      EXTERNAL         D02TKF, D02TVF, D02TXF, D02TYF, D02TZF, FFUN,
     +                 FJAC, GAFUN, GAJAC, GBFUN, GBJAC, GUESS
*     .. Common blocks ..
      COMMON           /PROBS/EPS, ALPHA, BETA
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02TZF Example Program Results'
      WRITE (NOUT,*)
      NMESH = 7
      MESH(1) = 0.0e0
      MESH(2) = 0.15e0
      MESH(3) = 0.3e0
      MESH(4) = 0.5e0
      MESH(5) = 0.7e0
      MESH(6) = 0.85e0
      MESH(NMESH) = 1.0e0
      IPMESH(1) = 1
      IPMESH(2) = 2
      IPMESH(3) = 1
      IPMESH(4) = 2
      IPMESH(5) = 1
      IPMESH(6) = 2
      IPMESH(NMESH) = 1
      ALPHA = -1.0e0/3.0e0
      BETA = 1.0e0/3.0e0
      TOL(1) = 1.0e-5
      EPS = 1.0e-3
      M(1) = 2
      IFAIL = 0
      CALL D02TVF(NEQ,M,NLBC,NRBC,NCOL,TOL,MXMESH,NMESH,MESH,IPMESH,
     +            WORK,LRWORK,IWORK,LIWORK,IFAIL)
      IFAIL = -1
      DO 40 J = 1, 2
         EPS = 0.1e0*EPS
```

```
         WRITE (NOUT,99997) TOL(1), EPS
         IFAIL = -1
         CALL D02TKF(FFUN,FJAC,GAFUN,GBFUN,GAJAC,GBJAC,GUESS,WORK,IWORK,
     +               IFAIL)
         FAILED = IFAIL .NE. 0
         IFAIL = 0
         CALL D02TZF(MXMESH,NMESH,MESH,IPMESH,ERMX,IERMX,IJERMX,WORK,
     +               IWORK,IFAIL)
         WRITE (NOUT,99996) NMESH, ERMX, IERMX, IJERMX
         IF (FAILED) GO TO 60
         WRITE (NOUT,99999)
         DO 20 I = 1, NMESH, 2
            CALL D02TYF(MESH(I),Y,NEQ,MMAX,WORK,IWORK,IFAIL)
            WRITE (NOUT,99998) MESH(I), Y(1,0), Y(1,1)
   20    CONTINUE
         IF (J.LT.2) THEN
            NMESH = (NMESH+1)/2
            CALL D02TXF(MXMESH,NMESH,MESH,IPMESH,WORK,IWORK,IFAIL)
         END IF
   40 CONTINUE
   60 CONTINUE
      STOP
*
99999 FORMAT (/' Solution and derivative at every second point:',
     +       /'     ',' x          u           u''')
99998 FORMAT (' ',F8.3,2F11.5)
99997 FORMAT (//' Tolerance = ',e8.1,'  EPS = ',e10.3)
99996 FORMAT (/' Used a mesh of ',I4,' points',/' Maximum error = ',
     +       e10.2,'  in interval ',I4,' for component ',I4)
      END
      SUBROUTINE FFUN(X,Y,NEQ,M,F)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              F(NEQ), Y(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Scalars in Common ..
      real              ALPHA, BETA, EPS
*     .. Common blocks ..
      COMMON            /PROBS/EPS, ALPHA, BETA
*     .. Executable Statements ..
      F(1) = (Y(1,0)-Y(1,0)*Y(1,1))/EPS
      RETURN
      END
      SUBROUTINE FJAC(X,Y,NEQ,M,DF)
*     .. Scalar Arguments ..
      real              X
      INTEGER           NEQ
*     .. Array Arguments ..
      real              DF(NEQ,NEQ,0:*), Y(NEQ,0:*)
      INTEGER           M(NEQ)
*     .. Scalars in Common ..
      real              ALPHA, BETA, EPS
*     .. Local Scalars ..
      real              FAC, MACHEP, PTRB
      INTEGER           I, J, K
*     .. Local Arrays ..
```

9.3 Example Results

```
D02TZF Example Program Results



Tolerance =  0.1E-04  EPS =  0.100E-03

Used a mesh of   25 points
Maximum error =   0.21E-05  in interval   16 for component    1

Solution and derivative at every second point:
     x        u          u'
   0.000   -0.33333    1.00000
   0.075   -0.25833    1.00000
   0.150   -0.18333    1.00000
   0.225   -0.10833    1.00002
   0.300   -0.03332    1.00372
   0.400   -0.00001    0.00084
   0.500    0.00000    0.00000
   0.600    0.00001    0.00084
   0.700    0.03332    1.00372
   0.775    0.10833    1.00002
   0.850    0.18333    1.00000
   0.925    0.25833    1.00000
   1.000    0.33333    1.00000


Tolerance =  0.1E-04  EPS =  0.100E-04

Used a mesh of   49 points
Maximum error =   0.21E-05  in interval   32 for component    1

Solution and derivative at every second point:
     x        u          u'
   0.000   -0.33333    1.00014
   0.038   -0.29583    1.00018
   0.075   -0.25833    1.00022
   0.113   -0.22083    1.00029
   0.150   -0.18333    1.00040
   0.188   -0.14583    1.00059
   0.225   -0.10833    1.00098
   0.262   -0.07083    1.00202
   0.300   -0.03333    1.00745
   0.350   -0.00001    0.00354
   0.400    0.00000    0.00000
   0.450    0.00000    0.00000
   0.500    0.00000    0.00000
   0.550    0.00000    0.00000
   0.600    0.00000    0.00000
   0.650    0.00001    0.00354
   0.700    0.03333    1.00745
   0.737    0.07083    1.00202
   0.775    0.10833    1.00098
   0.813    0.14583    1.00059
   0.850    0.18333    1.00040
   0.888    0.22083    1.00029
   0.925    0.25833    1.00022
```

```
0.963    0.29583    1.00018
1.000    0.33333    1.00014
```

D02XJF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02XJF interpolates components of the solution of a system of first-order ordinary differential equations from information provided by the integrators in the subchapter D02M-D02N.

2. Specification

```
      SUBROUTINE D02XJF (XSOL, SOL, M, YSAVE, NEQMAX, NY2DIM, NEQ, X,
     1                   NQU, HU, H, IFAIL)
      INTEGER           M, NEQMAX, NY2DIM, NEQ, NQU, IFAIL
      real              XSOL, SOL(M), YSAVE(NEQMAX,NY2DIM), X, HU, H
```

3. Description

D02XJF evaluates the first m components of the solution of a system of ordinary differential equations at any point using natural polynomial interpolation based on information generated by the integrator. This information must be passed unchanged to D02XJF. D02XJF should not normally be used to extrapolate outside the range of values obtained from the above routines.

4. References

See the Chapter Introduction.

5. Parameters

1: XSOL – ***real***. *Input*

On entry: the point at which the first m components of the solution are to be evaluated. XSOL should not be an extrapolation point, that is XSOL should satisfy $(\text{XSOL}-\text{X})\times\text{HU} \leq 0.0$. Extrapolation is permitted but not recommended.

2: SOL(M) – ***real*** array. *Output*

On exit: the calculated value of the ith component of the solution at XSOL, for $i = 1,2,\ldots,m$.

3: M – INTEGER. *Input*

On entry: the number of components, m, of the solution whose values at XSOL are required. The first M components are evaluated.

Constraint: $1 \leq \text{M} \leq \text{NEQ}$.

4: YSAVE(NEQMAX,NY2DIM) – ***real*** array. *Input*

On entry: the values provided in the parameter YSAVE on return from the integrator.

5: NEQMAX – INTEGER. *Input*

On entry: the value used for the parameter NEQMAX when calling the integrator.

Constraint: $\text{NEQMAX} \geq 1$.

6: NY2DIM – INTEGER. *Input*

On entry: the value used for the parameter NY2DIM when calling the integrator.

Constraint: $\text{NY2DIM} \geq \text{NQU} + 1$.

7: NEQ – INTEGER. *Input*

On entry: the value used for the parameter NEQ when calling the integrator.

Constraint: 1 ≤ NEQ ≤ NEQMAX.

8: X – ***real***. *Input*

On entry: the latest value at which the solution has been computed, as provided in the parameter TCURR on return from the optional output D02NYF.

9: NQU – INTEGER. *Input*

On entry: the order of the method used up to the latest value at which the solution has been computed, as provided in the parameter NQU on return from the optional output D02NYF.

Constraint: NQU ≥ 1.

10: HU – ***real***. *Input*

On entry: the last successful step used, that is the step used in the integration to get to X, as provided in the parameter HU on return from the optional output D02NYF.

11: H – ***real***. *Input*

On entry: the next step size to be attempted in the integration, as provided in the parameter H on return from the optional output D02NYF.

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

If D02XJF is to be used for extrapolation, IFAIL must be set to 1 before entry. It is then essential to test the value of IFAIL on exit for IFAIL = 1 or 2.

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, M < 1,
or NEQ < 1,
or NEQMAX < 1,
or NEQ > NEQMAX,
or M > NEQ,
or NQU < 1,
or NY2DIM < NQU + 1.

IFAIL = 2

On entry, HU = 0.0 or H = 0.0. This error can only occur if H and HU have been changed by the user or possibly if the integrator has failed before calling D02XJF.

IFAIL = 3

D02XJF has been called for extrapolation. Before returning with this error exit, the value of the solution at XSOL is calculated and placed in SOL.

7. Accuracy

The solution values returned will be of a similar accuracy to those computed by the integrator.

8. Further Comments

This routine is that employed for prediction purposes internally by the integrator. It is supplied for purposes of consistency only. Users are recommended to employ the C^1 interpolant provided by D02XKF wherever possible.

9. Example

See the example for routine D02NGF.

D02XKF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02XKF interpolates components of the solution of a system of first-order ordinary differential equations from information provided by the integrators in the subchapter D02M-D02N. It provides C^1 interpolation suitable for general use.

2. Specification

```
      SUBROUTINE D02XKF (XSOL, SOL, M, YSAVE, NEQMAX, NY2DIM, ACOR,
     1                   NEQ, X, NQU, HU, H, IFAIL)
      INTEGER          M, NEQMAX, IW, NEQ, NQU, IFAIL
      real             XSOL, SOL(M), YSAVE(NEQMAX,NY2DIM), ACOR(NEQMAX),
     1                 X, HU, H
```

3. Description

D02XKF evaluates the first m components of the solution of a system of ordinary differential equations at any point using C^1 polynomial interpolation based on information generated by the integrator. This information must be passed unchanged to D02XKF. D02XKF should not normally be used to extrapolate outside the range of values obtained from the above routines.

It may be used with the D02N routines only when the BDF integration method is being employed (setup routine D02NVF), provided the Petzold error test was not selected.

4. References

See the Chapter Introduction.

5. Parameters

1: XSOL – ***real***. *Input*

On entry: the point at which the first m components of the solution are to be evaluated. XSOL should not be an extrapolation point, that is XSOL should satisfy (XSOL−X)×HU ≤ 0.0. Extrapolation is permitted but not recommended.

2: SOL(M) – ***real*** array. *Output*

On exit: the calculated value of the ith component of the solution at XSOL, for $i = 1,2,...,m$.

3: M – INTEGER. *Input*

On entry: the number of components of the solution whose values at XSOL are required. The first m components are evaluated.

Constraint: 1 ≤ M ≤ NEQ.

4: YSAVE(NEQMAX,NY2DIM) – ***real*** array. *Input*

On entry: the values provided in the parameter YSAVE on return from the integrator.

5: NEQMAX – INTEGER. *Input*

On entry: the value used for the parameter NEQMAX when calling the integrator.

Constraint: NEQMAX ≥ 1.

6: NY2DIM – INTEGER. *Input*

On entry: the value used for the parameter NY2DIM when calling the integrator.

Constraint: NY2DIM ≥ NQU + 1.

7: ACOR(NEQMAX) – ***real*** array. *Input*

On entry: the value returned in position (NEQMAX+50+i), for i = 1,2,...,NEQ of the parameter RWORK returned by the integrator. If one of the forward communication D02N routines is being employed and D02XKF is to be used in the user-supplied MONITR routine, then ACOR(i) must contain the value given in position (i,2) of the MONITR argument ACOR, for i = 1,2,...,NEQ.

8: NEQ – INTEGER. *Input*

On entry: the value used for the parameter NEQ when calling the integrator.

Constraint: 1 ≤ NEQ ≤ NEQMAX.

9: X – ***real***. *Input*

On entry: the latest value at which the solution has been computed, as provided in the parameter TCURR on return from the optional output D02NYF.

10: NQU – INTEGER. *Input*

On entry: the order of the method used up to the latest value at which the solution has been computed, as provided in the parameter NQU on return from the optional output D02NYF.

Constraint: NQU ≥ 1.

11: HU – ***real***. *Input*

On entry: the last successful step used, that is the step used in the integration to get to X, as provided in the parameter HU on return from the optional output D02NYF.

12: H – ***real***. *Input*

On entry: the next step size to be attempted in the integration, as provided in the parameter H on return from the optional output D02NYF.

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

If D02XKF is to be used for extrapolation, IFAIL must be set to 1 before entry. It is then essential to test the value of IFAIL on exit for IFAIL = 1 or 2.

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, M < 1,
or NEQ < 1,
or NEQMAX < 1,
or NEQ > NEQMAX,
or M > NEQ,
or NQU < 1,
or NY2DIM < NQU + 1,
or the BDF integrator was not previously used,
or the Petzold error test, if applicable, was used.

IFAIL = 2

On entry, HU = 0.0 or H = 0.0. This error can only occur if H and HU have been changed by the user or possibly if the integrator has failed before calling D02XKF.

IFAIL = 3

D02XKF has been called for extrapolation. Before returning with this error exit, the value of the solution at XSOL is calculated and placed in SOL.

7. Accuracy

The solution values returned will be of a similar accuracy to those computed by the integrator.

8. Further Comments

D02XKF provides a C^1 interpolant and as such is ideal for most applications, for example for tabulation and root-finding. In general D02XKF should be preferred to D02XJF for interpolation as the latter provides only a C^0 interpolant. D02XJF is the natural interpolant employed by the BDF method and it is supplied only to permit the user to reproduce the internal values used by the integrator.

9. Example

See the examples for D02NDF and D02NMF.

D02ZAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D02ZAF calculates the weighted norm of the local error estimate from inside a MONITR routine called from an integrator in the D02M-D02N subchapter.

2. Specification

```
real FUNCTION D02ZAF (NEQ, V, W, IFAIL)
INTEGER         NEQ, IFAIL
real            V(NEQ), W(NEQ)
```

3. Description

This function is for use with the forward communication integrators D02NBF, D02NCF, D02NDF, D02NGF, D02NHF and D02NJF and the reverse communication integrators D02NMF and D02NNF. It must be used only inside the user-supplied routine MONITR (if this option is selected) for the forward communication routines or on the equivalent return for the reverse communication routines. It may be used to evaluate the norm of the scaled local error estimate, $\|v\|$, where the weights used are contained in w and the norm used is as defined by an earlier call to the integrator setup routine (D02MVF, D02NVF or D02NWF). Its use is described under the description of MONITR in the specifications for the forward communication integrators mentioned above.

4. References

None.

5. Parameters

1: NEQ – INTEGER. *Input*

On entry: the number of differential equations, as defined for the integrator being used.

2: V(NEQ) – ***real*** array. *Input*

On entry: the vector, the weighted norm of which is to be evaluated by D02ZAF. V is calculated internally by the integrator being used.

3: W(NEQ) – ***real*** array. *Input*

On entry: the weights, calculated internally by the integrator, to be used in the norm evaluation.

4: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to –1 before entry. **It is then essential to test the value of IFAIL on exit.** To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

The value of the norm would either overflow or is close to overflowing. A value close to the square root of the largest number on the computer is returned.

7. Accuracy

The result is calculated close to ***machine precision*** except in the case when the routine exits with IFAIL = 1.

8. Further Comments

This routine should only be used within the user-supplied MONITR subroutine associated with the integrators in the D02M-D02N subchapter. Its use and only valid calling sequence are fully documented in the description of this MONITR subroutine in the routine documents for the integrators.

9. Example

None.

Chapter D03 – Partial Differential Equations

Note. Please refer to the Users' Note for your implementation to check that a routine is available.

Routine Name	Mark of Introduction	Purpose
D03EAF	7	Elliptic PDE, Laplace's equation, 2-D arbitrary domain
D03EBF	7	Elliptic PDE, solution of finite difference equations by SIP, five-point 2-D molecule, iterate to convergence
D03ECF	8	Elliptic PDE, solution of finite difference equations by SIP for seven-point 3-D molecule, iterate to convergence
D03EDF	12	Elliptic PDE, solution of finite difference equations by a multigrid technique
D03EEF	13	Discretize a second order elliptic PDE on a rectangle
D03FAF	14	Elliptic PDE, Helmholtz equation, 3-D Cartesian co-ordinates
D03MAF	7	Triangulation of a plane region
D03PCF	15	General system of parabolic PDEs, method of lines, finite differences, one space variable
D03PDF	15	General system of parabolic PDEs, method of lines, Chebyshev C^0 collocation, one space variable
D03PEF	16	General system of first order PDEs, method of lines, Keller box discretisation, one space variable
D03PFF	17	General system of convection-diffusion PDEs with source terms in conservative form, method of lines, upwind scheme using numerical flux function based on Riemann solver, one space variable
D03PHF	15	General system of parabolic PDEs, coupled DAEs, method of lines, finite differences, one space variable
D03PJF	15	General system of parabolic PDEs, coupled DAEs, method of lines, Chebyshev C^0 collocation, one space variable
D03PKF	16	General system of first order PDEs, coupled DAEs, method of lines, Keller box discretisation, one space variable
D03PLF	17	General system of convection-diffusion PDEs with source terms in conservative form, coupled DAEs, method of lines, upwind scheme using numerical flux function based on Riemann solver, one space variable
D03PPF	16	General system of parabolic PDEs, coupled DAEs, method of lines, finite differences, remeshing, one space variable
D03PRF	16	General system of first order PDEs, coupled DAEs, method of lines, Keller box discretisation, remeshing, one space variable
D03PSF	17	General system of convection-diffusion PDEs with source terms in conservative form, coupled DAEs, method of lines, upwind scheme using numerical flux function based on Riemann solver, remeshing, one space variable
D03PUF	17	Roe's approximate Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PVF	17	Osher's approximate Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PWF	18	Modified HLL Riemann solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PXF	18	Exact Riemann Solver for Euler equations in conservative form, for use with D03PFF, D03PLF and D03PSF
D03PYF	15	PDEs, spatial interpolation with D03PDF or D03PJF
D03PZF	15	PDEs, spatial interpolation with D03PCF, D03PEF, D03PFF, D03PHF, D03PKF, D03PLF, D03PPF, D03PRF or D03PSF

D03RAF	18	General system of second order PDEs, method of lines, finite differences, remeshing, two space variables, rectangular region
D03RBF	18	General system of second order PDEs, method of lines, finite differences, remeshing, two space variables, rectilinear region
D03RYF	18	Check initial grid data in D03RBF
D03RZF	18	Extract grid data from D03RBF
D03UAF	7	Elliptic PDE, solution of finite difference equations by SIP, five-point 2-D molecule, one iteration
D03UBF	8	Elliptic PDE, solution of finite difference equations by SIP, seven-point 3-D molecule, one iteration

Chapter D03

Partial Differential Equations

Contents

1 Scope of the Chapter

This chapter is concerned with the numerical solution of partial differential equations.

2 Background to the Problems

The definition of a partial differential equation problem includes not only the equation itself but also the domain of interest and appropriate subsidiary conditions. Indeed, partial differential equations are usually classified as elliptic, hyperbolic or parabolic according to the form of the equation **and** the form of the subsidiary conditions which must be assigned to produce a well-posed problem. Ultimately it is hoped that this chapter will contain routines for the solution of equations of each of these types together with automatic mesh generation routines and other utility routines particular to the solution of partial differential equations. The routines in this chapter will often call upon routines from other chapters, such as Chapter F04 (Simultaneous Linear Equations) and Chapter D02 (Ordinary Differential Equations).

The classification of partial differential equations is easily described in the case of **linear** equations of the **second order** in two independent variables, i.e., equations of the form

$$au_{xx} + 2bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0, \tag{1}$$

where a, b, c, d, e, f and g are functions of x and y only. Equation (1) is called elliptic, hyperbolic or parabolic according to whether $ac - b^2$ is positive, negative or zero, respectively. Useful definitions of the concepts of elliptic, hyperbolic and parabolic character can also be given for differential equations in more than two independent variables, for systems and for nonlinear differential equations.

For **elliptic** equations, of which Laplace's equation

$$u_{xx} + u_{yy} = 0 \tag{2}$$

is the simplest example of second order, the subsidiary conditions take the form of **boundary** conditions, i.e., conditions which provide information about the solution at all points of a **closed** boundary. For example, if equation (2) holds in a plane domain D bounded by a contour C, a solution u may be sought subject to the condition

$$u = f \quad \text{on } C, \tag{3}$$

where f is a given function. The condition (3) is known as a Dirichlet boundary condition. Equally common is the Neumann boundary condition

$$u' = g \quad \text{on } C, \tag{4}$$

which is one form of a more general condition

$$u' + fu = g \quad \text{on } C, \tag{5}$$

where u' denotes the derivative of u normal to the contour C, and f and g are given functions. Provided that f and g satisfy certain restrictions, condition (5) yields a well-posed **boundary value problem** for Laplace's equation. In the case of the Neumann problem, one further piece of information, e.g. the value of u at a particular point, is necessary for uniqueness of the solution. Boundary conditions similar to the above are applicable to more general second-order elliptic equations, whilst two such conditions are required for equations of fourth order.

For **hyperbolic** equations, the wave equation

$$u_{tt} - u_{xx} = 0 \tag{6}$$

is the simplest example of second order. It is equivalent to a first-order system

$$u_t - v_x = 0, \qquad v_t - u_x = 0. \tag{7}$$

The subsidiary conditions may take the form of **initial** conditions, i.e., conditions which provide information about the solution at points on a suitable **open** boundary. For example, if equation (6) is satisfied for $t > 0$, a solution u may be sought such that

$$u(x,0) = f(x), \qquad u_t(x,0) = g(x), \tag{8}$$

where f and g are given functions. This is an example of an **initial value problem**, sometimes known as Cauchy's problem.

For **parabolic** equations, of which the heat conduction equation

$$u_t - u_{xx} = 0 \tag{9}$$

is the simplest example, the subsidiary conditions always include some of **initial** type and may also include some of **boundary** type. For example, if equation (9) is satisfied for $t > 0$ and $0 < x < 1$, a solution u may be sought such that

$$u(x,0) = f(x), \qquad 0 < x < 1, \tag{10}$$

and

$$u(0,t) = 0, \qquad u(1,t) = 1, \qquad t > 0. \tag{11}$$

This is an example of a mixed **initial/boundary value problem**.

For all types of partial differential equations, finite difference methods (Mitchell and Griffiths [6]) and finite element methods (Wait and Mitchell [11]) are the most common means of solution and such methods obviously feature prominently either in this chapter or in the companion NAG Finite Element Library. Some of the utility routines in this chapter are concerned with the solution of the large sparse systems of equations which arise from finite difference and finite element methods.

Alternative methods of solution are often suitable for special classes of problems. For example, the method of characteristics is the most common for hyperbolic equations involving time and one space dimension (Smith [9]). The method of lines (see Mikhlin and Smolitsky [5]) may be used to reduce a parabolic equation to a (stiff) system of ordinary differential equations, which may be solved by means of routines from Chapter D02 – Ordinary Differential Equations. Similarly, integral equation or boundary element methods (Jaswon and Symm [3]) are frequently used for elliptic equations. Typically, in the latter case, the solution of a boundary value problem is represented in terms of certain boundary functions by an integral expression which satisfies the differential equation throughout the relevant domain. The boundary functions are obtained by applying the given boundary conditions to this representation. Implementation of this method necessitates discretization of only the boundary of the domain, the dimensionality of the problem thus being effectively reduced by one. The boundary conditions yield a full system of simultaneous equations, as opposed to the sparse systems yielded by finite difference and finite element methods, but the full system is usually of much lower order. Solution of this system yields the boundary functions, from which the solution of the problem may be obtained, by quadrature, as and where required.

3 Recommendations on Choice and Use of Available Routines

Note. Refer to the Users' Note for your implementation to check that a routine is available.

The choice of routine will depend first of all upon the type of partial differential equation to be solved. At present no special allowances are made for problems with boundary singularities such as may arise at corners of domains or at points where boundary conditions change. For such problems results should be treated with caution.

Users may wish to construct their own partial differential equation solution software for problems not solvable by the routines described in Section 3.1 to Section 3.6 below. In such cases users can employ appropriate routines from the Linear Algebra Chapters to solve the resulting linear systems; see Section 3.8 for further details.

3.1 Elliptic Equations

The routine D03EAF solves Laplace's equation in two dimensions, equation (2), by an integral equation method. This routine is applicable to an arbitrary domain bounded internally or externally by one or more closed contours, when the value of either the unknown function u or its normal derivative u' is given at each point of the boundary.

The routines D03EBF and D03ECF solve a system of simultaneous algebraic equations of five-point and seven-point molecule form (Mikhlin and Smolitsky [5]) on two-dimensional and three-dimensional topologically-rectangular meshes respectively, using Stone's Strongly Implicit Procedure (SIP). These routines, which make repeated calls of the utility routines D03UAF and D03UBF respectively, may be used to solve any boundary value problem whose finite difference representation takes the appropriate form.

The routine D03EDF solves a system of seven-point difference equations in a rectangular grid (in two dimensions), using the multigrid iterative method. The equations are supplied by the user, and the seven-point form allows cross-derivative terms to be represented (see Mitchell and Griffiths [6]). The method is particularly efficient for large systems of equations with diagonal dominance and should be preferred to D03EBF whenever it is appropriate for the solution of the problem.

The routine D03EEF discretizes a second-order equation on a two-dimensional rectangular region using finite differences and a seven-point molecule. The routine allows for cross-derivative terms, Dirichlet, Neumann or mixed boundary conditions, and either central or upwind differences. The resulting seven-diagonal difference equations are in a form suitable for passing directly to the multigrid routine D03EDF, although other solution methods could just as easily be used.

The routine D03FAF, based on the routine HW3CRT from FISHPACK (Swarztrauber and Sweet [10]), solves the Helmholtz equation in a three-dimensional cuboidal region, with any combination of Dirichlet, Neumann or periodic boundary conditions. The method used is based on the fast Fourier transform algorithm, and is likely to be particularly efficient on vector-processing machines.

3.2 Hyperbolic Equations

See Section 3.6.

3.3 Parabolic Equations

There are five routines available for solving parabolic equations in one space dimension: D03PCF, D03PDF, D03PHF, D02PJF and D03PPF. Equations may include nonlinear terms but the true derivative u_t should occur linearly and equations should usually contain a second-order space derivative u_{xx}. There are certain restrictions on the coefficients to try to ensure that the problems posed can be solved by the above routines.

The method of solution is to discretize the space derivatives using finite differences or collocation, and to solve the resulting system of ordinary differential equations using a 'stiff' solver.

D03PCF and D03PDF can solve a system of parabolic (and possibly elliptic) equations of the form

$$\sum_{j=1}^{n} P_{ij}(x,t,U,U_x)\frac{\partial U_j}{\partial t} + Q_i(x,t,U,U_x) = x^{-m}\frac{\partial}{\partial x}(x^m R_i(x,t,U,U_x)),$$

where $i = 1,2,\ldots,n$, $a \leq x \leq b$, $t \geq t_0$.

The parameter m allows the routine to handle different coordinate systems easily (Cartesian, cylindrical polars and spherical polars). D03PCF uses a finite differences spatial discretization and D03PDF uses a collocation spatial discretization.

D03PHF and D03PJF are similar to D03PCF and D03PDF respectively, except that they provide scope for coupled differential-algebraic systems. This extended functionality allows for the solution of more complex and more general problems, e.g. periodic boundary conditions and integro-differential equations.

D03PPF is similar to D03PHF but allows remeshing to take place in the spatial direction. This facility can be very useful when the nature of the solution in the spatial direction varies considerably over time.

For parabolic systems in two space dimensions see Section 3.5.

3.4 First Order Systems in One Space Dimension

There are three routines available for solving systems of first-order partial differential equations: D03PEF, D03PKF and D03PRF. Equations may include nonlinear terms but the time derivative should occur linearly. There are certain restrictions on the coefficients to ensure that the problems posed can be solved by the above routines.

The method of solution is to discretize the space derivatives using the Keller box scheme and to solve the resulting system of ordinary differential equations using a 'stiff' solver.

D03PEF is designed to solve a system of the form

$$\sum_{j=1}^{n} P_{ij}(x,t,U,U_x)\frac{\partial U_j}{\partial t} + Q_i(x,t,U,U_x) = 0,$$

where $i = 1,2,\ldots,n$, $a \leq x \leq b$, $t \geq t_0$.

D03PKF is similar to D03PEF except that it provides scope for coupled differential algebraic systems. This extended functionality allows for the solution of more complex problems.

D03PRF is similar to D03PKF but allows remeshing to take place in the spatial direction. This facility can be very useful when the nature of the solution in the spatial direction varies considerably over time.

D03PEF, D03PKF or D03PRF may also be used to solve systems of higher or mixed order partial differential equations which have been reduced to first order. Note that in general these routines are unsuitable for hyperbolic first-order equations, for which an appropriate upwind discretization scheme should be used (see Section 3.6 for example).

3.5 Second Order Systems in Two Space Dimensions

There are two routines available for solving nonlinear second order time-dependent systems in two space dimensions: D03RAF and D03RBF. These reoutines are formally applicable to the general nonlinear system:

$$F_j(t,x,y,u,u_t,u_x,u_y,u_{xx},u_{xy},u_{yy}) = 0$$

where $j = 1,2,\ldots,$NPDE, $(x,y) \in \Omega$, $t_0 \leq t \leq t_{out}$. However, they should not be used to solve purely hyperbolic systems, or time-independent problems.

D03RAF solves the nonlinear system in a rectangular domain, while D03RBF solves in a rectilinear region, i.e., a domain bounded by perpendicular straight lines.

Both routines use the method of lines and solve the resulting system of ordinary differential equations using a backward differentiation formula (BDF) method, modified Newton method, and BiCGSTAB iterative linear solver. Local uniform grid refinement is used to improve accuracy.

Utility routines D03RYF and D03RZF may be used in conjunction with D03RBF to check the user-supplied initial mesh, and extract mesh co-ordinate data.

3.6 Convection-diffusion Systems

There are three routines available for solving systems of convection-diffusion equations with optional source terms: D03PFF, D03PLF, D03PSF. Equations may include nonlinear terms but the time derivative should occur linearly. There are certain restrictions on the coefficients to ensure that the problems posed can be solved by the above routines, in particular the system must be posed in conservative form (see below). The routines may also be used to solve hyperbolic convection-only systems.

Convection terms are discretized using an upwind scheme involving a numerical flux function based on the solution of a Riemann problem at each mesh point [4]; and diffusion and source terms are discretized using central differences. The resulting system of ordinary differential equations is solved using a 'stiff' solver. In the case of Euler equations for a perfect gas various approximate and exact Riemann solvers are provided in D03PUF, D03PVF, D03PWF and D03PXF. These routines may be used in conjunction with D03PFF, D03PLF and D03PSF.

D03PFF is designed to solve systems of the form

$$\sum_{j=1}^{n} P_{ij}(x,t,U)\frac{\partial U_j}{\partial t} + \frac{\partial}{\partial x}F_i(x,t,U) = C_i(x,t,U)\frac{\partial}{\partial x}D_i(x,t,U,U_x) + S_i(x,t,U),$$

or hyperbolic convection-only systems of the form

$$\sum_{j=1}^{n} P_{ij}(x,t,U)\frac{\partial U_j}{\partial t} + \frac{\partial F_i(x,t,U)}{\partial x} = 0,$$

where $i = 1, 2, \ldots, n$, $a \leq x \leq b$, $t \geq t_0$.

D03PLF is similar to D03PFF except that it provides scope for coupled differential algebraic systems. This extended functionality allows for the solution of more complex problems.

D03PSF is similar to D03PLF but allows remeshing to take place in the spatial direction. This facility can be very useful when the nature of the solution in the spatial direction varies considerably over time.

3.7 Automatic Mesh Generation

The routine D03MAF places a triangular mesh over a given two-dimensional region. The region may have any shape and may include holes. It may also be used in conjunction with routines from the NAG Finite Element Library.

3.8 Utility Routines

D03UAF (D03UBF) calculates, by the Strongly Implicit Procedure, an approximate correction to a current estimate of the solution of a system of simultaneous algebraic equations for which the iterative update matrix is of five (seven) point molecule form on a two- (three-) dimensional topologically-rectangular mesh.

Routines are available in the Linear Algebra Chapters for the direct and iterative solution of linear equations. Here we point to some of the routines that may be of use in solving the linear systems that arise from finite difference or finite element approximations to partial differential equation solutions. Chapters F01, F04 and F11 should be consulted for further information and for the appropriate routine documents. Decision trees for the solution of linear systems are given in Section 3.6 of the the F04 Chapter Introduction.

The following routines allow the direct solution of symmetric positive-definite systems:

Band	F07HDF and F07HEF
Variable band (skyline)	F01MCF and F04MCF
Tridiagonal	F04FAF
Sparse	F11JAF* and F11JBF

(* the description of F11JBF explains how F11JAF should be called to obtain a direct method)

and the following routines allow the iterative solution of symmetric positive-definite and symmetric-indefinite systems:

Sparse	F11GAF, F11GBF, F11GCF, F11JAF, F11JCF and F11JEF

The latter two routines above are black box routines which include Incomplete Cholesky, SSOR or Jacobi preconditioning.

The following routines allow the direct solution of nonsymmetric systems:

Band	F07BDF and F07BEF
Almost block-diagonal	F01LHF and F04LHF
Tridiagonal	F01LEF and F04LEF, or F04EAF
Sparse	F01BRF (and F01BSF) and F04AXF

and the following routines allow the iterative solution of nonsymmetric systems:

Sparse	F11BAF, F11BBF, F11BCF, F11DAF, F11DCF and F11DEF

The latter two routines above are black box routines which include incomplete LU, SSOR and Jacobi preconditioning.

The routines D03PZF and D03PYF use linear interpolation to compute the solution to a parabolic problem and its first derivative at the user-specified points. D03PZF may be used in conjunction with D03PCF, D03PEF, D03PHF, D03PKF, D03PPF and D03PRF. D03PYF may be used in conjunction with D03PDF and D03PJF.

D03RYF and D03RZF are utility routines for use in conjunction with D03RBF. They can be called to check the user-specified initial mesh and to extract mesh co-ordinate data.

4 Index

5 Routines Withdrawn or Scheduled for Withdrawal

Since Mark 13 the following routines have been withdrawn. Advice on replacing calls to these routines is given in the document 'Advice on Replacement Calls for Withdrawn/Superseded Routines'.

D03PAF D03PBF D03PGF

6 References

[1] Ames W F (1977) *Nonlinear Partial Differential Equations in Engineering* Academic Press (2nd Edition)

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

[3] Jaswon M A and Symm G T (1977) *Integral Equation Methods in Potential Theory and Elastostatics* Academic Press

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

[5] Mikhlin S G and Smolitsky K L (1967) *Approximate Methods for the Solution of Differential and Integral Equations* Elsevier

[6] Mitchell A R and Griffiths D F (1980) *The Finite Difference Method in Partial Differential Equations* Wiley

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

[8] Richtmyer R D and Morton K W (1967) *Difference Methods for Initial-value Problems* Interscience (2nd Edition)

[9] Smith G D (1985) *Numerical Solution of Partial Differential Equations: Finite Difference Methods* Oxford University Press (3rd Edition)

[10] Swarztrauber P N and Sweet R A (1979) Efficient Fortran subprograms for the solution of separable elliptic partial differential equations *ACM Trans. Math. Software* **5** 352–364

[11] Wait R and Mitchell A R (1985) *Finite Element Analysis and Application* Wiley

---

D03EAF – NAG Fortran Library Routine Document

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

D03EAF solves Laplace's equation in two dimensions for an arbitrary domain bounded internally or externally by one or more closed contours, given the value of either the unknown function or its normal derivative (into the domain) at each point of the boundary.

2 Specification

```
      SUBROUTINE D03EAF(STAGE1, EXT, DORM, N, P, Q, X, Y, N1P1, PHI,
     1                  PHID, ALPHA, C, IC, NP4, ICINT, NP1, IFAIL)
      INTEGER           N, N1P1, IC, NP4, ICINT(NP1), NP1, IFAIL
      real              P, Q, X(N1P1), Y(N1P1), PHI(N), PHID(N), ALPHA,
     1                  C(IC,NP4)
      LOGICAL           STAGE1, EXT, DORM
```

3 Description

The routine uses an integral equation method, based upon Green's formula, which yields the solution, ϕ, within the domain, given its value or that of its normal derivative at each point of the boundary (except possibly at a finite number of discrete points). The solution is obtained in two stages. The first, which is executed once only, determines the complementary boundary values, i.e., ϕ, where its normal derivative is known and vice versa. The second stage is entered once for each point at which the solution is required.

The boundary is divided into a number of intervals in each of which ϕ and its normal derivative are approximated by constants. Of these half are evaluated by applying the given boundary conditions at one 'nodal' point within each interval while the remainder are determined (in stage 1) by solving a set of simultaneous linear equations. Here this is achieved by means of auxiliary routines F01BKF and F04AUF, which will yield the least-squares solution of an overdetermined system of equations as well as the unique solution of a square non-singular system.

In exterior domains the solution behaves as $c+s\log r+O(1/r)$ as r tends to infinity, where c is a constant, s is the total integral of the normal derivative around the boundary and r is the radial distance from the origin of co-ordinates. For the Neumann problem (when the normal derivative is given along the whole boundary) s is fixed by the boundary conditions whilst c is chosen by the user. However, for a Dirichlet problem (when ϕ is given along the whole boundary) or for a mixed problem, stage 1 produces a value of c for which $s=0$; then as r tends to infinity the solution tends to the constant c.

4 References

[1] Symm G T and Pitfield R A (1974) Solution of Laplace's equation in two dimensions *NPL Report NAC 44* National Physical Laboratory

5 Parameters

1: STAGE1 — LOGICAL *Input*

On entry: indicates whether the routine call is for stage 1 of the computation as defined in Section 3. If STAGE1 = .TRUE., then the call is for stage 1. If STAGE1 = .FALSE., then the call is for stage 2.

2: EXT — LOGICAL *Input*

On entry: the form of the domain. If EXT = .TRUE., the domain is unbounded. Otherwise the domain is an interior one.

3: DORM — LOGICAL *Input*

On entry: the form of the boundary conditions. If DORM = .TRUE., then the problem is a Dirichlet or mixed boundary value problem. Otherwise it is a Neumann problem.

4: N — INTEGER *Input*

On entry: the number of intervals into which the boundary is divided (see Section 7 and Section 8).

5: P — ***real*** *Input*

6: Q — ***real*** *Input*

On entry: to stage 2, P and Q must specify the x and y co-ordinates respectively of a point at which the solution is required.

When STAGE1 is .TRUE., P and Q are ignored.

7: X(N1P1) — ***real*** array *Input*

8: Y(N1P1) — ***real*** array *Input*

On entry: the x and y co-ordinates respectively of points on the one or more closed contours which define the domain of the problem.

Note. Each contour is described in such a manner that the subscripts of the co-ordinates increase when the domain is kept on the left. The final point on each contour coincides with the first and, if a further contour is to be described, the co-ordinates of this point are repeated in the arrays. In this way each interval is defined by three points, the second of which (the nodal point) always has an even subscript. In the case of the interior Neumann problem, the outermost boundary contour must be given first, if there is more than one.

9: N1P1 — INTEGER *Input*

On entry: the value 2 × (N+M−1, where M denotes the number of closed contours which make up the boundary.

10: PHI(N) — ***real*** array *Input/Output*

On entry: for stage 1, PHI must contain the nodal values of ϕ or its normal derivative (into the domain) as prescribed in each interval. For stage 2 it must retain its output values from stage 1.

On exit: from stage 1, it contains the constants which approximate ϕ in each interval. It remains unchanged on exit from stage 2.

11: PHID(N) — ***real*** array *Input/Output*

On entry: for stage 1, PHID(i) must hold the value 0.0 or 1.0 according as PHI(i) contains a value of ϕ or its normal derivative, for $i = 1, 2, \ldots$,N. For stage 2 it must retain its output values from stage 1.

On exit: from stage 1, PHID contains the constants which approximate the normal derivative of ϕ in each interval. It remains unchanged on exit from stage 2.

12: ALPHA — ***real*** *Input/Output*

On entry: for stage 1, the use of ALPHA depends on the nature of the problem:

DORM = .TRUE. – ALPHA need not be set.

DORM = .FALSE. and EXT = .TRUE. – ALPHA must contain the prescribed constant c (see Section 3).

DORM = .FALSE. and EXT = .FALSE. – ALPHA must contain an appropriate value (often zero) for the integral of ϕ around the outermost boundary.

For stage 2, on every call ALPHA must contain the value returned at stage 1.

On exit: from stage 1:

EXT = .FALSE. – ALPHA contains 0.0.

EXT = .TRUE. and DORM = .FALSE. – ALPHA is unchanged.
EXT = .TRUE. and DORM = .TRUE. – ALPHA contains a computed estimate for c.

From stage 2:

ALPHA contains the computed value of ϕ at the point (P,Q).

13: C(IC,NP4) — ***real*** array *Workspace*

14: IC — INTEGER *Input*

On entry: the first dimension of the array C as declared in the (sub)program from which D03EAF is called.

Constraint: IC $\geq$ N + 1.

15: NP4 — INTEGER *Input*

On entry: the value N + 4.

16: ICINT(NP1) — INTEGER array *Workspace*

17: NP1 — INTEGER *Input*

On entry: the value N + 1.

18: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

Invalid tolerance used in an internal call to an auxiliary routine:

ICINT(1) = 0

indicates too large a tolerance.

ICINT(1) > 0

indicates too small a tolerance.

Note. That this error is only possible in stage 1, and the circumstances under which it may occur cannot be foreseen. In the event of a failure, it is suggested that the user change the scale of the domain of the problem and apply the routine again.

IFAIL = 2

Incorrect rank obtained by an auxiliary routine; ICINT(1) contains the computed rank.

7 Accuracy

The accuracy of the computed solution depends upon how closely ϕ and its normal derivative may be approximated by constants in each interval of the boundary and upon how well the boundary contours are represented by polygons with vertices at the selected points (X(i),Y(i)), $i = 1, 2, \ldots, 2(N+M) - 1$.

Consequently, in general, the accuracy increases as the boundary is subdivided into smaller and smaller intervals and by comparing solutions for successive subdivisions one may obtain an indication of the error in these solutions.

Alternatively, since the point of maximum error always lies on the boundary of the domain, an estimate of the error may be obtained by computing ϕ at a sufficient number of points on the boundary where the true solution is known. The latter method (not applicable to the Neumann problem) is most useful in the case where ϕ alone is prescribed on the boundary (the Dirichlet problem).

8 Further Comments

The time taken by the routine for stage 1, which is executed once only, is roughly proportional to N^2, being dominated by the time taken to compute the coefficients. The time for each stage 2 application of the routine is proportional to N.

The intervals into which the boundary is divided need not be of equal lengths.

For many practical problems useful results may be obtained with 20 to 40 intervals per boundary contour.

9 Example

An interior Neumann problem to solve Laplace's equation in the domain bounded externally by the triangle with vertices (3,0), (−3,0) and (0,4), and internally by the triangle with vertices (2,1), (−2,1) and (0,3), given that the normal derivative of the solution ϕ is zero on each side of each triangle and, for uniqueness that the total integral of ϕ around the outer triangle is 16 (the length of the contour).

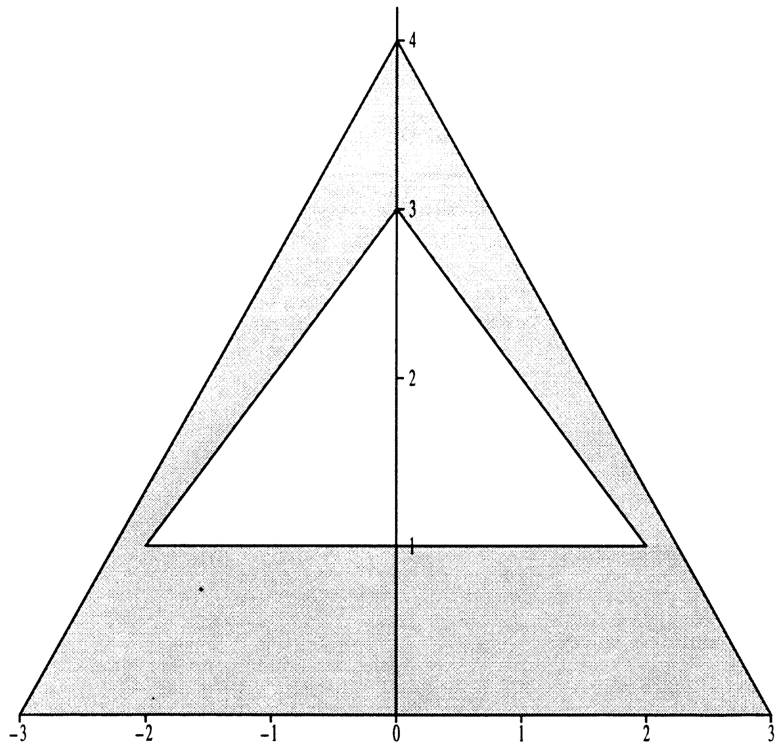


Figure 1

This trivial example has the obvious solution $\phi = 1$ throughout the domain. However it provides a useful illustration of data input for a doubly-connected domain. The solution is computed at one corner of each triangle and at one point inside the domain.

The program is written to handle any of the different types of problem that the routine can solve. The array dimensions must be increased for larger problems.

9.1 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.

```
*     D03EAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N, M, NP1, IC, NP4, N1, N1P1
      PARAMETER        (N=6,M=2,NP1=N+1,IC=N+1,NP4=N+4,N1=2*(N+M)-2,
     +                 N1P1=N1+1)
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
*     .. Local Scalars ..
      real             ALPHA, C, P, Q
      INTEGER          I, IFAIL, J, NPTS
      LOGICAL          DORM, EXT, STGONE
*     .. Local Arrays ..
      real             C1(IC,NP4), PHI(N), PHID(N), X(N1P1), Y(N1P1)
      INTEGER          ICINT(NP1)
*     .. External Subroutines ..
      EXTERNAL         D03EAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03EAF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
      READ (NIN,*) EXT, DORM
      STGONE = .TRUE.
      WRITE (NOUT,*)
      IF ( .NOT. EXT .AND. .NOT. DORM) THEN
         READ (NIN,*) ALPHA
         WRITE (NOUT,*) 'Interior Neumann problem'
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Total integral =', ALPHA
      ELSE
         IF (EXT .AND. .NOT. DORM) THEN
            READ (NIN,*) ALPHA
            WRITE (NOUT,*) 'Exterior Neumann problem'
            WRITE (NOUT,*)
            WRITE (NOUT,99998) 'C=', ALPHA
         END IF
      END IF
      DO 20 I = 1, N1 + 1
         READ (NIN,*) X(I), Y(I)
   20 CONTINUE
      DO 40 I = 1, N
         READ (NIN,*) PHI(I), PHID(I)
   40 CONTINUE
      IFAIL = 1
*
      CALL D03EAF(STGONE,EXT,DORM,N,P,Q,X,Y,N1P1,PHI,PHID,ALPHA,C1,IC,
     +            NP4,ICINT,NP1,IFAIL)
*
      IF (IFAIL.NE.0) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,99996) 'Error in D03EAF IFAIL = ', IFAIL
         WRITE (NOUT,*)
         WRITE (NOUT,99996) 'The value of RANK is ', ICINT(1)
         STOP
```

```
      END IF
      C = ALPHA
      IF (EXT .AND. DORM) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Exterior problem'
         WRITE (NOUT,*)
         WRITE (NOUT,99999) 'Computed C =', C
      END IF
      J = 2
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Nodes'
      WRITE (NOUT,*)
     +  '          X              Y              PHI             PHID'
      DO 60 I = 1, N
         IF (X(J).EQ.X(J-1) .AND. Y(J).EQ.Y(J-1)) J = J + 2
         WRITE (NOUT,99997) X(J), Y(J), PHI(I), PHID(I)
         J = J + 2
   60 CONTINUE
      STGONE = .FALSE.
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Selected points'
      WRITE (NOUT,*) '          X              Y              PHI'
      READ (NIN,*) NPTS
      DO 80 I = 1, NPTS
         READ (NIN,*) P, Q
         ALPHA = C
*
         CALL D03EAF(STGONE,EXT,DORM,N,P,Q,X,Y,N1P1,PHI,PHID,ALPHA,C1,
     +               IC,NP4,ICINT,NP1,IFAIL)
*
         WRITE (NOUT,99997) P, Q, ALPHA
   80 CONTINUE
      STOP
*
99999 FORMAT (1X,A,F15.2)
99998 FORMAT (1X,A,e15.4)
99997 FORMAT (1X,4F15.2)
99996 FORMAT (1X,A,I2)
      END
```

9.2 Program Data

```
D03EAF Example Program Data
F F
 16.0
  3.0  0.0
  1.5  2.0
  0.0  4.0
 -1.5  2.0
 -3.0  0.0
  0.0  0.0
  3.0  0.0
  3.0  0.0
  2.0  1.0
  0.0  1.0
 -2.0  1.0
 -1.0  2.0
  0.0  3.0
```

```
  1.0  2.0
  2.0  1.0
  0.0  1.0
  0.0  1.0
  0.0  1.0
  0.0  1.0
  0.0  1.0
  0.0  1.0
3
  2.0  1.0
  2.5  0.5
  3.0  0.0
```

9.3 Program Results

```
D03EAF Example Program Results

Interior Neumann problem

Total integral =          16.00

Nodes
           X              Y              PHI            PHID
           1.50           2.00           1.00           0.00
          -1.50           2.00           1.00           0.00
           0.00           0.00           1.00           0.00
           0.00           1.00           1.00           0.00
          -1.00           2.00           1.00           0.00
           1.00           2.00           1.00           0.00

Selected points
           X              Y              PHI
           2.00           1.00           1.00
           2.50           0.50           1.00
           3.00           0.00           1.00
```

D03EBF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D03EBF uses the Strongly Implicit Procedure to calculate the solution to a system of simultaneous algebraic equations of five-point molecule form on a two-dimensional topologically-rectangular mesh. ('Topological' means that a polar grid, for example (r,θ), can be used, being equivalent to a rectangular box.)

2. Specification

```
      SUBROUTINE D03EBF (N1, N2, N1M, A, B, C, D, E, Q, T, APARAM, ITMAX,
     1                   ITCOUN, ITUSED, NDIR, IXN, IYN, CONRES, CONCHN,
     2                   RESIDS, CHNGS, WRKSP1, WRKSP2, WRKSP3, IFAIL)
      INTEGER         N1, N2, N1M, ITMAX, ITCOUN, ITUSED, NDIR, IXN,
     1                IYN, IFAIL
      real            A(N1M,N2), B(N1M,N2), C(N1M,N2), D(N1M,N2),
     1                E(N1M,N2), Q(N1M,N2), T(N1M,N2), APARAM, CONRES,
     2                CONCHN, RESIDS(ITMAX), CHNGS(ITMAX),
     3                WRKSP1(N1M,N2), WRKSP2(N1M,N2), WRKSP3(N1M,N2)
```

3. Description

Given a set of simultaneous equations

$$Mt = q \qquad (1)$$

(which could be nonlinear) derived, for example, from a finite difference representation of a two-dimensional elliptic partial differential equation and its boundary conditions, the routine determines the values of the dependent variable t. q is a known vector of length $n_1 \times n_2$ and M is a square $(n_1 \times n_2)$ by $(n_1 \times n_2)$ matrix.

The equations must be of five diagonal form:

$$a_{ij}t_{i,j-1} + b_{ij}t_{i-1,j} + c_{ij}t_{ij} + d_{ij}t_{i+1,j} + e_{ij}t_{i,j+1} = q_{ij}$$

for $i = 1,2,...,n_1$; $j = 1,2,...,n_2$, provided $c_{ij} \neq 0.0$. Indeed, if $c_{ij} = 0.0$, then the equation is assumed to be

$$t_{ij} = q_{ij}.$$

For example, if $n_1 = 3$ and $n_2 = 2$, the equations take the form:

$$\begin{bmatrix} c_{11} & d_{11} & & e_{11} & & \\ b_{21} & c_{21} & d_{21} & & e_{21} & \\ & b_{31} & c_{31} & & & e_{31} \\ a_{12} & & & c_{12} & d_{12} & \\ & a_{22} & & b_{22} & c_{22} & d_{22} \\ & & a_{32} & & b_{32} & c_{32} \end{bmatrix} \begin{bmatrix} t_{11} \\ t_{21} \\ t_{31} \\ t_{12} \\ t_{22} \\ t_{32} \end{bmatrix} = \begin{bmatrix} q_{11} \\ q_{21} \\ q_{31} \\ q_{12} \\ q_{22} \\ q_{32} \end{bmatrix}$$

The system is solved iteratively, from a starting approximation $t^{(1)}$, by the formulae

$$r^{(n)} = q - Mt^{(n)}$$

$$Ms^{(n)} = r^{(n)}$$

$$t^{(n+1)} = t^{(n)} + s^{(n)}.$$

Thus $r^{(n)}$ is the residual of the nth approximate solution $t^{(n)}$, and $s^{(n)}$ is the up-date change vector. The calling program supplies an initial approximation for the values of the dependent variable in the array T, the coefficients of the five-point molecule system of equations in the arrays A, B, C, D and E, and the source terms in the array Q. The routine derives the residual of the latest approximate solution and then uses the approximate *LU* factorization of the Strongly Implicit Procedure with the necessary acceleration parameter adjustment by calling D03UAF at each iteration. D03EBF combines the newly derived change with the old approximation to obtain the new approximate solution for t. The new solution is checked for convergence against the user-supplied convergence criteria and if these have not been achieved the iterative cycle is repeated. Convergence is based on both the maximum absolute normalised residuals (calculated with reference to the previous approximate solution as these are calculated at the commencement of each iteration) and on the maximum absolute change made to the values of t.

Problems in topologically non-rectangular regions can be solved using the routine by surrounding the region by a circumscribing topological rectangle. The equations for the nodal values external to the region of interest are set to zero (i.e. $c_{ij} = t_{ij} = 0$) and the boundary conditions are incorporated into the equations for the appropriate nodes.

If there is no better initial approximation when starting the iterative cycle, an array of all zeros can be used as the initial approximation.

The routine can be used to solve linear elliptic equations in which case the arrays A, B, C, D, E and Q are constants and for which a single call provides the required solution. It can also be used to solve nonlinear elliptic equations in which case some or all of these arrays may require updating during the progress of the iterations as more accurate solutions are derived. The routine will then have to be called repeatedly in an outer iterative cycle. Dependent on the nonlinearity, some under relaxation of the coefficients and/or source terms may be needed during their recalculation using the new estimates of the solution.

The routine can also be used to solve each step of a time-dependent parabolic equation in two space dimensions. The solution at each time step can be expressed in terms of an elliptic equation if the Crank-Nicolson or other form of implicit time integration is used.

Neither diagonal dominance, nor positive-definiteness, of the matrix M formed from the arrays A, B, C, D, E is necessary to ensure convergence.

For problems in which the solution is not unique in the sense that an arbitrary constant can be added to the solution, for example Laplace's equation with all Neumann boundary conditions, a parameter is incorporated so that the solution can be rescaled by subtracting a specified nodal value from the whole solution t after the completion of every iteration to keep rounding errors to a minimum for those cases when the convergence is slow.

4. References

[1] JACOBS, D.A.H.
The strongly implicit procedure for the numerical solution of parabolic and elliptic partial differential equations.
Central Electricity Research Laboratory Note RD/L/N66/72, 1972.

[2] STONE, H.L.
Iterative solution of implicit approximations of multidimensional partial differential equations.
SIAM J. Numer. Anal., 5, pp. 530-558, 1968.

5. Parameters

1: N1 – INTEGER. *Input*

On entry: the number of nodes in the first co-ordinate direction, n_1.

Constraint: N1 > 1.

2: N2 – INTEGER. *Input*

On entry: the number of nodes in the second co-ordinate direction, n_2.

Constraint: N2 > 1.

3: N1M – INTEGER. *Input*

On entry: the first dimension of the arrays A, B, C, D, E, Q, T, WRKSP1, WRKSP2 and WRKSP3, as declared in the (sub)program from which D03EBF is called.

Constraint: N1M $\geq$ N1.

4: A(N1M,N2) – ***real*** array. *Input*

On entry: A(i,j) must contain the coefficient of the 'southerly' term involving $t_{i,j-1}$ in the (i,j)th equation of the system (1), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of A for j = 1 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

5: B(N1M,N2) – ***real*** array. *Input*

On entry: B(i,j) must contain the coefficient of the 'westerly' term involving $t_{i-1,j}$ in the (i,j)th equation of the system (1), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of B for i = 1 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

6: C(N1M,N2) – ***real*** array. *Input*

On entry: C(i,j) must contain the coefficient of the 'central' term involving t_{ij} in the (i,j)th equation of the system (1), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of C are checked to ensure that they are non-zero. If any element is found to be zero, the corresponding algebraic equation is assumed to be $t_{ij} = q_{ij}$. This feature can be used to define the equations for nodes at which, for example, Dirichlet boundary conditions are applied, or for nodes external to the problem of interest, by setting C(i,j) = 0.0 at appropriate points, and the corresponding value of Q(i,j) to the appropriate value, namely the prescribed value of T(i,j) in the Dirichlet case or zero at an external point.

7: D(N1M,N2) – ***real*** array. *Input*

On entry: D(i,j) must contain the coefficient of the 'easterly' term involving $t_{i+1,j}$ in the (i,j)th equation of the system (1), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of D for i = N1 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

8: E(N1M,N2) – ***real*** array. *Input*

On entry: E(i,j) must contain the coefficient of the 'northerly' term involving $t_{i,j+1}$ in the (i,j)th equation of the system (1), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of E for j = N2 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

9: Q(N1M,N2) – ***real*** array. *Input*

On entry: Q(i,j) must contain q_{ij} for i = 1,2,...,N1; j = 1,2,...,N2, i.e. the source term values at the nodal points for the system (1).

10: T(N1M,N2) – ***real*** array. *Input/Output*

On entry: T(i,j) must contain the element t_{ij} of the approximate solution to the equations supplied by the calling program as an initial starting value, for i = 1,2,...,N1; j = 1,2,...,N2.

If no better approximation is known, an array of zeros can be used.

On exit: the solution derived by the routine.

11: APARAM – ***real.*** *Input*

On entry: the iteration acceleration factor. A value of 1.0 is adequate for most typical problems. However, if convergence is slow, the value can be reduced, typically to 0.2 or 0.1. If divergence is obtained, the value can be increased, typically to 2.0, 5.0 or 10.0.

Constraint: $0.0 < \text{APARAM} \le ((\text{N1}-1)^2+(\text{N2}-1)^2)/2.0$.

12: ITMAX – INTEGER. *Input*

On entry: the maximum number of iterations to be used by the routine in seeking the solution. A reasonable value might be 30 if N1 = N2 = 10 or 100 if N1 = N2 = 50.

13: ITCOUN – INTEGER. *Input/Output*

On entry: on the first call of D03EBF, ITCOUN must be set to 0. On subsequent entries, its value must be unchanged from the previous call.

On exit: its value is increased by the number of iterations used on this call (namely ITUSED). It therefore stores the accumulated number of iterations actually used. For subsequent calls for the same problem, i.e. with the same N1 and N2 but possibly different coefficients and/or source terms, as occur with nonlinear systems or with time-dependent systems, ITCOUN is the accumulated number of iterations. This applies to the second and subsequent calls to D03EBF. In this way a suitable cycling of the sequence of iteration parameters is obtained in the calls to D03UAF.

14: ITUSED – INTEGER. *Output*

On exit: the number of iterations actually used on that call.

15: NDIR – INTEGER. *Input*

On entry: indicates whether or not the system of equations has a unique solution. For systems which have a unique solution, NDIR must be set to any non-zero value. For systems derived from, for example, Laplace's equation with all Neumann boundary conditions, i.e. problems in which an arbitrary constant can be added to the solution, NDIR should be set to 0 and the values of the next two parameters must be specified. For such problems the routine subtracts the value of the function derived at the node (IXN, IYN) from the whole solution after each iteration to reduce the possibility of large rounding errors. The user must also ensure that for such problems the appropriate consistency condition on the source terms Q is satisfied.

16: IXN – INTEGER. *Input*

On entry: IXN is ignored unless NDIR is equal to zero, in which case it must specify the first index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

17: IYN – INTEGER. *Input*

On entry: IYN is ignored unless NDIR is equal to zero, in which case it must specify the second index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

18: CONRES – ***real.*** *Input*

On entry: the convergence criterion to be used on the maximum absolute value of the normalised residual vector components. The latter is defined as the residual of the algebraic equation divided by the central coefficient when the latter is not equal to 0.0, and defined as the residual when the central coefficient is zero.

Clearly CONRES should not be less than a reasonable multiple of the ***machine precision.***

19: CONCHN – ***real***. *Input*

On entry: the convergence criterion to be used on the maximum absolute value of the change made at each iteration to the elements of the array T, namely the dependent variable. Clearly CONCHN should not be less than a reasonable multiple of the ***machine precision*** multiplied by the maximum value of T attained.

Convergence is achieved when both the convergence criteria are satisfied. The user can therefore set convergence on either the residual or on the change, or (as is recommended) on a requirement that both are below prescribed limits.

20: RESIDS(ITMAX) – ***real*** array. *Output*

On exit: the maximum absolute value of the residuals calculated at the ith iteration, for $i = 1,2,...,$ITUSED. The user who wants to know the maximum absolute residual of the solution which is returned must calculate this in the calling program. The sequence of values RESIDS indicates the rate of convergence.

21: CHNGS(ITMAX) – ***real*** array. *Output*

On exit: the maximum absolute value of the changes made to the components of the dependent variable T at the ith iteration, for $i = 1,2,...,$ITUSED. The sequence of values CHNGS indicates the rate of convergence.

22: WRKSP1(N1M,N2) – ***real*** array. *Workspace*
23: WRKSP2(N1M,N2) – ***real*** array. *Workspace*
24: WRKSP3(N1M,N2) – ***real*** array. *Workspace*

25: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, N1 < 2,
or N2 < 2.

IFAIL = 2

On entry, N1M < N1.

IFAIL = 3

On entry, APARAM ≤ 0.0.

IFAIL = 4

On entry, APARAM > $((N1-1)^2+(N2-1)^2)/2.0$.

IFAIL = 5

Convergence was not achieved after ITMAX iterations.

7. Accuracy

The improvement in accuracy for each iteration depends on the size of the system and on the condition of the up-date matrix characterised by the five-diagonal coefficient arrays. The ultimate accuracy obtainable depends on the above factors and on the ***machine precision***. The rate of convergence obtained with the Strongly Implicit Procedure is not always smooth because of the cyclic use of nine acceleration parameters. The convergence may become slow with very large problems, for example when N1 = N2 = 60. The final accuracy may be judged approximately from the rate of convergence determined from the sequence of values returned in CHNGS and the magnitude of the maximum absolute value of the change vector on the last iteration stored in CHNGS(ITUSED).

8. Further Comments

The time taken by the routine per iteration is approximately proportional to N1×N2.

Convergence may not always be obtained when the problem is very large and/or the coefficients of the equations have widely disparate values. The latter case is often associated with a near ill-conditioned matrix.

9. Example

To solve Laplace's equation in a rectangle with a non-uniform grid spacing in the x and y co-ordinate directions and with Dirichlet boundary conditions specifying the function on the perimeter of the rectangle equal to

$$e^{(1.0+x)/y(n_2)}\times\cos(y/y(n_2)).$$

9.1. 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.

```
*     D03EBF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N1, N2, N1M, ITMAX
      PARAMETER        (N1=6,N2=10,N1M=N1,ITMAX=18)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             APARAM, CONCHN, CONRES
      INTEGER          I, IFAIL, ITCOUN, ITUSED, IXN, IYN, J, NDIR
*     .. Local Arrays ..
      real             A(N1M,N2), B(N1M,N2), C(N1M,N2), CHNGS(ITMAX),
     +                 D(N1M,N2), E(N1M,N2), Q(N1M,N2), RESIDS(ITMAX),
     +                 T(N1M,N2), WRKSP1(N1M,N2), WRKSP2(N1M,N2),
     +                 WRKSP3(N1M,N2), X(N1M), Y(N2)
*     .. External Subroutines ..
      EXTERNAL         D03EBF
*     .. Intrinsic Functions ..
      INTRINSIC        COS, EXP
*     .. Data statements ..
      DATA             X(1), X(2), X(3), X(4), X(5), X(6)/0.0e0, 1.0e0,
     +                 3.0e0, 6.0e0, 10.0e0, 15.0e0/
      DATA             Y(1), Y(2), Y(3), Y(4), Y(5), Y(6), Y(7), Y(8),
     +                 Y(9), Y(10)/0.0e0, 1.0e0, 3.0e0, 6.0e0, 10.0e0,
     +                 15.0e0, 21.0e0, 28.0e0, 36.0e0, 45.0e0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03EBF Example Program Results'
      WRITE (NOUT,*)
      APARAM = 1.0e0
      ITCOUN = 0
      NDIR = 1
      CONRES = 0.1e-5
      CONCHN = 0.1e-5
*     Set up difference equation coefficients, source terms and
*     initial conditions.
      DO 40 J = 1, N2
         DO 20 I = 1, N1
            IF ((I.NE.1) .AND. (I.NE.N1) .AND. (J.NE.1) .AND. (J.NE.N2))
     +          THEN
```

```
*              Specification for internal nodes
               A(I,J) = 2.0e0/((Y(J)-Y(J-1))*(Y(J+1)-Y(J-1)))
               E(I,J) = 2.0e0/((Y(J+1)-Y(J))*(Y(J+1)-Y(J-1)))
               B(I,J) = 2.0e0/((X(I)-X(I-1))*(X(I+1)-X(I-1)))
               D(I,J) = 2.0e0/((X(I+1)-X(I))*(X(I+1)-X(I-1)))
               C(I,J) = -A(I,J) - B(I,J) - D(I,J) - E(I,J)
               Q(I,J) = 0.0e0
               T(I,J) = 0.0e0
            ELSE
*              Specification for boundary nodes
               A(I,J) = 0.0e0
               B(I,J) = 0.0e0
               C(I,J) = 0.0e0
               D(I,J) = 0.0e0
               E(I,J) = 0.0e0
               Q(I,J) = EXP((X(I)+1.0e0)/Y(N2))*COS(Y(J)/Y(N2))
               T(I,J) = 0.0e0
            END IF
   20    CONTINUE
   40 CONTINUE
      WRITE (NOUT,*) 'Iteration      Maximum        Maximum'
      WRITE (NOUT,*) ' number        residual        change'
      IFAIL = 1
*
      CALL D03EBF(N1,N2,N1M,A,B,C,D,E,Q,T,APARAM,ITMAX,ITCOUN,ITUSED,
     +            NDIR,IXN,IYN,CONRES,CONCHN,RESIDS,CHNGS,WRKSP1,WRKSP2,
     +            WRKSP3,IFAIL)
*
      WRITE (NOUT,99999) (I,RESIDS(I),CHNGS(I),I=1,ITUSED)
*     Check error flag
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Table of calculated function values'
         WRITE (NOUT,*)
         WRITE (NOUT,*)
     +' I    1          2          3          4          5          6'
         WRITE (NOUT,*) ' J'
         DO 60 J = 1, N2
            WRITE (NOUT,99998) J, (T(I,J),I=1,N1)
   60    CONTINUE
      ELSE
   80    WRITE (NOUT,99997) 'Error in D03EBF  IFAIL =', IFAIL
      END IF
      STOP
*
99999 FORMAT (2X,I2,10X,e11.4,4X,e11.4)
99998 FORMAT (1X,I2,1X,6(F9.3,2X))
99997 FORMAT (1X,A,I4)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D03EBF Example Program Results

Iteration      Maximum        Maximum
 number        residual        change
  1           0.1427E+01     0.1427E+01
  2           0.6671E-02     0.2176E-01
  3           0.8422E-03     0.1621E-02
  4           0.7635E-04     0.1810E-03
  5           0.5434E-05     0.1199E-04
  6           0.6471E-06     0.1245E-05
  7           0.5467E-07     0.1081E-06
```

Table of calculated function values

I / J	1	2	3	4	5	6
1	1.022	1.045	1.093	1.168	1.277	1.427
2	1.022	1.045	1.093	1.168	1.277	1.427
3	1.020	1.043	1.091	1.166	1.274	1.424
4	1.013	1.036	1.083	1.158	1.266	1.414
5	0.997	1.020	1.066	1.140	1.246	1.392
6	0.966	0.988	1.033	1.104	1.207	1.348
7	0.913	0.934	0.976	1.044	1.141	1.274
8	0.831	0.850	0.888	0.950	1.038	1.160
9	0.712	0.728	0.762	0.814	0.890	0.994
10	0.552	0.565	0.591	0.631	0.690	0.771

D03ECF – NAG Fortran Library Routine Document

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

D03ECF uses the Strongly Implicit Procedure to calculate the solution to a system of simultaneous algebraic equations of seven-point molecule form on a three-dimensional topologically-rectangular mesh. ('Topological' means that a polar grid, for example, can be used if it is equivalent to a rectangular box.)

2 Specification

```
      SUBROUTINE D03ECF(N1, N2, N3, N1M, N2M, A, B, C, D, E, F, G, Q, T,
     1                  APARAM, ITMAX, ITCOUN, ITUSED, NDIR, IXN, IYN,
     2                  IZN, CONRES, CONCHN, RESIDS, CHNGS, WRKSP1,
     3                  WRKSP2, WRKSP3, WRKSP4, IFAIL)
      INTEGER           N1, N2, N3, N1M, N2M, ITMAX, ITCOUN, ITUSED,
     1                  NDIR, IXN, IYN, IZN, IFAIL
      real              A(N1M,N2M,N3), B(N1M,N2M,N3), C(N1M,N2M,N3),
     1                  D(N1M,N2M,N3), E(N1M,N2M,N3), F(N1M,N2M,N3),
     2                  G(N1M,N2M,N3), Q(N1M,N2M,N3), T(N1M,N2M,N3),
     3                  APARAM, CONRES, CONCHN, RESIDS(ITMAX),
     4                  CHNGS(ITMAX), WRKSP1(N1M,N2M,N3),
     5                  WRKSP2(N1M,N2M,N3), WRKSP3(N1M,N2M,N3),
     6                  WRKSP4(N1M,N2M,N3)
```

3 Description

Given a set of simultaneous equations

$$Mt = q \tag{1}$$

(which could be nonlinear) derived, for example, from a finite difference representation of a three-dimensional elliptic partial differential equation and its boundary conditions, the routine determines the values of the dependent variable t. M is a square $(n_1 \times n_2 \times n_3)$ by $(n_1 \times n_2 \times n_3)$ matrix and q is a known vector of length $(n_1 \times n_2 \times n_3)$.

The equations must be of seven-diagonal form:

$$a_{ijk}t_{ij,k-1} + b_{ijk}t_{i,j-1,k} + c_{ijk}t_{i-1,jk} + d_{ijk}t_{ijk} + e_{ijk}t_{i+1,jk} + f_{ijk}t_{i,j+1,k} + g_{ijk}t_{ij,k+1} = q_{ijk}$$

for $i = 1,2,\ldots,n_1$; $j = 1,2,\ldots,n_2$ and $k = 1,2,\ldots,n_3$, provided that $d_{ijk} \neq 0.0$.

Indeed, if $d_{ijk} = 0.0$, then the equation is assumed to be:

$$t_{ijk} = q_{ijk}.$$

The system is solved iteratively from a starting approximation $t^{(1)}$ by the formulae:

$$\begin{aligned} r^{(n)} &= q - Mt^{(n)} \\ Ms^{(n)} &= r^{(n)} \\ t^{(n+1)} &= t^{(n)} + s^{(n)}. \end{aligned}$$

Thus $r^{(n)}$ is the residual of the nth approximate solution $t^{(n)}$, and $s^{(n)}$ is the up-date change vector.

The calling program supplies an initial approximation for the values of the dependent variable in the array T, the coefficients of the seven-point molecule system of equations in the arrays A, B, C, D, E, F and G, and the source terms in the array Q. The routine derives the residual of the latest approximate solution, and then uses the approximate LU factorization of the Strongly Implicit Procedure with the necessary acceleration parameter adjustment by calling D03UBF at each iteration. D03ECF combines the newly derived change with the old approximation to obtain the new approximate solution for t. The new solution is checked for convergence against the user-supplied convergence criteria, and if these have not been satisfied, the iterative cycle is repeated. Convergence is based on both the maximum absolute normalised residuals (calculated with reference to the previous approximate solution as these are calculated at the commencement of each iteration) and on the maximum absolute change made to the values of t.

Problems in topologically non-rectangular-box-shaped regions can be solved using the routine by surrounding the region by a circumscribing topologically rectangular box. The equations for the nodal values external to the region of interest are set to zero (i.e., $d_{ijk} = t_{ijk} = 0$) and the boundary conditions are incorporated into the equations for the appropriate nodes.

If there is no better initial approximation when starting the iterative cycle, one can use an array of zeros as the initial approximation.

The routine can be used to solve linear elliptic equations in which case the arrays A, B, C, D, E, F, G and Q remain constant and for which a single call provides the required solution. It can also be used to solve nonlinear elliptic equations, in which case some or all of these arrays may require updating during the progress of the iterations as more accurate solutions are derived. The routine will then have to be called repeatedly in an outer iterative cycle. Dependent on the nonlinearity, some under-relaxation of the coefficients and/or source terms may be needed during their recalculation using the new estimates of the solution.

The routine can also be used to solve each step of a time-dependent parabolic equation in three space dimensions. The solution at each time step can be expressed in terms of an elliptic equation if the Crank–Nicolson or other form of implicit time integration is used.

Neither diagonal dominance, nor positive definiteness, of the matrix M formed from the arrays A, B, C, D, E, F, G is necessary to ensure convergence.

For problems in which the solution is not unique in the sense that an arbitrary constant can be added to the solution (for example Poisson's equation with all Neumann boundary conditions), a parameter is incorporated so that the solution can be rescaled. A specified nodal value is subtracted from the whole solution t after the completion of every iteration. This keeps rounding errors to a minimum for those cases when convergence is slow. For such problems there is generally an associated compatibility condition. For the example mentioned this compatibility condition equates the total net source within the region (i.e., the source integrated over the region) with the total net outflow across the boundaries defined by the Neumann conditions (i.e., the normal derivative integrated along the whole boundary). It is very important that the algebraic equations derived to model such a problem implement accurately the compatibility condition. If they do not, a net source or sink is very likely to be represented by the set of algebraic equations and no steady-state solution of the equations exists.

4 References

[1] Jacobs D A H (1972) The strongly implicit procedure for the numerical solution of parabolic and elliptic partial differential equations *Note RD/L/N66/72* Central Electricity Research Laboratory

[2] Stone H L (1968) Iterative solution of implicit approximations of multi-dimensional partial differential equations *SIAM J. Numer. Anal.* **5** 530–558

[3] Weinstein H G, Stone H L and Kwan T V (1969) Iterative procedure for solution of systems of parabolic and elliptic equations in three dimensions *Industrial and Engineering Chemistry Fundamentals* **8** 281–287

5 Parameters

1: N1 — INTEGER *Input*

On entry: the number of nodes in the first co-ordinate direction, n_1.

Constraint: N1 > 1.

2: N2 — INTEGER *Input*

On entry: the number of nodes in the second co-ordinate direction, n_2.

Constraint: N2 > 1.

3: N3 — INTEGER *Input*

On entry: the number of nodes in the third co-ordinate direction, n_3.

Constraint: N3 > 1.

4: N1M — INTEGER *Input*

On entry: the first dimension of all the three-dimensional arrays, as declared in the (sub)program from which D03ECF is called.

Constraint: N1M $\geq$ N1.

5: N2M — INTEGER *Input*

On entry: the second dimension of all the three-dimensional arrays, as declared in the (sub)program from which D03ECF is called.

Constraint: N2M $\geq$ N2.

6: A(N1M,N2M,N3) — ***real*** array *Input*

On entry: A(i,j,k) must contain the coefficient of $t_{ij,k-1}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of A for $k = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

7: B(N1M,N2M,N3) — ***real*** array *Input*

On entry: B(i,j,k) must contain the coefficient of $t_{i,j-1,k}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of B for $j = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

8: C(N1M,N2M,N3) — ***real*** array *Input*

On entry: C(i,j,k) must contain the coefficient of $t_{i-1,jk}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of C for $i = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

9: D(N1M,N2M,N3) — ***real*** array *Input*

On entry: D(i,j,k) must contain the coefficient of t_{ijk} (the 'central' term) in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of D are checked to ensure that they are non-zero. If any element is found to be zero, the corresponding algebraic equation is assumed to be $t_{ijk} = q_{ijk}$. This feature can be used to define the equations for nodes at which, for example, Dirichlet boundary conditions are applied, or for nodes external to the problem of interest. Setting D$(i,j,k) = 0.0$ at appropriate points, and the corresponding value of Q(i,j,k) to the appropriate value, namely the prescribed value of T(i,j,k) in the Dirichlet case, or to zero at an external point.

10: E(N1M,N2M,N3) — ***real*** array *Input*

On entry: E(i,j,k) must contain the coefficient of $t_{i+1,jk}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of E for $i =$ N1 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

11: F(N1M,N2M,N3) — ***real*** array *Input*

On entry: F(i,j,k) must contain the coefficient of $t_{i,j+1,k}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots$,N1; $j = 1,2,\ldots$,N2 and $k = 1,2,\ldots$,N3. The elements of F for j = N2 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

12: G(N1M,N2M,N3) — ***real*** array *Input*

On entry: G(i,j,k) must contain the coefficient of $t_{ij,k+1}$ in the (i,j,k)th equation of the system (1), for $i = 1,2,\ldots$,N1; $j = 1,2,\ldots$,N2 and $k = 1,2,\ldots$,N3. The elements of G for k = N3 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

13: Q(N1M,N2M,N3) — ***real*** array *Input*

On entry: Q(i,j,k) must contain q_{ijk} for $i = 1,2,\ldots$,N1; $j = 1,2,\ldots$,N2 and $k = 1,2,\ldots$,N3, i.e., the source-term values at the nodal points of the system (1).

14: T(N1M,N2M,N3) — ***real*** array *Input/Output*

On entry: T(i,j,k) must contain the element t_{ijk} of an approximate solution to the equations for $i = 1,2,\ldots$,N1; $j = 1,2,\ldots$,N2 and $k = 1,2,\ldots$,N3.

If no better approximation is known, an array of zeros can be used.

On exit: the solution derived by the routine.

15: APARAM — ***real*** *Input*

On entry: the iteration acceleration factor. A value of 1.0 is adequate for most typical problems. However, if convergence is slow, the value can be reduced, typically to 0.2 or 0.1. If divergence is obtained, the value can be increased, typically to 2.0, 5.0 or 10.0.

Constraint: 0.0 < APARAM ≤ $((\mathrm{N}-1)^2 + (\mathrm{N2}-1)^2 + (\mathrm{N3}-1)^2)/3.0$.

16: ITMAX — INTEGER *Input*

On entry: the maximum number of iterations to be used by the routine in seeking the solution. A reasonable value might be 20 for a problem with 3000 nodes and convergence criteria of about 10^{-3} of the original residual and change.

17: ITCOUN — INTEGER *Input/Output*

On entry: on the first call of D03ECF, ITCOUN must be set to 0. On subsequent entries, its value must be unchanged from the previous call.

On exit: its value is increased by the number of iterations used on this call (namely ITUSED). It therefore stores the accumulated number of iterations actually used.

For subsequent calls for the same problem, i.e., with the same N1, N2 and N3 but possibly different coefficients and/or source terms, as occur with nonlinear systems or with time-dependent systems, ITCOUN should not be reset, i.e., it must contain the accumulated number of iterations. In this way a suitable cycling of the sequence of iteration parameters is obtained in the calls to D03UBF.

18: ITUSED — INTEGER *Output*

On exit: the number of iterations actually used on that call.

19: NDIR — INTEGER *Input*

On entry: indicates whether or not the system of equations has a unique solution. For systems which have a unique solution, NDIR must be set to any non-zero value. For systems derived from problems to which an arbitrary constant can be added to the solution, for example Poisson's equation with all Neumann boundary conditions, NDIR should be set to 0 and the values of the next three parameters must be specified. For such problems the routine subtracts the value of the function derived at the node (IXN, IYN, IZN) from the whole solution after each iteration to reduce the possibility of large rounding errors. The user must also ensure for such problems that the appropriate compatibility condition on the source terms Q is satisfied. See the comments at the end of Section 3.

20: IXN — INTEGER *Input*

On entry: IXN is ignored unless NDIR is equal to zero, in which case it must specify the first index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

21: IYN — INTEGER *Input*

On entry: IYN is ignored unless NDIR is equal to zero, in which case it must specify the second index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

22: IZN — INTEGER *Input*

On entry: IZN is ignored unless NDIR is equal to zero, in which case it must specify the third index of the nodal point at which the solution is to be set to zero. The node should not correspond to a corner node, or to a node external to the region of interest.

23: CONRES — ***real*** *Input*

On entry: the convergence criterion to be used on the maximum absolute value of the normalised residual vector components. The latter is defined as the residual of the algebraic equation divided by the central coefficient when the latter is not equal to 0.0, and defined as the residual when the central coefficient is zero.

CONRES should not be less than a reasonable multiple of the ***machine precision***.

24: CONCHN — ***real*** *Input*

On entry: the convergence criterion to be used on the maximum absolute value of the change made at each iteration to the elements of the array T, namely the dependent variable. CONCHN should not be less than a reasonable multiple of the machine accuracy multiplied by the maximum value of T attained.

Convergence is achieved when both the convergence criteria are satisfied. The user can therefore set convergence on either the residual or on the change, or (as is recommended) on a requirement that both are below prescribed limits.

25: RESIDS(ITMAX) — ***real*** array *Output*

On exit: the maximum absolute value of the residuals calculated at the ith iteration, for $i = 1, 2, \ldots$,ITUSED. If the residual of the solution is sought the user must calculate this in the (sub)program from which D03ECF is called. The sequence of values RESIDS indicates the rate of convergence.

26: CHNGS(ITMAX) — ***real*** array *Output*

On exit: the maximum absolute value of the changes made to the components of the dependent variable T at the ith iteration, for $i = 1, 2, \ldots$,ITUSED. The sequence of values CHNGS indicates the rate of convergence.

27: WRKSP1(N1M,N2M,N3) — ***real*** array *Workspace*

28: WRKSP2(N1M,N2M,N3) — ***real*** array *Workspace*

29: WRKSP3(N1M,N2M,N3) — ***real*** array *Workspace*

30: WRKSP4(N1M,N2M,N3) — ***real*** array *Workspace*

31: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL $\neq 0$ on exit, users are recommended to set IFAIL to -1 before entry. **It is then essential to test the value of IFAIL on exit**. To suppress the output of an error message when soft failure occurs, set IFAIL to 1.

6 Error Indicators and Warnings

Errors or warnings specified by the routine:

IFAIL = 1

On entry, N1 < 2,
or N2 < 2,
or N3 < 2.

IFAIL = 2

On entry, N1M < N1,
or N2M < N2.

IFAIL = 3

On entry, APARAM $\leq$ 0.0.

IFAIL = 4

On entry, APARAM > $((N1-1)^2 + (N2-1)^2 + (N3-1)^2)/3.0$.

IFAIL = 5

Convergence was not achieved after ITMAX iterations.

7 Accuracy

The improvement in accuracy for each iteration depends on the size of the system and on the condition of the up-date matrix characterised by the seven-diagonal coefficient arrays. The ultimate accuracy obtainable depends on the above factors and on the ***machine precision***. The rate of convergence obtained with the Strongly Implicit Procedure is not always smooth because of the cyclic use of nine acceleration parameters. The convergence may become slow with very large problems. The final accuracy obtained may be judged approximately from the rate of convergence determined from the sequence of values returned in the arrays RESIDS and CHNGS and the magnitude of the maximum absolute value of the change vector on the last iteration stored in CHNGS(ITUSED).

8 Further Comments

The time taken by the routine per iteration is approximately proportional to N1 × N2 × N3.

Convergence may not always be obtained when the problem is very large and/or the coefficients of the equations have widely disparate values. The latter case is often associated with a near ill-conditioned matrix.

9 Example

To solve Laplace's equation in a rectangular box with a non-uniform grid spacing in the x, y, and z co-ordinate directions and with Dirichlet boundary conditions specifying the function on the surfaces of the box equal to

$$e^{(1.0+x)/y(n_2)} \times \cos(\sqrt{2}y/y(n_2)) \times e^{(-1.0-z)/y(n_2)}.$$

Note that this is the same problem as that solved in the example for D03UBF. The differences in the maximum residuals obtained at each iteration between the two test runs are explained by the fact that in D03ECF the residual at each node is normalised by dividing by the central coefficient, whereas this normalisation has not been used in the example program for D03UBF.

9.1 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.

```
*     D03ECF Example Program Text
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          N1, N2, N3, N1M, N2M, ITMAX
      PARAMETER        (N1=4,N2=5,N3=6,N1M=N1,N2M=N2,ITMAX=18)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             APARAM, CONCHN, CONRES, ROOT2
      INTEGER          I, IFAIL, ITCOUN, ITUSED, IXN, IYN, IZN, J, K,
     +                 NDIR
*     .. Local Arrays ..
      real             A(N1M,N2M,N3), B(N1M,N2M,N3), C(N1M,N2M,N3),
     +                 CHNGS(ITMAX), D(N1M,N2M,N3), E(N1M,N2M,N3),
     +                 F(N1M,N2M,N3), G(N1M,N2M,N3), Q(N1M,N2M,N3),
     +                 RESIDS(18), T(N1M,N2M,N3), WRKSP1(N1M,N2M,N3),
     +                 WRKSP2(N1M,N2M,N3), WRKSP3(N1M,N2M,N3),
     +                 WRKSP4(N1M,N2M,N3), X(N1), Y(N2), Z(N3)
*     .. External Subroutines ..
      EXTERNAL         D03ECF
*     .. Intrinsic Functions ..
      INTRINSIC        COS, EXP, SQRT
*     .. Data statements ..
      DATA             X(1), X(2), X(3), X(4)/0.0e0, 1.0e0, 3.0e0,
     +                 6.0e0/
      DATA             Y(1), Y(2), Y(3), Y(4), Y(5)/0.0e0, 1.0e0, 3.0e0,
     +                 6.0e0, 10.0e0/
      DATA             Z(1), Z(2), Z(3), Z(4), Z(5), Z(6)/0.0e0, 1.0e0,
     +                 3.0e0, 6.0e0, 10.0e0, 15.0e0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03ECF Example Program Results'
      WRITE (NOUT,*)
      ROOT2 = SQRT(2.0e0)
      APARAM = 1.0e0
      ITCOUN = 0
      NDIR = 1
      CONRES = 0.1e-5
      CONCHN = 0.1e-5
*     Set up difference equation coefficients, source terms and
*     initial approximation.
      DO 80 K = 1, N3
         DO 60 J = 1, N2
            DO 40 I = 1, N1
               IF ((I.NE.1) .AND. (I.NE.N1) .AND. (J.NE.1)
     +             .AND. (J.NE.N2) .AND. (K.NE.1) .AND. (K.NE.N3)) THEN
*                 Specification for internal nodes
                  A(I,J,K) = 2.0e0/((Z(K)-Z(K-1))*(Z(K+1)-Z(K-1)))
                  G(I,J,K) = 2.0e0/((Z(K+1)-Z(K))*(Z(K+1)-Z(K-1)))
                  B(I,J,K) = 2.0e0/((Y(J)-Y(J-1))*(Y(J+1)-Y(J-1)))
                  F(I,J,K) = 2.0e0/((Y(J+1)-Y(J))*(Y(J+1)-Y(J-1)))
                  C(I,J,K) = 2.0e0/((X(I)-X(I-1))*(X(I+1)-X(I-1)))
                  E(I,J,K) = 2.0e0/((X(I+1)-X(I))*(X(I+1)-X(I-1)))
                  D(I,J,K) = -A(I,J,K) - B(I,J,K) - C(I,J,K) - E(I,J,K)
     +                        - F(I,J,K) - G(I,J,K)
```

```
                  Q(I,J,K) = 0.0e0
                  T(I,J,K) = 0.0e0
               ELSE
*                 Specification for boundary nodes
   20             A(I,J,K) = 0.0e0
                  B(I,J,K) = 0.0e0
                  C(I,J,K) = 0.0e0
                  E(I,J,K) = 0.0e0
                  F(I,J,K) = 0.0e0
                  G(I,J,K) = 0.0e0
                  D(I,J,K) = 0.0e0
                  Q(I,J,K) = EXP((X(I)+1.0e0)/Y(N2))*COS(ROOT2*Y(J)
     +                       /Y(N2))*EXP((-Z(K)-1.0e0)/Y(N2))
                  T(I,J,K) = 0.0e0
               END IF
   40       CONTINUE
   60    CONTINUE
   80 CONTINUE
      WRITE (NOUT,*) 'Iteration      Maximum        Maximum'
      WRITE (NOUT,*) ' number        residual        change'
      IFAIL = 0
*
      CALL D03ECF(N1,N2,N3,N1M,N2M,A,B,C,D,E,F,G,Q,T,APARAM,ITMAX,
     +            ITCOUN,ITUSED,NDIR,IXN,IYN,IZN,CONRES,CONCHN,RESIDS,
     +            CHNGS,WRKSP1,WRKSP2,WRKSP3,WRKSP4,IFAIL)
*
      IF (ITUSED.NE.0) WRITE (NOUT,99999) (I,RESIDS(I),CHNGS(I),I=1,
     +    ITUSED)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Table of calculated function values'
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     + 'K  J   (I      T   ) (I      T   ) (I      T   ) (I      T   )'
      WRITE (NOUT,99998) ((K,J,(I,T(I,J,K),I=1,N1),J=1,N2),K=1,N3)
      STOP
*
99999 FORMAT (2X,I3,9X,e11.4,4X,e11.4)
99998 FORMAT ((1X,I1,I3,1X,4(1X,I3,2X,F8.3)))
      END
```

9.2 Program Data

None.

9.3 Program Results

```
D03ECF Example Program Results

Iteration      Maximum        Maximum
 number        residual        change
   1         0.1822E+01     0.1822E+01
   2         0.9025E-02     0.1970E-01
   3         0.1358E-02     0.1496E-02
   4         0.4013E-04     0.3848E-04
   5         0.5321E-05     0.5481E-05
   6         0.2695E-06     0.2333E-06
```

```
Table of calculated function values

 K  J   (I      T   ) (I      T   ) (I      T   ) (I      T   )
 1  1    1    1.000    2    1.105    3    1.350    4    1.822
 1  2    1    0.990    2    1.094    3    1.336    4    1.804
 1  3    1    0.911    2    1.007    3    1.230    4    1.661
 1  4    1    0.661    2    0.731    3    0.892    4    1.205
 1  5    1    0.156    2    0.172    3    0.211    4    0.284
 2  1    1    0.905    2    1.000    3    1.221    4    1.649
 2  2    1    0.896    2    0.990    3    1.210    4    1.632
 2  3    1    0.825    2    0.912    3    1.114    4    1.503
 2  4    1    0.598    2    0.662    3    0.809    4    1.090
 2  5    1    0.141    2    0.156    3    0.190    4    0.257
 3  1    1    0.741    2    0.819    3    1.000    4    1.350
 3  2    1    0.733    2    0.811    3    0.991    4    1.336
 3  3    1    0.675    2    0.747    3    0.913    4    1.230
 3  4    1    0.490    2    0.543    3    0.664    4    0.892
 3  5    1    0.116    2    0.128    3    0.156    4    0.211
 4  1    1    0.549    2    0.607    3    0.741    4    1.000
 4  2    1    0.543    2    0.601    3    0.734    4    0.990
 4  3    1    0.500    2    0.554    3    0.677    4    0.911
 4  4    1    0.363    2    0.402    3    0.492    4    0.661
 4  5    1    0.086    2    0.095    3    0.116    4    0.156
 5  1    1    0.368    2    0.407    3    0.497    4    0.670
 5  2    1    0.364    2    0.403    3    0.492    4    0.664
 5  3    1    0.335    2    0.371    3    0.454    4    0.611
 5  4    1    0.243    2    0.270    3    0.330    4    0.443
 5  5    1    0.057    2    0.063    3    0.077    4    0.105
 6  1    1    0.223    2    0.247    3    0.301    4    0.407
 6  2    1    0.221    2    0.244    3    0.298    4    0.403
 6  3    1    0.203    2    0.225    3    0.274    4    0.371
 6  4    1    0.148    2    0.163    3    0.199    4    0.269
 6  5    1    0.035    2    0.038    3    0.047    4    0.063
```

D03EDF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D03EDF solves seven-diagonal systems of linear equations which arise from the discretization of an elliptic partial differential equation on a rectangular region. This routine uses a multigrid technique.

2. Specification

```
      SUBROUTINE D03EDF (NGX, NGY, LDA, A, RHS, UB, MAXIT, ACC, US, U,
     1                   IOUT, NUMIT, IFAIL)
      INTEGER            NGX, NGY, LDA, MAXIT, IOUT, NUMIT, IFAIL
      real               A(LDA,7), RHS(LDA), UB(NGX*NGY), ACC, US(LDA),
     1                   U(LDA)
```

3. Description

D03EDF solves, by multigrid iteration, the seven-point scheme

$$\begin{aligned} & A^6_{ij}\, u_{i-1,j+1} + A^7_{ij}\, u_{i,j+1} \\ + & A^3_{ij}\, u_{i-1,j} + A^4_{ij}\, u_{ij} + A^5_{ij}\, u_{i+1,j} \\ & + A^1_{ij}\, u_{i,j-1} + A^2_{ij}\, u_{i+1,j-1} = f_{ij}, \qquad i = 1,2,\ldots,n_x;\ j = 1,2,\ldots,n_y, \end{aligned}$$

which arises from the discretization of an elliptic partial differential equation of the form

$$\alpha(x,y)U_{xx} + \beta(x,y)U_{xy} + \gamma(x,y)U_{yy} + \delta(x,y)U_x + \varepsilon(x,y)U_y + \phi(x,y)U = \psi(x,y)$$

and its boundary conditions, defined on a rectangular region. This we write in matrix form as

$$Au = f.$$

The algorithm is described in separate reports by Wesseling [2], [3] and McCarthy [1].

Systems of linear equations, matching the seven-point stencil defined above, are solved by a multigrid iteration. An initial estimate of the solution must be provided by the user. A zero guess may be supplied if no better approximation is available.

A 'smoother' based on incomplete Crout decomposition is used to eliminate the high frequency components of the error. A restriction operator is then used to map the system on to a sequence of coarser grids. The errors are then smoothed and prolongated (mapped onto successively finer grids). When the finest cycle is reached, the approximation to the solution is corrected. The cycle is repeated for MAXIT iterations or until the required accuracy, ACC, is reached.

D03EDF will automatically determine the number l of possible coarse grids, 'levels' of the multigrid scheme, for a particular problem. In other words, D03EDF determines the maximum integer l so that n_x and n_y can be expressed in the form

$$n_x = m2^{l-1} + 1,\ n_y = n2^{l-1} + 1, \qquad \text{with } m \geq 2 \text{ and } n \geq 2.$$

It should be noted that the rate of convergence improves significantly with the number of levels used (see McCarthy [1]), so that n_x and n_y should be carefully chosen so that n_x-1 and n_y-1 have factors of the form 2^l, with l as large as possible. For good convergence the integer l should be at least 2.

D03EDF has been found to be robust in application, but being an iterative method the problem of divergence can arise. For a strictly diagonally dominant matrix A

$$|A^4_{ij}| > \sum_{k\neq 4} |A^k_{ij}|, \qquad i = 1,2,\ldots,n_x;\ i = 1,2,\ldots,n_y$$

no such problem is foreseen. The diagonal dominance of A is not a necessary condition, but should this condition be strongly violated then divergence may occur. The quickest test is to try the routine.

4. References

[1] MCCARTHY, G.J.
Investigation into the Multigrid Code MGD1.
Harwell, Report AERE – R 10889, April 1983.

[2] WESSELING, P.
MGD1 – A Robust and Efficient Multigrid Method.
In: 'Multigrid Methods', Lecture Notes in Mathematics, (No. 960), pp. 614-630.
Springer-Verlag, Berlin, 1982.

[3] WESSELING, P.
Theoretical Aspects of a Multigrid Method.
SIAM J. Sci. Statist. Comput., 3, pp. 387-407, 1982.

5. Parameters

1: NGX – INTEGER. *Input*

On entry: the number of interior grid points in the x-direction, n_x. NGX−1 should preferably be divisible by as high a power of 2 as possible.

Constraint: NGX ≥ 3.

2: NGY – INTEGER. *Input*

On entry: the number of interior grid points in the y-direction, n_y. NGY−1 should preferably be divisible by as high a power of 2 as possible.

Constraint: NGY ≥ 3.

3: LDA – INTEGER. *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which D03EDF is called, which must also be a lower bound for the dimensions of the arrays RHS, US and U. It is always sufficient to set LDA ≥ (4×(NGX+1)×(NGY+1))/3, but slightly smaller values may be permitted, depending on the values of NGX and NGY. If on entry, LDA is too small, an error message gives the minimum permitted value. (LDA must be large enough to allow space for the coarse-grid approximations).

4: A(LDA,7) – ***real*** array. *Input/Output*

On entry: A(i+(j−1)×NGX,k) must be set to A_{ij}^k, for i = 1,2,...,NGX; j = 1,2,...,NGY and k = 1,2,...,7.

On exit: A is overwritten.

5: RHS(LDA) – ***real*** array. *Input/Output*

On entry: RHS(i+(j−1)×NGX) must be set to f_{ij}, for i = 1,2,...,NGX; j = 1,2,...,NGY.

On exit: the first NGX×NGY elements are unchanged and the rest of the array is used as workspace.

6: UB(NGX*NGY) – ***real*** array. *Input/Output*

On entry: UB(i+(j−1)×NGX) must be set to the initial estimate for the solution u_{ij}.

On exit: the corresponding component of the residual $r = f - Au$.

7: MAXIT – INTEGER. *Input*

On entry: the maximum permitted number of multigrid iterations. If MAXIT = 0, no multigrid iterations are performed, but the coarse-grid approximations and incomplete Crout decompositions are computed, and may be output if IOUT is set accordingly.

Constraint: MAXIT ≥ 0.

8: ACC – ***real.*** *Input*

On entry: the required tolerance for convergence of the residual 2-norm:

$$\|r\|_2 = \sqrt{\sum_{k=1}^{\mathrm{NGX}\times\mathrm{NGY}} (r_k)^2}$$

where $r = f - Au$ and u is the computed solution. Note that the norm is not scaled by the number of equations. The routine will stop after fewer than MAXIT iterations if the residual 2-norm is less than the specified tolerance. (If MAXIT > 0, at least one iteration is always performed.)

If on entry ACC = 0.0, then the ***machine precision*** is used as a default value for the tolerance; if ACC > 0.0, but ACC is less than the ***machine precision***, then the routine will stop when the residual 2-norm is less than the ***machine precision*** and IFAIL will be set to 4.

Constraint: ACC ≥ 0.0.

9: US(LDA) – ***real*** array. *Output*

On exit: the residual 2-norm, stored in element US(1).

10: U(LDA) – ***real*** array. *Output*

On exit: the computed solution u_{ij} is returned in U(i+(j–1)×NGX), for i = 1,2,...,NGX; j = 1,2,...,NGY.

11: IOUT – INTEGER. *Input*

On entry: controls the output of printed information to the advisory message unit as returned by X04ABF:

IOUT = 0

No output.

IOUT = 1

The solution u_{ij}, for i = 1,2,...,NGX; j = 1,2,...,NGY.

IOUT = 2

The residual 2-norm after each iteration, with the reduction factor over the previous iteration.

IOUT = 3

As for IOUT = 1 and IOUT = 2.

IOUT = 4

As for IOUT = 3, plus the final residual (as returned in UB).

IOUT = 5

As for IOUT = 4, plus the initial elements of A and RHS.

IOUT = 6

As for IOUT = 5, plus the Galerkin coarse grid approximations.

IOUT = 7

As for IOUT = 6, plus the incomplete Crout decompositions.

IOUT = 8

As for IOUT = 7, plus the residual after each iteration.

The elements A(p,k), the Galerkin coarse grid approximations and the incomplete Crout decompositions are output in the format:

Y-index = j
X-index = i A$(p,1)$ A$(p,2)$ A$(p,3)$ A$(p,4)$ A$(p,5)$ A$(p,6)$ A$(p,7)$

where $p = i + (j-1)\times$NGX, $i = 1,2,...,$NGX and $j = 1,2,...,$NGY.

The vectors U(p), UB(p), RHS(p) are output in matrix form with NGY rows and NGX columns. Where NGX > 10, the NGX values for a given j-value are produced in rows of 10. Values of IOUT > 4 may yield considerable amounts of output.

Constraint: 0 ≤ IOUT ≤ 8.

12: NUMIT – INTEGER. *Output*

On exit: the number of iterations performed.

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

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

IFAIL = 1

On entry, NGX < 3,
or NGY < 3,
or LDA is too small,
or ACC < 0.0,
or MAXIT < 0,
or IOUT < 0,
or IOUT > 8.

IFAIL = 2

MAXIT iterations have been performed with the residual 2-norm decreasing at each iteration but the residual 2-norm has not been reduced to less than the specified tolerance (see ACC). Examine the progress of the iteration by setting IOUT ≥ 2.

IFAIL = 3

As for IFAIL = 2, except that at one or more iterations the residual 2-norm did not decrease. It is likely that the method fails to converge for the given matrix A.

IFAIL = 4

On entry, ACC is less than the ***machine precision***. The routine terminated because the residual norm is less than the ***machine precision***.

7. Accuracy

See ACC (Section 5).

8. Further Comments

The rate of convergence of this routine is strongly dependent upon the number of levels, l, in the multigrid scheme, and thus the choice of NGX and NGY is very important. The user is advised to experiment with different values of NGX and NGY to see the effect they have on the rate of convergence; for example, using a value such as NGX = 65 (= 2^6+1) followed by NGX = 64 (for which l = 1).

9. Example

The program solves the elliptic partial differential equation

$$U_{xx} - \alpha U_{xy} + U_{yy} = -4, \qquad \alpha = 1.7$$

on the unit square $0 \le x,y \le 1$, with boundary conditions

$$U = 0 \text{ on } \begin{cases} x = 0, & (0 \le y \le 1) \\ y = 0, & (0 \le x \le 1) \\ y = 1, & (0 \le x \le 1) \end{cases} \quad U = 1 \text{ on } x = 1, \qquad 0 \le y \le 1.$$

For the equation to be elliptic, α must be less than 2.

The equation is discretized on a square grid with mesh spacing h in both directions using the following approximations:

NW	6 N	7	
W	3 O	4 E	5
	S	1 SE	2

$$U_{xx} \simeq \frac{1}{h^2}(U_{\mathrm{E}}-2U_{\mathrm{O}}+U_{\mathrm{W}})$$

$$U_{yy} \simeq \frac{1}{h^2}(U_{\mathrm{N}}-2U_{\mathrm{O}}+U_{\mathrm{S}})$$

$$U_{xy} \simeq \frac{1}{2h^2}(U_{\mathrm{N}}-U_{\mathrm{NW}}+U_{\mathrm{E}}-2U_{\mathrm{O}}+U_{\mathrm{W}}-U_{\mathrm{SE}}+U_{\mathrm{S}}).$$

Thus the following equations are solved:

$$\begin{aligned} &\tfrac{1}{2}\alpha u_{i-1,j+1} + (1-\tfrac{1}{2}\alpha)u_{i,j+1} \\ &+ (1-\tfrac{1}{2}\alpha)u_{i+1,j} + (-4+\alpha)u_{ij} + (1-\tfrac{1}{2}\alpha)u_{i+1,j} \\ &\qquad + (1-\tfrac{1}{2}\alpha)u_{i,j-1} + \tfrac{1}{2}\alpha u_{i+1,j-1} = -4h^2 \end{aligned}$$

9.1. 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.

```
*     D03EDF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      real             ALPHA
      PARAMETER        (ALPHA=1.7e0)
      INTEGER          LEVELS, NGX, NGY, LDA
      PARAMETER        (LEVELS=3,NGX=2**LEVELS+1,NGY=NGX,LDA=4*(NGX+1)
     +                 *(NGY+1)/3)
*     .. Local Scalars ..
      real             ACC, HX, HY
      INTEGER          I, IFAIL, IOUT, IX, IY, J, K, MAXIT, NUMIT
*     .. Local Arrays ..
      real             A(LDA,7), RHS(LDA), U(LDA), UB(NGX*NGY), US(LDA)
*     .. External Subroutines ..
      EXTERNAL         D03EDF, X04ABF
*     .. Intrinsic Functions ..
      INTRINSIC        real
```

```
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03EDF Example Program Results'
      WRITE (NOUT,*)
      ACC = 1.0e-4
      CALL X04ABF(1,NOUT)
      MAXIT = 15
*     ** Set IOUT.GE.2 to obtain intermediate output **
      IOUT = 0
      HX = 1.0e0/real(NGX+1)
      HY = 1.0e0/real(NGY+1)
      WRITE (NOUT,99999) 'NGX = ', NGX, '  NGY = ', NGY, '  ACC =', ACC,
     +  '  MAXIT = ', MAXIT
*     Set up operator, right-hand side and initial guess for
*     step-lengths HX and HY
      DO 40 J = 1, NGY
         DO 20 I = 1, NGX
            K = (J-1)*NGX + I
            A(K,1) = 1.0e0 - 0.5e0*ALPHA
            A(K,2) = 0.5e0*ALPHA
            A(K,3) = 1.0e0 - 0.5e0*ALPHA
            A(K,4) = -4.0e0 + ALPHA
            A(K,5) = 1.0e0 - 0.5e0*ALPHA
            A(K,6) = 0.5e0*ALPHA
            A(K,7) = 1.0e0 - 0.5e0*ALPHA
            RHS(K) = -4.0e0*HX*HY
            UB(K) = 0.0e0
   20    CONTINUE
   40 CONTINUE
*     Correction for the boundary conditions
*     Horizontal boundaries --
      DO 60 I = 2, NGX - 1
*        Boundary condition on Y=0 -- U=0
         IX = I
         A(IX,1) = 0.0e0
         A(IX,2) = 0.0e0
*        Boundary condition on Y=1 -- U=0
         IX = I + (NGY-1)*NGX
         A(IX,6) = 0.0e0
         A(IX,7) = 0.0e0
   60 CONTINUE
*     Vertical boundaries --
      DO 80 J = 2, NGY - 1
*        Boundary condition on X=0 -- U=0
         IY = (J-1)*NGX + 1
         A(IY,3) = 0.0e0
         A(IY,6) = 0.0e0
*        Boundary condition on X=1 -- U=1
         IY = J*NGX
         RHS(IY) = RHS(IY) - A(IY,5) - A(IY,2)
         A(IY,2) = 0.0e0
         A(IY,5) = 0.0e0
   80 CONTINUE
*     Now the four corners --
*     Bottom left corner
      K = 1
      A(K,1) = 0.0e0
      A(K,2) = 0.0e0
      A(K,3) = 0.0e0
      A(K,6) = 0.0e0
*     Top left corner
      K = 1 + (NGY-1)*NGX
      A(K,3) = 0.0e0
      A(K,6) = 0.0e0
      A(K,7) = 0.0e0
```

```
*     Bottom right corner
*     Use average value at discontinuity ( = 0.5 )
      K = NGX
      RHS(K) = RHS(K) - A(K,2)*0.5e0 - A(K,5)
      A(K,1) = 0.0e0
      A(K,2) = 0.0e0
      A(K,5) = 0.0e0
*     Top right corner
      K = NGX*NGY
      RHS(K) = RHS(K) - A(K,2) - A(K,5)
      A(K,2) = 0.0e0
      A(K,5) = 0.0e0
      A(K,6) = 0.0e0
      A(K,7) = 0.0e0
*     Solve the equations
      IFAIL = 0
*
      CALL D03EDF(NGX,NGY,LDA,A,RHS,UB,MAXIT,ACC,US,U,IOUT,NUMIT,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99998) 'Residual norm =', US(1)
      WRITE (NOUT,99997) 'Number of iterations =', NUMIT
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Solution'
      WRITE (NOUT,*)
      WRITE (NOUT,99996) '  I/J', (I,I=1,NGX)
      DO 100 J = 1, NGY
         WRITE (NOUT,99995) J, (U(I+(J-1)*NGX),I=1,NGX)
  100 CONTINUE
      STOP
*
99999 FORMAT (1X,A,I3,A,I3,A,1P,e10.2,A,I3)
99998 FORMAT (1X,A,1P,e12.2)
99997 FORMAT (1X,A,I5)
99996 FORMAT (1X,A,10I7,:)
99995 FORMAT (1X,I3,2X,10F7.3,:)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D03EDF Example Program Results

NGX =    9 NGY =    9 ACC =  1.00E-04  MAXIT =  15

Residual norm =     1.61E-05
Number of iterations =    4

Solution

  I/J      1      2      3      4      5      6      7      8      9
  1    0.024  0.047  0.071  0.095  0.120  0.148  0.185  0.261  0.579
  2    0.047  0.094  0.142  0.192  0.245  0.310  0.412  0.636  0.913
  3    0.071  0.142  0.215  0.292  0.378  0.489  0.663  0.862  0.969
  4    0.095  0.191  0.289  0.393  0.511  0.656  0.810  0.915  0.967
  5    0.119  0.239  0.361  0.486  0.616  0.741  0.836  0.895  0.939
  6    0.143  0.284  0.419  0.543  0.648  0.729  0.786  0.832  0.893
  7    0.164  0.315  0.438  0.527  0.593  0.641  0.682  0.734  0.823
  8    0.174  0.306  0.378  0.427  0.462  0.492  0.528  0.591  0.717
  9    0.155  0.202  0.229  0.248  0.264  0.282  0.313  0.376  0.523
```

When IOUT is set to 2 in the calling program, the routine produces intermediate output similar to the following:

```
D03EDF Example Program Results

NGX =    9  NGY =    9  ACC =  1.00E-04  MAXIT =  15

Iteration   0  Residual norm =  3.69E-01

Iteration   1  Residual norm =  1.35E-02  Reduction factor =  3.67E-02

Iteration   2  Residual norm =  9.10E-04  Reduction factor =  6.73E-02

Iteration   3  Residual norm =  1.18E-04  Reduction factor =  1.30E-01

Iteration   4  Residual norm =  1.61E-05  Reduction factor =  1.36E-01

Residual norm =    1.61E-05
Number of iterations =    4

Solution

  I/J      1      2      3      4      5      6      7      8      9
  1    0.024  0.047  0.071  0.095  0.120  0.148  0.185  0.261  0.579
  2    0.047  0.094  0.142  0.192  0.245  0.310  0.412  0.636  0.913
  3    0.071  0.142  0.215  0.292  0.378  0.489  0.663  0.862  0.969
  4    0.095  0.191  0.289  0.393  0.511  0.656  0.810  0.915  0.967
  5    0.119  0.239  0.361  0.486  0.616  0.741  0.836  0.895  0.939
  6    0.143  0.284  0.419  0.543  0.648  0.729  0.786  0.832  0.893
  7    0.164  0.315  0.438  0.527  0.593  0.641  0.682  0.734  0.823
  8    0.174  0.306  0.378  0.427  0.462  0.492  0.528  0.591  0.717
  9    0.155  0.202  0.229  0.248  0.264  0.282  0.313  0.376  0.523
```

D03EEF – NAG Fortran Library Routine Document

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

D03EEF discretizes a second-order elliptic partial differential equation (PDE) on a rectangular region.

2 Specification

```
      SUBROUTINE D03EEF(XMIN, XMAX, YMIN, YMAX, PDEF, BNDY, NGX, NGY,
     1                  LDA, A, RHS, SCHEME, IFAIL)
      INTEGER           NGX, NGY, LDA, IFAIL
      real              XMIN, XMAX, YMIN, YMAX, A(LDA,7), RHS(LDA)
      CHARACTER*1       SCHEME
      EXTERNAL          PDEF, BNDY
```

3 Description

D03EEF discretizes a second-order linear elliptic partial differential equation of the form

$$\alpha(x,y)\frac{\partial^2 U}{\partial x^2} + \beta(x,y)\frac{\partial^2 U}{\partial x \partial y} + \gamma(x,y)\frac{\partial^2 U}{\partial y^2} + \delta(x,y)\frac{\partial U}{\partial x} + \epsilon(x,y)\frac{\partial U}{\partial y} + \phi(x,y)U = \psi(x,y) \qquad (1)$$

on a rectangular region

$$\begin{aligned} x_A \leq x \leq x_B \\ y_A \leq y \leq y_B \end{aligned}$$

subject to boundary conditions of the form

$$a(x,y)U + b(x,y)\frac{\partial U}{\partial n} = c(x,y)$$

where $\frac{\partial U}{\partial n}$ denotes the outward pointing normal derivative on the boundary. Equation (1) is said to be elliptic if

$$4\alpha(x,y)\gamma(x,y) \geq (\beta(x,y))^2$$

for all points in the rectangular region. The linear equations produced are in a form suitable for passing directly to the multigrid routine D03EDF.

The equation is discretized on a rectangular grid, with n_x grid points in the x-direction and n_y grid points in the y-direction. The grid spacing used is therefore

$$\begin{aligned} h_x &= (x_B - x_A)/(n_x - 1) \\ h_y &= (y_B - y_A)/(n_y - 1) \end{aligned}$$

and the co-ordinates of the grid points (x_i, y_j) are

$$\begin{aligned} x_i &= x_A + (i-1)h_x, \;\; i = 1,2,\ldots,n_x, \\ y_j &= y_A + (j-1)h_y, \;\; j = 1,2,\ldots,n_y. \end{aligned}$$

At each grid point (x_i, y_j) six neighbouring grid points are used to approximate the partial differential equation, so that the equation is discretized on the seven-point stencil shown in Figure 1.

For convenience the approximation u_{ij} to the exact solution $U(x_i, y_j)$ is denoted by u_O, and the neighbouring approximations are labelled according to points of the compass as shown. Where numerical labels for the seven points are required, these are also shown.

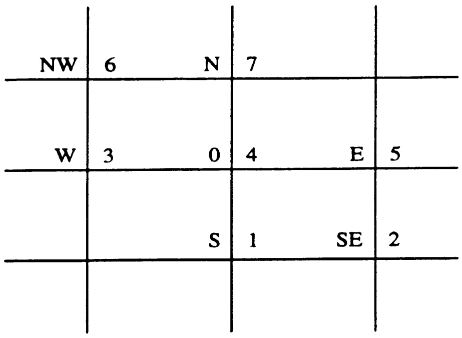


Figure 1

The following approximations are used for the second derivatives:

$$\frac{\partial^2 U}{\partial x^2} \simeq \frac{1}{h_x^2}(u_{\rm E} - 2u_{\rm O} + u_{\rm W})$$

$$\frac{\partial^2 U}{\partial y^2} \simeq \frac{1}{h_y^2}(u_{\rm N} - 2u_{\rm O} + u_{\rm S})$$

$$\frac{\partial^2 U}{\partial x \partial y} \simeq \frac{1}{2h_x h_y}(u_{\rm N} - u_{\rm NW} + u_{\rm E} - 2u_{\rm O} + u_{\rm W} - u_{\rm SE} + u_{\rm S}).$$

Two possible schemes may be used to approximate the first derivatives:

Central Differences

$$\frac{\partial U}{\partial x} \simeq \frac{1}{2h_x}(u_{\rm E} - u_{\rm W})$$

$$\frac{\partial U}{\partial y} \simeq \frac{1}{2h_y}(u_{\rm N} - u_{\rm S})$$

Upwind Differences

$$\frac{\partial U}{\partial x} \simeq \frac{1}{h_x}(u_{\rm O} - u_{\rm W}) \quad \text{if} \quad \delta(x,y) > 0$$

$$\frac{\partial U}{\partial x} \simeq \frac{1}{h_x}(u_{\rm E} - u_{\rm O}) \quad \text{if} \quad \delta(x,y) < 0$$

$$\frac{\partial U}{\partial y} \simeq \frac{1}{h_y}(u_{\rm N} - u_{\rm O}) \quad \text{if} \quad \epsilon(x,y) > 0$$

$$\frac{\partial U}{\partial y} \simeq \frac{1}{h_y}(u_{\rm O} - u_{\rm S}) \quad \text{if} \quad \epsilon(x,y) < 0.$$

Central differences are more accurate than upwind differences, but upwind differences may lead to a more diagonally dominant matrix for those problems where the coefficients of the first derivatives are significantly larger than the coefficients of the second derivatives.

The approximations used for the first derivatives may be written in a more compact form as follows:

$$\frac{\partial U}{\partial x} \simeq \frac{1}{2h_x}\left((k_x - 1)u_{\rm W} - 2k_x u_{\rm O} + (k_x + 1)u_{\rm E}\right)$$

$$\frac{\partial U}{\partial y} \simeq \frac{1}{2h_y}\left((k_y - 1)u_{\rm S} - 2k_y u_{\rm O} + (k_y + 1)u_{\rm N}\right)$$

where $k_x = \operatorname{sign}\delta$ and $k_y = \operatorname{sign}\epsilon$ for upwind differences, and $k_x = k_y = 0$ for central differences.

At all points in the rectangular domain, including the boundary, the coefficients in the partial differential equation are evaluated by calling the user-supplied subroutine PDEF, and applying the approximations.

This leads to a seven-diagonal system of linear equations of the form:

$$\begin{array}{lll} & A^6_{ij}u_{i-1,j+1} + A^7_{ij}u_{i,j+1} & \\ + & A^3_{ij}u_{i-1,j} + A^4_{ij}u_{ij} & + A^5_{ij}u_{i+1,j} \\ & \quad + A^1_{ij}u_{i,j-1} & + A^2_{ij}u_{i+1,j-1} = f_{ij},\ i = 1,2,\ldots,n_x; j = 1,2,\ldots,n_y, \end{array}$$

where the coefficients are given by

$$\begin{aligned}
A^1_{ij} &= \beta(x_i,y_j)\frac{1}{2h_xh_y} + \gamma(x_i,y_j)\frac{1}{h_y^2} + \epsilon(x_i,y_j)\frac{1}{2h_y}(k_y-1) \\
A^2_{ij} &= -\beta(x_i,y_j)\frac{1}{2h_xh_y} \\
A^3_{ij} &= \alpha(x_i,y_j)\frac{1}{h_x^2} + \beta(x_i,y_j)\frac{1}{2h_xh_y} + \delta(x_i,y_j)\frac{1}{2h_x}(k_x-1) \\
A^4_{ij} &= -\alpha(x_i,y_j)\frac{2}{h_x^2} - \beta(x_i,y_j)\frac{1}{h_xh_y} - \gamma(x_i,y_j)\frac{2}{h_y^2} - \delta(x_i,y_j)\frac{k_y}{h_x} - \epsilon(x_i,y_j)\frac{k_y}{h_y} - \phi(x_i,y_j) \\
A^5_{ij} &= \alpha(x_i,y_j)\frac{1}{h_x^2} + \beta(x_i,y_j)\frac{1}{2h_xh_y} + \delta(x_i,y_j)\frac{1}{2h_x}(k_x+1) \\
A^6_{ij} &= -\beta(x_i,y_j)\frac{1}{2h_xh_y} \\
A^7_{ij} &= \beta(x_i,y_j)\frac{1}{2h_xh_y} + \gamma(x_i,y_j)\frac{1}{h_y^2} + \epsilon(x_i,y_j)\frac{1}{2h_y}(k_y+1) \\
f_{ij} &= \psi(x_i,y_j)
\end{aligned}$$

These equations then have to be modified to take account of the boundary conditions. These may be Dirichlet (where the solution is given), Neumann (where the derivative of the solution is given), or mixed (where a linear combination of solution and derivative is given).

If the boundary conditions are Dirichlet, there are an infinity of possible equations which may be applied:

$$\mu u_{ij} = \mu f_{ij}\ ,\ \mu \neq 0. \tag{2}$$

If D03EDF is used to solve the discretized equations, it turns out that the choice of μ can have a dramatic effect on the rate of convergence, and the obvious choice $\mu = 1$ is not the best. Some choices may even cause the multigrid method to fail altogether. In practice it has been found that a value of the same order as the other diagonal elements of the matrix is best, and the following value has been found to work well in practice:

$$\mu = \min_{ij}\left(-\left\{\frac{2}{h_x^2} + \frac{2}{h_y^2}\right\}, A^4_{ij}\right).$$

If the boundary conditions are either mixed or Neumann (i.e., B $\neq 0$ on return from the user-supplied subroutine BNDY), then one of the points in the seven-point stencil lies outside the domain. In this case the normal derivative in the boundary conditions is used to eliminate the 'fictitious' point, u_{outside}:

$$\frac{\partial U}{\partial n} \simeq \frac{1}{2h}(u_{\text{outside}} - u_{\text{inside}}). \tag{3}$$

It should be noted that if the boundary conditions are Neumann and $\phi(x,y) \equiv 0$, then there is no unique solution. The routine returns with IFAIL = 5 in this case, and the seven-diagonal matrix is singular.

The four corners are treated separately. The user-supplied subroutine BNDY is called twice, once along each of the edges meeting at the corner. If both boundary conditions at this point are Dirichlet and the prescribed solution values agree, then this value is used in an equation of the form (2). If the prescribed solution is discontinuous at the corner, then the average of the two values is used. If one boundary condition is Dirichlet and the other is mixed, then the value prescribed by the Dirichlet condition is used in an equation of the form given above. Finally, if both conditions are mixed or Neumann, then two 'fictitious' points are eliminated using two equations of the form (3).

It is possible that equations for which the solution is known at all points on the boundary, have coefficients which are not defined on the boundary. Since this routine calls the user-supplied subroutine PDEF at **all** points in the domain, including boundary points, arithmetic errors may occur in the user's routine PDEF which this routine cannot trap. If the user has an equation with Dirichlet boundary conditions (i.e., B = 0 at all points on the boundary), but with PDE coefficients which are singular on the boundary, then D03EDF could be called directly only using interior grid points with the user's own discretization.

After the equations have been set up as described above, they are checked for diagonal dominance. That is to say,

$$|A_{ij}^4| > \sum_{k\neq 4} |A_{ij}^k|, \quad i = 1,2,\ldots,n_x; \; j = 1,2,\ldots,n_y.$$

If this condition is not satisfied then the routine returns with IFAIL = 6. The multigrid routine D03EDF may still converge in this case, but if the coefficients of the first derivatives in the partial differential equation are large compared with the coefficients of the second derivative, the user should consider using upwind differences (SCHEME = 'U').

Since this routine is designed primarily for use with D03EDF, this document should be read in conjunction with the document for that routine.

4 References

[1] Wesseling P (1982) MGD1 – A robust and efficient multigrid method *Multigrid Methods. Lecture Notes in Mathematics* **960** Springer-Verlag 614–630

5 Parameters

1: XMIN — ***real*** *Input*

2: XMAX — ***real*** *Input*

On entry: the lower and upper x co-ordinates of the rectangular region respectively, x_A and x_B.

Constraint: XMIN < XMAX.

3: YMIN — ***real*** *Input*

4: YMAX — ***real*** *Input*

On entry: the lower and upper y co-ordinates of the rectangular region respectively, y_A and y_B.

Constraint: YMIN < YMAX.

5: PDEF — SUBROUTINE, supplied by the user. *External Procedure*

PDEF must evaluate the functions $\alpha(x,y)$, $\beta(x,y)$, $\gamma(x,y)$, $\delta(x,y)$, $\epsilon(x,y)$, $\phi(x,y)$ and $\psi(x,y)$ which define the equation at a general point (x,y).

Its specification is:

```
SUBROUTINE PDEF(X, Y, ALPHA, BETA, GAMMA, DELTA, EPSLON, PHI, PSI)
real            X, Y, ALPHA, BETA, GAMMA, DELTA, EPSLON, PHI, PSI
```

1: X — ***real*** *Input*

2: Y — ***real*** *Input*

On entry: the x and y co-ordinates of the point at which the coefficients of the partial differential equation are to be evaluated.

3: ALPHA — ***real*** *Output*

4: BETA — ***real*** *Output*

5: GAMMA — ***real*** *Output*

6: DELTA — ***real*** *Output*

7: EPSLON — ***real*** *Output*
8: PHI — ***real*** *Output*
9: PSI — ***real*** *Output*

On exit: ALPHA, BETA, GAMMA, DELTA, EPSLON, PHI and PSI must be set to the values of $\alpha(x,y)$, $\beta(x,y)$, $\gamma(x,y)$, $\delta(x,y)$, $\epsilon(x,y)$, $\phi(x,y)$ and $\psi(x,y)$ respectively at the point specified by X and Y.

PDEF must be declared as EXTERNAL in the (sub)program from which D03EEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: BNDY — SUBROUTINE, supplied by the user. *External Procedure*

BNDY must evaluate the functions $a(x,y)$, $b(x,y)$, and $c(x,y)$ involved in the boundary conditions.

Its specification is:

```
SUBROUTINE BNDY(X, Y, A, B, C, IBND)
INTEGER        IBND
real           X, Y, A, B, C
```

1: X — ***real*** *Input*
2: Y — ***real*** *Input*

On entry: the x and y co-ordinates of the point at which the boundary conditions are to be evaluated.

3: A — ***real*** *Output*
4: B — ***real*** *Output*
5: C — ***real*** *Output*

On exit: A, B and C must be set to the values of the functions appearing in the boundary conditions.

6: IBND — INTEGER *Input*

On entry: specifies on which boundary the point (X,Y) lies. IBND = 0, 1, 2 or 3 according as the point lies on the bottom, right, top or left boundary.

BNDY must be declared as EXTERNAL in the (sub)program from which D03EEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: NGX — INTEGER *Input*
8: NGY — INTEGER *Input*

On entry: the number of interior grid points in the x- and y-directions respectively, n_x and n_y. If the seven-diagonal equations are to be solved by D03EDF, then NGX − 1 and NGY − 1 should preferably be divisible by as high a power of 2 as possible.

Constraint: NGX ≥ 3, NGY ≥ 3.

9: LDA — INTEGER *Input*

On entry: the first dimension of the array A as declared in the (sub)program from which D03EEF is called.

Constraint: if only the seven-diagonal equations are required, then LDA ≥ NGX × NGY. If a call to this routine is to be followed by a call to D03EDF to solve the seven-diagonal linear equations, LDA ≥ (4 × (NGX+1) × (NGY+1))/3.

Note. This routine only checks the former condition. D03EDF, if called, will check the latter condition.

10: A(LDA,7) — ***real*** array *Output*

On exit: A(i,j), for $i = 1,2,\ldots$,NGX × NGY; j = 1,2,...,7, contains the seven-diagonal linear equations produced by the discretization described above. If LDA > NGX × NGY, the remaining elements are not referenced by the routine, but if LDA ≥ (4 × (NGX+1) × (NGY+1))/3 then the array A can be passed directly to D03EDF, where these elements are used as workspace.

11: RHS(LDA) — ***real*** array *Output*

On exit: the first NGX × NGY elements contain the right-hand sides of the seven-diagonal linear equations produced by the discretization described above. If LDA > NGX × NGY, the remaining elements are not referenced by the routine, but if LDA ≥ (4 × (NGY+1) × (NGY+1))/3 then the array RHS can be passed directly to D03EDF, where these elements are used as workspace.

12: SCHEME — CHARACTER*1 *Input*

On entry: the type of approximation to be used for the first derivatives which occur in the partial differential equation.

If SCHEME = 'C', then central differences are used.

If SCHEME = 'U', then upwind differences are used.

Constraint: SCHEME = 'C' or 'U'.

Note. Generally speaking, if at least one of the coefficients multiplying the first derivatives (DELTA or EPSLON as returned by PDEF) are large compared with the coefficients multiplying the second derivatives, then upwind differences may be more appropriate. Upwind differences are less accurate than central differences, but may result in more rapid convergence for strongly convective equations. The easiest test is to try both schemes.

13: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then 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 specified by the routine:

IFAIL = 1

On entry, XMIN ≥ XMAX,
or YMIN ≥ YMAX,
or NGX < 3,
or NGY < 3,
or LDA < NGX × NGY,
or SCHEME is not one of 'C' or 'U'.

IFAIL = 2

At some point on the boundary there is a derivative in the boundary conditions (B ≠ 0 on return from a BNDY) and there is a non-zero coefficient of the mixed derivative $\frac{\partial^2 U}{\partial x \partial y}$ (BETA ≠ 0 on return from PDEF).

IFAIL = 3

A null boundary has been specified, i.e., at some point both A and B are zero on return from a call to BNDY.

IFAIL = 4

The equation is not elliptic, i.e., 4 × ALPHA × GAMMA < BETA^2 after a call to PDEF. The discretization has been completed, but the convergence of D03EDF cannot be guaranteed.

IFAIL = 5

The boundary conditions are purely Neumann (only the derivative is specified) and there is, in general, no unique solution.

IFAIL = 6

The equations were not diagonally dominant. (See Section 3).

7 Accuracy

Not applicable.

8 Further Comments

If this routine is used as a pre-processor to the multigrid routine D03EDF it should be noted that the rate of convergence of that routine is strongly dependent upon the number of levels in the multigrid scheme, and thus the choice of NGX and NGY is very important.

9 Example

The program solves the elliptic partial differential equation

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + 50\left\{\frac{\partial U}{\partial x} + \frac{\partial U}{\partial y}\right\} = f(x,y)$$

on the unit square $0 \le x, y \le 1$, with boundary conditions

$\frac{\partial U}{\partial n}$ given on $x = 0$ and $y = 0$,
U given on $x = 1$ and $y = 1$.

The function $f(x,y)$ and the exact form of the boundary conditions are derived from the exact solution $U(x,y) = \sin x \sin y$.

The equation is first solved using central differences. Since the coefficients of the first derivatives are large, the linear equations are not diagonally dominated, and convergence is slow. The equation is solved a second time with upwind differences, showing that convergence is more rapid, but the solution is less accurate.

9.1 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.

```
*     D03EEF Example Program Text
*     Mark 16 Revised. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          LEVELS, NGX, NGY, LDA
      PARAMETER        (LEVELS=3,NGX=2**LEVELS+1,NGY=NGX,LDA=4*(NGX+1)
```

```
     +                 *(NGY+1)/3)
*     .. Arrays in Common ..
      real             USER(6)
*     .. Local Scalars ..
      real             ACC, HX, HY, PI, RMSERR, XMAX, XMIN, YMAX, YMIN
      INTEGER          I, IFAIL, IOUT, J, MAXIT, NUMIT
*     .. Local Arrays ..
      real             A(LDA,7), RHS(LDA), U(LDA), UB(NGX*NGY), US(LDA),
     +                 X(NGX*NGY), Y(NGX*NGY)
*     .. External Functions ..
      real             FEXACT, X01AAF
      EXTERNAL         FEXACT, X01AAF
*     .. External Subroutines ..
      EXTERNAL         BNDY, D03EDF, D03EEF, PDEF
*     .. Intrinsic Functions ..
      INTRINSIC        real, SQRT
*     .. Common blocks ..
      COMMON           /BLOCK1/USER
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03EEF Example Program Results'
      WRITE (NOUT,*)
      PI = X01AAF(0.0e0)
*
*     USER(1) .. USER(6) contain the coefficients ALPHA, BETA, GAMMA,
*     DELTA, EPSLON and PHI appearing in the example partial
*     differential equation. They are stored in COMMON for use in PDEF.
*
      USER(1) = 1.0e0
      USER(2) = 0.0e0
      USER(3) = 1.0e0
      USER(4) = 50.0e0
      USER(5) = 50.0e0
      USER(6) = 0.0e0
*
      XMIN = 0.0e0
      XMAX = 1.0e0
      YMIN = 0.0e0
      YMAX = 1.0e0
      HX = (XMAX-XMIN)/real(NGX-1)
      HY = (YMAX-YMIN)/real(NGY-1)
      DO 40 I = 1, NGX
         DO 20 J = 1, NGY
            X(I+(J-1)*NGX) = XMIN + real(I-1)*HX
            Y(I+(J-1)*NGX) = YMIN + real(J-1)*HY
   20    CONTINUE
   40 CONTINUE
*
*     Discretize the equations
*
      IFAIL = -1
*
      CALL D03EEF(XMIN,XMAX,YMIN,YMAX,PDEF,BNDY,NGX,NGY,LDA,A,RHS,
     +            'Central',IFAIL)
*
*     Set the initial guess to zero
*
      DO 60 I = 1, NGX*NGY
         UB(I) = 0.0e0
```

```
   60 CONTINUE
*
*     Solve the equations
*
*     ** set IOUT.GE.2 to obtain intermediate output from D03EDF **
*
      IOUT = 0
      ACC = 1.0e-6
      MAXIT = 50
      IFAIL = -1
*
      CALL D03EDF(NGX,NGY,LDA,A,RHS,UB,MAXIT,ACC,US,U,IOUT,NUMIT,IFAIL)
*
*     Print out the solution
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Exact solution above computed solution'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) '  I/J', (I,I=1,NGX)
      RMSERR = 0.0e0
      DO 100 J = NGY, 1, -1
         WRITE (NOUT,*)
         WRITE (NOUT,99999) J, (FEXACT(X(I+(J-1)*NGX),Y(I+(J-1)*NGX)),
     +     I=1,NGX)
         WRITE (NOUT,99999) J, (U(I+(J-1)*NGX),I=1,NGX)
         DO 80 I = 1, NGX
            RMSERR = RMSERR + (FEXACT(X(I+(J-1)*NGX),Y(I+(J-1)*NGX))
     +               -U(I+(J-1)*NGX))**2
   80    CONTINUE
  100 CONTINUE
      RMSERR = SQRT(RMSERR/real(NGX*NGY))
      WRITE (NOUT,*)
      WRITE (NOUT,99997) 'Number of Iterations = ', NUMIT
      WRITE (NOUT,99996) 'RMS Error = ', RMSERR
*
*     Now discretize and solve the equations using upwind differences
*
      IFAIL = -1
*
      CALL D03EEF(XMIN,XMAX,YMIN,YMAX,PDEF,BNDY,NGX,NGY,LDA,A,RHS,
     +            'Upwind',IFAIL)
*
      IFAIL = -1
*
*     Set the initial guess to zero
*
      DO 120 I = 1, NGX*NGY
         UB(I) = 0.0e0
  120 CONTINUE
*
      CALL D03EDF(NGX,NGY,LDA,A,RHS,UB,MAXIT,ACC,US,U,IOUT,NUMIT,IFAIL)
*
*     Print the solution
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Exact solution above computed solution'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) '  I/J', (I,I=1,NGX)
```

```
      RMSERR = 0.0e0
      DO 160 J = NGY, 1, -1
         WRITE (NOUT,*)
         WRITE (NOUT,99999) J, (FEXACT(X(I+(J-1)*NGX),Y(I+(J-1)*NGX)),
     +     I=1,NGX)
         WRITE (NOUT,99999) J, (U(I+(J-1)*NGX),I=1,NGX)
         DO 140 I = 1, NGX
            RMSERR = RMSERR + (FEXACT(X(I+(J-1)*NGX),Y(I+(J-1)*NGX))
     +               -U(I+(J-1)*NGX))**2
  140    CONTINUE
  160 CONTINUE
      RMSERR = SQRT(RMSERR/real(NGX*NGY))
      WRITE (NOUT,*)
      WRITE (NOUT,99997) 'Number of Iterations = ', NUMIT
      WRITE (NOUT,99996) 'RMS Error = ', RMSERR
      STOP
*
99999 FORMAT (1X,I3,2X,10F7.3,:/(6X,10F7.3))
99998 FORMAT (1X,A,10I7,:/(6X,10I7))
99997 FORMAT (1X,A,I3)
99996 FORMAT (1X,A,1P,e10.2)
      END
*
      SUBROUTINE PDEF(X,Y,ALPHA,BETA,GAMMA,DELTA,EPSLON,PHI,PSI)
*     .. Scalar Arguments ..
      real              ALPHA, BETA, DELTA, EPSLON, GAMMA, PHI, PSI, X, Y
*     .. Arrays in Common ..
      real              USER(6)
*     .. Intrinsic Functions ..
      INTRINSIC         COS, SIN
*     .. Common blocks ..
      COMMON            /BLOCK1/USER
*     .. Executable Statements ..
      ALPHA = USER(1)
      BETA = USER(2)
      GAMMA = USER(3)
      DELTA = USER(4)
      EPSLON = USER(5)
      PHI = USER(6)
*
      PSI = (-ALPHA-GAMMA+PHI)*SIN(X)*SIN(Y) + BETA*COS(X)*COS(Y) +
     +      DELTA*COS(X)*SIN(Y) + EPSLON*SIN(X)*COS(Y)
*
      RETURN
      END
*
      SUBROUTINE BNDY(X,Y,A,B,C,IBND)
*     .. Parameters ..
      INTEGER           BOTTOM, RIGHT, TOP, LEFT
      PARAMETER         (BOTTOM=0,RIGHT=1,TOP=2,LEFT=3)
*     .. Scalar Arguments ..
      real              A, B, C, X, Y
      INTEGER           IBND
*     .. Intrinsic Functions ..
      INTRINSIC         SIN
*     .. Executable Statements ..
      IF (IBND.EQ.TOP .OR. IBND.EQ.RIGHT) THEN
*
```

```
*        Solution prescribed
*
         A = 1.0e0
         B = 0.0e0
         C = SIN(X)*SIN(Y)
      ELSE IF (IBND.EQ.BOTTOM) THEN
*
*        Derivative prescribed
*
         A = 0.0e0
         B = 1.0e0
         C = -SIN(X)
      ELSE IF (IBND.EQ.LEFT) THEN
*
*        Derivative prescribed
*
         A = 0.0e0
         B = 1.0e0
         C = -SIN(Y)
      END IF
*
      RETURN
      END
*
      real FUNCTION FEXACT(X,Y)
*     .. Scalar Arguments ..
      real                 X, Y
*     .. Intrinsic Functions ..
      INTRINSIC            SIN
*     .. Executable Statements ..
      FEXACT = SIN(X)*SIN(Y)
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D03EEF Example Program Results

 ** The linear equations were not diagonally dominated
 ** ABNORMAL EXIT from NAG Library routine D03EEF: IFAIL =     6
 ** NAG soft failure - control returned

 Exact solution above computed solution

   I/J      1      2      3      4      5      6      7      8      9

   9    0.000  0.105  0.208  0.308  0.403  0.492  0.574  0.646  0.708
   9    0.000  0.105  0.208  0.308  0.403  0.492  0.574  0.646  0.708

   8    0.000  0.096  0.190  0.281  0.368  0.449  0.523  0.589  0.646
   8    0.000  0.095  0.190  0.281  0.368  0.449  0.523  0.589  0.646

   7    0.000  0.085  0.169  0.250  0.327  0.399  0.465  0.523  0.574
   7    0.000  0.084  0.168  0.249  0.326  0.398  0.464  0.523  0.574
```

```
  6    0.000  0.073  0.145  0.214  0.281  0.342  0.399  0.449  0.492
  6   -0.001  0.072  0.144  0.213  0.280  0.342  0.398  0.449  0.492

  5    0.000  0.060  0.119  0.176  0.230  0.281  0.327  0.368  0.403
  5   -0.001  0.059  0.118  0.174  0.229  0.280  0.326  0.368  0.403

  4    0.000  0.046  0.091  0.134  0.176  0.214  0.250  0.281  0.308
  4   -0.001  0.044  0.089  0.133  0.174  0.213  0.249  0.281  0.308

  3    0.000  0.031  0.061  0.091  0.119  0.145  0.169  0.190  0.208
  3   -0.001  0.029  0.060  0.089  0.118  0.144  0.168  0.190  0.208

  2    0.000  0.016  0.031  0.046  0.060  0.073  0.085  0.096  0.105
  2   -0.001  0.014  0.029  0.044  0.059  0.072  0.084  0.095  0.105

  1    0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
  1   -0.001 -0.001 -0.001 -0.001 -0.001 -0.001  0.000  0.000  0.000

Number of Iterations =  10
RMS Error =   7.92E-04

Exact solution above computed solution

  I/J      1      2      3      4      5      6      7      8      9

  9    0.000  0.105  0.208  0.308  0.403  0.492  0.574  0.646  0.708
  9    0.000  0.105  0.208  0.308  0.403  0.492  0.574  0.646  0.708

  8    0.000  0.096  0.190  0.281  0.368  0.449  0.523  0.589  0.646
  8   -0.002  0.093  0.186  0.276  0.362  0.443  0.517  0.585  0.646

  7    0.000  0.085  0.169  0.250  0.327  0.399  0.465  0.523  0.574
  7   -0.005  0.078  0.160  0.239  0.316  0.388  0.455  0.517  0.574

  6    0.000  0.073  0.145  0.214  0.281  0.342  0.399  0.449  0.492
  6   -0.008  0.063  0.132  0.200  0.266  0.329  0.388  0.443  0.492

  5    0.000  0.060  0.119  0.176  0.230  0.281  0.327  0.368  0.403
  5   -0.011  0.047  0.103  0.159  0.214  0.266  0.316  0.362  0.403

  4    0.000  0.046  0.091  0.134  0.176  0.214  0.250  0.281  0.308
  4   -0.013  0.030  0.074  0.117  0.159  0.200  0.239  0.276  0.308

  3    0.000  0.031  0.061  0.091  0.119  0.145  0.169  0.190  0.208
  3   -0.015  0.014  0.044  0.074  0.103  0.132  0.160  0.186  0.208

  2    0.000  0.016  0.031  0.046  0.060  0.073  0.085  0.096  0.105
  2   -0.016 -0.001  0.014  0.030  0.047  0.063  0.078  0.093  0.105

  1    0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
  1   -0.016 -0.016 -0.015 -0.013 -0.011 -0.008 -0.005 -0.002  0.000

Number of Iterations =   4
RMS Error =   1.05E-02
```

D03FAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D03FAF solves the Helmholtz equation in Cartesian co-ordinates in three dimensions using the standard seven-point finite difference approximation. This routine is designed to be particularly efficient on vector processors.

2. Specification

```
      SUBROUTINE D03FAF (XS, XF, L, LBDCND, BDXS, BDXF, YS, YF,
     1                   M, MBDCND, BDYS, BDYF, ZS, ZF, N, NBDCND, BDZS,
     2                   BDZF, LAMBDA, LDIMF, MDIMF, F, PERTRB, W, LWRK,
     3                   IFAIL)
      INTEGER          L, LBDCND, M, MBDCND, N, NBDCND, LDIMF, MDIMF,
     1                 LWRK, IFAIL
      real             XS, XF, BDXS(MDIMF,N+1), BDXF(MDIMF,N+1), YS, YF,
     1                 BDYS(LDIMF,N+1), BDYF(LDIMF,N+1), ZS, ZF,
     2                 BDZS(LDIMF,M+1), BDZF(LDIMF,M+1), LAMBDA,
     3                 F(LDIMF,MDIMF,N+1), PERTRB, W(LWRK)
```

3. Description

D03FAF solves the three-dimensional Helmholtz equation in cartesian co-ordinates:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + \lambda u = f(x,y,z).$$

This subroutine forms the system of linear equations resulting from the standard seven-point finite difference equations, and then solves the system using a method based on the fast Fourier transform (FFT) described by Swarztrauber [1]. This subroutine is based on the routine HW3CRT from FISHPACK (see Swarztrauber and Sweet [2]).

More precisely, the routine replaces all the second derivatives by second-order central difference approximations, resulting in a block tridiagonal system of linear equations. The equations are modified to allow for the prescribed boundary conditions. Either the solution or the derivative of the solution may be specified on any of the boundaries, or the solution may be specified to be periodic in any of the three dimensions. By taking the discrete Fourier transform in the x- and y-directions, the equations are reduced to sets of tridiagonal systems of equations. The Fourier transforms required are computed using the multiple FFT routines found in Chapter C06 of the NAG Fortran Library.

4. References

[1] SWARZTRAUBER, P.N.
Fast Poisson Solvers.
In: 'Studies in Numerical Analysis', G.H. Golub, (Ed.).
Mathematical Association of America, 1984.

[2] SWARZTRAUBER, P.N. and SWEET, R.A.
Efficient Fortran Subprograms for the Solution of Separable Elliptic Partial Differential Equations.
ACM Trans. Math. Softw., 5, pp. 352-364, 1979.

5. Parameters

1: XS – ***real***. *Input*

On entry: the lower bound of the range of x, i.e. XS $\le x \le$ XF.

Constraint: XS < XF.

2: XF – ***real***. *Input*

On entry: the upper bound of the range of x, i.e. XS $\le x \le$ XF.

Constraint: XS < XF.

3: L – INTEGER. *Input*

On entry: the number of panels into which the interval (XS,XF) is subdivided. Hence, there will be L+1 grid points in the x-direction given by x_i = XS + $(i-1)\times\delta x$, for $i = 1,2,\ldots,$L+1, where δx = (XF−XS)/L is the panel width.

Constraint: L $\ge$ 5.

4: LBDCND – INTEGER. *Input*

On entry: indicates the type of boundary conditions at x = XS and x = XF.

LBDCND = 0

if the solution is periodic in x, i.e. $u(\text{XS},y,z) = u(\text{XF},y,z)$.

LBDCND = 1

if the solution is specified at x = XS and x = XF.

LBDCND = 2

if the solution is specified at x = XS and the derivative of the solution with respect to x is specified at x = XF.

LBDCND = 3

if the derivative of the solution with respect to x is specified at x = XS and x = XF.

LBDCND = 4

if the derivative of the solution with respect to x is specified at x = XS and the solution is specified at x = XF.

Constraint: 0 $\le$ LBDCND $\le$ 4.

5: BDXS(MDIMF,N+1) – ***real*** array. *Input*

On entry: the values of the derivative of the solution with respect to x at x = XS. When LBDCND = 3 or 4, BDXS(j,k) = $u_x(\text{XS},y_j,z_k)$, for j = 1,2,...,M+1; k = 1,2,...,N+1.

When LBDCND has any other value, BDXS is not referenced.

6: BDXF(MDIMF,N+1) – ***real*** array. *Input*

On entry: the values of the derivative of the solution with respect to x at x = XF. When LBDCND = 2 or 3, BDXF(j,k) = $u_x(\text{XF},y_j,z_k)$, for j = 1,2,...,M+1; k = 1,2,...,N+1.

When LBDCND has any other value, BDXF is not referenced.

7: YS – ***real***. *Input*

On entry: the lower bound of the range of y, i.e. YS $\le y \le$ YF.

Constraint: YS < YF.

8: YF – ***real***. *Input*

On entry: the upper bound of the range of y, i.e. YS $\le y \le$ YF.

Constraint: YS < YF.

9: M – INTEGER. *Input*

On entry: the number of panels into which the interval (YS,YF) is subdivided. Hence, there will be M+1 grid points in the y-direction given by y_j = YS + $(j-1)\times\delta y$ for j = 1,2,...,M+1, where δy = (YF−YS)/M is the panel width.

Constraint: M ≥ 5.

10: MBDCND – INTEGER. *Input*

On entry: indicates the type of boundary conditions at y = YS and y = YF.

MBDCND = 0

if the solution is periodic in y, i.e. $u(x,\text{YF},z) = u(x,\text{YS},z)$.

MBDCND = 1

if the solution is specified at y = YS and y = YF.

MBDCND = 2

if the solution is specified at y = YS and the derivative of the solution with respect to y is specified at y = YF.

MBDCND = 3

if the derivative of the solution with respect to y is specified at y = YS and y = YF.

MBDCND = 4

if the derivative of the solution with respect to y is specified at y = YS and the solution is specified at y = YF.

Constraint: 0 ≤ MBDCND ≤ 4.

11: BDYS(LDIMF,N+1) – ***real*** array. *Input*

On entry: the values of the derivative of the solution with respect to y at y = YS. When MBDCND = 3 or 4, BDYS(i,k) = $u_y(x_i,\text{YS},z_k)$, for i = 1,2,...,L+1; k = 1,2,...,N+1.

When MBDCND has any other value, BDYS is not referenced.

12: BDYF(LDIMF,N+1) – ***real*** array. *Input*

On entry: the values of the derivative of the solution with respect to y at y = YF. When MBDCND = 2 or 3, BDYF(i,k) = $u_y(x_i,\text{YF},z_k)$, for i = 1,2,...,L+1; k = 1,2,...,N+1.

When MBDCND has any other value, BDYF is not referenced.

13: ZS – ***real***. *Input*

On entry: the lower bound of the range of z, i.e. ZS ≤ z ≤ ZF.

Constraint: ZS < ZF.

14: ZF – ***real***. *Input*

On entry: the upper bound of the range of z, i.e. ZS ≤ z ≤ ZF.

Constraint: ZS < ZF.

15: N – INTEGER. *Input*

On entry: the number of panels into which the interval (ZS,ZF) is subdivided. Hence, there will be N+1 grid points in the z-direction given by z_k = ZS + $(k-1)\times\delta z$, for k = 1,2,...,N+1, where δz = (ZF−ZS)/N is the panel width.

Constraint: N ≥ 5.

16: NBDCND – INTEGER. *Input*

On entry: specifies the type of boundary conditions at z = ZS and z = ZF.

NBDCND = 0

if the solution is periodic in z, i.e. $u(x,y,\mathrm{ZF}) = u(x,y,\mathrm{ZS})$.

NBDCND = 1

if the solution is specified at z = ZS and z = ZF.

NBDCND = 2

if the solution is specified at z = ZS and the derivative of the solution with respect to z is specified at z = ZF.

NBDCND = 3

if the derivative of the solution with respect to z is specified at z = ZS and z = ZF.

NBDCND = 4

if the derivative of the solution with respect to z is specified at z = ZS and the solution is specified at z = ZF.

Constraint: 0 ≤ NBDCND ≤ 4.

17: BDZS(LDIMF,M+1) – ***real*** array. *Input*

On entry: the values of the derivative of the solution with respect to z at z = ZS. When NBDCND = 3 or 4, BDZS$(i,j) = u_z(x_i,y_j,\mathrm{ZS})$, for i = 1,2,...,L+1; j = 1,2,...,M+1.

When NBDCND has any other value, BDZS is not referenced.

18: BDZF(LDIMF,M+1) – ***real*** array. *Input*

On entry: the values of the derivative of the solution with respect to z at z = ZF. When NBDCND = 2 or 3, BDZF$(i,j) = u_z(x_i,y_j,\mathrm{ZF})$, for i = 1,2,...,L+1; j = 1,2,...,M+1.

When NBDCND has any other value, BDZF is not referenced.

19: LAMBDA – ***real***. *Input*

On entry: the constant λ in the Helmholtz equation. For certain positive values of λ a solution to the differential equation may not exist, and close to these values the solution of the discretized problem will be extremely ill-conditioned. If $\lambda > 0$, then D03FAF will set IFAIL to 3, but will still attempt to find a solution. However, since in general the values of λ for which no solution exists cannot be predicted a priori, the user is advised to treat any results computed with $\lambda > 0$ with great caution.

20: LDIMF – INTEGER. *Input*

On entry: the first dimension of the arrays F, BDYS, BDYF, BDZS and BDZF as declared in the (sub)program from which D03FAF is called.

Constraint: LDIMF ≥ L + 1.

21: MDIMF – INTEGER. *Input*

On entry: the second dimension of the array F and the first dimension of the arrays BDXS and BDXF as declared in the (sub)program from which D03FAF is called.

Constraint: MDIMF ≥ M + 1.

22: F(LDIMF,MDIMF,N+1) – ***real*** array. *Input/Output*

On entry: the values of the right-side of the Helmholtz equation and boundary values (if any).

$\mathrm{F}(i,j,k) = f(x_i,y_j,z_k)$ i = 2,3,...,L, j = 2,3,...,M and k = 2,3,...,N.

On the boundaries F is defined by

LBDCND	F(1,*j*,*k*)	F(L+1,*j*,*k*)	
0	$f(XS,y_j,z_k)$	$f(XS,y_j,z_k)$	
1	$u(XS,y_j,z_k)$	$u(XF,y_j,z_k)$	
2	$u(XS,y_j,z_k)$	$f(XF,y_j,z_k)$	j = 1,2,...,M+1
3	$f(XS,y_j,z_k)$	$f(XF,y_j,z_k)$	k = 1,2,...,N+1
4	$f(XS,y_j,z_k)$	$u(XF,y_j,z_k)$	

MBDCND	F(*i*,1,*k*)	F(*i*,M+1,*k*)	
0	$f(x_i,YS,z_k)$	$f(x_i,YS,z_k)$	
1	$u(x_i,YS,z_k)$	$u(x_i,YF,z_k)$	
2	$u(x_i,YS,z_k)$	$f(x_i,YF,z_k)$	i = 1,2,...,L+1
3	$f(x_i,YS,z_k)$	$f(x_i,YF,z_k)$	k = 1,2,...,N+1
4	$f(x_i,YS,z_k)$	$u(x_i,YF,z_k)$	

NBDCND	F(*i*,*j*,1)	F(*i*,*j*,N+1)	
0	$f(x_i,y_j,ZS)$	$f(x_i,y_j,ZS)$	
1	$u(x_i,y_j,ZS)$	$u(x_i,y_j,ZF)$	
2	$u(x_i,y_j,ZS)$	$f(x_i,y_j,ZF)$	i = 1,2,...,L+1
3	$f(x_i,y_j,ZS)$	$f(x_i,y_j,ZF)$	j = 1,2,...,M+1
4	$f(x_i,y_j,ZS)$	$u(x_i,y_j,ZF)$	

Note: if the table calls for both the solution u and the right-hand side f on a boundary, then the solution must be specified.

On exit: F contains the solution $u(i,j,k)$ of the finite difference approximation for the grid point (x_i,y_j,z_k) for i = 1,2,...,L+1, j = 1,2,...,M+1 and k = 1,2,...,N+1.

23: PERTRB – ***real***. *Output*

On exit: PERTRB = 0, unless a solution to Poisson's equation ($\lambda = 0$) is required with a combination of periodic or derivative boundary conditions (LBDCND, MBDCND and NBDCND = 0 or 3). In this case a solution may not exist. PERTRB is a constant, calculated and subtracted from the array F, which ensures that a solution exists. D03FAF then computes this solution, which is a least-squares solution to the original approximation. This solution is not unique and is unnormalised. The value of PERTRB should be small compared to the right-hand side F, otherwise a solution has been obtained to an essentially different problem. This comparison should always be made to insure that a meaningful solution has been obtained.

24: W(LWRK) – ***real*** array. *Workspace*

25: LWRK – INTEGER. *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03FAF is called. 2×(N+1)×max(L,M) + 3×L + 3×M + 4×N + 6 is an upper bound on the required size of W. If LWRK is too small, the routine exits with IFAIL = 2, and if on entry IFAIL = 0 or IFAIL = −1, a message is output giving the exact value of LWRK required to solve the current problem.

26: 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 or gives a warning (see Section 6).

For this routine, because the values of output parameters may be useful even if IFAIL ≠ 0 on exit, users are recommended to set IFAIL to −1 before entry. **It is then essential to test the value of IFAIL on exit.**

6. Error Indicators and Warnings

Errors or warnings specified by the routine:

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

IFAIL = 1

On entry, XS ≥ XF,
or L < 5,
or LBDCND < 0,
or LBDCND > 4,
or YS ≥ YF,
or M < 5,
or MBDCND < 0,
or MBDCND > 4,
or ZS ≥ ZF,
or N < 5,
or NBDCND < 0,
or NBDCND > 4,
or LDIMF < L + 1,
or MDIMF < M + 1.

IFAIL = 2

On entry, LWRK is too small.

IFAIL = 3

On entry, $\lambda > 0$.

7. Accuracy

Not applicable.

8. Further Comments

The execution time is roughly proportional to $L \times M \times N \times (\log_2 L + \log_2 M + 5)$, but also depends on input parameters LBDCND and MBDCND.

9. Example

The example solves the Helmholz equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + \lambda u = f(x,y,z)$$

for $(x,y,z) \in [0,1]\times[0,2\pi]\times[0,\frac{\pi}{2}]$ where $\lambda = -2$, and $f(x,y,z)$ is derived from the exact solution

$$u(x,y,z) = x^4 \sin y \cos z.$$

The equation is subject to the following boundary conditions, again derived from the exact solution given above.

$u(0,y,z)$ and $u(1,y,z)$ are prescribed (i.e. LBDCND = 1).

$u(x,0,z) = u(x,2\pi,z)$ (i.e. MBDCND = 0).

$u(x,y,0)$ and $u_x(x,y,\frac{\pi}{2})$ are prescribed (i.e. NBDCND = 2).

9.1. 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.

```
*     D03FAF Example Program Text
*     Mark 14 Release.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          L, M, N, MAXLM, LDIMF, MDIMF, LWRK
      PARAMETER        (L=16,M=32,N=20,MAXLM=32,LDIMF=L+1,MDIMF=M+1,
     +                 LWRK=2*(N+1)*MAXLM+3*L+3*M+4*N+6000)
*     .. Local Scalars ..
      real             DX, DY, DZ, ERROR, LAMBDA, PERTRB, PI, T, XF, XS,
     +                 YF, YS, ZF, ZS
      INTEGER          I, IFAIL, J, K, LBDCND, MBDCND, NBDCND
*     .. Local Arrays ..
      real             BDXF(MDIMF,N+1), BDXS(MDIMF,N+1),
     +                 BDYF(LDIMF,N+1), BDYS(LDIMF,N+1),
     +                 BDZF(LDIMF,M+1), BDZS(LDIMF,M+1),
     +                 F(LDIMF,MDIMF,N+1), W(LWRK), X(L+1), Y(M+1),
     +                 Z(N+1)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         D03FAF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, COS, real, SIN
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03FAF Example Program Results'
      LAMBDA = -2.0e0
      XS = 0.0e0
      XF = 1.0e0
      LBDCND = 1
      YS = 0.0e0
      PI = X01AAF(PI)
      YF = 2.0e0*PI
      MBDCND = 0
      ZS = 0.0e0
      ZF = PI/2.0e0
      NBDCND = 2
*
*     Define the grid points for later use.
*
      DX = (XF-XS)/real(L)
      DO 20 I = 1, L + 1
         X(I) = XS + real(I-1)*DX
   20 CONTINUE
      DY = (YF-YS)/real(M)
      DO 40 J = 1, M + 1
         Y(J) = YS + real(J-1)*DY
   40 CONTINUE
      DZ = (ZF-ZS)/real(N)
      DO 60 K = 1, N + 1
         Z(K) = ZS + real(K-1)*DZ
   60 CONTINUE
*
*     Define the array of derivative boundary values.
*
      DO 100 J = 1, M + 1
         DO 80 I = 1, L + 1
            BDZF(I,J) = -X(I)**4*SIN(Y(J))
   80    CONTINUE
  100 CONTINUE
*
```

```
*     Note that for this example all other boundary arrays are
*     dummy variables.
*
*     We define the function boundary values in the F array.
*
      DO 140 K = 1, N + 1
         DO 120 J = 1, M + 1
            F(1,J,K) = 0.0e0
            F(L+1,J,K) = SIN(Y(J))*COS(Z(K))
  120    CONTINUE
  140 CONTINUE
      DO 180 J = 1, M + 1
         DO 160 I = 1, L + 1
            F(I,J,1) = X(I)**4*SIN(Y(J))
  160    CONTINUE
  180 CONTINUE
*
*     Define the values of the right hand side of the Helmholtz
*     equation.
*
      DO 240 K = 2, N + 1
         DO 220 J = 1, M + 1
            DO 200 I = 2, L
               F(I,J,K) = 4.0e0*X(I)**2*(3.0e0-X(I)**2)*SIN(Y(J))
     +                    *COS(Z(K))
  200       CONTINUE
  220    CONTINUE
  240 CONTINUE
*
*     Call D03FAF to generate and solve the finite difference equation.
*
      IFAIL = 0
*
      CALL D03FAF(XS,XF,L,LBDCND,BDXS,BDXF,YS,YF,M,MBDCND,BDYS,BDYF,ZS,
     +            ZF,N,NBDCND,BDZS,BDZF,LAMBDA,LDIMF,MDIMF,F,PERTRB,W,
     +            LWRK,IFAIL)
*
*     Compute discretization error.  The exact solution to the
*     problem is
*
*        U(X,Y,Z) = X**4*SIN(Y)*COS(Z)
*
      ERROR = 0.0e0
      DO 300 K = 1, N + 1
         DO 280 J = 1, M + 1
            DO 260 I = 1, L + 1
               T = ABS(F(I,J,K)-X(I)**4*SIN(Y(J))*COS(Z(K)))
               IF (T.GT.ERROR) ERROR = T
  260       CONTINUE
  280    CONTINUE
  300 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'Maximum component of discretization error =',
     +  ERROR
      STOP
*
99999 FORMAT (1X,A,1P,e13.6)
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D03FAF Example Program Results

 Maximum component of discretization error = 5.176553E-04
```

D03MAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D03MAF places a triangular mesh over a given two-dimensional region. The region may have any shape, including one with holes.

2. Specification

```
      SUBROUTINE D03MAF (H, M, N, NB, NPTS, PLACES, INDEX, IDIM, IN, DIST,
     1                   LD, IFAIL)
      INTEGER          M, N, NB, NPTS, INDEX(4,IDIM), IDIM, IN, LD,
     1                 IFAIL
      real             H, PLACES(2,IDIM), DIST(4,LD)
      EXTERNAL         IN
```

3. Description

This subroutine begins with a uniform triangular grid as shown in Figure 1 and assumes that the region to be triangulated lies within the rectangle given by the inequalities

$$0 < x < \sqrt{3}(m-1)h, \quad 0 < y < (n-1)h.$$

This rectangle is drawn in bold in Figure 1. The region is specified by the user's function IN which must determine whether any given point (x,y) lies in the region. The uniform grid is processed columnwise, with (x_1,y_1) preceding (x_2,y_2) if $x_1 < x_2$ or $x_1 = x_2, y_1 < y_2$. Points near the boundary are moved onto it and points well outside the boundary are omitted. The direction of movement is chosen to avoid pathologically thin triangles. The points accepted are numbered in exactly the same order as the corresponding points of the uniform grid were scanned. The output consists of the x,y co-ordinates of all grid points and integers indicating whether they are internal and to which other points they are joined by triangle sides.

The mesh size h must be chosen small enough for the essential features of the region to be apparent from testing all points of the original uniform grid for being inside the region. For instance if any hole is within $2h$ of another hole or the outer boundary then a triangle may be found with all vertices within $\frac{1}{2}h$ of a boundary. Such a triangle is taken to be external to the region so the effect will be to join the hole to another hole or to the external region.

Further details of the algorithm are given in the references.

4. References

[1] REID, J.K.
On the Construction and Convergence of a Finite-element Solution of Laplace's Equation.
J. Inst. Math. Appl., 9, pp. 1-13, 1972.

[2] REID, J.K.
Fortran Subroutines for the Solutions of Laplace's Equation over a General Routine in Two Dimensions.
Harwell Report TP.422, 1970.

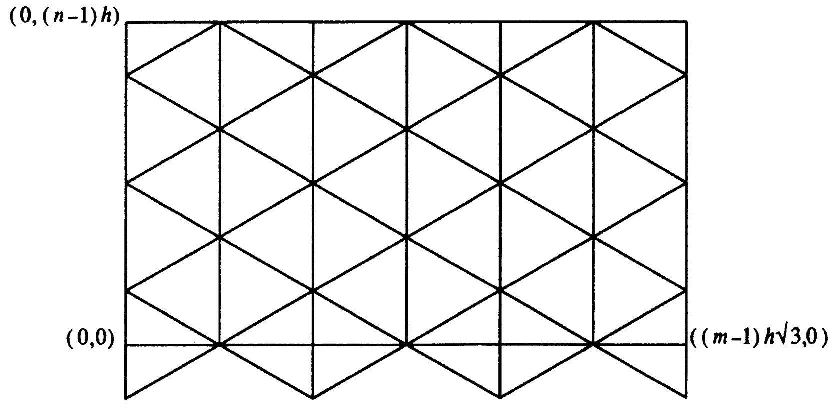


Figure 1
The original grid for $m = n = 4$

5. Parameters

1: H – ***real***. *Input*

On entry: the required length, h, for the sides of the triangles of the uniform mesh.

2: M – INTEGER. *Input*
3: N – INTEGER. *Input*

On entry: values m and n such that all points (x,y) inside the region satisfy the inequalities

$0 \le x \le \sqrt{3}(m-1)h,$

$0 \le y \le (n-1)h.$

Constraint: M, N > 2.

4: NB – INTEGER. *Input*

On entry: the number of times a triangle side is bisected to find a point on the boundary. A value of 10 is adequate for most purposes (see Section 7).

Constraint: NB ≥ 1.

5: NPTS – INTEGER. *Output*

On exit: the number of points in the triangulation.

6: PLACES(2,IDIM) – ***real*** array. *Output*

On exit: the x and y co-ordinates respectively of the ith point of the triangulation.

7: INDEX(4,IDIM) – INTEGER array. *Output*

On exit: INDEX$(1,i)$ contains i if point i is inside the region and $-i$ if it is on the boundary. For each triangle side between points i and j with $j > i$, INDEX(k,i), $k > 1$, contains j or $-j$ according to whether point j is internal or on the boundary. There can never be more than three such points. If there are less, then some values INDEX(k,i), $k > 1$, are zero.

8: IDIM – INTEGER. *Input*

On entry: the second dimension of the arrays PLACES and INDEX as declared in the (sub)program from which D03MAF is called.

Constraint: IDIM ≥ NPTS.

9: IN – INTEGER FUNCTION, supplied by the user. *External Procedure*

IN must return the value 1 if the given point (X,Y) lies inside the region, and 0 if it lies outside.

Its specification is:

```
INTEGER FUNCTION  IN(X, Y)
real              X, Y
```

1: X – ***real***. *Input*

2: Y – ***real***. *Input*

On entry: the co-ordinates of the given point.

IN must be declared as EXTERNAL in the (sub)program from which D03MAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: DIST(4,LD) – ***real*** array. *Workspace*

11: LD – INTEGER. *Input*

On entry: the second dimension of the array DIST as declared in the (sub)program from which D03MAF is called.

Constraint: LD $\geq$ 4N.

12: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

IDIM is too small.

IFAIL = 2

A point inside the region violates one of the constraints (see parameters M and N above).

IFAIL = 3

LD is too small.

IFAIL = 4

M $\leq$ 2.

IFAIL = 5

N $\leq$ 2.

IFAIL = 6

NB $\leq$ 0.

7. Accuracy

Points are moved onto the boundary by bisecting a triangle side NB times. The accuracy is therefore $h \times 2^{-NB}$.

8. Further Comments

The time taken by the routine is approximately proportional to $m \times n$.

9. Example

The following program triangulates the circle with centre (7.0,7.0) and radius 6.0 using a basic grid size $h = 4.0$.

9.1. 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.

```
*     D03MAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          IDIM, LD
      PARAMETER        (IDIM=100,LD=20)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             H
      INTEGER          I, IFAIL, J, M, N, NB, NPTS
*     .. Local Arrays ..
      real             DIST(4,LD), PLACES(2,IDIM)
      INTEGER          INDEX(4,IDIM)
*     .. External Functions ..
      INTEGER          IN1
      EXTERNAL         IN1
*     .. External Subroutines ..
      EXTERNAL         D03MAF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03MAF Example Program Results'
      WRITE (NOUT,*)
      H = 4.0e0
      M = 3
      N = 5
      NB = 10
      IFAIL = 0
*
      CALL D03MAF(H,M,N,NB,NPTS,PLACES,INDEX,IDIM,IN1,DIST,LD,IFAIL)
*
      WRITE (NOUT,*) '  I    X(I)      Y(I)'
      DO 20 I = 1, NPTS
         WRITE (NOUT,99999) I, PLACES(1,I), PLACES(2,I)
   20 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Index'
      DO 40 I = 1, NPTS
         WRITE (NOUT,99998) (INDEX(J,I),J=1,4)
   40 CONTINUE
      STOP
*
99999 FORMAT (1X,I3,2F10.6)
99998 FORMAT (1X,4I5)
      END
*
      INTEGER FUNCTION IN1(X,Y)
*     Circular domain
*     .. Scalar Arguments ..
      real                 X, Y
*     .. Executable Statements ..
      IF ((X-7.0e0)**2+(Y-7.0e0)**2.LE.36.0e0) THEN
         IN1 = 1
      ELSE
         IN1 = 0
      END IF
      RETURN
      END
```

9.2. Program Data

None.

9.3. Program Results

```
D03MAF Example Program Results

  I    X(I)      Y(I)
  1  1.013182  6.584961
  2  1.412366  9.184570
  3  2.268242  3.309570
  4  3.464102  8.000000
  5  3.584195 11.930664
  6  6.928203  1.001953
  7  6.928203  6.000000
  8  6.928203 10.000000
  9  6.928203 12.998047
 10 11.686269  3.252930
 11 10.392305  8.000000
 12 10.392305 11.947266
 13 12.978541  6.506836
 14 12.562443  9.252930

Index
    -1    -3     4    -2
    -2     4    -5     0
    -3    -6     7     4
     4     7     8    -5
    -5     8    -9     0
    -6     0   -10     7
     7   -10    11     8
     8    11   -12    -9
    -9   -12     0     0
   -10     0   -13    11
    11   -13   -14   -12
   -12   -14     0     0
   -13     0     0   -14
   -14     0     0     0
```

D03PCF – NAG Fortran Library Routine Document

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

D03PCF integrates a system of linear or nonlinear parabolic partial differential equations (PDEs) in one space variable. The spatial discretisation is performed using finite differences, and the method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs). The resulting system is solved using a backward differentiation formula method.

2 Specification

```
      SUBROUTINE D03PCF(NPDE, M, TS, TOUT, PDEDEF, BNDARY, U, NPTS, X,
     1                  ACC, W, NW, IW, NIW, ITASK, ITRACE, IND, IFAIL)
      INTEGER           NPDE, M, NPTS, NW, IW(NIW), NIW, ITASK, ITRACE,
     1                  IND, IFAIL
      real              TS, TOUT, U(NPDE,NPTS), X(NPTS), ACC, W(NW)
      EXTERNAL          PDEDEF, BNDARY
```

3 Description

D03PCF integrates the system of parabolic equations:

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

where $P_{i,j}$, Q_i and R_i depend on x, t, U, U_x and the vector U is the set of solution values

$$U(x,t) = [U_1(x,t), ..., U_{\mathrm{NPDE}}(x,t)]^T, \tag{2}$$

and the vector U_x is its partial derivative with respect to x. Note that $P_{i,j}$, Q_i and R_i must not depend on $\frac{\partial U}{\partial t}$.

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 $b = x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$. The co-ordinate system in space is defined by the value of m; $m = 0$ for Cartesian co-ordinates, $m = 1$ for cylindrical polar co-ordinates and $m = 2$ for spherical polar co-ordinates. The mesh should be chosen in accordance with the expected behaviour of the solution.

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

The initial values of the functions $U(x,t)$ must be given at $t = t_0$. The functions R_i, for $i = 1,2,...,\mathrm{NPDE}$, which may be thought of as fluxes, are also used in the definition of the boundary conditions for each equation. The boundary conditions must have the form

$$\beta_i(x,t)R_i(x,t,U,U_x) = \gamma_i(x,t,U,U_x), \quad i = 1,2,...,\mathrm{NPDE}, \tag{3}$$

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

The boundary conditions must be specified in a subroutine BNDARY provided by the user.

The problem is subject to the following restrictions:

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

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

(iii) The evaluation of the functions $P_{i,j}$, Q_i and R_i is done at the mid-points of the mesh intervals by calling the routine PDEDEF for each mid-point in turn. Any discontinuities in these functions **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$;

(iv) At least one of the functions $P_{i,j}$ must be non-zero so that there is a time derivative present in the problem; and

(v) 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 8 below.

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 co-ordinates, 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 × NPTS ODEs in the time direction. This system is then integrated forwards in time using a backward differentiation formula method.

4 References

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

[2] 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

[3] 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

[4] Dew P M and Walsh J (1981) A set of library routines for solving parabolic equations in one space variable *ACM Trans. Math. Software* **7** 295–314

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system to be solved.

Constraint: NPDE ≥ 1.

2: M — INTEGER *Input*

On entry: the co-ordinate system used, m:

M = 0
indicates Cartesian co-ordinates,

M = 1
indicates cylindrical polar co-ordinates,

M = 2
indicates spherical polar co-ordinates.

Constraint: $0 \leq \mathrm{M} \leq 2$.

3: 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.

4: TOUT — ***real*** *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 compute the functions $P_{i,j}$, Q_i and R_i which define the system of PDEs. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PCF.

Its specification is:

```
      SUBROUTINE PDEDEF(NPDE, T, X, U, UX, P, Q, R, IRES)
      INTEGER           NPDE, IRES
      real              T, X, U(NPDE), UX(NPDE), P(NPDE,NPDE), Q(NPDE),
     1                  R(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 $i = 1,2,\ldots,\mathrm{NPDE}$.

5: UX(NPDE) — ***real*** array *Input*

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

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

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

7: Q(NPDE) — ***real*** array *Output*

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

8: R(NPDE) — ***real*** array *Output*

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

9: 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 D03PCF 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 D03PCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

BNDARY must compute the functions β_i and γ_i which define the boundary conditions as in equation (3).

Its specification is:

```
SUBROUTINE BNDARY(NPDE, T, U, UX, IBND, BETA, GAMMA, IRES)
INTEGER            NPDE, IBND, IRES
real               T, U(NPDE), UX(NPDE), BETA(NPDE), GAMMA(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: U(NPDE) — ***real*** array *Input*

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

4: UX(NPDE) — ***real*** array *Input*

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

5: IBND — INTEGER *Input*

On entry: IBND determines the position of the boundary conditions. If IBND = 0, then BNDARY must set up the coefficients of the left-hand boundary $x = a$. Any other value of IBND indicates that BNDARY must set up the coefficients of the right-hand boundary, $x = b$.

6: BETA(NPDE) — ***real*** 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.

7: GAMMA(NPDE) — ***real*** array *Output*

On exit: GAMMA(i) must be set to the value of $\gamma_i(x,t,U,U_x)$ at the boundary specified by 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 D03PCF 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 D03PCF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

On entry: the initial values of $U(x,t)$ at t = TS and the mesh points X(j), for j = 1,2,...,NPTS.

On exit: U(i,j) will contain the computed solution at t = TS.

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 spatial 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).

10: ACC — ***real*** *Input*

On entry: a positive quantity for controlling the local error estimate in the time integration. If $\mathrm{E}(i,j)$ is the estimated error for U_i at the jth mesh point, the error test is:

$$|\mathrm{E}(i,j)| = \mathrm{ACC} \times (1.0 + |\mathrm{U}(i,j)|).$$

Constraint: ACC > 0.0.

11: W(NW) — ***real*** array *Workspace*

12: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PCF is called.

Constraint: $\mathrm{NW} \geq (10+6\times\mathrm{NPDE})\times\mathrm{NPDE}\times\mathrm{NPTS}+(21+3\times\mathrm{NPDE})\times\mathrm{NPDE}+7\times\mathrm{NPTS}+54$.

13: 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 each of the boundary conditions.

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

IW(4) contains the order of the last backward differentiation formula method used.

IW(5) contains the number of Newton iterations performed by the time integrator. Each iteration involves an ODE residual evaluation followed by a back-substitution using the LU decomposition of the Jacobian matrix.

The rest of the array is used as workspace.

14: NIW — INTEGER *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D03PCF is called.

Constraint: NIW = NPDE × NPTS + 24.

15: ITASK — INTEGER *Input*

On entry: specifies 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.

ITASK = 2
: one step and return.

ITASK = 3
: stop at first internal integration point at or beyond t = TOUT.

Constraint: 1 ≤ ITASK ≤ 3.

16: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03PCF 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 subchapter D02M-N of the NAG Fortran Library.

17: 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 D03PCF.

Constraint: $0 \leq$ IND ≤ 1.

On exit: IND = 1.

18: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TOUT ≤ TS,
or (TOUT − TS) is too small,
or ITASK ≠ 1, 2 or 3,
or M ≠ 0, 1 or 2,
or M > 0 and X(1) < 0.0,
or X(i), for $i = 1, 2, \ldots,$ NPTS are not ordered,
or NPTS < 3,
or NPDE < 1,
or ACC ≤ 0.0,
or IND ≠ 0 or 1,
or NW is too small,
or NIW is too small,
or D03PCF called initially with IND = 1.

IFAIL = 2

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

IFAIL = 3

In the underlying ODE solver, there were repeated errors or corrector convergence test failures on an attempted step, before completing the requested task. The problem may have a singularity or ACC is too small for the integration to continue. Integration was successful as far as t = TS.

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 the user-supplied subroutines PDEDEF or BNDARY, when the residual in the underlying ODE solver was being evaluated.

IFAIL = 5

In solving the ODE system, a singular Jacobian has been encountered. The user should check his 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 or BNDARY. Integration was successful as far as t = TS.

IFAIL = 7

The value of ACC is so small that the routine is unable to start the integration in time.

IFAIL = 8

In one of the user-supplied routines, PDEDEF 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 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).

IFAIL = 12

Not applicable.

IFAIL = 13

Not applicable.

IFAIL = 14

The flux function R_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 accuracy parameter, ACC.

8 Further Comments

The routine is designed to solve parabolic systems (possibly including some elliptic equations) with second-order derivatives in space. The parameter specification allows the user 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 D03PEF.

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

9 Example

We use the example given in Dew and Walsh [4] which consists of an elliptic-parabolic pair of PDEs. The problem was originally derived from a single third-order in space PDE. The elliptic equation is

$$\frac{1}{r}\frac{\partial}{\partial r}\left(r^2\frac{\partial U_1}{\partial r}\right) = 4\alpha\left(U_2 + r\frac{\partial U_2}{\partial r}\right)$$

and the parabolic equation is

$$(1-r^2)\frac{\partial U_2}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(r\left(\frac{\partial U_2}{\partial r} - U_2U_1\right)\right)$$

where $(r,t) \in [0,1] \times [0,1]$. The boundary conditions are given by

$$U_1 = \frac{\partial U_2}{\partial r} = 0 \text{ at } r = 0,$$

and

$$\frac{\partial}{\partial r}(rU_1) = 0 \text{ and } U_2 = 0 \text{ at } r = 1.$$

The first of these boundary conditions implies that the flux term in the second PDE, $\left(\frac{\partial U_2}{\partial r} - U_2U_1\right)$, is zero at $r = 0$.

The initial conditions at $t = 0$ are given by

$$U_1 = 2\alpha r \text{ and } U_2 = 1.0, \text{ for } r \in [0,1].$$

The value $\alpha = 1$ was used in the problem definition. A mesh of 20 points was used with a circular mesh spacing to cluster the points towards the right-hand side of the spatial interval, $r = 1$.

9.1 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.

```
*     D03PCF Example Program Text
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, INTPTS, ITYPE, NEQN, NIW, NWK, NW
      PARAMETER        (NPDE=2,NPTS=20,INTPTS=6,ITYPE=1,NEQN=NPDE*NPTS,
     +                 NIW=NEQN+24,NWK=(10+6*NPDE)*NEQN,
     +                 NW=NWK+(21+3*NPDE)*NPDE+7*NPTS+54)
*     .. Scalars in Common ..
      real             ALPHA
```

```
*     .. Local Scalars ..
      real             ACC, HX, PI, PIBY2, TOUT, TS
      INTEGER          I, IFAIL, IND, IT, ITASK, ITRACE, M
*     .. Local Arrays ..
      real             U(NPDE,NPTS), UOUT(NPDE,INTPTS,ITYPE), W(NW),
     +                 X(NPTS), XOUT(INTPTS)
      INTEGER          IW(NIW)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PCF, D03PZF, PDEDEF, UINIT
*     .. Intrinsic Functions ..
      INTRINSIC        SIN
*     .. Common blocks ..
      COMMON           /VBLE/ALPHA
*     .. Data statements ..
      DATA             XOUT(1)/0.0e+0/, XOUT(2)/0.40e+0/,
     +                 XOUT(3)/0.6e+0/, XOUT(4)/0.8e+0/,
     +                 XOUT(5)/0.9e+0/, XOUT(6)/1.0e+0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PCF Example Program Results'
      ACC = 1.0e-3
      M = 1
      ITRACE = 0
      ALPHA = 1.0e0
      IND = 0
      ITASK = 1
*
*     Set spatial mesh points
*
      PIBY2 = 0.5e0*X01AAF(PI)
      HX = PIBY2/(NPTS-1)
      X(1) = 0.0e0
      X(NPTS) = 1.0e0
      DO 20 I = 2, NPTS - 1
         X(I) = SIN(HX*(I-1))
   20 CONTINUE
*
*     Set initial conditions
*
      TS = 0.0e0
      TOUT = 0.1e-4
      WRITE (NOUT,99999) ACC, ALPHA
      WRITE (NOUT,99998) (XOUT(I),I=1,6)
*
*     Set the initial values
*
      CALL UINIT(U,X,NPTS)
      DO 40 IT = 1, 5
         IFAIL = -1
         TOUT = 10.0e0*TOUT
*
         CALL D03PCF(NPDE,M,TS,TOUT,PDEDEF,BNDARY,U,NPTS,X,ACC,W,NW,IW,
     +               NIW,ITASK,ITRACE,IND,IFAIL)
*
*        Interpolate at required spatial points
*
```

```
         CALL D03PZF(NPDE,M,U,NPTS,X,XOUT,INTPTS,ITYPE,UOUT,IFAIL)
         WRITE (NOUT,99996) TOUT, (UOUT(1,I,1),I=1,INTPTS)
         WRITE (NOUT,99995) (UOUT(2,I,1),I=1,INTPTS)
   40 CONTINUE
*
*     Print integration statistics
*
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (//' Accuracy requirement  = ',e12.5,/' Parameter ALPHA =',
     +       '         ',e12.3,/)
99998 FORMAT ('   T  /  X   ',6F8.4,/)
99997 FORMAT (' Number of integration steps in time                   ',
     +       I4,/' Number of residual evaluations of resulting ODE sys',
     +       'tem',I4,/' Number of Jacobian evaluations               ',
     +       '          ',I4,/' Number of iterations of nonlinear solve',
     +       'r             ',I4,/)
99996 FORMAT (1X,F6.4,' U(1)',6F8.4)
99995 FORMAT (8X,'U(2)',6F8.4,/)
      END
*
      SUBROUTINE UINIT(U,X,NPTS)
*     Routine for PDE initial conditon
*     .. Scalar Arguments ..
      INTEGER           NPTS
*     .. Array Arguments ..
      real              U(2,NPTS), X(NPTS)
*     .. Scalars in Common ..
      real              ALPHA
*     .. Local Scalars ..
      INTEGER           I
*     .. Common blocks ..
      COMMON            /VBLE/ALPHA
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         U(1,I) = 2.0e0*ALPHA*X(I)
         U(2,I) = 1.0e0
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,DUDX,P,Q,R,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NPDE
*     .. Array Arguments ..
      real              DUDX(NPDE), P(NPDE,NPDE), Q(NPDE), R(NPDE),
     +                  U(NPDE)
*     .. Scalars in Common ..
      real              ALPHA
*     .. Common blocks ..
      COMMON            /VBLE/ALPHA
*     .. Executable Statements ..
      Q(1) = 4.0e0*ALPHA*(U(2)+X*DUDX(2))
      Q(2) = 0.0e+0
      R(1) = X*DUDX(1)
      R(2) = DUDX(2) - U(1)*U(2)
```

```
      P(1,1) = 0.0e+0
      P(1,2) = 0.0e0
      P(2,1) = 0.0e+0
      P(2,2) = 1.0e0 - X*X
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,U,UX,IBND,BETA,GAMMA,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NPDE
*     .. Array Arguments ..
      real              BETA(NPDE), GAMMA(NPDE), U(NPDE), UX(NPDE)
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         BETA(1) = 0.0e+0
         BETA(2) = 1.0e+0
         GAMMA(1) = U(1)
         GAMMA(2) = -U(1)*U(2)
      ELSE
         BETA(1) = 1.0e0
         BETA(2) = 0.0e+0
         GAMMA(1) = -U(1)
         GAMMA(2) = U(2)
      END IF
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D03PCF Example Program Results


 Accuracy requirement  =  0.10000E-02
 Parameter ALPHA =          0.100E+01

   T  /  X    0.0000  0.4000  0.6000  0.8000  0.9000  1.0000

 0.0001 U(1)  0.0000  0.8008  1.1988  1.5990  1.7958  1.8485
        U(2)  0.9997  0.9995  0.9994  0.9988  0.9663  0.0000

 0.0010 U(1)  0.0000  0.7982  1.1940  1.5841  1.7179  1.6734
        U(2)  0.9969  0.9952  0.9937  0.9484  0.6385  0.0000

 0.0100 U(1)  0.0000  0.7676  1.1239  1.3547  1.3635  1.2830
        U(2)  0.9627  0.9495  0.8754  0.5537  0.2908  0.0000

 0.1000 U(1)  0.0000  0.3908  0.5007  0.5297  0.5120  0.4744
        U(2)  0.5468  0.4299  0.2995  0.1479  0.0724  0.0000

 1.0000 U(1)  0.0000  0.0007  0.0008  0.0008  0.0008  0.0007
        U(2)  0.0010  0.0007  0.0005  0.0002  0.0001  0.0000
```

```
Number of integration steps in time                    78
Number of residual evaluations of resulting ODE system 378
Number of Jacobian evaluations                          25
Number of iterations of nonlinear solver               190
```

D03PDF – NAG Fortran Library Routine Document

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

D03PDF integrates a system of linear or nonlinear parabolic partial differential equations (PDEs) in one space variable. The spatial discretisation is performed using a Chebyshev C^0 collocation method, and the method of lines is employed to reduce the the PDEs to a system of ordinary differential equations (ODEs). The resulting system is solved using a backward differentiation formula method.

2 Specification

```
      SUBROUTINE D03PDF(NPDE, M, TS, TOUT, PDEDEF, BNDARY, U, NBKPTS,
     1                  XBKPTS, NPOLY, NPTS, X, UINIT, ACC, W, NW, IW,
     2                  NIW, ITASK, ITRACE, IND, IFAIL)
      INTEGER           NPDE, M, NBKPTS, NPOLY, NPTS, NW, IW(NIW), NIW,
     1                  ITASK, ITRACE, IND, IFAIL
      real              TS, TOUT, U(NPDE,NPTS), XBKPTS(NBKPTS), X(NPTS),
     1                  ACC, W(NW)
      EXTERNAL          PDEDEF, BNDARY, UINIT
```

3 Description

D03PDF integrates the system of parabolic equations:

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

where $P_{i,j}$, Q_i and R_i depend on x, t, U, U_x and the vector U is the set of solution values

$$U(x,t) = [U_1(x,t),...,U_{\mathrm{NPDE}}(x,t)]^T, \tag{2}$$

and the vector U_x is its partial derivative with respect to x. Note that $P_{i,j}$, Q_i and R_i must not depend on $\frac{\partial U}{\partial t}$.

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 $b = x_{\mathrm{NBKPTS}}$ are the leftmost and rightmost of a user-defined set of break-points $x_1, x_2, \ldots, x_{\mathrm{NBKPTS}}$. The co-ordinate system in space is defined by the value of m; $m = 0$ for Cartesian co-ordinates, $m = 1$ for cylindrical polar co-ordinates and $m = 2$ for spherical polar co-ordinates.

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

The initial values of the functions $U(x,t)$ must be given at $t = t_0$, and must be specified in a subroutine UINIT.

The functions R_i, for i = 1,2,...,NPDE, which may be thought of as fluxes, are also used in the definition of the boundary conditions for each equation. The boundary conditions must have the form

$$\beta_i(x,t)R_i(x,t,U,U_x) = \gamma_i(x,t,U,U_x),\ i = 1,2,...,\mathrm{NPDE}, \tag{3}$$

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

The boundary conditions must be specified in a subroutine BNDARY provided by the user. Thus, the problem is subject to the following restrictions:

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

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

(iii) The evaluation of the functions $P_{i,j}$, Q_i and R_i is done at both the break-points and internally selected points for each element in turn, that is $P_{i,j}$, Q_i and R_i are evaluated twice at each break-point. Any discontinuities in these functions **must** therefore be at one or more of the break-points $x_1, x_2, \ldots, x_{\mathrm{NBKPTS}}$;

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

(v) 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 8 below.

The parabolic equations are approximated by a system of ODEs in time for the values of U_i at the mesh points. This ODE system is obtained by approximating the PDE solution between each pair of break-points by a Chebyshev polynomial of degree NPOLY. The interval between each pair of break-points is treated by D03PDF as an element, and on this element, a polynomial and its space and time derivatives are made to satisfy the system of PDEs at NPOLY − 1 spatial points, which are chosen internally by the code and the break-points. In the case of just one element, the break-points are the boundaries. The user-defined break-points and the internally selected points together define the mesh. The smallest value that NPOLY can take is one, in which case, the solution is approximated by piecewise linear polynomials between consecutive break-points and the method is similar to an ordinary finite element method.

In total there are (NBKPTS − 1) × NPOLY + 1 mesh points in the spatial direction, and NPDE × ((NBKPTS − 1) × NPOLY + 1) ODEs in the time direction; one ODE at each break-point for each PDE component and (NPOLY − 1) ODEs for each PDE component between each pair of break-points. The system is then integrated forwards in time using a backward differentiation formula method.

4 References

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

[2] Berzins M and Dew P M (1991) Algorithm 690: Chebyshev polynomial software for elliptic-parabolic systems of PDEs *ACM Trans. Math. Software* **17** 178–206

[3] Zaturska N B, Drazin P G and Banks W H H (1988) On the flow of a viscous fluid driven along a channel by a suction at porous walls *Fluid Dynamics Research* **4**

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system to be solved.

Constraint: NPDE ≥ 1.

2: M — INTEGER *Input*

On entry: the co-ordinate system used:

M = 0
  indicates Cartesian co-ordinates,

M = 1
  indicates cylindrical polar co-ordinates,

M = 2
  indicates spherical polar co-ordinates.

Constraint: 0 ≤ M ≤ 2.

3: 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.

4: TOUT — ***real*** *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 compute the values of the functions $P_{i,j}$, Q_i and R_i which define the system of PDEs. The functions may depend on x, t, U and U_x and must be evaluated at a set of points.

Its specification is:

```
      SUBROUTINE PDEDEF(NPDE, T, X, NPTL, U, UX, P, Q, R, IRES)
      INTEGER            NPDE, NPTL, IRES
      real               T, X(NPTL), U(NPDE,NPTL), UX(NPDE,NPTL),
     1                   P(NPDE,NPDE,NPTL), Q(NPDE,NPTL), R(NPDE,NPTL)
```

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(NPTL) — ***real*** array *Input*

On entry: contains a set of mesh points at which $P_{i,j}$, Q_i and R_i are to be evaluated. X(1) and X(NPTL) contain successive user-supplied break-points and the elements of the array will satisfy X(1) < X(2) < ... < X(NPTL).

4: NPTL — INTEGER *Input*

On entry: the number of points at which evaluations are required (the value of NPOLY + 1).

5: U(NPDE,NPTL) — ***real*** array *Input*

On entry: U(i,j) contains the value of the component $U_i(x,t)$ where x = X(j), for i = 1,2,...,NPDE; j = 1,2,...,NPTL.

6: UX(NPDE,NPTL) — ***real*** array *Input*

On entry: UX(i,j) contains the value of the component $\frac{\partial U_i(x,t)}{\partial x}$ where x = X(j), for i = 1,2,...,NPDE; j = 1,2,...,NPTL.

7: P(NPDE,NPDE,NPTL) — ***real*** array *Output*

On exit: P(i,j,k) must be set to the value of $P_{i,j}(x,t,U,U_x)$ where x = X(k), for i,j = 1,2,...,NPDE; k = 1,2,...,NPTL.

8: Q(NPDE,NPTL) — ***real*** array *Output*

On exit: Q(i,j) must be set to the value of $Q_i(x,t,U,U_x)$ where x = X(j), for i = 1,2,...,NPDE; j = 1,2,...,NPTL.

9: R(NPDE,NPTL) — ***real*** array *Output*

On exit: R(i,j) must be set to the value of $R_i(x,t,U,U_x)$ where x = X(j), for i = 1,2,...,NPDE; j = 1,2,...,NPTL.

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 D03PDF 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 D03PDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

BNDARY must compute the functions β_i and γ_i which define the boundary conditions as in equation (3).

Its specification is:

```
SUBROUTINE BNDARY(NPDE, T, U, UX, IBND, BETA, GAMMA, IRES)
INTEGER          NPDE, IBND, IRES
real             T, U(NPDE), UX(NPDE), BETA(NPDE), GAMMA(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: U(NPDE) — ***real*** array *Input*

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

4: UX(NPDE) — ***real*** array *Input*

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

5: IBND — INTEGER *Input*

On entry: specifies which boundary conditions are to be evaluated. If IBND = 0, then BNDARY must set up the coefficients of the left-hand boundary $x = a$. Any other value of IBND indicates that BNDARY must set up the coefficients of the right-hand boundary, $x = b$.

6: BETA(NPDE) — ***real*** 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.

7: GAMMA(NPDE) — ***real*** array *Output*

On exit: GAMMA(i) must be set to the value of $\gamma_i(x,t,U,U_x)$ at the boundary specified by 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 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)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 D03PDF 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 D03PDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: U(NPDE,NPTS) — ***real*** array *Output*

On exit: $U(i, j)$ will contain the computed solution at t = TS.

8: NBKPTS — INTEGER *Input*

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

Constraint: NBKPTS $\geq$ 2.

9: XBKPTS(NBKPTS) — ***real*** array *Input*

On entry: the values of the break-points in the space direction. XBKPTS(1) must specify the left-hand boundary, a, and XBKPTS(NBKPTS) must specify the right-hand boundary, b.

Constraint: XBKPTS(1) < XBKPTS(2) < ... < XBKPTS(NBKPTS).

10: NPOLY — INTEGER *Input*

On entry: the degree of the Chebyshev polynomial to be used in approximating the PDE solution between each pair of break-points.

Constraint: 1 $\leq$ NPOLY $\leq$ 49.

11: NPTS — INTEGER *Input*

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

Constraint: NPTS = (NBKPTS − 1) × NPOLY + 1.

12: X(NPTS) — ***real*** array *Output*

On exit: the mesh points chosen by D03PDF in the spatial direction. The values of X will satisfy X(1) < X(2) < ... < X(NPTS).

13: UINIT — SUBROUTINE, supplied by the user. *External Procedure*

UINIT must compute the initial values of the PDE components $U_i(x_j, t_0)$, for i = 1,2,...,NPDE; j = 1,2,...,NPTS.

Its specification is:

```
SUBROUTINE UINIT(NPDE, NPTS, X, U)
INTEGER          NPDE, NPTS
real             X(NPTS), U(NPDE,NPTS)
```

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: X(NPTS) — ***real*** array *Input*

On entry: X(j), contains the values of the jth mesh point, for $j = 1, 2, \ldots,$ NPTS.

4: U(NPDE,NPTS) — ***real*** array *Output*

On exit: U(i,j) must be set to the initial value $U_i(x_j, t_0)$, for i = 1,2,...,NPDE; j = $1, 2, \ldots,$ NPTS.

UINIT must be declared as EXTERNAL in the (sub)program from which D03PDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

14: ACC — ***real*** *Input*

On entry: a positive quantity for controlling the local error in the time integration. If E(i,j) is the estimated error for U_i at the jth mesh point, the error test is:

$$|\mathrm{E}(i,j)| = \mathrm{ACC} \times (1.0 + |\mathrm{U}(i,j)|).$$

Constraint: ACC > 0.0.

15: W(NW) — ***real*** array *Workspace*

16: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PDF is called.

Constraints:

$$\mathrm{NW} \geq 11 \times \mathrm{NPDE} \times \mathrm{NPTS} + 50 + \mathrm{NWKRES} + \mathrm{LENODE}, \text{ where}$$
$$\mathrm{NWKRES} = 3 \times (\mathrm{NPOLY}+1)^2 + (\mathrm{NPOLY}+1) \times [\mathrm{NPDE}^2 + 6 \times \mathrm{NPDE} + \mathrm{NBKPTS} + 1] + 13 \times \mathrm{NPDE} + 5, \text{ and}$$
$$\mathrm{LENODE} = \mathrm{NPDE} \times \mathrm{NPTS} \times [3 \times \mathrm{NPDE} \times (\mathrm{NPOLY}+1) - 2].$$

17: 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 last backward differentiation formula method used.

IW(5) contains the number of Newton iterations performed by the time integrator. Each iteration involves an ODE residual evaluation followed by a back-substitution using the *LU* decomposition of the Jacobian matrix.

18: NIW — INTEGER *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D03PDF is called.

Constraint: NIW = NPDE × NPTS + 24.

19: ITASK — INTEGER *Input*

On entry: specifies 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.

ITASK = 2
: one step only and return.

ITASK = 3
: stop at first internal integration point at or beyond t = TOUT.

Constraint: $1 \leq$ ITASK ≤ 3.

20: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03PDF 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 subchapter D02M-N of the NAG Fortran Library.

21: 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 D03PDF.

Constraint: $0 \leq$ IND ≤ 1.

On exit: IND = 1.

22: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TOUT ≤ TS,
or (TOUT − TS) is too small,
or ITASK ≠ 1, 2 or 3,
or M ≠ 0, 1 or 2,
or M > 0 and XBKPTS(1) < 0.0,
or NPDE < 1,
or NBKPTS < 2,
or NPOLY < 1 or NPOLY > 49,

or NPTS $\neq$ (NBKPTS $-$ 1) $\times$ NPOLY + 1,
or ACC $\leq$ 0.0,
or IND $\neq$ 0 or 1,
or break-points XBKPTS(i) are not ordered,
or NW or NIW are too small,
or IND = 1 on initial entry to D03PDF.

IFAIL = 2

The underlying ODE solver cannot make any further progress across the integration range from the current point t = TS with the supplied value of ACC. The components of U contain the computed values at the current point t = TS.

IFAIL = 3

In the underlying ODE solver, there were repeated errors or corrector convergence test failures on an attempted step, before completing the requested task. The problem may have a singularity or ACC is too small for the integration to continue. Integration was successful as far as t = TS.

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 the user-supplied subroutines PDEDEF or BNDARY, when the residual in the underlying ODE solver was being evaluated.

IFAIL = 5

In solving the ODE system, a singular Jacobian has been encountered. The user should check his 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 or BNDARY. Integration was successful as far as t = TS.

IFAIL = 7

The value of ACC is so small that the routine is unable to start the integration in time.

IFAIL = 8

In one of the user-supplied routines, PDEDEF 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 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 error message unit).

IFAIL = 12

Not applicable.

IFAIL = 13

Not applicable.

IFAIL = 14

The flux function R_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 the degree of the polynomial approximation NPOLY, and on both the number of break-points and on their distribution in space. In the time integration only the local error over a single step is controlled and so the accuracy over a number of steps cannot be guaranteed. The user should therefore test the effect of varying the accuracy parameter, ACC.

8 Further Comments

The routine is designed to solve parabolic systems (possibly including elliptic equations) with second-order derivatives in space. The parameter specification allows the user 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.

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

9 Example

The problem consists of a fourth order PDE which can be written as a pair of second order elliptic-parabolic PDEs for $U_1(x,t)$ and $U_2(x,t)$,

$$0 = \frac{\partial^2 U_1}{\partial x^2} - U_2 \tag{4}$$

$$\frac{\partial U_2}{\partial t} = \frac{\partial^2 U_2}{\partial x^2} + U_2 \frac{\partial U_1}{\partial x} - U_1 \frac{\partial U_2}{\partial x} \tag{5}$$

where $-1 \leq x \leq 1$ and $t \geq 0$. The boundary conditions are given by

$$\frac{\partial U_1}{\partial x} = 0 \text{ and } U_1 = 1 \text{ at } x = -1, \text{ and}$$

$$\frac{\partial U_1}{\partial x} = 0 \text{ and } U_1 = -1 \text{ at } x = 1.$$

The initial conditions at $t = 0$ are given by

$$U_1 = -\sin\frac{\pi x}{2} \text{ and } U_2 = \frac{\pi^2}{4}\sin\frac{\pi x}{2}.$$

The absence of boundary conditions for $U_2(x,t)$ does not pose any difficulties provided that the derivative flux boundary conditions are assigned to the first PDE (4) which has the correct flux, $\frac{\partial U_1}{\partial x}$. The conditions on $U_1(x,t)$ at the boundaries are assigned to the second PDE by setting $\beta_2 = 0.0$ in equation (3) and placing the Dirichlet boundary conditions on $U_1(x,t)$ in the function γ_2.

9.1 Example 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.

```
*     D03PDF Example Program Text
*     Mark 15 Release. NAG Copyright 1991.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NBKPTS, NEL, NPDE, NPOLY, M, NPTS, NEQN, NIW,
     +                 NPL1, NWKRES, MU, LENODE, NW, ITYPE, INTPTS
      PARAMETER        (NBKPTS=10,NEL=NBKPTS-1,NPDE=2,NPOLY=3,M=0,
     +                 NPTS=NEL*NPOLY+1,NEQN=NPDE*NPTS,NIW=NEQN+24,
     +                 NPL1=NPOLY+1,NWKRES=3*NPL1*NPL1+NPL1*
     +                 (NPDE*NPDE+6*NPDE+NBKPTS+1)+13*NPDE+5,
     +                 MU=NPDE*(NPOLY+1)-1,LENODE=(3*MU+1)*NEQN,
     +                 NW=11*NEQN+50+NWKRES+LENODE,ITYPE=1,INTPTS=6)
*     .. Scalars in Common ..
      real             PIBY2
*     .. Local Scalars ..
      real             ACC, PI, TOUT, TS
      INTEGER          I, IFAIL, IND, IT, ITASK, ITRACE
*     .. Local Arrays ..
      real             U(NPDE,NPTS), UOUT(NPDE,INTPTS,ITYPE), W(NW),
     +                 X(NPTS), XBKPTS(NBKPTS), XOUT(6)
      INTEGER          IW(NIW)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PDF, D03PYF, PDEDEF, UINIT
*     .. Common blocks ..
      COMMON           /PIVAL/PIBY2
*     .. Data statements ..
      DATA             XOUT(1)/-1.0e+0/, XOUT(2)/-0.6e+0/,
     +                 XOUT(3)/-0.2e+0/, XOUT(4)/0.2e+0/,
     +                 XOUT(5)/0.6e+0/, XOUT(6)/1.0e+0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PDF Example Program Results'
      PIBY2 = 0.5e0*X01AAF(PI)
      ACC = 1.0e-4
      ITRACE = 0
*
*     Set the break-points
*
      DO 20 I = 1, NBKPTS
         XBKPTS(I) = -1.0e0 + (I-1.0e0)*2.0e0/(NBKPTS-1.0e0)
   20 CONTINUE
*
      IND = 0
      ITASK = 1
      TS = 0.0e0
      TOUT = 0.1e-4
      WRITE (NOUT,99999) NPOLY, NEL
      WRITE (NOUT,99998) ACC, NPTS
      WRITE (NOUT,99997) (XOUT(I),I=1,6)
*
*     Loop over output values of t
```

```
*
      DO 40 IT = 1, 5
         IFAIL = -1
         TOUT = 10.0e0*TOUT
*
         CALL D03PDF(NPDE,M,TS,TOUT,PDEDEF,BNDARY,U,NBKPTS,XBKPTS,NPOLY,
     +               NPTS,X,UINIT,ACC,W,NW,IW,NIW,ITASK,ITRACE,IND,
     +               IFAIL)
*
*        Interpolate at required spatial points
*
         CALL D03PYF(NPDE,U,NBKPTS,XBKPTS,NPOLY,NPTS,XOUT,INTPTS,ITYPE,
     +               UOUT,W,NW,IFAIL)
         WRITE (NOUT,99996) TS, (UOUT(1,I,1),I=1,INTPTS)
         WRITE (NOUT,99995) (UOUT(2,I,1),I=1,INTPTS)
   40 CONTINUE
*
*     Print integration statistics
*
      WRITE (NOUT,99994) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (' Polynomial degree =',I4,'   No. of elements = ',I4)
99998 FORMAT (' Accuracy requirement = ',e9.3,'  Number of points = ',
     +       I5,/)
99997 FORMAT ('  T /   X    ',6F8.4,/)
99996 FORMAT (1X,F6.4,' U(1)',6F8.4)
99995 FORMAT (8X,'U(2)',6F8.4,/)
99994 FORMAT (' Number of integration steps in time                   ',
     +       I4,/' Number of residual evaluations of resulting ODE sys',
     +       'tem',I4,/' Number of Jacobian evaluations                ',
     +       '          ',I4,/' Number of iterations of nonlinear solve',
     +       'r              ',I4,/)
      END
*
      SUBROUTINE UINIT(NPDE,NPTS,X,U)
*     .. Scalar Arguments ..
      INTEGER          NPDE, NPTS
*     .. Array Arguments ..
      real             U(NPDE,NPTS), X(NPTS)
*     .. Scalars in Common ..
      real             PIBY2
*     .. Local Scalars ..
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        SIN
*     .. Common blocks ..
      COMMON           /PIVAL/PIBY2
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         U(1,I) = -SIN(PIBY2*X(I))
         U(2,I) = -PIBY2*PIBY2*U(1,I)
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,NPTL,U,DUDX,P,Q,R,IRES)
*     .. Scalar Arguments ..
```

```
      real              T
      INTEGER           IRES, NPDE, NPTL
*     .. Array Arguments ..
      real              DUDX(NPDE,NPTL), P(NPDE,NPDE,NPTL),
     +                  Q(NPDE,NPTL), R(NPDE,NPTL), U(NPDE,NPTL),
     +                  X(NPTL)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      DO 20 I = 1, NPTL
         Q(1,I) = U(2,I)
         Q(2,I) = U(1,I)*DUDX(2,I) - DUDX(1,I)*U(2,I)
         R(1,I) = DUDX(1,I)
         R(2,I) = DUDX(2,I)
         P(1,1,I) = 0.0e0
         P(1,2,I) = 0.0e0
         P(2,1,I) = 0.0e0
         P(2,2,I) = 1.0e0
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,U,UX,IBND,BETA,GAMMA,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NPDE
*     .. Array Arguments ..
      real              BETA(NPDE), GAMMA(NPDE), U(NPDE), UX(NPDE)
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         BETA(1) = 1.0e0
         GAMMA(1) = 0.0e0
         BETA(2) = 0.0e0
         GAMMA(2) = U(1) - 1.0e0
      ELSE
         BETA(1) = 1.0e+0
         GAMMA(1) = 0.0e0
         BETA(2) = 0.0e0
         GAMMA(2) = U(1) + 1.0e0
      END IF
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
 D03PDF Example Program Results
 Polynomial degree =   3   No. of elements =    9
 Accuracy requirement = 0.100E-03  Number of points =    28

  T /   X    -1.0000 -0.6000 -0.2000  0.2000  0.6000  1.0000

 0.0001 U(1)  1.0000  0.8090  0.3090 -0.3090 -0.8090 -1.0000
        U(2) -2.4850 -1.9957 -0.7623  0.7623  1.9957  2.4850
```

```
0.0010 U(1)  1.0000  0.8085  0.3088 -0.3088 -0.8085 -1.0000
       U(2) -2.5583 -1.9913 -0.7606  0.7606  1.9913  2.5583

0.0100 U(1)  1.0000  0.8051  0.3068 -0.3068 -0.8051 -1.0000
       U(2) -2.6962 -1.9481 -0.7439  0.7439  1.9481  2.6962

0.1000 U(1)  1.0000  0.7951  0.2985 -0.2985 -0.7951 -1.0000
       U(2) -2.9022 -1.8339 -0.6338  0.6338  1.8339  2.9022

1.0000 U(1)  1.0000  0.7939  0.2972 -0.2972 -0.7939 -1.0000
       U(2) -2.9233 -1.8247 -0.6120  0.6120  1.8247  2.9233

Number of integration steps in time                   50
Number of residual evaluations of resulting ODE system 407
Number of Jacobian evaluations                         18
Number of iterations of nonlinear solver              122
```

D03PEF – NAG Fortran Library Routine Document

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

D03PEF integrates a system of linear or nonlinear, first-order, time-dependent partial differential equations (PDEs) in one space variable. The spatial discretisation is performed using the Keller box scheme and the method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs). The resulting system is solved using a Backward Differentiation Formula (BDF) method.

2 Specification

```
 SUBROUTINE D03PEF(NPDE, TS, TOUT, PDEDEF, BNDARY, U, NPTS, X,
1                  NLEFT, ACC, W, NW, IW, NIW, ITASK, ITRACE, IND,
2                  IFAIL)
 INTEGER           NPDE, NPTS, NLEFT, NW, IW(NIW), NIW, ITASK,
1                  ITRACE, IND, IFAIL
 real              TS, TOUT, U(NPDE,NPTS), X(NPTS), ACC, W(NW)
 EXTERNAL          PDEDEF, BNDARY
```

3 Description

D03PEF integrates the system of first-order PDEs

$$G_i(x,t,U,U_x,U_t) = 0,\ i = 1,2,\ldots,\mathrm{NPDE}. \tag{1}$$

In particular the functions G_i must have the general form:

$$G_i = \sum_{j=1}^{\mathrm{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + Q_i,\quad i = 1,2,\ldots,\mathrm{NPDE},\quad a \le x \le b,\ t \ge t_0, \tag{2}$$

where $P_{i,j}$ and Q_i depend on x, t, U, U_x and the vector U is the set of solution values

$$U(x,t) = [U_1(x,t),\ldots,U_{\mathrm{NPDE}}(x,t)]^T, \tag{3}$$

and the vector U_x is its partial derivative with respect to x. Note that $P_{i,j}$ and Q_i must not depend on $\frac{\partial U}{\partial t}$.

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 $b = x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$. The mesh should be chosen in accordance with the expected behaviour of the solution.

The PDE system which is defined by the functions G_i must be specified in a subroutine PDEDEF supplied by the user.

The initial values of the functions $U(x,t)$ must be given at $t = t_0$. For a first-order system of PDEs, only one boundary condition is required for each PDE component U_i. The NPDE boundary conditions are separated into NLEFT at the left-hand boundary $x = a$, and NRIGHT at the right-hand boundary $x = b$, such that NLEFT + NRIGHT = NPDE. The position of the boundary condition for each component should be chosen with care; the general rule is that if the characteristic direction of U_i at the left-hand boundary (say) points into the interior of the solution domain, then the boundary condition for U_i should be specified at the left-hand boundary. Incorrect positioning of boundary conditions generally results in initialisation or integration difficulties in the underlying time integration routines.

The boundary conditions have the form:

$$G_i^L(x,t,U,U_t) = 0,\ \text{at } x = a,\quad i = 1,2,\ldots,\mathrm{NLEFT} \tag{4}$$

at the left-hand boundary, and

$$G_i^R(x,t,U,U_t) = 0, \text{ at } x = b, \quad i = 1,2,\ldots,\text{NRIGHT} \tag{5}$$

at the right-hand boundary.

Note that the functions G_i^L and G_i^R must not depend on U_x, since spatial derivatives are not determined explicitly in the Keller box scheme [3]. If the problem involves derivative (Neumann) boundary conditions then it is generally possible to restate such boundary conditions in terms of permissible variables. Also note that G_i^L and G_i^R must be linear with respect to time derivatives, so that the boundary conditions have the general form:

$$\sum_{j=1}^{\text{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + S_i^L = 0, \; i = 1,2,\ldots,\text{NLEFT} \tag{6}$$

at the left-hand boundary, and

$$\sum_{j=1}^{\text{NPDE}} E_{i,j}^R \frac{\partial U_j}{\partial t} + S_i^R = 0, \; i = 1,2,\ldots,\text{NRIGHT} \tag{7}$$

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

The boundary conditions must be specified in a subroutine BNDARY provided by the user.

The problem is subject to the following restrictions:

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

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

(iii) The evaluation of the function G_i is done at the mid-points of the mesh intervals by calling the routine PDEDEF for each mid-point in turn. Any discontinuities in the function **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\text{NPTS}}$;

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

In this method of lines approach the Keller box scheme [3] is applied to each PDE in the space variable only, resulting in a system of ODEs in time for the values of U_i at each mesh point. In total there are NPDE × NPTS ODEs in the time direction. This system is then integrated forwards in time using a BDF method.

4 References

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

[2] 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

[3] Keller H B (1970) A new difference scheme for parabolic problems *Numerical Solutions of Partial Differential Equations* (ed J Bramble) **2** Academic Press 327–350

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

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system to be solved.

Constraint: NPDE ≥ 1.

2: TS — ***real*** *Input/Output*

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

Constraint: TS < TOUT.

On exit: the value of t corresponding to the solution values in U. Normally 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 compute the functions G_i which define the system of PDEs. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PEF.

Its specification is:

```
SUBROUTINE PDEDEF(NPDE, T, X, U, UT, UX, RES, IRES)
INTEGER           NPDE, IRES
real              T, X, U(NPDE), UT(NPDE), UX(NPDE), RES(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: UT(NPDE) — ***real*** array *Input*

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

6: UX(NPDE) — ***real*** array *Input*

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

7: RES(NPDE) — ***real*** array *Output*

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

$$G_i = \sum_{j=1}^{\text{NPDE}} P_{i,j} \frac{\partial U_j}{\partial t}, \tag{8}$$

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

$$G_i = \sum_{j=1}^{\text{NPDE}} P_{i,j} \frac{\partial U_j}{\partial t} + Q_i, \tag{9}$$

i.e., all terms in equation (2).

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

8: IRES — INTEGER *Input/Output*

On entry: the form of G_i that must be returned in the array RES. If IRES = −1, then equation (8) above must be used. If IRES = 1, then equation (9) above must be used.

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 (IFAIL) set to 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 D03PEF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

PDEDEF must be declared as EXTERNAL in the (sub)program from which D03PEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: BNDARY — SUBROUTINE, supplied by the user. *External Procedure*

BNDARY must compute the functions G_i^L and G_i^R which define the boundary conditions as in equations (4) and (5).

Its specification is:

```
SUBROUTINE BNDARY(NPDE, T, IBND, NOBC, U, UT, RES, IRES)
INTEGER          NPDE, IBND, NOBC, IRES
real             T, U(NPDE), UT(NPDE), RES(NOBC)
```

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: IBND — INTEGER *Input*

On entry: IBND determines the position of the boundary conditions. If IBND = 0, then BNDARY must compute the left-hand boundary condition at $x = a$. Any other value of IBND indicates that BNDARY must compute the right-hand boundary condition at $x = b$.

4: NOBC — INTEGER *Input*

On entry: NOBC specifies the number of boundary conditions at the boundary specified by IBND.

5: U(NPDE) — ***real*** array *Input*

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

6: UT(NPDE) — ***real*** array *Input*

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

7: RES(NOBC) — ***real*** array *Output*

On exit: RES(i) must contain the ith component of G^L or G^R, depending on the value of IBND, for $i = 1,2,\ldots,$NOBC, where G^L is defined as

$$G_i^L = \sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t}, \tag{10}$$

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

$$G_i^L = \sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + S_i^L, \tag{11}$$

i.e., all terms in equation (6), and similarly for G_i^R.
The definitions of G^L and G^R are determined by the input value of IRES.

8: IRES — INTEGER *Input/Output*

On entry: the form G_i^L (or G_i^R) that must be returned in the array RES. If IRES = −1, then equation (10) above must be used. If IRES = 1, then equation (11) above must be used.

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 (IFAIL) set to 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 D03PEF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

BNDARY must be declared as EXTERNAL in the (sub)program from which D03PEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: U(NPDE,NPTS) — ***real*** array *Input/Output*

On entry: the initial values of $U(x,t)$ at t = TS and the mesh points X(j), for $j = 1,2,\ldots,$NPTS.

On exit: U(i,j) will contain the computed solution at t = TS.

7: NPTS — INTEGER *Input*

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

Constraint: NPTS $\geq$ 3.

8: X(NPTS) — ***real*** array *Input*

On entry: the mesh points in the spatial 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).

9: NLEFT — INTEGER *Input*

On entry: the number of boundary conditions at the left-hand mesh point X(1).

Constraint: 0 $\leq$ NLEFT $\leq$ NPDE.

10: ACC — ***real*** *Input*

On entry: a positive quantity for controlling the local error estimate in the time integration. If E(i,j) is the estimated error for U_i at the jth mesh point, the error test is:

$$|\mathrm{E}(i,j)| = \mathrm{ACC} \times (1.0 + |\mathrm{U}(i,j)|).$$

Constraint: ACC > 0.0.

11: W(NW) — ***real*** array *Workspace*

12: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PEF is called.

Constraint: NW $\geq$ (4 × NPDE + NLEFT + 14) × NPDE × NPTS + (3 × NPDE + 21) × NPDE + 7 × NPTS + 54.

IFAIL = 5

In solving the ODE system, a singular Jacobian has been encountered. The user should check their problem formulation.

IFAIL = 6

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

IFAIL = 7

The value of ACC is so small that the routine is unable to start the integration in time.

IFAIL = 8

In one of the user-supplied routines, PDEDEF 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 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).

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 accuracy parameter, ACC.

8 Further Comments

The Keller box scheme can be used to solve higher-order problems which have been reduced to first order by the introduction of new variables (see the example problem in D03PKF). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite-difference scheme (D03PCF/D03PHF for example), but at the expense of increased CPU time due to the larger number of function evaluations required.

It should be noted that the Keller box scheme, in common with other central-difference schemes, may be unsuitable for some hyperbolic first-order problems such as the apparently simple linear advection equation $U_t + aU_x = 0$, where a is a constant, resulting in spurious oscillations due to the lack of dissipation. This type of problem requires a discretisation scheme with upwind weighting (D03PFF for example), or the addition of a second-order artificial dissipation term.

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

9 Example

This example is the simple first-order system

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

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

for $t \in [0,1]$ and $x \in [0,1]$.

The initial conditions are

$$U_1(x,0) = \exp(x), \quad U_2(x,0) = \sin(x),$$

and the Dirichlet boundary conditions for U_1 at $x = 0$ and U_2 at $x = 1$ are given by the exact solution:

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

$$U_2(x,t) = \exp(x-3t) - \exp(x+t) + \frac{1}{2}\left\{\sin(x+t) + \sin(x-3t)\right\}.$$

9.1 Example 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.

```
*     D03PEF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, NLEFT, NEQN, NIW, NWKRES, NW
      PARAMETER        (NPDE=2,NPTS=41,NLEFT=1,NEQN=NPDE*NPTS,
     +                 NIW=NEQN+24,NWKRES=NPDE*(NPTS+21+3*NPDE)
     +                 +7*NPTS+4,NW=11*NEQN+(4*NPDE+NLEFT+2)
     +                 *NEQN+50+NWKRES)
*     .. Local Scalars ..
      real             ACC, TOUT, TS
      INTEGER          I, IFAIL, IND, IT, ITASK, ITRACE
*     .. Local Arrays ..
      real             EU(NPDE,NPTS), U(NPDE,NPTS), W(NW), X(NPTS)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PEF, EXACT, PDEDEF, UINIT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PEF Example Program Results'
      ITRACE = 0
      ACC = 0.1e-5
      WRITE (NOUT,99997) ACC, NPTS
*
*     Set spatial-mesh points
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
      WRITE (NOUT,99999) X(5), X(13), X(21), X(29), X(37)
*
      IND = 0
      ITASK = 1
*
      CALL UINIT(NPDE,NPTS,X,U)
*
*     Loop over output value of t
      TS = 0.0e0
      TOUT = 0.0e0
```

```
      DO 40 IT = 1, 5
         TOUT = 0.2e0*IT
         IFAIL = -1
*
         CALL D03PEF(NPDE,TS,TOUT,PDEDEF,BNDARY,U,NPTS,X,NLEFT,ACC,W,NW,
     +               IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
*        Check against the exact solution
*
         CALL EXACT(TOUT,NPDE,NPTS,X,EU)
*
         WRITE (NOUT,99998) TS
         WRITE (NOUT,99995) U(1,5), U(1,13), U(1,21), U(1,29), U(1,37)
         WRITE (NOUT,99994) EU(1,5), EU(1,13), EU(1,21), EU(1,29),
     +     EU(1,37)
         WRITE (NOUT,99993) U(2,5), U(2,13), U(2,21), U(2,29), U(2,37)
         WRITE (NOUT,99992) EU(2,5), EU(2,13), EU(2,21), EU(2,29),
     +     EU(2,37)
   40 CONTINUE
      WRITE (NOUT,99996) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (' X        ',5F10.4,/)
99998 FORMAT (' T = ',F5.2)
99997 FORMAT (//'  Accuracy requirement =',e10.3,' Number of points = ',
     +       I3,/)
99996 FORMAT (' Number of integration steps in time = ',I6,/' Number o',
     +       'f function evaluations = ',I6,/' Number of Jacobian eval',
     +       'uations =',I6,/' Number of iterations = ',I6,/)
99995 FORMAT (' Approx U1',5F10.4)
99994 FORMAT (' Exact  U1',5F10.4)
99993 FORMAT (' Approx U2',5F10.4)
99992 FORMAT (' Exact  U2',5F10.4,/)
      END
*
      SUBROUTINE UINIT(NPDE,NPTS,X,U)
*     Routine for PDE initial values
*     .. Scalar Arguments ..
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              U(NPDE,NPTS), X(NPTS)
*     .. Local Scalars ..
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         EXP, SIN
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         U(1,I) = EXP(X(I))
         U(2,I) = SIN(X(I))
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,UDOT,DUDX,RES,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NPDE
```

```
*     .. Array Arguments ..
      real               DUDX(NPDE), RES(NPDE), U(NPDE), UDOT(NPDE)
*     .. Executable Statements ..
      IF (IRES.EQ.-1) THEN
         RES(1) = UDOT(1)
         RES(2) = UDOT(2)
      ELSE
         RES(1) = UDOT(1) + DUDX(1) + DUDX(2)
         RES(2) = UDOT(2) + 4.0e0*DUDX(1) + DUDX(2)
      END IF
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,IBND,NOBC,U,UDOT,RES,IRES)
*     .. Scalar Arguments ..
      real               T
      INTEGER            IBND, IRES, NOBC, NPDE
*     .. Array Arguments ..
      real               RES(NOBC), U(NPDE), UDOT(NPDE)
*     .. Intrinsic Functions ..
      INTRINSIC          EXP, SIN
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         IF (IRES.EQ.-1) THEN
            RES(1) = 0.0e0
         ELSE
            RES(1) = U(1) - 0.5e0*(EXP(T)+EXP(-3.0e0*T)) -
     +               0.25e0*(SIN(-3.0e0*T)-SIN(T))
         END IF
      ELSE
         IF (IRES.EQ.-1) THEN
            RES(1) = 0.0e0
         ELSE
            RES(1) = U(2) - EXP(1.0e0-3.0e0*T) + EXP(1.0e0+T) -
     +               0.5e0*(SIN(1.0e0-3.0e0*T)+SIN(1.0e0+T))
         END IF
      END IF
      RETURN
      END
*
      SUBROUTINE EXACT(T,NPDE,NPTS,X,U)
*     Exact solution (for comparison purposes)
*     .. Scalar Arguments ..
      real               T
      INTEGER            NPDE, NPTS
*     .. Array Arguments ..
      real               U(NPDE,NPTS), X(NPTS)
*     .. Local Scalars ..
      INTEGER            I
*     .. Intrinsic Functions ..
      INTRINSIC          EXP, SIN
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         U(1,I) = 0.5e0*(EXP(X(I)+T)+EXP(X(I)-3.0e0*T)) +
     +            0.25e0*(SIN(X(I)-3.0e0*T)-SIN(X(I)+T))
         U(2,I) = EXP(X(I)-3.0e0*T) - EXP(X(I)+T) + 0.5e0*(SIN(X(I)
     +            -3.0e0*T)+SIN(X(I)+T))
   20 CONTINUE
```

```
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
D03PEF Example Program Results


 Accuracy requirement = 0.100E-05 Number of points =  41

X            0.1000    0.3000    0.5000    0.7000    0.9000

T =  0.20
Approx U1    0.7845    1.0010    1.2733    1.6115    2.0281
Exact  U1    0.7845    1.0010    1.2733    1.6115    2.0281
Approx U2   -0.8352   -0.8159   -0.8367   -0.9128   -1.0609
Exact  U2   -0.8353   -0.8160   -0.8367   -0.9129   -1.0609

T =  0.40
Approx U1    0.6481    0.8533    1.1212    1.4627    1.8903
Exact  U1    0.6481    0.8533    1.1212    1.4627    1.8903
Approx U2   -1.5216   -1.6767   -1.8934   -2.1917   -2.5944
Exact  U2   -1.5217   -1.6767   -1.8935   -2.1917   -2.5945

T =  0.60
Approx U1    0.6892    0.8961    1.1747    1.5374    1.9989
Exact  U1    0.6892    0.8962    1.1747    1.5374    1.9989
Approx U2   -2.0047   -2.3434   -2.7677   -3.3002   -3.9680
Exact  U2   -2.0048   -2.3436   -2.7678   -3.3003   -3.9680

T =  0.80
Approx U1    0.8977    1.1247    1.4320    1.8349    2.3514
Exact  U1    0.8977    1.1247    1.4320    1.8349    2.3512
Approx U2   -2.3403   -2.8675   -3.5110   -4.2960   -5.2536
Exact  U2   -2.3405   -2.8677   -3.5111   -4.2961   -5.2537

T =  1.00
Approx U1    1.2470    1.5206    1.8828    2.3528    2.9519
Exact  U1    1.2470    1.5205    1.8829    2.3528    2.9518
Approx U2   -2.6229   -3.3338   -4.1998   -5.2505   -6.5218
Exact  U2   -2.6232   -3.3340   -4.2001   -5.2507   -6.5219

Number of integration steps in time =    149
Number of function evaluations =    399
Number of Jacobian evaluations =    13
Number of iterations =    323
```

D03PFF – NAG Fortran Library Routine Document

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

3 Description

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

$$\sum_{j=1}^{\mathrm{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t}+\frac{\partial F_i}{\partial x}=C_i\frac{\partial D_i}{\partial x}+S_i, \qquad (1)$$

or the hyperbolic convection-only system:

$$\frac{\partial U_i}{\partial t}+\frac{\partial F_i}{\partial x}=0, \qquad (2)$$

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

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

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 $b=x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a user-defined mesh $x_1,x_2,\ldots,x_{\mathrm{NPTS}}$. 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, D03PHF and D03PPF, 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

$x_{j-\frac{1}{2}} = (x_{j-1} + x_j)/2$ for $j = 2, 3, \ldots,$ 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 [2]). 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 $y = x - x_{j-\frac{1}{2}}$, i.e., $y = 0$ corresponds to $x = x_{j-\frac{1}{2}}$, 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 [2] and [5]. Roe's scheme [4] 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

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

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 [4]).

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

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

at the left-hand boundary, and

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

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) $t_0 < t_{out}$, 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 × 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 [1] and the references therein.

4 References

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

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

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

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

[5] 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

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 $P_{i,j}(x,t,U)$, 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 $i = 1,2,\ldots,$NPDE.

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

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

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

On exit: FLUX(i) must be set to the numerical flux $\hat{F}_i$, for $i = 1,2,\ldots,$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 $x = \text{X}(j)$, for $i = 1,2,\ldots,\text{NPDE}$; $j = 1,2,3$. If IBND $\neq$ 0, then U(i,j) contains the value of the component $U_i(x,t)$ at $x = \text{X}(\text{NPTS} - j + 1)$, for $i = 1,2,\ldots,\text{NPDE}$; $j = 1,2,3$.

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 $i = 1,2,\ldots,\text{NPDE}$.

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 $x = \text{X}(j)$ and $t = \text{TS}$; for $i = 1,2,\ldots,\text{NPDE}$; $j = 1,2,\ldots,\text{NPTS}$.

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

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: X(1) < X(2) < ... < X(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 $\mathrm{E}(i,j)$ 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 $\geq$ 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: 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 $\leq$ 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 subchapter 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: $0 \le \text{IND} \le 1$.

On exit: IND = 1.

19: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TS ≥ TOUT,
or TOUT − TS is too small,
or ITASK ≠ 1, 2, or 3,
or NPTS < 3,
or NPDE < 1,
or IND ≠ 0 or 1,
or incorrectly defined user mesh, i.e., $X(i) \ge X(i+1)$ for some $i = 1, 2, \ldots, \text{NPTS}-1$,
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 $\geq$ 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 [1].

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 and Section 9.2. 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 Section 9.1.1 and Section 9.2.1 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 $[\,1 \;\; 2\,]^T$ and $[\,-1 \;\; 2\,]^T$ 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

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

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

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

that is

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

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
*
```

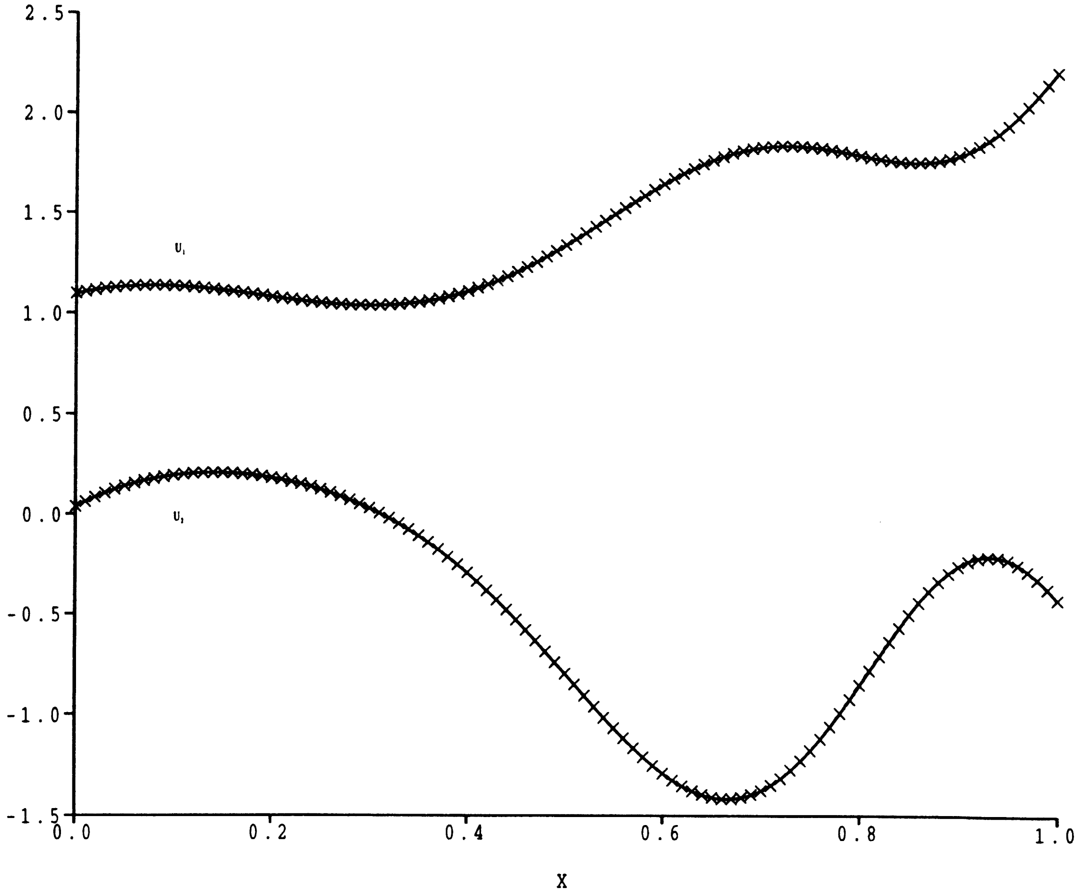
9.1.2 Program Data

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 $U(-1,t) = 3.0$ and $U(1,t) = 5.0$ and the initial condition is $U(x,0) = x+4$. The integration is run to steady state at which the solution is known to be $U = 4$ 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

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

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
```

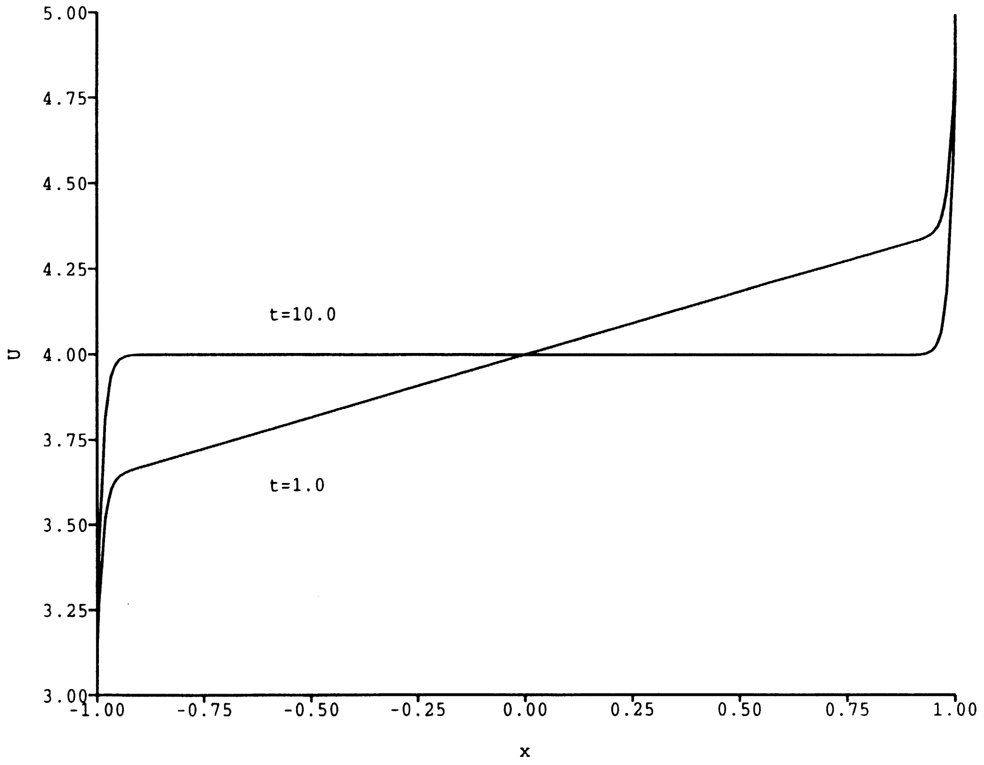
```
      END
*
      SUBROUTINE BNDRY2(NPDE,NPTS,T,X,U,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,3), X(NPTS)
*     .. Executable Statements ..
      IF (IBND.EQ.0) 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

```
 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
```


5.00
4.75
4.50
4.25
4.00
3.75
3.50
3.25
3.00
u
t=10.0
t=1.0
-1.00
-0.75
-0.50
-0.25
0.00
0.25
0.50
0.75
1.00
x

D03PHF – NAG Fortran Library Routine Document

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

D03PHF integrates a system of linear or nonlinear parabolic partial differential equations (PDEs) in one space variable, with scope for coupled ordinary differential equations (ODEs). The spatial discretisation 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 method or a Theta method (switching between Newton's method and functional iteration).

2 Specification

```
      SUBROUTINE D03PHF(NPDE, M, TS, TOUT, PDEDEF, BNDARY, U, NPTS, X,
     1                  NCODE, ODEDEF, NXI, XI, NEQN, RTOL, ATOL, ITOL,
     2                  NORM, LAOPT, ALGOPT, W, NW, IW, NIW, ITASK,
     3                  ITRACE, IND, IFAIL)
      INTEGER           NPDE, M, NPTS, NCODE, NXI, NEQN, ITOL, NW,
     1                  IW(NIW), NIW, ITASK, ITRACE, IND, IFAIL
      real              TS, TOUT, U(NEQN), X(NPTS), XI(*), RTOL(*),
     1                  ATOL(*), ALGOPT(30), W(NW)
      CHARACTER*1       NORM, LAOPT
      EXTERNAL          PDEDEF, BNDARY, ODEDEF
```

3 Description

D03PHF integrates the system of parabolic-elliptic equations and coupled ODEs

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

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

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

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

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

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

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

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, where $a = x_1$ and $b = x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$. The co-ordinate system in space is defined by the values of m; $m = 0$ for Cartesian co-ordinates, $m = 1$ for cylindrical polar co-ordinates and $m = 2$ for spherical polar co-ordinates.

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

The initial values of the functions $U(x,t)$ and $V(t)$ must be given at $t = t_0$.

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

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

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

The boundary conditions must be specified in a subroutine BNDARY provided by the user. 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,...,NCODE, may only appear **linearly** in the functions Q_i, for i = 1,2,...,NPDE, with a similar restriction for γ;

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

(iii) $t_0 < t_{out}$, 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,...,NPTS, by calling the routine PDEDEF for each mid-point in turn. Any discontinuities in these functions **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$;

(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;

(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 8 below.

The algebraic-differential equation system which is defined by the functions F_i must be specified in a subroutine ODEDEF supplied by the user. The user 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 co-ordinates, 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 × NPTS + NCODE ODEs in time direction. This system is then integrated forwards in time using a backward differentiation formula (BDF) or a Theta method.

4 References

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

[2] 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

[3] 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

[4] 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

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE ≥ 1.

2: M — INTEGER *Input*

On entry: the co-ordinate system used:

M = 0
  indicates Cartesian co-ordinates,

M = 1
  indicates cylindrical polar co-ordinates,

M = 2
  indicates spherical polar co-ordinates.

Constraint: $0 \leq$ M ≤ 2.

3: 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.

4: TOUT — ***real*** *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 D03PHF.

Its specification is:

```
      SUBROUTINE PDEDEF(NPDE, T, X, U, UX, NCODE, V, VDOT, P, Q, R, IRES)
      INTEGER            NPDE, NCODE, IRES
      real               T, X, U(NPDE), UX(NPDE), V(*), VDOT(*),
     1                   P(NPDE,NPDE), Q(NPDE), R(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 $\frac{\partial U_i(x,t)}{\partial x}$, for $i = 1,2,\ldots,$NPDE.

6: NCODE — INTEGER *Input*

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

7: V(*) — ***real*** array *Input*

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

8: VDOT(*) — ***real*** array *Input*

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

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

9: P(NPDE,NPDE) — ***real*** array *Output*

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

10: Q(NPDE) — ***real*** array *Output*

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

11: R(NPDE) — ***real*** array *Output*

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

12: 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 D03PHF 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 D03PHF 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 β_i and γ_i which describe the boundary conditions, as given in (4).

Its specification is:

```
      SUBROUTINE BNDARY(NPDE, T, U, UX, NCODE, V, VDOT, IBND, BETA,
     1                  GAMMA, IRES)
      INTEGER           NPDE, NCODE, IBND, IRES
      real              T, U(NPDE), UX(NPDE), V(*), VDOT(*), BETA(NPDE),
     1                  GAMMA(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: U(NPDE) — ***real*** array *Input*

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

4: UX(NPDE) — ***real*** array *Input*

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

5: NCODE — INTEGER *Input*

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

6: V(*) — ***real*** array *Input*

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

7: VDOT(*) — ***real*** array *Input*

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

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

8: IBND — INTEGER *Input*

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

9: BETA(NPDE) — ***real*** 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*** array *Output*

On entry: GAMMA(i) must be set to the value of $\gamma_i(x,t,U,U_x,V,\dot{V})$ 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, 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 D03PHF 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 D03PHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: U(NEQN) — ***real*** array *Input/Output*

On entry: the initial values of the dependent variables defined as follows:
U(NPDE $\times$ $(j-1)+i$) contain $U_i(x_j,t_0)$, for $i = 1,2,\ldots,$NPDE; $j = 1,2,\ldots,$NPTS and U(NPTS $\times$ NPDE $+\, i$) contain $V_i(t_0)$, for $i = 1,2,\ldots,$NCODE.

On exit: the computed solution $U_i(x_j,t)$, for i = 1,2,...,NPDE; j = 1,2,...,NPTS, and $V_k(t)$, for k = 1,2,...,NCODE, evaluated at t = TS.

8: NPTS — INTEGER *Input*

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

Constraint: NPTS $\geq$ 3.

9: X(NPTS) — ***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: X(1) < X(2) < ... < X(NPTS).

10: NCODE — INTEGER *Input*

On entry: the number of coupled ODE components.

Constraint: NCODE ≥ 0.

11: ODEDEF — SUBROUTINE, supplied by the user. *External Procedure*

ODEDEF must evaluate the functions F, which define the system of ODEs, as given in (3). If the user wishes to compute the solution of a system of PDEs only (i.e., NCODE = 0), ODEDEF must be the dummy routine D03PCK. (D03PCK is included in the NAG Fortran Library; however, its name may be implementation-dependent: see the the Users' Note for your implementation for details.)

Its specification is:

```
      SUBROUTINE ODEDEF(NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,
     1                  RCP, UCPT, UCPTX, F, IRES)
      INTEGER           NPDE, NCODE, NXI, IRES
      real              T, V(*), VDOT(*), XI(*), UCP(NPDE,*),
     1                  UCPX(NPDE,*), RCP(NPDE,*), UCPT(NPDE,*),
     2                  UCPTX(NPDE,*), F(*)
```

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: NCODE — INTEGER *Input*

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

4: V(*) — ***real*** array *Input*

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

5: VDOT(*) — ***real*** array *Input*

On entry: 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(*) — ***real*** array *Input*

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

8: UCP(NPDE,*) — ***real*** array *Input*

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

9: UCPX(NPDE,*) — ***real*** array *Input*

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

10: RCP(NPDE,*) — ***real*** 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,...,NPDE; j = 1,2,...,NXI.

11: UCPT(NPDE,*) — ***real*** array *Input*

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

12: UCPTX(NPDE,∗) — ***real*** array *Input*

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

13: F(∗) — ***real*** array *Output*

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

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

or

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

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. If IRES = 1, then the equation (5) above must be used. If IRES = −1, then the equation (6) above must be used.

On exit: should usually remain unchanged. However, the user 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)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 D03PHF returns to the calling (sub)program with the error indicator set to IFAIL = 4.

ODEDEF must be declared as EXTERNAL in the (sub)program from which D03PHF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

12: NXI — INTEGER *Input*

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

Constraints:

NXI = 0 if NCODE = 0,
NXI ≥ 0 if NCODE > 0.

13: XI(∗) — ***real*** array *Input*

Note: the dimension of the array XI must be at least max(1, NXI).

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

Constraint: X(1) ≤ XI(1) < XI(2) < ... < XI(NXI) ≤ X(NPTS).

14: NEQN — INTEGER *Input*

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

Constraint: NEQN = NPDE × NPTS + NCODE.

15: RTOL(∗) — ***real*** 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) ≥ 0 for all relevant i.

16: ATOL(*) — ***real*** 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.

Constraint: ATOL(i) ≥ 0 for all relevant i.

17: ITOL — INTEGER *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D03PHF 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,...,NEQN.

The choice of norm used is defined by the parameter NORM, see below.

Constraint: $1 \leq$ ITOL ≤ 4.

18: NORM — CHARACTER*1 *Input*

On entry: the type of norm to be used. Two options are available:

'M' – maximum norm.

'A' – averaged L_2 norm.

If U_{norm} denotes the norm of the vector U of length NEQN, then for the averaged L_2 norm

$$U_{\text{norm}} = \sqrt{\frac{1}{\text{NEQN}} \sum_{i=1}^{\text{NEQN}} (U(i)/w_i)^2},$$

while for the maximum norm

$$U_{\text{norm}} = \max_i |U(i)/w_i|.$$

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

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

19: LAOPT — CHARACTER*1 *Input*

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

'F' – full matrix routines to be used;

'B' – banded matrix routines to be used;

'S' – sparse matrix routines to be used.

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

Note: the user is recommended to use the banded option when no coupled ODEs are present (NCODE = 0).

20: ALGOPT(30) — ***real*** array *Input*

On entry: ALGOPT may be set to control various options available in the integrator. If the user wishes 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 parameter 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 initialise 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, i.e., 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 < 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.

21: W(NW) — ***real*** array *Workspace*

22: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PHF is called. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',
  NW ≥ NEQN × NEQN + NEQN + NWKRES + LENODE,
LAOPT = 'B',
  NW ≥ (3×MLU+1) × NEQN + NWKRES + LENODE,
LAOPT = 'S',
  NW ≥ 4 × NEQN + 11 × NEQN/2 + 1 + NWKRES + LENODE.

where MLU = the lower or upper half bandwidths, and

MLU = 2 × NPDE−1, for PDE problems only, and

MLU = NEQN−1, for coupled PDE/ODE problems.

NWKRES = NPDE × (NPTS+6×NXI+3×NPDE+15) + NXI + NCODE + 7 × NPTS + 2 when NCODE > 0, and NXI > 0.

NWKRES = NPDE × (NPTS+3×NPDE+21) × NCODE + 7 × NPTS+3 when NCODE > 0, and NXI = 0.

NWKRES = NPDE × (NPTS+3×NPDE+21) + 7 × NPTS + 4 when NCODE = 0.

LENODE = (6+int(ALGOPT(2))) × NEQN+50, when the BDF method is used and

LENODE = 9 × NEQN+50, when the Theta method is used.

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

23: IW(NIW) — INTEGER array *Output*

On entry: 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.

24: NIW — INTEGER *Input*

On entry: the dimension of the array IW. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',
  NIW $\geq$ 24,

LAOPT = 'B',
  NIW $\geq$ NEQN+24,

LAOPT = 'S',
  NIW $\geq 25 \times$ NEQN+24.

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

25: 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.

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 parameter ALGOPT.

ITASK = 5
  take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter ALGOPT.

Constraint: $1 \leq$ ITASK ≤ 5.

26: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03PHF 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 subchapter D02M–N of the NAG Fortran Library.

27: 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 D03PHF.

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

On exit: IND = 1.

28: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TOUT−TS is too small,
or ITASK ≠ 1, 2, 3, 4 or 5,
or M ≠ 0, 1 or 2,
or at least one of the coupling points defined in array XI is outside the interval [X(1),X(NPTS)],
or M > 0 and X(1) < 0.0,
or NPTS < 3,
or NPDE < 1,
or NORM ≠ 'A' or 'M',
or LAOPT ≠ 'F', 'B' or 'S',
or ITOL ≠ 1, 2, 3 or 4,
or IND ≠ 0 or 1,
or incorrectly defined user mesh, i.e., X(i) ≥ X(i+1) for some $i = 1, 2, \ldots$,NPTS−1,
or NW or NIW are too small,
or NCODE and NXI are incorrectly defined,
or IND = 1 on initial entry to D03PHF,
or NEQN ≠ NPDE × NPTS+NCODE,
or either an element of RTOL or ATOL < 0.0,
or all the elements of RTOL and ATOL are zero.

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 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, 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. The user should check his 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, 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 the user-supplied routines, PDEDEF, BNDARY or ODEDEF, IRES was set to an invalid value.

IFAIL = 9

A serious error has occurred in an internal call to D02NNF. Check problem specifications 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 ATOL and RTOL 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 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 advisory 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) has been taken.

IFAIL = 13

Some error weights w_i became zero during the time integration (see 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 succesful 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 NIW or NW was not sufficient (more detailed information may be directed to the current error message unit).

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 accuracy parameters ATOL and RTOL.

8 Further Comments

The parameter specification allows the user 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 D03PKF.

The time taken by the routine depends on the complexity of the parabolic system and on the accuracy requested. For a given system and a fixed accuracy it is approximately proportional to NEQN.

9 Example

This problem provides a simple coupled system of one PDE and one ODE.

$$(V_1)^2\frac{\partial U_1}{\partial t} - xV_1\dot{V}_1\frac{\partial U_1}{\partial x} = \frac{\partial^2 U_1}{\partial x^2}$$

$$\dot{V}_1 = V_1U_1 + \frac{\partial U_1}{\partial x} + 1 + t,$$

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

The left boundary condition at $x = 0$ is

$$\frac{\partial U_1}{\partial x} = -V_1 \exp t.$$

The right boundary condition at $x = 1$ is

$$\frac{\partial U_1}{\partial x} = -V_1\dot{V}_1$$

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

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

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

9.1 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.

```
*     D03PHF Example Program Text
*     Mark 16 Revised. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, NCODE, M, NXI, NEQN, NIW, NWKRES,
     +                 LENODE, NW
      PARAMETER        (NPDE=1,NPTS=21,NCODE=1,M=0,NXI=1,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=24,
     +                 NWKRES=NPDE*(NPTS+6*NXI+3*NPDE+15)
     +                 +NCODE+NXI+7*NPTS+2,LENODE=11*NEQN+50,
```

```
     +                 NW=NEQN*NEQN+NEQN+NWKRES+LENODE)
*     .. Scalars in Common ..
      real             TS
*     .. Local Scalars ..
      real             TOUT
      INTEGER          I, IFAIL, IND, IT, ITASK, ITOL, ITRACE
      LOGICAL          THETA
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), EXY(NPTS), RTOL(1), U(NEQN),
     +                 W(NW), X(NPTS), XI(1)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PHF, EXACT, ODEDEF, PDEDEF, UVINIT
*     .. Common blocks ..
      COMMON           /TAXIS/TS
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PHF Example Program Results'
      ITRACE = 0
      ITOL = 1
      ATOL(1) = 1.0e-4
      RTOL(1) = ATOL(1)
      WRITE (NOUT,99997) ATOL, NPTS
*
*     Set break-points
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
      XI(1) = 1.0e0
      NORM = 'A'
      LAOPT = 'F'
      IND = 0
      ITASK = 1
*
*     Set THETA to .TRUE. if the Theta integrator is required
*
      THETA = .FALSE.
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
      IF (THETA) THEN
         ALGOPT(1) = 2.0e0
      ELSE
         ALGOPT(1) = 0.0e0
      END IF
*
*     Loop over output value of t
*
      TS = 1.0e-4
      TOUT = 0.0e0
      WRITE (NOUT,99999) X(1), X(5), X(9), X(13), X(21)
      CALL UVINIT(NPDE,NPTS,X,U,NCODE,NEQN)
      DO 60 IT = 1, 5
         TOUT = 0.1e0*(2**IT)
         IFAIL = -1
*
```

```
         CALL D03PHF(NPDE,M,TS,TOUT,PDEDEF,BNDARY,U,NPTS,X,NCODE,ODEDEF,
     +               NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,ALGOPT,W,NW,
     +               IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
*        Check against the exact solution
*
         CALL EXACT(TOUT,NPTS,X,EXY)
         WRITE (NOUT,99998) TS
         WRITE (NOUT,99995) U(1), U(5), U(9), U(13), U(21), U(22)
         WRITE (NOUT,99994) EXY(1), EXY(5), EXY(9), EXY(13), EXY(21), TS
   60 CONTINUE
      WRITE (NOUT,99996) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT ('  X        ',5F9.3,/)
99998 FORMAT (' T = ',F6.3)
99997 FORMAT (//'  Simple coupled PDE using BDF ',/'  Accuracy require',
     +       'ment =',e10.3,' Number of points = ',I4,/)
99996 FORMAT (' Number of integration steps in time = ',I6,/' Number o',
     +       'f function evaluations = ',I6,/' Number of Jacobian eval',
     +       'uations =',I6,/' Number of iterations = ',I6,/)
99995 FORMAT (1X,'App.  sol.  ',F7.3,4F9.3,'  ODE sol. =',F8.3)
99994 FORMAT (1X,'Exact sol.  ',F7.3,4F9.3,'  ODE sol. =',F8.3,/)
      END
*
      SUBROUTINE UVINIT(NPDE,NPTS,X,U,NCODE,NEQN)
*     Routine for PDE initial values
*     .. Scalar Arguments ..
      INTEGER           NCODE, NEQN, NPDE, NPTS
*     .. Array Arguments ..
      real              U(NEQN), X(NPTS)
*     .. Scalars in Common ..
      real              TS
*     .. Local Scalars ..
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Common blocks ..
      COMMON            /TAXIS/TS
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         U(I) = EXP(TS*(1.0e0-X(I))) - 1.0e0
   20 CONTINUE
      U(NEQN) = TS
      RETURN
      END
*
      SUBROUTINE ODEDEF(NPDE,T,NCODE,V,VDOT,NXI,XI,UCP,UCPX,RCP,UCPT,
     +                  UCPTX,F,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IRES, NCODE, NPDE, NXI
*     .. Array Arguments ..
      real              F(*), RCP(NPDE,*), UCP(NPDE,*), UCPT(NPDE,*),
     +                  UCPTX(NPDE,*), UCPX(NPDE,*), V(*), VDOT(*),
     +                  XI(*)
*     .. Executable Statements ..
      IF (IRES.EQ.1) THEN
```

```
         F(1) = VDOT(1) - V(1)*UCP(1,1) - UCPX(1,1) - 1.0e0 - T
      ELSE IF (IRES.EQ.-1) THEN
         F(1) = VDOT(1)
      END IF
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,UX,NCODE,V,VDOT,P,Q,R,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              P(NPDE,NPDE), Q(NPDE), R(NPDE), U(NPDE),
     +                  UX(NPDE), V(*), VDOT(*)
*     .. Executable Statements ..
      P(1,1) = V(1)*V(1)
      R(1) = UX(1)
      Q(1) = -X*UX(1)*V(1)*VDOT(1)
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,U,UX,NCODE,V,VDOT,IBND,BETA,GAMMA,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              BETA(NPDE), GAMMA(NPDE), U(NPDE), UX(NPDE),
     +                  V(*), VDOT(*)
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Executable Statements ..
      BETA(1) = 1.0e0
      IF (IBND.EQ.0) THEN
         GAMMA(1) = -V(1)*EXP(T)
      ELSE
         GAMMA(1) = -V(1)*VDOT(1)
      END IF
      RETURN
      END
*
      SUBROUTINE EXACT(TIME,NPTS,X,U)
*     Exact solution (for comparison purpose)
*     .. Scalar Arguments ..
      real             TIME
      INTEGER          NPTS
*     .. Array Arguments ..
      real             U(NPTS), X(NPTS)
*     .. Local Scalars ..
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         U(I) = EXP(TIME*(1.0e0-X(I))) - 1.0e0
   20 CONTINUE
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D03PHF Example Program Results


  Simple coupled PDE using BDF
  Accuracy requirement = 0.100E-03 Number of points =   21

  X              0.000     0.200     0.400     0.600     1.000

 T =  0.200
 App.  sol.      0.222     0.174     0.128     0.084     0.001  ODE sol. =    0.200
 Exact sol.      0.221     0.174     0.127     0.083     0.000  ODE sol. =    0.200

 T =  0.400
 App.  sol.      0.494     0.379     0.273     0.176     0.002  ODE sol. =    0.400
 Exact sol.      0.492     0.377     0.271     0.174     0.000  ODE sol. =    0.400

 T =  0.800
 App.  sol.      1.229     0.901     0.622     0.384     0.008  ODE sol. =    0.798
 Exact sol.      1.226     0.896     0.616     0.377     0.000  ODE sol. =    0.800

 T =  1.600
 App.  sol.      3.959     2.610     1.629     0.917     0.027  ODE sol. =    1.594
 Exact sol.      3.953     2.597     1.612     0.896     0.000  ODE sol. =    1.600

 T =  3.200
 App.  sol.     23.470    11.974     5.886     2.665     0.074  ODE sol. =    3.184
 Exact sol.     23.533    11.936     5.821     2.597     0.000  ODE sol. =    3.200

 Number of integration steps in time =     32
 Number of function evaluations =    443
 Number of Jacobian evaluations =    15
 Number of iterations =    106
```

D03PJF – NAG Fortran Library Routine Document

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

D03PJF integrates a system of linear or nonlinear parabolic partial differential equations (PDEs), in one space variable with scope for coupled ordinary differential equations (ODEs). The spatial discretisation is performed using a Chebyshev C^0 collocation method, and the method of lines is employed to reduce the PDEs to a system of ODEs. The resulting system is solved using a backward differentiation formula (BDF) method or a Theta method (switching between Newton's method and functional iteration).

2 Specification

```
 SUBROUTINE D03PJF(NPDE, M, TS, TOUT, PDEDEF, BNDARY, U, NBKPTS,
1                  XBKPTS, NPOLY, NPTS, X, NCODE, ODEDEF, NXI, XI,
2                  NEQN, UVINIT, RTOL, ATOL, ITOL, NORM, LAOPT,
3                  ALGOPT, W, NW, IW, NIW, ITASK, ITRACE, IND,
4                  IFAIL)
 INTEGER           NPDE, M, NBKPTS, NPOLY, NPTS, NCODE, NXI, NEQN,
1                  ITOL, NW, IW(NIW), NIW, ITASK, ITRACE, IND, IFAIL
 real              TS, TOUT, U(NEQN), XBKPTS(NBKPTS), X(NPTS),
1                  XI(*), RTOL(*), ATOL(*), ALGOPT(30), W(NW)
 CHARACTER*1       NORM, LAOPT
 EXTERNAL          PDEDEF, BNDARY, ODEDEF, UVINIT
```

3 Description

D03PJF integrates the system of parabolic-elliptic equations and coupled ODEs

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

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

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

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

and the vector U_x is the partial derivative with respect to x. Note that $P_{i,j}$, Q_i and R_i must not depend on $\frac{\partial U}{\partial t}$. The vector V is the set of ODE solution values

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

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 $F = [F_1, \ldots, F_{\mathrm{NCODE}}]^T$, G is a vector of length NCODE, A is an NCODE by NCODE matrix, B is an NCODE by ($n_\xi \times$ NPDE) matrix and the entries in G, A and B may depend on t, ξ, U^*, U_x^* and V. In practice the user needs only to supply a vector of information to define the ODEs and not the matrices A and B. (See Section 5 for the specification of the user-supplied procedure 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 $b = x_{\mathrm{NBKPTS}}$ are the leftmost and rightmost of a user-defined set of break-points $x_1, x_2, \ldots, x_{\mathrm{NBKPTS}}$. The co-ordinate system in space is defined by the value of m; $m = 0$ for Cartesian co-ordinates, $m = 1$ for cylindrical polar co-ordinates and $m = 2$ for spherical polar co-ordinates.

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

The initial values of the functions $U(x,t)$ and $V(t)$ must be given at $t = t_0$. These values are calculated in a user-supplied subroutine, UVINIT.

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

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

where $x = a$ or $x = b$. The functions γ_i may only depend **linearly** on $\dot{V}$.

The boundary conditions must be specified in a subroutine BNDARY provided by the user.

The algebraic-differential equation system which is defined by the functions F_i must be specified in a subroutine ODEDEF supplied by the user. The user must also specify the coupling points ξ in the array XI. Thus, the problem is subject to the following restrictions:

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

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

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

(iv) The evaluation of the functions $P_{i,j}$, Q_i and R_i is done at both the break-points and internally selected points for each element in turn, that is $P_{i,j}$, Q_i and R_i are evaluated twice at each break-point. Any discontinuities in these functions **must** therefore be at one or more of the mesh points;

(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;

(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 either by specifying the solution at $x = 0.0$ or by specifying a zero flux there, that is $\beta_i = 1.0$ and $\gamma_i = 0.0$.

The parabolic equations are approximated by a system of ODEs in time for the values of U_i at the mesh points. This ODE system is obtained by approximating the PDE solution between each pair of break-points by a Chebyshev polynomial of degree NPOLY. The interval between each pair of break-points is treated by D03PJF as an element, and on this element, a polynomial and its space and time derivatives are made to satisfy the system of PDEs at NPOLY − 1 spatial points, which are chosen internally by the code and the break-points. The user-defined break-points and the internally selected points together define the mesh. The smallest value that NPOLY can take is one, in which case, the solution is approximated by piecewise linear polynomials between consecutive break-points and the method is similar to an ordinary finite element method.

In total there are (NBKPTS − 1) × NPOLY + 1 mesh points in the spatial direction, and NPDE × ((NBKPTS − 1) × NPOLY + 1) + NCODE ODEs in the time direction; one ODE at each break-point for each PDE component, NPOLY − 1 ODEs for each PDE component between each pair of break-points, and NCODE coupled ODEs. The system is then integrated forwards in time using a Backward Differentiation Formula (BDF) method or a Theta method.

4 References

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

[2] Berzins M and Dew P M (1991) Algorithm 690: Chebyshev polynomial software for elliptic-parabolic systems of PDEs *ACM Trans. Math. Software* **17** 178–206

[3] Berzins M, Dew P M and Furzeland R M (1988) Software tools for time-dependent equations in simulation and optimisation of large systems *Proc. IMA Conf. Simulation and Optimization* (ed A J Osiadcz) Clarendon Press, Oxford 35–50

[4] 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

[5] Zaturska N B, Drazin P G and Banks W H H (1988) On the flow of a viscous fluid driven along a channel by a suction at porous walls *Fluid Dynamics Research* **4**

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE ≥ 1.

2: M — INTEGER *Input*

On entry: the co-ordinate system used:

M=0
 indicates Cartesian co-ordinates,

M=1
 indicates cylindrical polar co-ordinates,

M=2
 indicates spherical polar co-ordinates.

Constraint: $0 \leq \text{M} \leq 2$.

3: 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.

4: TOUT — ***real*** *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 compute 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 $\dot{V}$. The functions must be evaluated at a set of points.

Its specification is:

```
      SUBROUTINE PDEDEF(NPDE, T, X, NPTL, U, UX, NCODE, V, VDOT, P, Q,
     1                  R, IRES)
      INTEGER           NPDE, NPTL, NCODE, IRES
      real              T, X(NPTL), U(NPDE,NPTL), UX(NPDE,NPTL), V(*),
     1                  VDOT(*), P(NPDE,NPDE,NPTL), Q(NPDE,NPTL),
     2                  R(NPDE,NPTL)
```

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(NPTL) — ***real*** array *Input*

On entry: contains a set of mesh points at which $P_{i,j}$, Q_i and R_i are to be evaluated. X(1) and X(NPTL) contain successive user-supplied break-points and the elements of the array will satisfy X(1) < X(2) < ... < X(NPTL).

4: NPTL — INTEGER *Input*

On entry: the number of points at which evaluations are required (the value NPOLY + 1).

5: U(NPDE,NPTL) — ***real*** array *Input*

On entry: U(i,j) contains the value of the component $U_i(x,t)$ where x = X(j), for i = 1,2,...,NPDE; j = 1,2,...,NPTL.

6: UX(NPDE,NPTL) — ***real*** array *Input*

On entry: UX(i,j) contains the value of the component $\frac{\partial U_i(x,t)}{\partial x}$ where x = X(j), for i = 1,2,...,NPDE; j = 1,2,...,NPTL.

7: NCODE — INTEGER *Input*

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

8: V(*) — ***real*** array *Input*

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

9: VDOT(*) — ***real*** array *Input*

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

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

10: P(NPDE,NPDE,NPTL) — ***real*** array *Output*

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

11: Q(NPDE,NPTL) — ***real*** array *Output*

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

12: R(NPDE,NPTL) — ***real*** array *Output*

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

13: 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 D03PJF 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 D03PJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

BNDARY must compute the functions β_i and γ_i which define the boundary conditions as in equation (4).

Its specification is:

```
      SUBROUTINE BNDARY(NPDE, T, U, UX, NCODE, V, VDOT, IBND, BETA,
     1                  GAMMA, IRES)
      INTEGER           NPDE, NCODE, IBND, IRES
      real              T, U(NPDE), UX(NPDE), V(*), VDOT(*), BETA(NPDE),
     1                  GAMMA(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: U(NPDE) — ***real*** array *Input*

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

4: UX(NPDE) — ***real*** array *Input*

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

5: NCODE — INTEGER *Input*

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

6: V(*) — ***real*** array *Input*

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

7: VDOT(*) — ***real*** 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. If IBND $= 0$, then BNDARY must set up the coefficients of the left-hand boundary $x = a$. If IBND $\neq 0$, then BNDARY must set up the coefficients on the right-hand boundary, $x = b$.

9: BETA(NPDE) — ***real*** 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*** array *Output*

On exit: GAMMA(i) must be set to the value of $\gamma_i(x,t,U,U_x,V,\dot{V})$ 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, 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 D03PJF 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 D03PJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: U(NEQN) — ***real*** array *Output*

On exit: the computed solution $U_i(x_j,t)$, for i = 1,2,...,NPDE; j = 1,2,...,NPTS and $V_k(t)$, for k = 1,2,...,NCODE, evaluated at t = TS, as follows:

U(NPDE × $(j-1)+i$) contain $U_i(x_j,t)$, for i = 1,2,...,NPDE; j = 1,2,...,NPTS and

U(NPTS × NPDE + i) contain $V_i(t)$, for i = 1,2,...,NCODE.

8: NBKPTS — INTEGER *Input*

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

Constraint: NBKPTS $\geq$ 2.

9: XBKPTS(NBKPTS) — ***real*** array *Input*

On entry: the values of the break-points in the space direction. XBKPTS(1) must specify the left-hand boundary, a, and XBKPTS(NBKPTS) must specify the right-hand boundary, b.

Constraint: XBKPTS(1) < XBKPTS(2) < ... < XBKPTS(NBKPTS).

10: NPOLY — INTEGER *Input*

On entry: the degree of the Chebyshev polynomial to be used in approximating the PDE solution between each pair of break-points.

Constraint: 1 $\leq$ NPOLY $\leq$ 49.

11: NPTS — INTEGER *Input*

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

Constraint: NPTS = (NBKPTS − 1) × NPOLY + 1.

12: X(NPTS) — ***real*** array *Output*

On exit: the nesh points chosen by D03PJF in the spatial direction. The values of X will satisfy X(1) < X(2) < ... < X(NPTS).

13: NCODE — INTEGER *Input*

On entry: the number of coupled ODEs components.

Constraint: NCODE $\geq$ 0.

14: ODEDEF — SUBROUTINE, supplied by the user. *External Procedure*

ODEDEF must evaluate the functions F, which define the system of ODEs, as given in (3). If the user wishes to compute the solution of a system of PDEs only (i.e., NCODE = 0), ODEDEF must be the dummy routine D03PCK. (D03PCK 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 ODEDEF(NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,
     1                  RCP, UCPT, UCPTX, F, IRES)
      INTEGER           NPDE, NCODE, NXI, IRES
      real              T, V(*), VDOT(*), XI(*), UCP(NPDE,*),
     1                  UCPX(NPDE,*), RCP(NPDE,*), UCPT(NPDE,*),
     2                  UCPTX(NPDE,*), F(*)
```

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: NCODE — INTEGER *Input*

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

4: V(*) — ***real*** array *Input*

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

5: VDOT(*) — ***real*** array *Input*

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

6: NXI — INTEGER *Input*

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

7: XI(*) — ***real*** array *Input*

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

8: UCP(NPDE,*) — ***real*** array *Input*

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

9: UCPX(NPDE,*) — ***real*** array *Input*

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

10: RCP(NPDE,*) — ***real*** array *Input*

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

11: UCPT(NPDE,*) — ***real*** array *Input*

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

12: UCPTX(NPDE,*) — ***real*** array *Input*

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

13: F(*) — ***real*** array *Output*

On exit: F(i) must contain the ith component of F, for $i = 1,2,...,$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. If IRES = 1, then equation (5) above must be used. If IRES = −1, then equation (6) above must be used.

On exit: should usually remain unchanged. However, the user 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)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 D03PJF returns to the calling (sub)program with the error indicator set to IFAIL = 4.

ODEDEF must be declared as EXTERNAL in the (sub)program from which D03PJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

15: NXI — INTEGER *Input*

On entry: number of ODE/PDE coupling points.

Constraints:

NXI = 0 if NCODE = 0.
NXI ≥ 0 if NCODE > 0.

16: XI(∗) — ***real*** array *Input*

Note: the dimension of the array XI must be at least max(1,NXI).

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

Constraint: XBKPTS(1) ≤ XI(1) < XI(2) < ... < XI(NXI) ≤ XBKPTS(NBKPTS).

17: NEQN — INTEGER *Input*

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

Constraint: NEQN = NPDE × NPTS + NCODE

18: UVINIT — SUBROUTINE, supplied by the user. *External Procedure*

UVINIT must compute the initial values of the PDE and the ODE components $U_i(x_j, t_0)$, for $i = 1,2,\ldots,$NPDE; $j = 1,2,\ldots,$NPTS, and $V_k(t_0)$, for $k = 1,2,\ldots,$NCODE.

Its specification is:

```
SUBROUTINE UVINIT(NPDE, NPTS, X, U, NCODE, V)
INTEGER            NPDE, NPTS, NCODE
real               X(NPTS), U(NPDE,NPTS), V(*)
```

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: X(NPTS) — ***real*** array *Input*

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

4: U(NPDE,NPTS) — ***real*** array *Output*

On exit: U(i,j) must be set to the initial value $U_i(x_j,t_0)$, for $i = 1,2,\ldots,$NPDE; $j = 1,2,\ldots,$NPTS.

5: NCODE — INTEGER *Input*

On entry: the number of coupled ODEs.

6: V($*$) — ***real*** array *Input*

On exit: V(i) must be set to the initial values of the components $V_i(t_0)$, for $i = 1,2,\ldots,$NCODE.

UVINIT must be declared as EXTERNAL in the (sub)program from which D03PJF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

19: RTOL($*$) — ***real*** 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) ≥ 0 for all relevant i.

20: ATOL($*$) — ***real*** 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.

Constraint: ATOL(i) ≥ 0 for all relevant i.

21: ITOL — INTEGER *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D03PJF 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 parameter NORM, see below.

Constraint: $1 \leq$ ITOL ≤ 4.

22: NORM — CHARACTER*1 *Input*

On entry: the type of norm to be used. Two options are available:

'M' – maximum 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

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

while for the maximum norm

$$U_{\text{norm}} = \max_i |U(i)/w_i|.$$

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

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

23: LAOPT — CHARACTER*1 *Input*

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

'F' – full matrix routines to be used;
'B' – banded matrix routines to be used;
'S' – sparse matrix routines to be used.

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

Note. The user is recommended to use the banded option when no coupled ODEs are present (NCODE = 0).

24: ALGOPT(30) — ***real*** array *Input*

On entry: ALGOPT may be set to control various options available in the integrator. If the user wishes 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 parameter 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 initialise 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, i.e., 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 < 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. ALGOPT(30) must be greater than zero, otherwise the default value is used. 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.

25: W(NW) — ***real*** array *Workspace*

26: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PJF is called. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',
  NW $\geq$ NEQN $\times$ NEQN + NEQN + NWKRES + LENODE,
LAOPT = 'B',
  NW $\geq$ (3$\times$MLU+1) $\times$ NEQN + NWKRES + LENODE,
LAOPT = 'S',
  NW $\geq$ 4 $\times$ NEQN + 11 $\times$ NEQN/2 + 1 + NWKRES + LENODE.

Where MLU = the lower or upper half bandwidths, and

MLU = (NPOLY+1) × NPDE−1, for PDE problems only, and,

MLU = NEQN−1, for coupled PDE/ODE problems.

$$\begin{aligned}\text{NWKRES} = {} & 3 \times (\text{NPOLY} + 1)^2 \\ & + (\text{NPOLY} + 1) \times [\text{NPDE}^2 + 6 \times \text{NPDE} + \text{NBKPTS} + 1] \\ & + 8 \times \text{NPDE} + \text{NXI} \times (5 \times \text{NPDE} + 1) + \text{NCODE} + 3,\end{aligned}$$

when NCODE > 0, and NXI > 0.

$$\begin{aligned}\text{NWKRES} = {} & 3 \times (\text{NPOLY} + 1)^2 \\ & + (\text{NPOLY} + 1) \times [\text{NPDE}^2 + 6 \times \text{NPDE} + \text{NBKPTS} + 1] \\ & + 13 \times \text{NPDE} + \text{NCODE} + 4,\end{aligned}$$

when NCODE > 0, and NXI = 0.

$$\begin{aligned}\text{NWKRES} = {} & 3 \times (\text{NPOLY} + 1)^2 \\ & + (\text{NPOLY} + 1) \times [\text{NPDE}^2 + 6 \times \text{NPDE} + \text{NBKPTS} + 1] \\ & + 13 \times \text{NPDE} + 5,\end{aligned}$$

when NCODE = 0.

LENODE = (6+int(ALGOPT(2))}) × NEQN+50, when the BDF method is used and,

LENODE = 9 × NEQN+50, when a Theta method is used.

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

27: 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.

28: NIW — INTEGER *Input*

On entry: the dimension of array IW. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',

NIW ≥ 24,

LAOPT = 'B',

NIW ≥ NEQN+24,

LAOPT = 'S',

NIW ≥ 25 × NEQN+24.

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

29: 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.

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 parameter ALGOPT.

ITASK = 5
: take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter ALGOPT.

Constraint: $1 \le$ ITASK ≤ 5.

30: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03PJF 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 subchapter D02M-N of the NAG Fortran Library.

31: 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 D03PJF.

Constraint: $0 \le$ IND ≤ 1.

On exit: IND = 1.

32: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

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 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 point in array XI is outside the interval [XBKPTS(1),XBKPTS(NBKPTS)],

or NPTS $\neq$ (NBKPTS $-$ 1) $\times$ NPOLY + 1,

or NBKPTS < 2,

or NPDE $\leq$ 0,

or NORM $\neq$ 'A' or 'M',

or ITOL $\neq$ 1, 2, 3 or 4,

or NPOLY < 1 or NPOLY > 49,

or NCODE and NXI are incorrectly defined,

or NEQN $\neq$ NPDE $\times$ NPTS + NCODE,

or LAOPT $\neq$ 'F', 'B' or 'S',

or IND $\neq$ 0 or 1,

or incorrectly defined user break-points, i.e., XBKPTS(i) $\geq$ XBKPTS(i + 1), for some $i = 1, 2, \ldots,$ NBKPTS $-$ 1,

or NW or NIW are too small,

or the ODE integrator has not been correctly defined; check ALGOPT parameter.

or IND = 1 on initial entry to D03PJF,

or either an element of RTOL or ATOL < 0.0,

or all the elements of RTOL and ATOL are zero.

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 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 the user-supplied subroutines 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. The user should check his 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, 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 the user-supplied routines, PDEDEF, BNDARY or ODEDEF, 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 ATOL and RTOL 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 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).

IFAIL = 12

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

IFAIL = 13

Some error weights w_i became zero during the time integration (see 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 succesful 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 NIW or NW was not sufficient (more detailed information may be directed to the current error message unit).

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 accuracy parameter ATOL and RTOL.

8 Further Comments

The parameter specification allows the user 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.

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

9 Example

This problem provides a simple coupled system of one PDE and one ODE.

$$(V_1)^2\frac{\partial U_1}{\partial t} - xV_1\dot{V}_1\frac{\partial U_1}{\partial x} = \frac{\partial^2 U_1}{\partial x^2}$$

$$\dot{V}_1 = V_1U_1 + \frac{\partial U_1}{\partial x} + 1 + t,$$

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

The left boundary condition at $x = 0$ is

$$\frac{\partial U_1}{\partial x} = -V_1 \exp t.$$

The right boundary condition at $x = 1$ is

$$U_1 = -V_1 \dot{V}_1.$$

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

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

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

9.1 Example 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.

```
*     D03PJF Example Program Text
*     Mark 16 Revised. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NBKPTS, NEL, NPDE, NPOLY, NPTS, NCODE, M, NXI,
     +                 NEQN, NIW, NPL1, NWKRES, LENODE, NW
      PARAMETER        (NBKPTS=11,NEL=NBKPTS-1,NPDE=1,NPOLY=2,
     +                 NPTS=NEL*NPOLY+1,NCODE=1,M=0,NXI=1,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=24,NPL1=NPOLY+1,
     +                 NWKRES=3*NPL1*NPL1+NPL1*
     +                 (NPDE*NPDE+6*NPDE+NBKPTS+1)+8*NPDE+NXI*(5*NPDE+1)
     +                 +NCODE+3,LENODE=11*NEQN+50,
     +                 NW=NEQN*NEQN+NEQN+NWKRES+LENODE)
*     .. Scalars in Common ..
      real             TS
*     .. Local Scalars ..
      real             TOUT
      INTEGER          I, IFAIL, IND, IT, ITASK, ITOL, ITRACE
      LOGICAL          THETA
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), EXY(NBKPTS), RTOL(1),
     +                 U(NEQN), W(NW), X(NPTS), XBKPTS(NBKPTS), XI(1)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PJF, EXACT, ODEDEF, PDEDEF, UVINIT
*     .. Common blocks ..
      COMMON           /TAXIS/TS
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PJF Example Program Results'
      ITRACE = 0
      ITOL = 1
      ATOL(1) = 1.0e-4
      RTOL(1) = ATOL(1)
      WRITE (NOUT,99999) NPOLY, NEL
      WRITE (NOUT,99996) ATOL, NPTS
*
```

```
*     Set break-points
*
      DO 20 I = 1, NBKPTS
         XBKPTS(I) = (I-1.0e0)/(NBKPTS-1.0e0)
   20 CONTINUE
*
      XI(1) = 1.0e0
      NORM = 'A'
      LAOPT = 'F'
      IND = 0
      ITASK = 1
*
*     Set THETA to .TRUE. if the Theta integrator is required
*
      THETA = .FALSE.
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
      IF (THETA) THEN
         ALGOPT(1) = 2.0e0
      ELSE
         ALGOPT(1) = 0.0e0
      END IF
*
*     Loop over output value of t
*
      TS = 1.0e-4
      TOUT = 0.0e0
      WRITE (NOUT,99998) XBKPTS(1), XBKPTS(3), XBKPTS(5), XBKPTS(7),
     +  XBKPTS(11)
      DO 60 IT = 1, 5
         TOUT = 0.1e0*(2**IT)
         IFAIL = -1
*
         CALL D03PJF(NPDE,M,TS,TOUT,PDEDEF,BNDARY,U,NBKPTS,XBKPTS,NPOLY,
     +               NPTS,X,NCODE,ODEDEF,NXI,XI,NEQN,UVINIT,RTOL,ATOL,
     +               ITOL,NORM,LAOPT,ALGOPT,W,NW,IW,NIW,ITASK,ITRACE,
     +               IND,IFAIL)
*
*        Check against the exact solution
*
         CALL EXACT(TOUT,NBKPTS,XBKPTS,EXY)
         WRITE (NOUT,99997) TS
         WRITE (NOUT,99994) U(1), U(5), U(9), U(13), U(21), U(22)
         WRITE (NOUT,99993) EXY(1), EXY(3), EXY(5), EXY(7), EXY(11), TS
   60 CONTINUE
      WRITE (NOUT,99995) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (' Degree of Polynomial =',I4,'   No. of elements =',I4,/)
99998 FORMAT ('  X        ',5F9.3,/)
99997 FORMAT (' T = ',F6.3)
99996 FORMAT (//'  Simple coupled PDE using BDF ',/'  Accuracy require',
     +       'ment =',e10.3,' Number of points = ',I4,/)
99995 FORMAT (' Number of integration steps in time = ',I6,/' Number o',
     +       'f function evaluations = ',I6,/' Number of Jacobian eval',
     +       'uations =',I6,/' Number of iterations = ',I6,/)
99994 FORMAT (1X,'App.  sol.  ',F7.3,4F9.3,'  ODE sol. =',F8.3)
```

```
99993 FORMAT (1X,'Exact sol.  ',F7.3,4F9.3,'  ODE sol. =',F8.3,/)
      END
*
      SUBROUTINE UVINIT(NPDE,NPTS,X,U,NCODE,V)
*     Routine for PDE initial values (start time is 0.1D-6)
*     .. Scalar Arguments ..
      INTEGER           NCODE, NPDE, NPTS
*     .. Array Arguments ..
      real              U(NPDE,NPTS), V(*), X(NPTS)
*     .. Scalars in Common ..
      real              TS
*     .. Local Scalars ..
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Common blocks ..
      COMMON            /TAXIS/TS
*     .. Executable Statements ..
      V(1) = TS
      DO 20 I = 1, NPTS
         U(1,I) = EXP(TS*(1.0e0-X(I))) - 1.0e0
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE ODEDEF(NPDE,T,NCODE,V,VDOT,NXI,XI,UCP,UCPX,RCP,UCPT,
     +                  UCPTX,F,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IRES, NCODE, NPDE, NXI
*     .. Array Arguments ..
      real              F(*), RCP(NPDE,*), UCP(NPDE,*), UCPT(NPDE,*),
     +                  UCPTX(NPDE,*), UCPX(NPDE,*), V(*), VDOT(*),
     +                  XI(*)
*     .. Executable Statements ..
      IF (IRES.EQ.1) THEN
         F(1) = VDOT(1) - V(1)*UCP(1,1) - UCPX(1,1) - 1.0e0 - T
      ELSE IF (IRES.EQ.-1) THEN
         F(1) = VDOT(1)
      END IF
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,NPTL,U,DUDX,NCODE,V,VDOT,P,Q,R,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IRES, NCODE, NPDE, NPTL
*     .. Array Arguments ..
      real              DUDX(NPDE,NPTL), P(NPDE,NPDE,NPTL),
     +                  Q(NPDE,NPTL), R(NPDE,NPTL), U(NPDE,NPTL), V(*),
     +                  VDOT(*), X(NPTL)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      DO 20 I = 1, NPTL
         P(1,1,I) = V(1)*V(1)
         R(1,I) = DUDX(1,I)
         Q(1,I) = -X(I)*DUDX(1,I)*V(1)*VDOT(1)
```

```
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,U,UX,NCODE,V,VDOT,IBND,BETA,GAMMA,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              BETA(NPDE), GAMMA(NPDE), U(NPDE), UX(NPDE),
     +                  V(*), VDOT(*)
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Executable Statements ..
      BETA(1) = 1.0e0
      IF (IBND.EQ.0) THEN
         GAMMA(1) = -V(1)*EXP(T)
      ELSE
         GAMMA(1) = -V(1)*VDOT(1)
      END IF
      RETURN
      END
*
      SUBROUTINE EXACT(TIME,NPTS,X,U)
*     Exact solution (for comparison purposes)
*     .. Scalar Arguments ..
      real             TIME
      INTEGER          NPTS
*     .. Array Arguments ..
      real             U(NPTS), X(NPTS)
*     .. Local Scalars ..
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         U(I) = EXP(TIME*(1.0e0-X(I))) - 1.0e0
   20 CONTINUE
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
 D03PJF Example Program Results
 Degree of Polynomial =   2   No. of elements =  10



  Simple coupled PDE using BDF
  Accuracy requirement = 0.100E-03 Number of points =   21

  X             0.000     0.200     0.400     0.600     1.000
```

```
T =  0.200
App.  sol.    0.222    0.174    0.128    0.084    0.000  ODE sol. =    0.200
Exact sol.    0.221    0.174    0.127    0.083    0.000  ODE sol. =    0.200

T =  0.400
App.  sol.    0.492    0.378    0.272    0.174    0.000  ODE sol. =    0.400
Exact sol.    0.492    0.377    0.271    0.174    0.000  ODE sol. =    0.400

T =  0.800
App.  sol.    1.226    0.897    0.616    0.377    0.000  ODE sol. =    0.800
Exact sol.    1.226    0.896    0.616    0.377    0.000  ODE sol. =    0.800

T =  1.600
App.  sol.    3.954    2.597    1.612    0.896   -0.001  ODE sol. =    1.600
Exact sol.    3.953    2.597    1.612    0.896    0.000  ODE sol. =    1.600

T =  3.200
App.  sol.   23.534   11.931    5.815    2.590   -0.008  ODE sol. =    3.202
Exact sol.   23.533   11.936    5.821    2.597    0.000  ODE sol. =    3.200

Number of integration steps in time =     32
Number of function evaluations =    446
Number of Jacobian evaluations =    15
Number of iterations =    105
```

D03PKF – NAG Fortran Library Routine Document

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

D03PKF integrates a system of linear or nonlinear, first-order, time-dependent partial differential equations (PDEs) in one space variable, with scope for coupled ordinary differential equations (ODEs). The spatial discretisation is performed using the Keller box scheme and the method of lines is employed to reduce the PDEs to a system of ODEs. The resulting system is solved using a Backward Differentiation Formula (BDF) method or a Theta method (switching between Newton's method and functional iteration).

2 Specification

```
      SUBROUTINE D03PKF(NPDE, TS, TOUT, PDEDEF, BNDARY, U, NPTS, X,
     1                  NLEFT, NCODE, ODEDEF, NXI, XI, NEQN, RTOL, ATOL,
     2                  ITOL, NORM, LAOPT, ALGOPT, W, NW, IW, NIW,
     3                  ITASK, ITRACE, IND, IFAIL)
      INTEGER           NPDE, NPTS, NLEFT, NCODE, NXI, NEQN, ITOL, NW,
     1                  IW(NIW), NIW, ITASK, ITRACE, IND, IFAIL
      real              TS, TOUT, U(NEQN), X(NPTS), XI(*), RTOL(*),
     1                  ATOL(*), ALGOPT(30), W(NW)
      CHARACTER*1       NORM, LAOPT
      EXTERNAL          PDEDEF, BNDARY, ODEDEF
```

3 Description

D03PKF integrates the system of first-order PDEs and coupled ODEs

$$G_i(x,t,U,U_x,U_t,V,\dot{V}) = 0, \; i = 1,2,\ldots,\mathrm{NPDE}, \; a \leq x \leq b, \; t \geq t_0, \tag{1}$$

$$F_i(t,V,\dot{V},\xi,U^*,U_x^*,U_t^*) = 0, \; i = 1,2,\ldots,\mathrm{NCODE}. \tag{2}$$

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

$$G_i = \sum_{j=1}^{\mathrm{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} Q_{i,j}\dot{V}_j + R_i = 0, \; i = 1,2,\ldots,\mathrm{NPDE}, \tag{3}$$

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

The vector U is the set of PDE solution values

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

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

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

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

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

$$F = A - B\dot{V} - CU_t^*, \tag{4}$$

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

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, where $a = x_1$ and $b = x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$.

The PDE system which is defined by the functions G_i must be specified in the user-supplied subroutine PDEDEF.

The initial values of the functions $U(x,t)$ and $V(t)$ must be given at $t = t_0$.

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

The boundary conditions have the form:

$$G_i^L(x,t,U,U_t,V,\dot{V}) = 0 \text{ at } x = a, \; i = 1,2,\ldots,\mathrm{NLEFT}, \tag{5}$$

at the left-hand boundary, and

$$G_i^R(x,t,U,U_t,V,\dot{V}) = 0 \text{ at } x = b, \; i = 1,2,\ldots,\mathrm{NRIGHT}, \tag{6}$$

at the right-hand boundary.

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

$$\sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^L \dot{V}_j + S_i^L = 0, \; i = 1,2,\ldots,\mathrm{NLEFT}, \tag{7}$$

at the left-hand boundary, and

$$\sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^R \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^R \dot{V}_j + S_i^R = 0, \; i = 1,2,\ldots,\mathrm{NRIGHT}, \tag{8}$$

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

The boundary conditions must be specified in a subroutine BNDARY provided by the user.

The problem is subject to the following restrictions:

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

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

(iii) The evaluation of the function G_i is done approximately at the mid-points of the mesh X(i), for i = 1,2,...,NPTS, by calling the routine PDEDEF for each mid-point in turn. Any discontinuities in the function **must** therefore be at one or more of the mesh points $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$;

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

The algebraic-differential equation system which is defined by the functions F_i must be specified in the user-supplied subroutine ODEDEF. The user 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. In this method of lines approach the Keller box scheme [4] is applied to each PDE in the space variable only, resulting in a system of ODEs in time for the values of U_i at each mesh point. In total there are NPDE × NPTS + NCODE ODEs in time direction. This system is then integrated forwards in time using a Backward Differentiation Formula (BDF) or a Theta method.

4 References

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

[2] 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

[3] 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

[4] Keller H B (1970) A new difference scheme for parabolic problems *Numerical Solutions of Partial Differential Equations* (ed J Bramble) **2** Academic Press 327–350

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

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE $\geq$ 1.

2: TS — ***real*** *Input/Output*

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

Constraint: TS < TOUT.

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

3: TOUT — ***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 G_i which define the system of PDEs. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PKF.

Its specification is:

```
      SUBROUTINE PDEDEF(NPDE, T, X, U, UT, UX, NCODE, V, VDOT, RES, IRES)
      INTEGER           NPDE, NCODE, IRES
      real              T, X, U(NPDE), UT(NPDE), UX(NPDE), V(*),
     1                  VDOT(*), RES(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: UT(NPDE) — ***real*** array *Input*

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

6: UX(NPDE) — ***real*** array *Input*

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

7: NCODE — INTEGER *Input*

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

8: V(*) — ***real*** array *Input*

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

9: VDOT(*) — ***real*** array *Input*

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

10: RES(NPDE) — ***real*** array *Output*

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

$$G_i = \sum_{j=1}^{\mathrm{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} Q_{i,j}\dot{V}_j, \quad (9)$$

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

$$G_i = \sum_{j=1}^{\mathrm{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} Q_{i,j}\dot{V}_j + R_i, \quad (10)$$

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

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

11: IRES — INTEGER *Input/Output*

On entry: the form of G_i that must be returned in the array RES. If IRES $= -1$, then equation (9) above must be used. If IRES $= 1$, then equation (10) above must be used.

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 (IFAIL) set to 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 D03PKF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

PDEDEF must be declared as EXTERNAL in the (sub)program from which D03PKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: BNDARY — SUBROUTINE, supplied by the user. *External Procedure*

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

Its specification is:

```
      SUBROUTINE BNDARY(NPDE, T, IBND, NOBC, U, UT, NCODE, V, VDOT, RES,
     1                  IRES)
      INTEGER           NPDE, IBND, NOBC, NCODE, IRES
      real              T, U(NPDE), UT(NPDE), V(*), VDOT(*), RES(NOBC)
```

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: IBND — INTEGER *Input*

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

4: NOBC — INTEGER *Input*

On entry: NOBC specifies the number of boundary conditions at the boundary specified by IBND.

5: U(NPDE) — ***real*** array *Input*

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

6: UT(NPDE) — ***real*** array *Input*

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

7: NCODE — INTEGER *Input*

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

8: V(*) — ***real*** array *Input*

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

9: VDOT(*) — ***real*** array *Input*

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

Note: VDOT(i), for $i = 1,2,\ldots,$NCODE, may only appear linearly as in (7) and (8).

10: RES(NOBC) — ***real*** array *Output*

On exit: RES(i) must contain the ith component of G^L or G^R, depending on the value of IBND, for $i = 1,2,\ldots,$NOBC, where G^L is defined as

$$G_i^L = \sum_{j=1}^{\text{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{NCODE}} H_{i,j}^L \dot{V}_j, \tag{11}$$

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

$$G_i^L = \sum_{j=1}^{\text{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{NCODE}} H_{i,j}^L \dot{V}_j + S_i^L, \tag{12}$$

i.e., all terms in equation (7), and similarly for G_i^R.

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

11: IRES — INTEGER *Input/Output*

On entry: the form of G_i^L (or G_i^R) that must be returned in the array RES. If IRES = −1, then equation (11) above must be used. If IRES = 1, then equation (12) above must be used.

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 (IFAIL) set to 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 D03PKF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

BNDARY must be declared as EXTERNAL in the (sub)program from which D03PKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: U(NEQN) — ***real*** array *Input/Output*

On entry: the initial values of the dependent variables defined as follows:

U(NPDE × $(j-1)+i$) contain $U_i(x_j, t_0)$, for $i = 1, 2, \ldots$, NPDE; $j = 1, 2, \ldots$, NPTS and

U(NPTS × NPDE + i) contain $V_i(t_0)$, for $i = 1, 2, \ldots$, NCODE.

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

7: NPTS — INTEGER *Input*

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

Constraint: NPTS ≥ 3.

8: 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: X(1) < X(2) < ... < X(NPTS).

9: NLEFT — INTEGER *Input*

On entry: the number of boundary conditions at the left-hand mesh point X(1).

Constraint: 0 ≤ NLEFT ≤ NPDE.

10: NCODE — INTEGER *Input*

On entry: the number of coupled ODE components.

Constraint: NCODE ≥ 0.

11: ODEDEF — SUBROUTINE, supplied by the user. *External Procedure*

ODEDEF must evaluate the functions F, which define the system of ODEs, as given in (4). If the user wishes to compute the solution of a system of PDEs only (i.e., NCODE = 0), ODEDEF must be the dummy routine D03PEK. (D03PEK 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 ODEDEF(NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,
     1                  UCPT, F, IRES)
      INTEGER           NPDE, NCODE, NXI, IRES
      real              T, V(*), VDOT(*), XI(*), UCP(NPDE,*),
     1                  UCPX(NPDE,*), UCPT(NPDE,*), F(*)
```

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: NCODE — INTEGER *Input*

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

4: V(*) — ***real*** array *Input*

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

5: VDOT(*) — ***real*** array *Input*

On entry: 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(*) — ***real*** array *Input*

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

8: UCP(NPDE,*) — ***real*** array *Input*

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

9: UCPX(NPDE,*) — ***real*** array *Input*

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

10: UCPT(NPDE,*) — ***real*** array *Input*

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

11: F(*) — ***real*** array *Output*

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

$$F = -B\dot{V} - CU_t^*, \qquad (13)$$

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

$$F = A - B\dot{V} - CU_t^*, \qquad (14)$$

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

12: IRES — INTEGER *Input/Output*

On entry: the form of F that must be returned in the array F. If IRES $= -1$, then equation (13) above must be used. If IRES $= 1$, then equation (14) above must be used.

On exit: should usually remain unchanged. However, the user 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)program with the error indicator (IFAIL) set to 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 D03PKF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

ODEDEF must be declared as EXTERNAL in the (sub)program from which D03PKF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

12: NXI — INTEGER *Input*

Constraints:

NXI = 0 for NCODE = 0.
NXI ≥ 0 for NCODE > 0.

13: XI(*) — ***real*** array *Input*

Note: the dimension of the array XI must be at least max(1,NXI).

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

Constraint: X(1) ≤ XI(1) < XI(2) < ... < XI(NXI) ≤ X(NPTS).

14: NEQN — INTEGER *Input*

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

Constraint: NEQN = NPDE × NPTS + NCODE.

15: RTOL(*) — ***real*** 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) ≥ 0 for all relevant i.

16: ATOL(*) — ***real*** 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 for all relevant i.

Corresponding elements of ATOL and RTOL should not both be 0.0.

17: ITOL — INTEGER *Input*

On entry: a value to indicate the form of the local error test. ITOL indicates to D03PKF 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) × |U(i)| + ATOL(1)
2	scalar	vector	RTOL(1) × |U(i)| + ATOL(i)
3	vector	scalar	RTOL(i) × |U(i)| + ATOL(1)
4	vector	vector	RTOL(i) × |U(i)| + 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 parameter NORM, see below.

Constraint: $1 \leq$ ITOL ≤ 4.

18: NORM — CHARACTER*1 *Input*

On entry: the type of norm to be used. Two options are available:

'M' - maximum norm.
'A' - averaged L_2 norm.

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

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

while for the maximum norm

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

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

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

19: LAOPT — CHARACTER*1 *Input*

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

'F' - full matrix routines to be used;
'B' - banded matrix routines to be used;
'S' - sparse matrix routines to be used.

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

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

20: ALGOPT(30) — ***real*** array *Input*

On entry: ALGOPT may be set to control various options available in the integrator. If the user wishes 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 parameter 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 initialise 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, i.e., 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 < 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. ALGOPT(30) must be greater than zero, otherwise the default value is used. 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.

21: W(NW) — ***real*** array *Workspace*

22: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PKF is called. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',
$\quad$ NW $\geq$ NEQN $\times$ NEQN + NEQN + NWKRES + LENODE,
LAOPT = 'B',
$\quad$ NW $\geq$ (2 $\times$ ML + MU + 2) $\times$ NEQN + NWKRES + LENODE,
LAOPT = 'S',
$\quad$ NW $\geq$ 4 $\times$ NEQN + 11 $\times$ NEQN/2 + 1 + NWKRES + LENODE,

where ML and MU are the lower and upper half bandwidths, given by ML = NPDE + NLEFT − 1, MU = 2 × NPDE − NLEFT − 1 for problems involving PDEs only, and ML = MU = NEQN − 1, for coupled PDE/ODE problems.

NWKRES = NPDE × (6 × NXI + 3 × NPDE + NPTS + 15) + NXI + NCODE + 7 × NPTS + 2

when NCODE > 0, and NXI > 0.

NWKRES = NPDE × (3 × NPDE + NPTS + 21) + NCODE + 7 × NPTS + 3

when NCODE > 0, and NXI = 0.

NWKRES = NPDE × (3 × NPDE + NPTS + 21) + 7 × NPTS + 4

when NCODE = 0.

LENODE = (6 + int(ALGOPT(2))) × NEQN + 50, when the BDF method is used and,

LENODE = 9 × NEQN + 50, when the Theta method is used.

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

23: 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.

The rest of the array is used as workspace.

24: NIW — INTEGER *Input*

On entry: the dimension of the array IW. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',
  NIW $\geq$ 24,
LAOPT = 'B',
  NIW $\geq$ NEQN + 24,
LAOPT = 'S',
  NIW $\geq 25 \times$ NEQN + 24.

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

25: ITASK — INTEGER *Input*

On entry: the task tp 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.
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 parameter ALGOPT.
ITASK = 5
  take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter ALGOPT.

Constraint: $1 \leq$ ITASK ≤ 5.

26: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03PKF and the underlying ODE solver as follows:

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 = 1, 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.

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

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

Users are advised to set ITRACE = 0, unless they are experienced with the subchapter D02M-N of the NAG Fortran Library.

27: 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 D03PKF.

Constraint: $0 \le \text{IND} \le 1$.

On exit: IND = 1.

28: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, (TOUT − TS) is too small,
or ITASK $\neq$ 1, 2, 3, 4 or 5,
or at least one of the coupling points defined in array XI is outside the interval [X(1),X(NPTS)],
or NPTS < 3,
or NPDE < 1,
or NLEFT not in range 0 to NPDE,
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 incorrectly defined user mesh, i.e., $X(i) \geq X(i+1)$ for some $i = 1, 2, \ldots, \text{NPTS} - 1$,
or NW or NIW are too small,
or NCODE and NXI are incorrectly defined,
or IND = 1 on initial entry to D03PKF,
or an element of RTOL or ATOL < 0.0,
or corresponding elements of ATOL and RTOL are both 0.0,
or NEQN $\neq$ NPDE $\times$ NPTS + NCODE.

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. Incorrect positioning 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, BNDARY or ODEDEF, when the residual in the underlying ODE solver was being evaluated. Incorrect positioning of boundary conditions may also result in this error.

IFAIL = 5

In solving the ODE system, a singular Jacobian has been encountered. The user should check their problem formulation.

IFAIL = 6

When evaluating the residual in solving the ODE system, IRES was set to 2 in one of the user-supplied subroutines 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 the user-supplied routines, PDEDEF, BNDARY or ODEDEF, 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 ATOL and RTOL 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 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 advisory 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) has been taken.

IFAIL = 13

Some error weights w_i became zero during the time integration (see 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 succesful as far as t = TS.

IFAIL = 14

Not applicable.

IFAIL = 15

When using the sparse option, the value of NIW or NW was insufficient (more detailed information may be directed to the current error message unit).

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 accuracy parameters, ATOL and RTOL.

8 Further Comments

The Keller box scheme can be used to solve higher-order problems which have been reduced to first order by the introduction of new variables (see the example in Section 9 below). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite-difference scheme (D03PCF/D03PHF for example), but at the expense of increased CPU time due to the larger number of function evaluations required.

It should be noted that the Keller box scheme, in common with other central-difference schemes, may be unsuitable for some hyperbolic first-order problems such as the apparently simple linear advection equation $U_t + aU_x = 0$, where a is a constant, resulting in spurious oscillations due to the lack of dissipation. This type of problem requires a discretisation scheme with upwind weighting (D03PLF for example), or the addition of a second-order artificial dissipation term.

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

9 Example

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

$$(V_1)^2 \frac{\partial U_1}{\partial t} - xV_1\dot{V}_1U_2 - \frac{\partial U_2}{\partial x} = 0,$$

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

$$\dot{V}_1 - V_1U_1 - U_2 - 1 - t = 0,$$

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

$$U_2 = -V_1 \exp t,$$

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

$$U_2 = -V_1\dot{V}_1.$$

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

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

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

This problem is exactly the same as the D03PHF example problem, but reduced to first order by the introduction of a second PDE variable (as mentioned in Section 8).

9.1 Example 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.

```
*     D03PKF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, NCODE, NXI, NLEFT, NEQN, NIW, NWKRES,
     +                 LENODE, NW
      PARAMETER        (NPDE=2,NPTS=21,NCODE=1,NXI=1,NLEFT=1,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=24,
     +                 NWKRES=NPDE*(NPTS+6*NXI+3*NPDE+15)
     +                 +NCODE+NXI+7*NPTS+2,LENODE=11*NEQN+50,
     +                 NW=NEQN*NEQN+NEQN+NWKRES+LENODE)
*     .. Scalars in Common ..
      real             TS
*     .. Local Scalars ..
      real             TOUT
      INTEGER          I, IFAIL, IND, IT, ITASK, ITOL, ITRACE
      LOGICAL          THETA
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), EXY(NEQN), RTOL(1), U(NEQN),
     +                 W(NW), X(NPTS), XI(1)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PKF, EXACT, ODEDEF, PDEDEF, UVINIT
*     .. Common blocks ..
      COMMON           /TAXIS/TS
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PKF Example Program Results'
      ITRACE = 0
      ITOL = 1
      ATOL(1) = 0.1e-3
      RTOL(1) = ATOL(1)
      WRITE (NOUT,99997) ATOL, NPTS
*
*     Set spatial-mesh points
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
      XI(1) = 1.0e0
      NORM = 'A'
      LAOPT = 'F'
      IND = 0
      ITASK = 1
*
*     Set THETA to .TRUE. if the Theta integrator is required
*
      THETA = .FALSE.
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
      IF (THETA) THEN
```

```
         ALGOPT(1) = 2.0e0
      ELSE
         ALGOPT(1) = 0.0e0
      END IF
      ALGOPT(1) = 1.0e0
      ALGOPT(13) = 0.5e-2
*
*     Loop over output value of t
*
      TS = 1.0e-4
      TOUT = 0.0e0
      WRITE (NOUT,99999) X(1), X(5), X(9), X(13), X(21)
*
      CALL UVINIT(NPDE,NPTS,X,U,NCODE,NEQN)
*
      DO 60 IT = 1, 5
         TOUT = 0.1e0*(2**IT)
         IFAIL = -1
*
         CALL D03PKF(NPDE,TS,TOUT,PDEDEF,BNDARY,U,NPTS,X,NLEFT,NCODE,
     +               ODEDEF,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,
     +               ALGOPT,W,NW,IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
*        Check against the exact solution
*
         CALL EXACT(TOUT,NEQN,NPTS,X,EXY)
*
         WRITE (NOUT,99998) TS
         WRITE (NOUT,99995) U(1), U(9), U(17), U(25), U(41), U(43)
         WRITE (NOUT,99994) EXY(1), EXY(9), EXY(17), EXY(25), EXY(41),
     +     TS
   60 CONTINUE
      WRITE (NOUT,99996) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (' X       ',5F9.3,/)
99998 FORMAT (' T = ',F6.3)
99997 FORMAT (//'  Accuracy requirement =',e10.3,' Number of points = ',
     +       I3,/)
99996 FORMAT (' Number of integration steps in time = ',I6,/' Number o',
     +       'f function evaluations = ',I6,/' Number of Jacobian eval',
     +       'uations =',I6,/' Number of iterations = ',I6,/)
99995 FORMAT (1X,'App.  sol.  ',F7.3,4F9.3,'  ODE sol. =',F8.3)
99994 FORMAT (1X,'Exact sol.  ',F7.3,4F9.3,'  ODE sol. =',F8.3,/)
      END
*
      SUBROUTINE UVINIT(NPDE,NPTS,X,U,NCODE,NEQN)
*     Routine for PDE initial values
*     .. Scalar Arguments ..
      INTEGER           NCODE, NEQN, NPDE, NPTS
*     .. Array Arguments ..
      real              U(NEQN), X(NPTS)
*     .. Scalars in Common ..
      real              TS
*     .. Local Scalars ..
      INTEGER           I, K
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
```

```
*     .. Common blocks ..
      COMMON           /TAXIS/TS
*     .. Executable Statements ..
      K = 1
      DO 20 I = 1, NPTS
         U(K) = EXP(TS*(1.0e0-X(I))) - 1.0e0
         U(K+1) = -TS*EXP(TS*(1.0e0-X(I)))
         K = K + 2
   20 CONTINUE
      U(NEQN) = TS
      RETURN
      END
*
      SUBROUTINE ODEDEF(NPDE,T,NCODE,V,VDOT,NXI,XI,UCP,UCPX,UCPT,F,IRES)
*     .. Scalar Arguments ..
      real             T
      INTEGER          IRES, NCODE, NPDE, NXI
*     .. Array Arguments ..
      real             F(*), UCP(NPDE,*), UCPT(NPDE,*), UCPX(NPDE,*),
     +                 V(*), VDOT(*), XI(*)
*     .. Executable Statements ..
      IF (IRES.EQ.-1) THEN
         F(1) = VDOT(1)
      ELSE
         F(1) = VDOT(1) - V(1)*UCP(1,1) - UCP(2,1) - 1.0e0 - T
      END IF
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,UDOT,UX,NCODE,V,VDOT,RES,IRES)
*     .. Scalar Arguments ..
      real             T, X
      INTEGER          IRES, NCODE, NPDE
*     .. Array Arguments ..
      real             RES(NPDE), U(NPDE), UDOT(NPDE), UX(NPDE), V(*),
     +                 VDOT(*)
*     .. Executable Statements ..
      IF (IRES.EQ.-1) THEN
         RES(1) = V(1)*V(1)*UDOT(1) - X*U(2)*V(1)*VDOT(1)
         RES(2) = 0.0e0
      ELSE
         RES(1) = V(1)*V(1)*UDOT(1) - X*U(2)*V(1)*VDOT(1) - UX(2)
         RES(2) = U(2) - UX(1)
      END IF
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,IBND,NOBC,U,UDOT,NCODE,V,VDOT,RES,IRES)
*     .. Scalar Arguments ..
      real             T
      INTEGER          IBND, IRES, NCODE, NOBC, NPDE
*     .. Array Arguments ..
      real             RES(NOBC), U(NPDE), UDOT(NPDE), V(*), VDOT(*)
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         IF (IRES.EQ.-1) THEN
```

```
            RES(1) = 0.0e0
         ELSE
            RES(1) = U(2) + V(1)*EXP(T)
         END IF
      ELSE
         IF (IRES.EQ.-1) THEN
            RES(1) = V(1)*VDOT(1)
         ELSE
            RES(1) = U(2) + V(1)*VDOT(1)
         END IF
      END IF
      RETURN
      END
*
      SUBROUTINE EXACT(TIME,NEQN,NPTS,X,U)
*     Exact solution (for comparison purposes)
*     .. Scalar Arguments ..
      real              TIME
      INTEGER           NEQN, NPTS
*     .. Array Arguments ..
      real              U(NEQN), X(NPTS)
*     .. Local Scalars ..
      INTEGER           I, K
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Executable Statements ..
      K = 1
      DO 20 I = 1, NPTS
         U(K) = EXP(TIME*(1.0e0-X(I))) - 1.0e0
         K = K + 2
   20 CONTINUE
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
 D03PKF Example Program Results


  Accuracy requirement = 0.100E-03 Number of points =  21

  X             0.000    0.200    0.400    0.600    1.000

 T =  0.200
 App.  sol.     0.222    0.174    0.128    0.084    0.000  ODE sol. =    0.200
 Exact sol.     0.221    0.174    0.127    0.083    0.000  ODE sol. =    0.200

 T =  0.400
 App.  sol.     0.492    0.377    0.271    0.174    0.000  ODE sol. =    0.400
 Exact sol.     0.492    0.377    0.271    0.174    0.000  ODE sol. =    0.400

 T =  0.800
 App.  sol.     1.226    0.896    0.616    0.377    0.000  ODE sol. =    0.800
 Exact sol.     1.226    0.896    0.616    0.377    0.000  ODE sol. =    0.800
```

```
T =  1.600
App.  sol.    3.952    2.595    1.610    0.895   -0.001  ODE sol. =    1.600
Exact sol.    3.953    2.597    1.612    0.896    0.000  ODE sol. =    1.600

T =  3.200
App.  sol.   23.522   11.918    5.807    2.588   -0.004  ODE sol. =    3.197
Exact sol.   23.533   11.936    5.821    2.597    0.000  ODE sol. =    3.200

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

D03PLF – NAG Fortran Library Routine Document

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

D03PLF integrates a system of linear or nonlinear convection-diffusion equations in one space dimension, with optional source terms and scope for coupled ordinary differential equations (ODEs). 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 partial differential equations (PDEs) to a system of ODEs, and the resulting system is solved using a backward differentiation formula (BDF) method or a Theta method.

2 Specification

```
      SUBROUTINE D03PLF(NPDE, TS, TOUT, PDEDEF, NUMFLX, BNDARY, U, NPTS,
     1                  X, NCODE, ODEDEF, NXI, XI, NEQN, RTOL, ATOL,
     2                  ITOL, NORM, LAOPT, ALGOPT, W, NW, IW, NIW,
     3                  ITASK, ITRACE, IND, IFAIL)
      INTEGER           NPDE, NPTS, NCODE, NXI, NEQN, ITOL, NW, IW(NIW),
     1                  NIW, ITASK, ITRACE, IND, IFAIL
      real              TS, TOUT, U(NEQN), X(NPTS), XI(*), RTOL(*),
     1                  ATOL(*), ALGOPT(30), W(NW)
      CHARACTER*1       NORM, LAOPT
      EXTERNAL          PDEDEF, NUMFLX, BNDARY, ODEDEF
```

3 Description

D03PLF integrates the system of convection-diffusion equations in conservative form:

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

or the hyperbolic convection-only system:

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

for $i = 1,2,\ldots,\mathrm{NPDE}$, $a \leq x \leq b$, $t \geq t_0$, where the vector U is the set of PDE solution values

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

The optional coupled ODEs are of the general form

$$R_i(t,V,\dot{V},\xi,U^*,U_x^*,U_t^*) = 0, \quad i = 1,2,...,\mathrm{NCODE}, \tag{3}$$

where the vector V is the set of ODE solution values

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

$\dot{V}$ denotes its derivative with respect to time, and U_x is the spatial derivative of U.

In (1), $P_{i,j}$, F_i and C_i depend on x, t, U and V; D_i depends on x, t, U, U_x and V; and S_i depends on x, t, U, V and **linearly** on $\dot{V}$. Note that $P_{i,j}$, F_i, C_i and S_i must not depend on any space derivatives, and $P_{i,j}$, F_i, C_i and D_i must not depend on any 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.

In (3), 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 PDE spatial mesh points. U^*, U_x^* and U_t^* are the functions

U, U_x and U_t evaluated at these coupling points. Each R_i may depend only linearly on time derivatives. Hence (3) may be written more precisely as

$$R = L - M\dot{V} - NU_t^*, \tag{4}$$

where $R = [R_1, \ldots, R_{\mathrm{NCODE}}]^T$, L is a vector of length NCODE, M is an NCODE by NCODE matrix, N is an NCODE by ($n_\xi \times$ NPDE) matrix and the entries in L, M and N may depend on t, ξ, U^*, U_x^* and V. In practice the user only needs to supply a vector of information to define the ODEs and not the matrices L, M and N. (See Section 5 for the specification of the user-supplied procedure ODEDEF).

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, where $a = x_1$ and $b = x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$. The initial values of the functions $U(x,t)$ and $V(t)$ 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, D03PHF and D03PPF, 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 $x_{j-\frac{1}{2}} = (x_{j-1} + x_j)/2$ for $j = 2, 3, \ldots,$ NPTS. The left and right values are calculated by D03PLF from two adjacent mesh points using a standard upwind technique combined with a Van Leer slope-limiter (see [2]). The physically correct value for $\hat{F}_i$ is derived from the solution of the Riemann problem given by

$$\frac{\partial U_i}{\partial t} + \frac{\partial F_i}{\partial y} = 0, \tag{5}$$

where $y = x - x_{j-\frac{1}{2}}$, i.e., $y = 0$ corresponds to $x = x_{j-\frac{1}{2}}$, 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 [2] and [5]. Roe's scheme [4] 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

$$\hat{F} = \frac{1}{2}(F_L + F_R) - \frac{1}{2}\sum_{k=1}^{\mathrm{NPDE}} \alpha_k |\lambda_k| e_k, \tag{6}$$

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

$$U_R - U_L = \sum_{k=1}^{\mathrm{NPDE}} \alpha_k e_k. \tag{7}$$

An example is given in Section 9 and in the D03PFF documentation.

If the system is nonlinear, Roe's scheme requires that a linearized Jacobian is found (see [4]).

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 D03PLP may be used for PDEDEF (D03PLP is included in the NAG Fortran Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details). D03PLP 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 conditions 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 when using banded matrix algebra the fixed bandwidth means that only linear extrapolation is allowed, i.e., using information at just two interior points adjacent to the boundary). 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. Another method of supplying numerical boundary conditions involves the solution of the characteristic equations associated with the outgoing characteristics. Examples of both methods can be found in Section 9 and in the D03PFF documentation.

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

$$G_i^L(x,t,U,V,\dot{V}) = 0 \quad \text{at} \quad x = a, \quad i = 1,2,...,\text{NPDE}, \tag{8}$$

at the left-hand boundary, and

$$G_i^R(x,t,U,V,\dot{V}) = 0 \quad \text{at} \quad x = b, \quad i = 1,2,...,\text{NPDE}, \tag{9}$$

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 algebraic-differential equation system which is defined by the functions R_i must be specified in a subroutine ODEDEF supplied by the user. The user must also specify the coupling points ξ (if any) in the array XI.

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 S_i, for $i = 1,2,\ldots,\text{NPDE}$, with a similar restriction for G_i^L and G_i^R;

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

(iii) $t_0 < t_{out}$, 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 $\text{NPDE} \times \text{NPTS} + \text{NCODE}$ ODEs in the time direction. This system is then integrated forwards in time using a BDF or Theta method, optionally switching between Newton's method and functional iteration (see [5]).

For further details of the scheme, see [1] and the references therein.

4 References

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

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

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

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

[5] 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

[6] Sod G A (1978) A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws *J. Comput. Phys.* **27** 1–31

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. $P_{i,j}$ and C_i may depend on x, t, U and V; D_i may depend on x, t, U, U_x and V; and S_i may depend on x, t, U, V and linearly on $\dot{V}$. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PLF. The actual argument D03PLP may be used for PDEDEF for problems in the form (2) (D03PLP 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, NCODE, V, VDOT, P, C, D, S,
     1                  IRES)
      INTEGER           NPDE, NCODE, IRES
      real              T, X, U(NPDE), UX(NPDE), V(*), VDOT(*),
     1                  P(NPDE,NPDE), C(NPDE), 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: NCODE — INTEGER *Input*

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

7: V(*) — ***real*** array *Input*

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

8: VDOT(*) — ***real*** 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 S_j, for $j = 1,2,\ldots,$NPDE.

9: P(NPDE,NPDE) — ***real*** array *Output*

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

10: C(NPDE) — ***real*** array *Output*

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

11: D(NPDE) — ***real*** array *Output*

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

12: S(NPDE) — ***real*** array *Output*

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

13: 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 D03PLF 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 D03PLF 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 D03PLF.

Its specification is:

```
SUBROUTINE NUMFLX(NPDE, T, X, NCODE, V, ULEFT, URIGHT, FLUX, IRES)
INTEGER          NPDE, NCODE, IRES
real             T, X, V(*), 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: NCODE — INTEGER *Input*

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

5: V(*) — ***real*** array *Input*

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

6: ULEFT(NPDE) — ***real*** array *Input*

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

7: URIGHT(NPDE) — ***real*** array *Input*

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

8: FLUX(NPDE) — ***real*** array *Output*

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

9: 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 D03PLF 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 D03PLF 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 (8) and (9).

Its specification is:

```
      SUBROUTINE BNDARY(NPDE, NPTS, T, X, U, NCODE, V, VDOT, IBND, G,
     1                  IRES)
      INTEGER           NPDE, NPTS, NCODE, IBND, IRES
      real              T, X(NPTS), U(NPDE,NPTS), V(*), VDOT(*), 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,NPTS) — ***real*** array *Input*

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

Note: if banded matrix algebra is to be used then the functions G_i^L and G_i^R may depend on the value of $U_i(x, t)$ at the boundary point and the two adjacent points only.

6: NCODE — INTEGER *Input*

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

7: V(*) — ***real*** array *Input*

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

8: VDOT(*) — ***real*** 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 G_j^L and G_j^R, for $j = 1, 2, \ldots,$ NPDE.

9: 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 $x = b$.

10: G(NPDE) — ***real*** array *Output*

On exit: G(i) must contain the ith component of either G_i^L or G_i^R in (8) and (9), depending on the value of 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, 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 D03PLF 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 D03PLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: U(NEQN) — ***real*** array *Input/Output*

On entry: the initial values of the dependent variables defined as follows:

U(NPDE $\times$ $(j - 1) + i$) contain $U_i(x_j, t_0)$, for $i = 1, 2, \ldots,$ NPDE; $j = 1, 2, \ldots,$ NPTS and

U(NPTS × NPDE + k) contain $V_k(t_0)$, for $k = 1,2,\ldots,$NCODE.

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

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

10: NCODE — INTEGER *Input*

On entry: the number of coupled ODE components.

Constraint: NCODE ≥ 0.

11: ODEDEF — SUBROUTINE, supplied by the user. *External Procedure*

ODEDEF must evaluate the functions R, which define the system of ODEs, as given in (4). If the user wishes to compute the solution of a system of PDEs only (i.e., NCODE = 0), ODEDEF must be the dummy routine D03PEK. (D03PEK 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 ODEDEF(NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,
     1                  UCPT, R, IRES)
      INTEGER           NPDE, NCODE, NXI, IRES
      real              T, V(*), VDOT(*), XI(*), UCP(NPDE,*),
     1                  UCPX(NPDE,*), UCPT(NPDE,*), R(*)
```

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: NCODE — INTEGER *Input*

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

4: V(*) — ***real*** array *Input*

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

5: VDOT(*) — ***real*** array *Input*

On entry: 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(*) — ***real*** array *Input*

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

8: UCP(NPDE,*) — ***real*** array *Input*

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

9: UCPX(NPDE,*) — ***real*** array *Input*

On entry: UCPX(i,j) contains the value of $\partial U_i(x,t)/\partial x$ at the coupling point $x = \xi_j$, for $i = 1,2,\ldots,$NPDE; $j = 1,2,\ldots,$NXI.

10: UCPT(NPDE,*) — ***real*** array *Input*

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

11: R(*) — ***real*** array *Output*

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

$$R = L - M\dot{V} - NU_t^*, \tag{10}$$

or

$$R = -M\dot{V} - NU_t^*. \tag{11}$$

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

12: IRES — INTEGER *Input/Output*

On entry: the form of R that must be returned in the array R. If IRES = 1, then equation (10) above must be used. If IRES = −1, then the equation (11) above must be used.

On exit: should usually remain unchanged. However, the user 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)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 D03PLF returns to the calling (sub)program with the error indicator set to IFAIL = 4.

ODEDEF must be declared as EXTERNAL in the (sub)program from which D03PLF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

12: NXI — INTEGER *Input*

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

Constraints:

NXI = 0 if NCODE = 0,
NXI ≥ 0 if NCODE > 0.

13: XI(*) — ***real*** array *Input*

Note: the dimension of the array XI must be at least max(1,NXI).

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

Constraint: X(1) ≤ XI(1) < XI(2) < ... < XI(NXI) ≤ X(NPTS).

14: NEQN — INTEGER *Input*

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

Constraint: NEQN = NPDE × NPTS + NCODE.

15: RTOL(*) — ***real*** 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.

16: ATOL(*) — ***real*** 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.

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

17: ITOL — INTEGER *Input*

On entry: a value to indicate the form of the local error test. If e_i is the estimated local error for U(i), $i = 1, 2, \ldots,$ NEQN, and $\|\ \|$ denotes the norm, then the error test to be satisfied is $\|e_i\| < 1.0$. ITOL indicates to D03PLF whether to interpret either or both of RTOL and ATOL as a vector or scalar in the formation of the weights w_i used in the calculation of the norm (see the description of the parameter NORM below):

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	$\mathrm{RTOL}(1) \times \lvert\mathrm{U}(i)\rvert + \mathrm{ATOL}(1)$
2	scalar	vector	$\mathrm{RTOL}(1) \times \lvert\mathrm{U}(i)\rvert + \mathrm{ATOL}(i)$
3	vector	scalar	$\mathrm{RTOL}(i) \times \lvert\mathrm{U}(i)\rvert + \mathrm{ATOL}(1)$
4	vector	vector	$\mathrm{RTOL}(i) \times \lvert\mathrm{U}(i)\rvert + \mathrm{ATOL}(i)$

Constraint: $1 \leq$ ITOL ≤ 4.

18: NORM — CHARACTER*1 *Input*

On entry: the type of norm to be used. Two options are available:

'1' – averaged L_1 norm.

'2' – averaged L_2 norm.

If $\mathrm{U}_{\mathrm{norm}}$ denotes the norm of the vector U of length NEQN, then for the averaged L_1 norm

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

and for the averaged L_2 norm

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

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

Constraint: NORM = '1' or '2'.

19: LAOPT — CHARACTER*1 *Input*

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

- 'F' – full matrix routines to be used;
- 'B' – banded matrix routines to be used;
- 'S' – sparse matrix routines to be used.

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

Note: the user is recommended to use the banded option when no coupled ODEs are present (NCODE = 0). Also, the banded option should not be used if the boundary conditions involve solution components at points other than the boundary and the immediately adjacent two points.

20: ALGOPT(30) — ***real*** array *Input*

On entry: ALGOPT may be set to control various options available in the integrator. If the user wishes 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 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 $V_i(t)$ 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 parameter 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 initialise 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, i.e. 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 < ALGOPT(29) < 1.0, with smaller values biasing the algorithm towards maintaining sparsity at the expense of numerical stability. If ALGOPT(29) lies outside the 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 the relative pivot threshold during subsequent Jacobian decompositions (see ALGOPT(29)) below which an internal error is invoked. ALGOPT(30) must be greater than zero, otherwise the default value is used. 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 matrix is found to be numerically singular (see ALGOPT(29)).

The default value is ALGOPT(30) = 0.0001.

21: W(NW) — ***real*** array *Workspace*

22: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PLF is called. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',

$$NW \geq NEQN \times NEQN + NEQN + NWKRES + LENODE,$$

LAOPT = 'B',

$$NW \geq (3 \times MLU+1) \times NEQN + NWKRES + LENODE,$$

LAOPT = 'S',

$$NW \geq 4 \times NEQN + 11 \times NEQN/2 + 1 + NWKRES + LENODE.$$

Where MLU = the lower or upper half bandwidths, and

MLU = 3 × NPDE−1, for PDE problems only, and,

MLU = NEQN−1, for coupled PDE/ODE problems.

NWKRES = NPDE × (2×NPTS+6×NXI+3×NPDE+26) + NXI + NCODE + 7 × NPTS+2

when NCODE > 0, and NXI > 0;

NWKRES = NPDE × (2×NPTS+3×NPDE+32) + NCODE + 7 × NPTS+3

when NCODE > 0, and NXI = 0;

NWKRES = NPDE × (2×NPTS+3×NPDE+32) + 7 × NPTS+4

when NCODE = 0.

LENODE = (6+int(ALGOPT(2))) × NEQN + 50, when the BDF method is used and,

LENODE = 9 × NEQN + 50, when the Theta method is used.

Note. When LAOPT = 'S', the value of NW may be too small when supplied to the integrator. An estimate of the minimum size of NW is printed on the current error message unit if ITRACE > 0 and the routine returns with IFAIL = 15.

23: 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 BDF method last used in the time integration, if applicable. When the Theta method is used IW(4) contains no useful information.

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.

24: NIW — INTEGER *Input*

On entry: the dimension of the array IW. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',

NIW ≥ 24,

LAOPT = 'B',

NIW ≥ NEQN + 24,

LAOPT = 'S',

NIW ≥ 25 × NEQN + 24.

Note. When LAOPT = 'S', the value of NIW may be too small when supplied to the integrator. An estimate of the minimum size of NIW is printed on the current error message unit if ITRACE > 0 and the routine returns with IFAIL = 15.

25: 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.

ITASK = 4

normal computation of output values U at t = TOUT but without overshooting $t = t_{\mathrm{crit}}$ where t_{crit} is described under the parameter ALGOPT.

ITASK = 5

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

Constraint: 1 ≤ ITASK ≤ 5.

26: ITRACE — INTEGER *Input*

stop at first internal integration point at or beyond t

On entry: the level of trace information required from D03PLF 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 subchapter D02M-N of the NAG Fortran Library.

27: 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 D03PLF.

Constraint: $0 \leq$ IND ≤ 1.

On exit: IND = 1.

28: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TS $\geq$ TOUT,

or TOUT $-$ TS is too small,

or ITASK $\neq$ 1, 2, 3, 4 or 5,

or at least one of the coupling points defined in array XI is outside the interval [X(1),X(NPTS)],

or the coupling points are not in strictly increasing order,

or NPTS < 3,

or NPDE < 1,

or LAOPT $\neq$ 'F' , 'B' or 'S',

or ITOL $\neq$ 1, 2, 3 or 4,

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 NCODE and NXI are incorrectly defined,

or IND = 1 on initial entry to D03PLF,

or NEQN $\neq$ NPDE $\times$ NPTS + NCODE, or an element of RTOL or ATOL < 0.0,

or corresponding elements of RTOL and ATOL are both 0.0,

or NORM ≠ '1' or '2'.

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. 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, BNDARY or ODEDEF, 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, 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 the user-supplied routines, PDEDEF, NUMFLX, BNDARY or ODEDEF, 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 ATOL and RTOL 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 ≠ 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 advisory message unit when ITRACE ≥ 1). 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) has been taken.

IFAIL = 13

Some error weights w_i became zero during the time integration (see 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

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.

IFAIL = 15

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

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 accuracy parameters, ATOL and RTOL.

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 ALGOPT(13). 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 [1].

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

9 Example

For this routine two examples are presented, in Section 9.1 and Section 9.2. In the example programs distributed to sites, there is a single example program for D03PLF, with a main program:

```
*     D03PLF Example Program Text
*     Mark 18 Revised.  NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PLF 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 Section 9.1.1 and Section 9.2.1 respectively.

9.1 Example 1

This example is a simple first-order system with coupled ODEs arising from the use of the characteristic equations for the numerical boundary conditions.

The PDEs are

$$\frac{\partial U_1}{\partial t} + \frac{\partial U_1}{\partial x} + 2\frac{\partial U_2}{\partial x} = 0,$$
$$\frac{\partial U_2}{\partial t} + 2\frac{\partial U_1}{\partial x} + \frac{\partial U_2}{\partial x} = 0,$$

for $x \in [0,1]$ and $t \geq 0$.

The PDEs have an exact solution given by

$$U_1(x,t) = f(x-3t) + g(x+t),\ U_2(x,t) = f(x-3t) - g(x+t),$$

where $f(z) = \exp(\pi z)\sin(2\pi z)$, $g(z) = \exp(-2\pi z)\cos(2\pi z)$.

The initial conditions are given by the exact solution.

The characteristic variables are $W_1 = U_1 - U_2$ and $W_2 = U_1 + U_2$, corresponding to the characteristics given by $dx/dt = -1$ and $dx/dt = 3$ respectively. Hence we require a physical boundary condition for W_2 at the left-hand boundary and for W_1 at the right-hand boundary (corresponding to the incoming characteristics), and a numerical boundary condition for W_1 at the left-hand boundary and for W_2 at the right-hand boundary (outgoing characteristics).

The physical boundary conditions are obtained from the exact solution, and the numerical boundary conditions are supplied in the form of the characteristic equations for the outgoing characteristics, that is

$$\frac{\partial W_1}{\partial t} - \frac{\partial W_1}{\partial x} = 0$$

at the left-hand boundary, and

$$\frac{\partial W_2}{\partial t} + 3\frac{\partial W_2}{\partial x} = 0$$

at the right-hand boundary.

In order to specify these boundary conditions, two ODE variables V_1 and V_2 are introduced, defined by

$$V_1(t) = W_1(0,t) = U_1(0,t) - U_2(0,t),$$
$$V_2(t) = W_2(1,t) = U_1(1,t) + U_2(1,t).$$

The coupling points are therefore at $x = 0$ and $x = 1$.

The numerical boundary conditions are now

$$\dot{V}_1 - \frac{\partial W_1}{\partial x} = 0$$

at the left-hand boundary, and

$$\dot{V}_2 + 3\frac{\partial W_2}{\partial x} = 0$$

at the right-hand boundary.

The spatial derivatives are evaluated at the appropriate boundary points in the BNDARY subroutine using one-sided differences (into the domain and therefore consistent with the characteristic directions).

The numerical flux is calculated using Roe's approximate Riemann solver (see Section 3 for details), giving

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

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, NCODE, NXI, NEQN, NIW, NW, OUTPTS
      PARAMETER        (NPDE=2,NPTS=141,NCODE=2,NXI=2,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=15700,NW=11000,OUTPTS=8)
*     .. Scalars in Common ..
      real             P
*     .. Local Scalars ..
      real             TOUT, TS, XX
      INTEGER          I, IFAIL, II, IND, ITASK, ITOL, ITRACE, J, NOP
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NEQN),
     +                 UE(NPDE,OUTPTS), UOUT(NPDE,OUTPTS), W(NW),
     +                 X(NPTS), XI(NXI), XOUT(OUTPTS)
      INTEGER          IW(NIW)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         BNDRY1, D03PLF, EXACT, NMFLX1, ODEDEF, PDEDEF
*     .. 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
      ITOL = 1
      NORM = '1'
      ATOL(1) = 0.1e-4
      RTOL(1) = 0.25e-3
      WRITE (NOUT,99995) NPTS, ATOL, RTOL
*
*     Initialise mesh ..
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
      XI(1) = 0.0e0
      XI(2) = 1.0e0
*
*     Set initial values ..
      TS = 0.0e0
      CALL EXACT(TS,U,NPDE,X,NPTS)
      U(NEQN-1) = U(1) - U(2)
      U(NEQN) = U(NEQN-2) + U(NEQN-3)
*
```

```
      LAOPT = 'S'
      IND = 0
      ITASK = 1
*
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
*     Theta integration
      ALGOPT(1) = 1.0e0
*     Sparse matrix algebra parameters
      ALGOPT(29) = 0.1e0
      ALGOPT(30) = 1.1e0
*
      TOUT = 0.5e0
      IFAIL = 0
*
      CALL D03PLF(NPDE,TS,TOUT,PDEDEF,NMFLX1,BNDRY1,U,NPTS,X,NCODE,
     +            ODEDEF,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,ALGOPT,W,
     +            NW,IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
*     Set output points ..
      NOP = 0
      DO 60 I = 1, NPTS, 20
         NOP = NOP + 1
         XOUT(NOP) = X(I)
   60 CONTINUE
*
      WRITE (NOUT,99996) TS
      WRITE (NOUT,99999)
*
      DO 80 I = 1, NOP
         II = 1 + 20*(I-1)
         J = NPDE*(II-1)
         UOUT(1,I) = U(J+1)
         UOUT(2,I) = U(J+2)
   80 CONTINUE
*
*     Check against exact solution ..
      CALL EXACT(TOUT,UE,NPDE,XOUT,NOP)
      DO 100 I = 1, NOP
         WRITE (NOUT,99998) XOUT(I), UOUT(1,I), UE(1,I), UOUT(2,I),
     +     UE(2,I)
  100 CONTINUE
      WRITE (NOUT,99997)
*
      WRITE (NOUT,99994) IW(1), IW(2), IW(3), IW(5)
      RETURN
*
99999 FORMAT (8X,'X',8X,'Approx U1',3X,'Exact U1',4X,'Approx U2',3X,
     +       'Exact U2',/)
99998 FORMAT (5(3X,F9.4))
99997 FORMAT (1X,e10.4,4(2X,e12.4))
99996 FORMAT (' T = ',F6.3)
99995 FORMAT (/' NPTS = ',I4,' ATOL = ',e10.3,' RTOL = ',e10.3,/)
99994 FORMAT (' Number of integration steps in time = ',I6,/' Number ',
     +       'of function evaluations = ',I6,/' Number of Jacobian ',
     +       'evaluations =',I6,/' Number of iterations = ',I6,/)
      END
```

```
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,UX,NCODE,V,VDOT,P,C,D,S,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              C(NPDE), D(NPDE), P(NPDE,NPDE), S(NPDE),
     +                  U(NPDE), UX(NPDE), V(*), VDOT(*)
*     .. Local Scalars ..
      INTEGER           I, J
*     .. Executable Statements ..
      DO 40 I = 1, NPDE
         C(I) = 1.0e0
         D(I) = 0.0e0
         S(I) = 0.0e0
         DO 20 J = 1, NPDE
            IF (I.EQ.J) THEN
               P(I,J) = 1.0e0
            ELSE
               P(I,J) = 0.0e0
            END IF
   20    CONTINUE
   40 CONTINUE
      RETURN
      END
*
      SUBROUTINE BNDRY1(NPDE,NPTS,T,X,U,NCODE,V,VDOT,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,NPTS), V(*), VDOT(*), X(NPTS)
*     .. Local Scalars ..
      real              DUDX
*     .. 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)
         G(1) = U(1,1) + U(2,1) - UE(1,1) - UE(2,1)
         DUDX = (U(1,2)-U(2,2)-U(1,1)+U(2,1))/(X(2)-X(1))
         G(2) = VDOT(1) - DUDX
      ELSE
         CALL EXACT(T,UE,NPDE,X(NPTS),1)
         G(1) = U(1,NPTS) - U(2,NPTS) - UE(1,1) + UE(2,1)
         DUDX = (U(1,NPTS)+U(2,NPTS)-U(1,NPTS-1)-U(2,NPTS-1))/(X(NPTS)
     +          -X(NPTS-1))
         G(2) = VDOT(2) + 3.0e0*DUDX
      END IF
      RETURN
      END
*
      SUBROUTINE NMFLX1(NPDE,T,X,NCODE,V,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
```

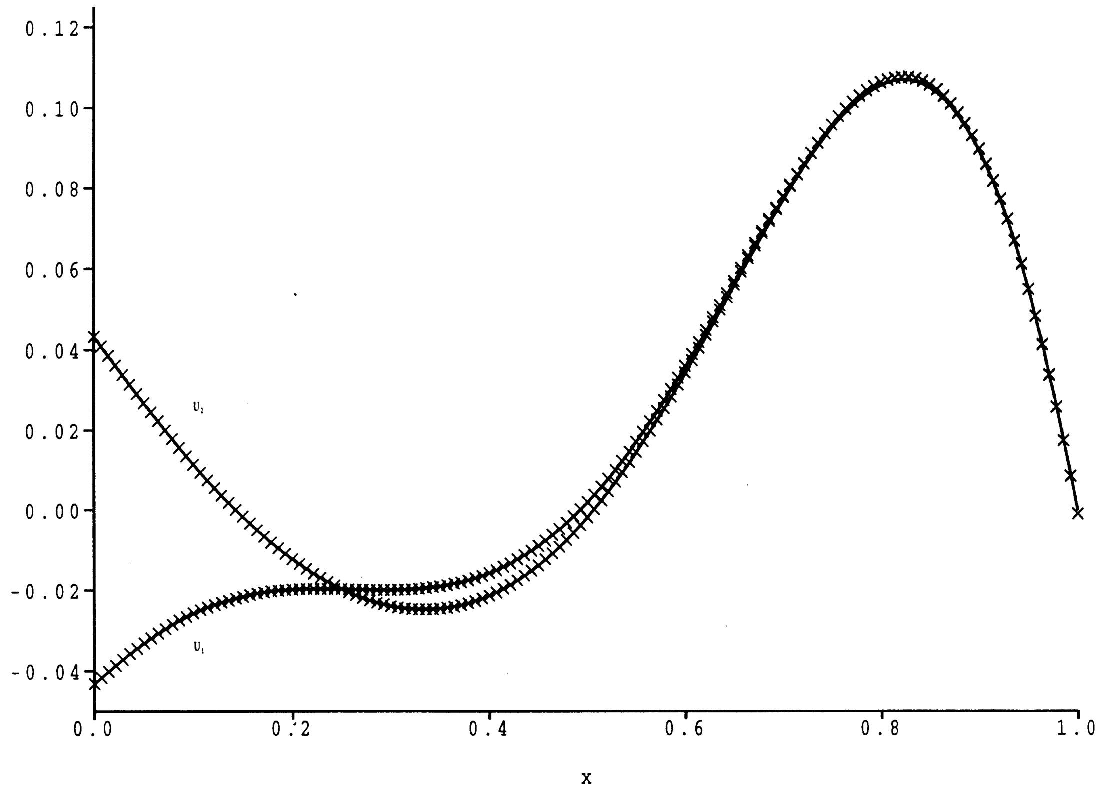
```
*     .. Array Arguments ..
      real              FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE), V(*)
*     .. Executable Statements ..
      FLUX(1) = 0.5e0*(3.0e0*ULEFT(1)-URIGHT(1)+3.0e0*ULEFT(2)+URIGHT(2)
     +          )
      FLUX(2) = 0.5e0*(3.0e0*ULEFT(1)+URIGHT(1)+3.0e0*ULEFT(2)-URIGHT(2)
     +          )
      RETURN
      END
*
      SUBROUTINE ODEDEF(NPDE,T,NCODE,V,VDOT,NXI,XI,UCP,UCPX,UCPT,F,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IRES, NCODE, NPDE, NXI
*     .. Array Arguments ..
      real              F(*), UCP(NPDE,*), UCPT(NPDE,*), UCPX(NPDE,*),
     +                  V(*), VDOT(*), XI(*)
*     .. Executable Statements ..
      IF (IRES.EQ.-1) THEN
         F(1) = 0.0e0
         F(2) = 0.0e0
      ELSE
         F(1) = V(1) - UCP(1,1) + UCP(2,1)
         F(2) = V(2) - UCP(1,2) - UCP(2,2)
      END IF
      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              F, G
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         COS, EXP, SIN
*     .. Common blocks ..
      COMMON            /PI/P
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         F = EXP(P*(X(I)-3.0e0*T))*SIN(2.0e0*P*(X(I)-3.0e0*T))
         G = EXP(-2.0e0*P*(X(I)+T))*COS(2.0e0*P*(X(I)+T))
         U(1,I) = F + G
         U(2,I) = F - G
   20 CONTINUE
      RETURN
      END
```

9.1.2 Program Data

None.

9.1.3 Program Results

```
D03PLF Example Program Results


Example 1


NPTS =  141 ATOL =  0.100E-04 RTOL =  0.250E-03

T =  0.500
       X        Approx U1   Exact U1    Approx U2   Exact U2

     0.0000      -0.0432     -0.0432      0.0432      0.0432
     0.1429      -0.0221     -0.0220      0.0001      0.0000
     0.2857      -0.0200     -0.0199     -0.0231     -0.0231
     0.4286      -0.0123     -0.0123     -0.0176     -0.0176
     0.5714       0.0248      0.0245      0.0226      0.0224
     0.7143       0.0834      0.0827      0.0831      0.0825
     0.8571       0.1043      0.1036      0.1045      0.1039
     1.0000      -0.0010     -0.0001     -0.0008      0.0001

Number of integration steps in time =     157
Number of function evaluations =   1166
Number of Jacobian evaluations =    17
Number of iterations =    415
```



9.2 Example 2

This example is the standard shock-tube test problem proposed by Sod [6] for the Euler equations of gas dynamics. The problem models the flow of a gas in a long tube following the sudden breakdown of a diaphragm separating two initial gas states at different pressures and densities. There is an exact solution to this problem which is not included explicitly as the calculation is quite lengthy. The PDEs are

$$\frac{\partial \rho}{\partial t} + \frac{\partial m}{\partial x} = 0,$$

$$\frac{\partial m}{\partial t} + \frac{\partial}{\partial x}\left(\frac{m^2}{\rho} + (\gamma - 1)\left(e - \frac{m^2}{2\rho}\right)\right) = 0,$$

$$\frac{\partial e}{\partial t} + \frac{\partial}{\partial x}\left(\frac{me}{\rho} + \frac{m}{\rho}(\gamma - 1)\left(e - \frac{m^2}{2\rho}\right)\right) = 0,$$

where ρ is the density; m is the momentum, such that $m = \rho u$, where u is the velocity; e is the specific energy; and γ is the (constant) ratio of specific heats. The pressure p is given by

$$p = (\gamma - 1)\left(e - \frac{\rho u^2}{2}\right).$$

The solution domain is $0 \leq x \leq 1$ for $0 < t \leq 0.2$, with the initial discontinuity at $x = 0.5$, and initial conditions

$$\rho(x,0) = 1, \quad m(x,0) = 0, \quad e(x,0) = 2.5, \quad \text{for } x < 0.5,$$
$$\rho(x,0) = 0.125, \quad m(x,0) = 0, \quad e(x,0) = 0.25, \quad \text{for } x > 0.5.$$

The solution is uniform and constant at both boundaries for the spatial domain and time of integration stated, and hence the physical and numerical boundary conditions are indistinguishable and are both given by the initial conditions above. The evaluation of the numerical flux for the Euler equations is not trivial; the Roe algorithm given in Section 3 can not be used directly as the Jacobian is nonlinear. However, an algorithm is available using the parameter-vector method (see [4]), and this is provided in the utility routine D03PUF. An alternative Approxiate Riemann Solver using Osher's scheme is provided in D03PVF. Either D03PUF or D03PVF can be called from the user-supplied NUMFLX subroutine.

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          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          NPDE, NPTS, NCODE, NXI, NEQN, NIW, NWKRES,
     +                 LENODE, MLU, NW
      PARAMETER        (NPDE=3,NPTS=141,NCODE=0,NXI=0,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=NEQN+24,
     +                 NWKRES=NPDE*(2*NPTS+3*NPDE+32)+7*NPTS+4,
     +                 LENODE=9*NEQN+50,MLU=3*NPDE-1,NW=(3*MLU+1)
     +                 *NEQN+NWKRES+LENODE)
*     .. Scalars in Common ..
      real             ELO, ERO, GAMMA, RLO, RRO
*     .. Local Scalars ..
      real             D, P, TOUT, TS, V
      INTEGER          I, IFAIL, IND, IT, ITASK, ITOL, ITRACE, K
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NPDE,NPTS),
```

```
     +                 UE(3,8), W(NW), X(NPTS), XI(1)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDRY2, D03PEK, D03PLF, D03PLP, NMFLX2, UVINIT
*     .. Common blocks ..
      COMMON           /INIT/ELO, ERO, RLO, RRO
      COMMON           /PARAMS/GAMMA
*     .. Save statement ..
      SAVE             /PARAMS/, /INIT/
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Example 2'
      WRITE (NOUT,*)
*     Skip heading in data file
      READ (NIN,*)
*
*     Problem parameters
      GAMMA = 1.4e0
      ELO = 2.5e0
      ERO = 0.25e0
      RLO = 1.0e0
      RRO = 0.125e0
      ITRACE = 0
      ITOL = 1
      NORM = '2'
      ATOL(1) = 0.5e-2
      RTOL(1) = 0.5e-3
      WRITE (NOUT,99994) GAMMA, ELO, ERO, RLO, RRO
      WRITE (NOUT,99996) NPTS, ATOL, RTOL
*
*     Initialise mesh
*
      DO 20 I = 1, NPTS
         X(I) = 1.0e0*(I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
*     Initial values of variables
      CALL UVINIT(NPDE,NPTS,X,U)
*
      XI(1) = 0.0e0
      LAOPT = 'B'
      IND = 0
      ITASK = 1
*
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
*     Theta integration
      ALGOPT(1) = 2.0e0
      ALGOPT(6) = 2.0e0
      ALGOPT(7) = 2.0e0
*     Max. time step
      ALGOPT(13) = 0.5e-2
*
      TS = 0.0e0
      WRITE (NOUT,99998)
      DO 100 IT = 1, 2
```

```
         TOUT = IT*0.1e0
         IFAIL = 0
*
         CALL D03PLF(NPDE,TS,TOUT,D03PLP,NMFLX2,BNDRY2,U,NPTS,X,NCODE,
     +               D03PEK,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,
     +               ALGOPT,W,NW,IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
         WRITE (NOUT,99997) TS
*
*        Read exact data at output points ..
         READ (NIN,*)
         DO 60 I = 1, 8
            READ (NIN,99999) UE(1,I), UE(2,I), UE(3,I)
   60    CONTINUE
*
*        Calculate density, velocity and pressure ..
         K = 0
         DO 80 I = 29, NPTS - 14, 14
            D = U(1,I)
            V = U(2,I)/D
            P = D*(GAMMA-1.0e0)*(U(3,I)/D-0.5e0*V**2)
            K = K + 1
            WRITE (NOUT,99993) X(I), D, UE(1,K), V, UE(2,K), P, UE(3,K)
   80    CONTINUE
  100 CONTINUE
*
      WRITE (NOUT,99995) IW(1), IW(2), IW(3), IW(5)
      RETURN
*
99999 FORMAT (3(1X,F6.4))
99998 FORMAT (4X,'X',4X,'APPROX D',1X,'EXACT D',2X,'APPROX V',1X,'EXAC',
     +       'T V',2X,'APPROX P',1X,'EXACT P')
99997 FORMAT (/' T = ',F6.3,/)
99996 FORMAT (/' NPTS = ',I4,' ATOL = ',e10.3,' RTOL = ',e10.3,/)
99995 FORMAT (/' Number of integration steps in time = ',I6,/' Number ',
     +       'of function evaluations = ',I6,/' Number of Jacobian ',
     +       'evaluations =',I6,/' Number of iterations = ',I6,/)
99994 FORMAT (/' GAMMA =',F6.3,'  ELO =',F6.3,'  ERO =',F6.3,'  RLO =',
     +       F6.3,'  RRO =',F6.3)
99993 FORMAT (1X,F6.4,6(3X,F6.4))
      END
*
      SUBROUTINE UVINIT(NPDE,NPTS,X,U)
*     .. Scalar Arguments ..
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              U(NPDE,NPTS), X(NPTS)
*     .. Scalars in Common ..
      real              ELO, ERO, RLO, RRO
*     .. Local Scalars ..
      INTEGER           I
*     .. Common blocks ..
      COMMON            /INIT/ELO, ERO, RLO, RRO
*     .. Save statement ..
      SAVE              /INIT/
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         IF (X(I).LT.0.5e0) THEN
```

```
            U(1,I) = RL0
            U(2,I) = 0.0e0
            U(3,I) = EL0
         ELSE IF (X(I).EQ.0.5e0) THEN
            U(1,I) = 0.5e0*(RL0+RR0)
            U(2,I) = 0.0e0
            U(3,I) = 0.5e0*(EL0+ER0)
         ELSE
            U(1,I) = RR0
            U(2,I) = 0.0e0
            U(3,I) = ER0
         END IF
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE BNDRY2(NPDE,NPTS,T,X,U,NCODE,V,VDOT,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,NPTS), V(*), VDOT(*), X(NPTS)
*     .. Scalars in Common ..
      real              EL0, ER0, RL0, RR0
*     .. Common blocks ..
      COMMON            /INIT/EL0, ER0, RL0, RR0
*     .. Save statement ..
      SAVE              /INIT/
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         G(1) = U(1,1) - RL0
         G(2) = U(2,1)
         G(3) = U(3,1) - EL0
      ELSE
         G(1) = U(1,NPTS) - RR0
         G(2) = U(2,NPTS)
         G(3) = U(3,NPTS) - ER0
      END IF
      RETURN
      END
*
      SUBROUTINE NMFLX2(NPDE,T,X,NCODE,V,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE), V(*)
*     .. Scalars in Common ..
      real              GAMMA
*     .. Local Scalars ..
      INTEGER           IFAIL
      CHARACTER         PATH, SOLVER
*     .. External Subroutines ..
      EXTERNAL          D03PUF, D03PVF
*     .. Common blocks ..
      COMMON            /PARAMS/GAMMA
*     .. Save statement ..
      SAVE              /PARAMS/
```

```
*     .. Executable Statements ..
      IFAIL = 0
      SOLVER = 'R'
      IF (SOLVER.EQ.'R') THEN
*       ROE SCHEME ..
         CALL D03PUF(ULEFT,URIGHT,GAMMA,FLUX,IFAIL)
      ELSE
*       OSHER SCHEME ..
         PATH = 'P'
         CALL D03PVF(ULEFT,URIGHT,GAMMA,PATH,FLUX,IFAIL)
      END IF
      RETURN
      END
```

9.2.2 Program Data

```
D03PLF Example Program Data
 D, V, P at selected output pts. For T = 0.1:
 1.0000 0.0000 1.0000
 1.0000 0.0000 1.0000
 0.8775 0.1527 0.8327
 0.4263 0.9275 0.3031
 0.2656 0.9275 0.3031
 0.1250 0.0000 0.1000
 0.1250 0.0000 0.1000
 0.1250 0.0000 0.1000
 For T = 0.2:
 1.0000 0.0000 1.0000
 0.8775 0.1527 0.8327
 0.6029 0.5693 0.4925
 0.4263 0.9275 0.3031
 0.4263 0.9275 0.3031
 0.2656 0.9275 0.3031
 0.2656 0.9275 0.3031
 0.1250 0.0000 0.1000
```

9.2.3 Program Results

```
 D03PLF Example Program Results



 Example 2


 GAMMA = 1.400  ELO = 2.500  ERO = 0.250  RLO = 1.000  RRO = 0.125

 NPTS =  141 ATOL =  0.500E-02 RTOL =  0.500E-03

    X     APPROX D EXACT D  APPROX V EXACT V  APPROX P EXACT P

 T =  0.100

 0.2000   1.0000   1.0000   0.0000   0.0000   1.0000   1.0000
 0.3000   1.0000   1.0000   0.0000   0.0000   1.0000   1.0000
 0.4000   0.8668   0.8775   0.1665   0.1527   0.8188   0.8327
 0.5000   0.4299   0.4263   0.9182   0.9275   0.3071   0.3031
```

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

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

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

The initial ($t = t_0$) values of the functions $U(x,t)$ and $V(t)$ must be specified in a subroutine UVINIT supplied by the user. 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 $a \leq x \leq b$, and not just the initial mesh points.

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

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

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

The boundary conditions must be specified in a subroutine BNDARY provided by the user. 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 $j = 1,2,\ldots,\mathrm{NCODE}$, may only appear **linearly** in the functions Q_i, for $i = 1,2,\ldots,\mathrm{NPDE}$, with a similar restriction for γ;

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

(iii) $t_0 < t_{out}$, 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 $i = 1,2,\ldots,\mathrm{NPTS}$, by calling the routine 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 non-zero 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 8 below.

The algebraic-differential equation system which is defined by the functions F_i must be specified in the user-supplied subroutine ODEDEF. The user 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 co-ordinates, 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 [4]) 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 the user to supply a subroutine MONITF which specifies in an analytical or numerical form the particular aspect of the solution behaviour the user wishes to track. This so-called monitor function is used by the routines 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.

The user specifies the frequency of mesh updates together with certain other criteria such as adjacent mesh ratios. Remeshing can be expensive and the user is 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. The subroutine 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

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

[2] 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

[3] 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

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

[5] 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 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE $\geq$ 1.

2: M — INTEGER *Input*

On entry: the co-ordinate system used:

M = 0
  indicates Cartesian co-ordinates,

M = 1
  indicates cylindrical polar co-ordinates,

M = 2
  indicates spherical polar co-ordinates.

Constraint: 0 $\leq$ M $\leq$ 2.

3: TS — ***real*** *Input/Output*

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

Constraint: TS < TOUT.

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

4: TOUT — ***real*** *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.

Its specification is:

```
      SUBROUTINE PDEDEF(NPDE, T, X, U, UX, NCODE, V, VDOT, P, Q, R, IRES)
      INTEGER           NPDE, NCODE, IRES
      real              T, X, U(NPDE), UX(NPDE), V(*), VDOT(*),
     1                  P(NPDE,NPDE), Q(NPDE), R(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 $i = 1,2,\ldots,\text{NPDE}$.

5: UX(NPDE) — ***real*** array *Input*

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

6: NCODE — INTEGER *Input*

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

7: V(*) — ***real*** array *Input*

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

8: VDOT(*) — ***real*** array *Input*

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

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

9: P(NPDE,NPDE) — ***real*** array *Output*

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

10: Q(NPDE) — ***real*** array *Output*

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

11: R(NPDE) — ***real*** array *Output*

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

12: 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 (IFAIL) set to 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 D03PPF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

PDEDEF must be declared as EXTERNAL in the (sub)program from which D03PPF 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 β_i and γ_i which describe the boundary conditions, as given in (4).

Its specification is:

```
SUBROUTINE BNDARY(NPDE, T, U, UX, NCODE, V, VDOT, IBND, BETA,
1                 GAMMA, IRES)
INTEGER           NPDE, NCODE, IBND, IRES
real              T, U(NPDE), UX(NPDE), V(*), VDOT(*), BETA(NPDE),
1                 GAMMA(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: U(NPDE) — ***real*** array *Input*

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

4: UX(NPDE) — ***real*** array *Input*

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

5: NCODE — INTEGER *Input*

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

6: V(∗) — ***real*** array *Input*

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

7: VDOT(∗) — ***real*** array *Input*

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

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

8: IBND — INTEGER *Input*

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

9: BETA(NPDE) — ***real*** array *Output*

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

10: GAMMA(NPDE) — ***real*** array *Output*

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

11: 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 (IFAIL) set to 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 D03PPF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

BNDARY must be declared as EXTERNAL in the (sub)program from which D03PPF is called. Parameters 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 $a \leq x \leq b$.

Its specification is:

```
SUBROUTINE UVINIT(NPDE, NPTS, NXI, X, XI, U, NCODE, V)
INTEGER           NPDE, NPTS, NXI, NCODE
real              X(NPTS), XI(*), U(NPDE,NPTS), V(*)
```

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*** array *Input*

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

5: XI(∗) — ***real*** array *Input*

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

6: U(NPDE,NPTS) — ***real*** array *Output*

On exit: U(i, j) must contain the value of component $U_i(x_j, t_0)$ for $i = 1, 2, \ldots,$ NPDE, $j = 1, 2, \ldots,$ NPTS.

7: NCODE — INTEGER *Input*

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

8: V(∗) — ***real*** array *Output*

On exit: V(i) must contain the value of component $V_i(t_0)$ for $i = 1, 2, \ldots,$ NCODE.

UVINIT must be declared as EXTERNAL in the (sub)program from which D03PPF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: U(NEQN) — ***real*** array *Output*

On exit: U(NPDE × $(j-1)+i$) contains the computed solution $U_i(x_j,t)$, for $i=1,2,\ldots,$NPDE; $j=1,2,\ldots,$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*** 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 ODEs in the system.

Constraint: NCODE ≥ 0.

12: ODEDEF — SUBROUTINE, supplied by the user. *External Procedure*

ODEDEF must evaluate the functions F, which define the system of ODEs, as given in (3). If the user wishes to compute the solution of a system of PDEs only (i.e., NCODE = 0), ODEDEF must be the dummy routine D03PCK. (D03PCK 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 ODEDEF(NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,
     1                  RCP, UCPT, UCPTX, F, IRES)
      INTEGER           NPDE, NCODE, NXI, IRES
      real              T, V(*), VDOT(*), XI(*), UCP(NPDE,*),
     1                  UCPX(NPDE,*), RCP(NPDE,*), UCPT(NPDE,*),
     2                  UCPTX(NPDE,*), F(*)
```

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: NCODE — INTEGER *Input*

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

4: V(*) — ***real*** array *Input*

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

5: VDOT(*) — ***real*** array *Input*

On entry: 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(*) — ***real*** array *Input*

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

8: UCP(NPDE,*) — ***real*** array *Input*

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

9: UCPX(NPDE,*) — ***real*** array *Input*

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

10: RCP(NPDE,*) — ***real*** 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; $j = 1,2,\ldots,$ NXI.

11: UCPT(NPDE,*) — ***real*** array *Input*

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

12: UCPTX(NPDE,*) — ***real*** array *Input*

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

13: F(*) — ***real*** array *Output*

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

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

or

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

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. If IRES = 1, then equation (5) above must be used. If IRES = −1, then equation (6) above must be used.

On exit: should usually remain unchanged. However, the user 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)program with the error indicator (IFAIL) set to 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 D03PPF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

ODEDEF must be declared as EXTERNAL in the (sub)program from which D03PPF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: NXI — INTEGER *Input*

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

Constraints:

NXI = 0 for NCODE = 0

NXI $\geq$ 0 for NCODE > 0.

14: XI(*) — ***real*** array *Input*

Note: the dimension of the array XI must be at least max(1,NXI).

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

Constraint: X(1) $\leq$ XI(1) $<$ XI(2) $< \ldots <$ 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*** 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) ≥ 0 for all relevant i.

17: ATOL(*) — ***real*** 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 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 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 parameter NORM, see below.

Constraint: $1 \leq$ ITOL ≤ 4.

19: NORM — CHARACTER*1 *Input*

On entry: the type of norm to be used. Two options are available:

'M' – maximum norm.
'A' – averaged L_2 norm.

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

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

while for the maximum norm

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

See the description of the ITOL parameter 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. The possible choices are:

'F' – full matrix routines to be used;
'B' – banded matrix routines to be used;
'S' – sparse matrix routines to be used.

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

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

21: ALGOPT(30) — ***real*** array *Input*

On entry: ALGOPT may be set to control various options available in the integrator. If the user wishes 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 parameter 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 initialise 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, i.e., 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 < 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. ALGOPT(30) must be greater than zero, otherwise the default value is used. 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. Remeshing can be switched off or on at specified times by using appropriate values for the parameters NRMESH and TRMESH at each call.

23: NXFIX — INTEGER *Input*

On entry: the number of fixed mesh points.

Constraint: 0 ≤ NXFIX ≤ 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*** array *Input*

Note: the dimension of the array XFIX must be at least max(1,NXFIX).

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

Constraint: XFIX(i) < XFIX(i+1), $i = 1,2,\ldots$,NXFIX−1}, and each fixed mesh point must coincide with a user-supplied initial mesh point, that is XFIX(i) = X(j) for some j, $2 \leq j \leq$ 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. The user 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 parameter DXMESH below. 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 (see below).

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 to give greater flexibility over the times of remeshing.

26: DXMESH — ***real*** *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

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

or

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

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

Constraint: DXMESH ≥ 0.0.

27: TRMESH — ***real*** *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 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: 0 ≤ IPMINF ≤ 2.

29: XRATIO — ***real*** *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

$$(x_i - x_{i-1})/\mathrm{XRATIO} < x_{i+1} - x_i < \mathrm{XRATIO} \times (x_i - x_{i-1})$$

Suggested value: XRATIO = 1.5.

Constraint: XRATIO > 1.0.

30: CONST — ***real*** *Input*

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

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

(see Furzeland [4]). CONST gives the user more control over the mesh distribution e.g. decreasing CONST allows more clustering. A typical value is 2/(NPTS − 1), but the user is 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: CONST = 2.0/(NPTS − 1).

Constraint: 0.1/(NPTS − 1) ≤ CONST ≤ 10.0/(NPTS − 1).

31: MONITF — SUBROUTINE, supplied by the user. *External Procedure*

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

If the user specifies REMESH = .FALSE., i.e., no remeshing, then MONITF will not be called and the dummy routine D03PCL may be used for MONITF. (D03PCL 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 MONITF(T, NPTS, NPDE, X, U, R, FMON)
      INTEGER            NPTS, NPDE
      real               T, X(NPTS), U(NPDE,NPTS), R(NPDE,NPTS),
     1                   FMON(NPTS)
```

1: T — ***real*** *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*** 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*** 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, $j = 1, 2, \ldots,$ NPTS.

6: R(NPDE,NPTS) — ***real*** 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, $j = 1, 2, \ldots,$ NPTS.

7: FMON(NPTS) — ***real*** array *Output*

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

Constraint: FMON(i) ≥ 0.

MONITF must be declared as EXTERNAL in the (sub)program from which D03PPF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

32: W(NW) — ***real*** array *Workspace*

33: NW — INTEGER *Input*

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

LAOPT = 'F',
$\quad$ NW $\geq$ NEQN $\times$ NEQN + NEQN + NWKRES + LENODE,
LAOPT = 'B',
$\quad$ NW $\geq$ (3 $\times$ MLU + 1) $\times$ NEQN + NWKRES + LENODE,
LAOPT = 'S',
$\quad$ NW $\geq$ 4 $\times$ NEQN + 11 $\times$ NEQN/2 + 1 + NWKRES + LENODE,

where MLU = the lower or upper half bandwidths,

MLU = 2 $\times$ NPDE − 1, for PDE problems only, and,

MLU = NEQN − 1, for coupled PDE/ODE problems.

NWKRES = NPDE $\times$ (3 $\times$ NPDE + 6 $\times$ NXI + NPTS + 15) + NXI + NCODE + 7 $\times$ NPTS + NXFIX + 1

when NCODE > 0, and NXI > 0.

NWKRES = NPDE $\times$ (3 $\times$ NPDE + NPTS + 21) + NCODE + 7 $\times$ NPTS + NXFIX + 2

when NCODE > 0, and NXI = 0.

NWKRES = NPDE $\times$ (3 $\times$ NPDE + NPTS + 21) + 7 $\times$ NPTS + NXFIX + 3

when NCODE = 0.

LENODE = (6 + int(ALGOPT(2))) $\times$ NEQN + 50, when the BDF method is used and,

LENODE = 9 $\times$ NEQN + 50, when the Theta method is used.

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

34: 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.

The rest of the array is used as workspace.

35: NIW — INTEGER *Input*

On entry: the dimension of the array IW. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',
    NIW $\geq$ 25 + NXFIX,
LAOPT = 'B',
    NIW $\geq$ NEQN + 25 + NXFIX,
LAOPT = 'S',
    NIW $\geq$ 25 $\times$ NEQN + 25 + NXFIX.

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

36: 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.
ITASK = 4
    normal computation of output values U at t = TOUT but without overshooting $t = t_{\mathrm{crit}}$, where t_{crit} is described under the parameter ALGOPT.
ITASK = 5
    take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter ALGOPT.

Constraint: $1 \leq$ ITASK ≤ 5.

37: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03PPF and the underlying ODE solver as follows:

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 = 1, 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.

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

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

Users are advised to set ITRACE = 0, unless they are experienced with the subchapter D02M-N of the NAG Fortran Library.

38: 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 and the remeshing parameters NRMESH, DXMESH, TRMESH, XRATIO and CONST may be reset between calls to D03PPF.

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

On exit: IND = 1.

39: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, (TOUT − TS) is too small,
- or ITASK ≠ 1, 2, 3, 4 or 5,
- or M ≠ 0, 1 or 2,
- or at least one of the coupling points defined in array XI is outside the interval [X(1),X(NPTS)],
- or M > 0 and X(1) < 0.0,
- or NPTS < 3,
- or NPDE < 1,
- or NORM ≠ 'A' or 'M',
- or LAOPT ≠ 'F', 'B' or 'S',
- or ITOL ≠ 1, 2, 3 or 4,
- or IND ≠ 0 or 1,
- or incorrectly defined user mesh, i.e., $X(i) \geq X(i+1)$ for some $i = 1, 2, \ldots, \text{NPTS} - 1$,
- or NW or NIW are too small,
- or NCODE and NXI are incorrectly defined,
- or IND = 1 on initial entry to D03PPF,
- or an element of RTOL or ATOL < 0.0,
- or corresponding elements of RTOL and ATOL are both 0.0,
- or NEQN ≠ NPDE × 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 ≠ 0, 1 or 2,
- or XRATIO ≤ 1.0,
- or CONST not in the range 0.1/(NPTS − 1) to 10/(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 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, 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. The user should check their problem formulation.

IFAIL = 6

When evaluating the residual in solving the ODE system, IRES was set to 2 in one of the user-supplied subroutines 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 the user-supplied routines, PDEDEF, BNDARY or ODEDEF, 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 ATOL and RTOL 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 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 advisory 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) has been taken.

IFAIL = 13

Some error weights w_i became zero during the time integration (see 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 succesful 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 NIW or NW was insufficient (more detailed information may be directed to the current error message unit).

IFAIL = 16

REMESH has been changed between calls to D03PPF, 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 accuracy parameters, ATOL and RTOL.

8 Further Comments

The parameter specification allows the user 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 by the routine 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.

9 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

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

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

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

9.1 Example 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.

```
*     D03PPF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, NCODE, M, NXI, NXFIX, NEQN, NIW,
     +                 NWKRES, LENODE, NW, INTPTS, ITYPE
      PARAMETER        (NPDE=1,NPTS=61,NCODE=0,M=0,NXI=0,NXFIX=0,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=25+NXFIX,
     +                 NWKRES=NPDE*(NPTS+3*NPDE+21)+7*NPTS+NXFIX+3,
     +                 LENODE=11*NEQN+50,NW=NEQN*NEQN+NEQN+NWKRES+
     +                 LENODE,INTPTS=5,ITYPE=1)
*     .. Scalars in Common ..
      real             E
*     .. Local Scalars ..
      real             CONST, DXMESH, TOUT, TRMESH, TS, XRATIO
      INTEGER          I, IFAIL, IND, IPMINF, IT, ITASK, ITOL, ITRACE,
     +                 NRMESH
      LOGICAL          REMESH, THETA
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NEQN),
     +                 UE(INTPTS), UOUT(NPDE,INTPTS,ITYPE), W(NW),
     +                 X(NPTS), XFIX(1), XI(1), XOUT(INTPTS)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PCK, D03PPF, D03PZF, EXACT, MONITF,
     +                 PDEDEF, UVINIT
*     .. Common blocks ..
      COMMON           /EPS/E
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PPF Example Program Results'
      E = 0.005e0
      ITRACE = 0
      ITOL = 1
      ATOL(1) = 0.5e-4
      RTOL(1) = ATOL(1)
      WRITE (NOUT,99998) ATOL, NPTS
*
*     Initialise mesh ..
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
*     Set remesh parameters..
*
      REMESH = .TRUE.
      NRMESH = 3
      DXMESH = 0.5e0
      CONST = 2.0e0/(NPTS-1.0e0)
      XRATIO = 1.5e0
      IPMINF = 0
*
```

```
      WRITE (NOUT,99993) NRMESH
      WRITE (NOUT,99992) E
      WRITE (NOUT,*)
*
      XI(1) = 0.0e0
      NORM = 'A'
      LAOPT = 'F'
      IND = 0
      ITASK = 1
*
*     Set THETA to .TRUE. if the Theta integrator is required
*
      THETA = .FALSE.
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
      IF (THETA) THEN
         ALGOPT(1) = 2.0e0
      ELSE
         ALGOPT(1) = 0.0e0
      END IF
*
*     Loop over output value of t
*
      TS = 0.0e0
      TOUT = 0.0e0
      DO 60 IT = 1, 5
         TOUT = 0.2e0*IT
         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,CONST,MONITF,W,NW,IW,NIW,ITASK,
     +               ITRACE,IND,IFAIL)
*
*        Set output points ..
         IF (IT.EQ.1) THEN
            XOUT(1) = 0.3e0
            XOUT(2) = 0.4e0
            XOUT(3) = 0.5e0
            XOUT(4) = 0.6e0
            XOUT(5) = 0.7e0
         ELSE IF (IT.EQ.2) THEN
            XOUT(1) = 0.4e0
            XOUT(2) = 0.5e0
            XOUT(3) = 0.6e0
            XOUT(4) = 0.7e0
            XOUT(5) = 0.8e0
         ELSE IF (IT.EQ.3) THEN
            XOUT(1) = 0.6e0
            XOUT(2) = 0.65e0
            XOUT(3) = 0.7e0
            XOUT(4) = 0.75e0
            XOUT(5) = 0.8e0
         ELSE IF (IT.EQ.4) THEN
            XOUT(1) = 0.7e0
            XOUT(2) = 0.75e0
```

```
            XOUT(3) = 0.8e0
            XOUT(4) = 0.85e0
            XOUT(5) = 0.9e0
         ELSE IF (IT.EQ.5) THEN
            XOUT(1) = 0.8e0
            XOUT(2) = 0.85e0
            XOUT(3) = 0.9e0
            XOUT(4) = 0.95e0
            XOUT(5) = 1.0e0
         END IF
*
         WRITE (NOUT,99999) TS
         WRITE (NOUT,99996) (XOUT(I),I=1,INTPTS)
*        Interpolate at output points ..
         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,I,1),I=1,INTPTS)
         WRITE (NOUT,99994) (UE(I),I=1,INTPTS)
*
   60 CONTINUE
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      STOP
*
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
*
      SUBROUTINE UVINIT(NPDE,NPTS,NXI,X,XI,U,NCODE,V)
*     .. Scalar Arguments ..
      INTEGER           NCODE, NPDE, NPTS, NXI
*     .. Array Arguments ..
      real              U(NPDE,NPTS), V(*), X(NPTS), XI(*)
*     .. Scalars in Common ..
      real              E
*     .. Local Scalars ..
      real              A, B, C, T
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Common blocks ..
      COMMON            /EPS/E
*     .. Executable Statements ..
      T = 0.0e0
      DO 20 I = 1, NPTS
         A = (X(I)-0.25e0-0.75e0*T)/(4.0e0*E)
         B = (0.9e0*X(I)-0.325e0-0.495e0*T)/(2.0e0*E)
```

```
         IF (A.GT.0.0e0 .AND. A.GT.B) THEN
            A = EXP(-A)
            C = (0.8e0*X(I)-0.4e0-0.24e0*T)/(4.0e0*E)
            C = EXP(C)
            U(1,I) = (0.5e0+0.1e0*C+A)/(1.0e0+C+A)
         ELSE IF (B.GT.0.0e0 .AND. B.GE.A) THEN
            B = EXP(-B)
            C = (-0.8e0*X(I)+0.4e0+0.24e0*T)/(4.0e0*E)
            C = EXP(C)
            U(1,I) = (0.1e0+0.5e0*C+B)/(1.0e0+C+B)
         ELSE
            A = EXP(A)
            B = EXP(B)
            U(1,I) = (1.0e0+0.5e0*A+0.1e0*B)/(1.0e0+A+B)
         END IF
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,UX,NCODE,V,VDOT,P,Q,R,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              P(NPDE,NPDE), Q(NPDE), R(NPDE), U(NPDE),
     +                  UX(NPDE), V(*), VDOT(*)
*     .. Scalars in Common ..
      real              E
*     .. Common blocks ..
      COMMON            /EPS/E
*     .. Executable Statements ..
      P(1,1) = 1.0e0
      R(1) = E*UX(1)
      Q(1) = U(1)*UX(1)
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,U,UX,NCODE,V,VDOT,IBND,BETA,GAMMA,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              BETA(NPDE), GAMMA(NPDE), U(NPDE), UX(NPDE),
     +                  V(*), VDOT(*)
*     .. Scalars in Common ..
      real              E
*     .. Local Scalars ..
      real              A, B, C, UE, X
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Common blocks ..
      COMMON            /EPS/E
*     .. Executable Statements ..
      BETA(1) = 0.0e0
      IF (IBND.EQ.0) THEN
         X = 0.0e0
         A = (X-0.25e0-0.75e0*T)/(4.0e0*E)
         B = (0.9e0*X-0.325e0-0.495e0*T)/(2.0e0*E)
```

```
            IF (A.GT.0.0e0 .AND. A.GT.B) THEN
               A = EXP(-A)
               C = (0.8e0*X-0.4e0-0.24e0*T)/(4.0e0*E)
               C = EXP(C)
               UE = (0.5e0+0.1e0*C+A)/(1.0e0+C+A)
            ELSE IF (B.GT.0.0e0 .AND. B.GE.A) THEN
               B = EXP(-B)
               C = (-0.8e0*X+0.4e0+0.24e0*T)/(4.0e0*E)
               C = EXP(C)
               UE = (0.1e0+0.5e0*C+B)/(1.0e0+C+B)
            ELSE
               A = EXP(A)
               B = EXP(B)
               UE = (1.0e0+0.5e0*A+0.1e0*B)/(1.0e0+A+B)
            END IF
         ELSE
            X = 1.0e0
            A = (X-0.25e0-0.75e0*T)/(4.0e0*E)
            B = (0.9e0*X-0.325e0-0.495e0*T)/(2.0e0*E)
            IF (A.GT.0.0e0 .AND. A.GT.B) THEN
               A = EXP(-A)
               C = (0.8e0*X-0.4e0-0.24e0*T)/(4.0e0*E)
               C = EXP(C)
               UE = (0.5e0+0.1e0*C+A)/(1.0e0+C+A)
            ELSE IF (B.GT.0.0e0 .AND. B.GE.A) THEN
               B = EXP(-B)
               C = (-0.8e0*X+0.4e0+0.24e0*T)/(4.0e0*E)
               C = EXP(C)
               UE = (0.1e0+0.5e0*C+B)/(1.0e0+C+B)
            ELSE
               A = EXP(A)
               B = EXP(B)
               UE = (1.0e0+0.5e0*A+0.1e0*B)/(1.0e0+A+B)
            END IF
         END IF
         GAMMA(1) = U(1) - UE
         RETURN
         END
*
      SUBROUTINE EXACT(T,X,NPTS,U)
*     Exact solution (for comparison purposes)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPTS
*     .. Array Arguments ..
      real              U(NPTS), X(NPTS)
*     .. Scalars in Common ..
      real              E
*     .. Local Scalars ..
      real              A, B, C
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Common blocks ..
      COMMON            /EPS/E
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         A = (X(I)-0.25e0-0.75e0*T)/(4.0e0*E)
```

```
         B = (0.9e0*X(I)-0.325e0-0.495e0*T)/(2.0e0*E)
         IF (A.GT.0.0e0 .AND. A.GT.B) THEN
            A = EXP(-A)
            C = (0.8e0*X(I)-0.4e0-0.24e0*T)/(4.0e0*E)
            C = EXP(C)
            U(I) = (0.5e0+0.1e0*C+A)/(1.0e0+C+A)
         ELSE IF (B.GT.0.0e0 .AND. B.GE.A) THEN
            B = EXP(-B)
            C = (-0.8e0*X(I)+0.4e0+0.24e0*T)/(4.0e0*E)
            C = EXP(C)
            U(I) = (0.1e0+0.5e0*C+B)/(1.0e0+C+B)
         ELSE
            A = EXP(A)
            B = EXP(B)
            U(I) = (1.0e0+0.5e0*A+0.1e0*B)/(1.0e0+A+B)
         END IF
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE MONITF(T,NPTS,NPDE,X,U,R,FMON)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              FMON(NPTS), R(NPDE,NPTS), U(NPDE,NPTS), X(NPTS)
*     .. Local Scalars ..
      real              DRDX, H
      INTEGER           I, K, L
*     .. Intrinsic Functions ..
      INTRINSIC         ABS, MAX, MIN
*     .. Executable Statements ..
      DO 20 I = 1, NPTS - 1
         K = MAX(1,I-1)
         L = MIN(NPTS,I+1)
         H = (X(L)-X(K))*0.5e0
*        Second derivative ..
         DRDX = (R(1,I+1)-R(1,I))/H
         FMON(I) = ABS(DRDX)
   20 CONTINUE
      FMON(NPTS) = FMON(NPTS-1)
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example 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
```

D03PRF – NAG Fortran Library Routine Document

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

D03PRF integrates a system of linear or nonlinear, first-order, time-dependent partial differential equations (PDEs) in one space variable, with scope for coupled ordinary differential equations (ODEs), and automatic adaptive spatial remeshing. The spatial discretisation is performed using the Keller box scheme [1] and the method of lines is employed to reduce the PDEs to a system of ODEs. The resulting system is solved using a Backward Differentiation Formula (BDF) method or a Theta method (switching between Newton's method and functional iteration).

2 Specification

```
      SUBROUTINE D03PRF(NPDE, TS, TOUT, PDEDEF, BNDARY, UVINIT, U, NPTS,
     1                  X, NLEFT, NCODE, ODEDEF, NXI, XI, NEQN, RTOL,
     2                  ATOL, ITOL, NORM, LAOPT, ALGOPT, REMESH, NXFIX,
     3                  XFIX, NRMESH, DXMESH, TRMESH, IPMINF, XRATIO,
     4                  CONST, MONITF, W, NW, IW, NIW, ITASK, ITRACE,
     5                  IND, IFAIL)
      INTEGER           NPDE, NPTS, NLEFT, NCODE, NXI, NEQN, ITOL,
     1                  NXFIX, NRMESH, IPMINF, NW, IW(NIW), NIW, ITASK,
     2                  ITRACE, IND, IFAIL
      real              TS, TOUT, U(NEQN), X(NPTS), XI(*), RTOL(*),
     1                  ATOL(*), ALGOPT(30), XFIX(*), DXMESH, TRMESH,
     2                  XRATIO, CONST, W(NW)
      LOGICAL           REMESH
      CHARACTER*1       NORM, LAOPT
      EXTERNAL          PDEDEF, BNDARY, UVINIT, ODEDEF, MONITF
```

3 Description

D03PRF integrates the system of first-order PDEs and coupled ODEs given by the master equations:

$$G_i(x,t,U,U_x,U_t,V,\dot{V}) = 0, \quad i = 1,2,...,\text{NPDE}, \ a \leq x \leq b, \ t \geq t_0, \tag{1}$$

$$F_i(t,V,\dot{V},\xi,U^*,U_x^*,U_t^*) = 0, \quad i = 1,2,...,\text{NCODE}. \tag{2}$$

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

$$G_i = \sum_{j=1}^{\text{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + \sum_{j=1}^{\text{NCODE}} Q_{i,j}\dot{V}_j + R_i = 0, \ i = 1,2,...,\text{NPDE}, \tag{3}$$

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

The vector U is the set of PDE solution values

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

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

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

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

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

$$F = A - B\dot{V} - CU_t^*, \tag{4}$$

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

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, where $a = x_1$ and $b = x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a mesh $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$ defined initially by the user and (possibly) adapted automatically during the integration according to user-specified criteria.

The PDE system which is defined by the functions G_i must be specified in the user-supplied subroutine PDEDEF.

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

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

The boundary conditions have the master equation form:

$$G_i^L(x,t,U,U_t,V,\dot{V}) = 0 \text{ at } x = a,\ i = 1,2,\ldots,\mathrm{NLEFT}, \tag{5}$$

at the left-hand boundary, and

$$G_i^R(x,t,U,U_t,V,\dot{V}) = 0 \text{ at } x = b,\ i = 1,2,\ldots,\mathrm{NRIGHT} \tag{6}$$

at the right-hand boundary.

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

$$\sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^L \dot{V}_j + S_i^L = 0,\ i = 1,2,\ldots,\mathrm{NLEFT}, \tag{7}$$

at the left-hand boundary, and

$$\sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^R \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^R \dot{V}_j + S_i^R = 0,\ i = 1,2,\ldots,\mathrm{NRIGHT}, \tag{8}$$

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

The boundary conditions must be specified in a subroutine BNDARY provided by the user.

The problem is subject to the following restrictions:

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

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

(iii) The evaluation of the function G_i is done approximately at the mid-points of the mesh X(i), for $i = 1, 2, \ldots,$ NPTS, by calling the routine PDEDEF for each mid-point in turn. Any discontinuities in the function **must** therefore be at one or more of the fixed mesh points specified by XFIX;

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

The algebraic-differential equation system which is defined by the functions F_i must be specified in the user-supplied subroutine ODEDEF. The user must also specify the coupling points ξ in the array XI.

The first order equations are approximated by a system of ODEs in time for the values of U_i at mesh points. In this method of lines approach the Keller box scheme is applied to each PDE in the space variable only, resulting in a system of ODEs in time for the values of U_i at each mesh point. In total there are 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 [6]) 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 the user to supply a subroutine MONITF which specifies in an analytic or numeric form the particular aspect of the solution behaviour the user wishes to track. This so-called monitor function is used by the routines to choose a mesh which equally distributes the integral of the monitor function over the domain. A typical choice of monitor function is the second space derivative of the solution value at each point (or some combination of the second space derivatives if more than one solution component), which results in refinement in regions where the solution gradient is changing most rapidly.

The user specifies the frequency of mesh updates along with certain other criteria such as adjacent mesh ratios. Remeshing can be expensive and the user is 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. The subroutine 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

[1] Keller H B (1970) A new difference scheme for parabolic problems *Numerical Solutions of Partial Differential Equations* (ed J Bramble) **2** Academic Press 327–350

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

[3] 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

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

[5] 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

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

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.

Constraint: TS < TOUT.

On exit: the value of t corresponding to the solution values in U. Normally 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*

On entry: PDEDEF must evaluate the functions G_i which define the system of PDEs. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PRF.

Its specification is:

```
      SUBROUTINE PDEDEF(NPDE, T, X, U, UDOT, UX, NCODE, V, VDOT, RES,
     1                  IRES)
      INTEGER           NPDE, NCODE, IRES
      real              T, X, U(NPDE), UDOT(NPDE), UX(NPDE), V(*),
     1                  VDOT(*), RES(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: UDOT(NPDE) — ***real*** array *Input*

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

6: UX(NPDE) — ***real*** array *Input*

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

7: NCODE — INTEGER *Input*

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

8: V(*) — ***real*** array *Input*

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

9: VDOT(*) — ***real*** array *Input*

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

10: RES(NPDE) — ***real*** array *Output*

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

$$G_i = \sum_{j=1}^{\mathrm{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} Q_{i,j}\dot{V}_j, \tag{9}$$

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

$$G_i = \sum_{j=1}^{\mathrm{NPDE}} P_{i,j}\frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} Q_{i,j}\dot{V}_j + R_i, \tag{10}$$

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

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

11: IRES — INTEGER *Input/Output*

On entry: the form of G_i that must be returned in the array RES. If IRES $= -1$, then equation (9) above must be used. If IRES $= 1$, then equation (10) above must be used.

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 (IFAIL) set to 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 D03PRF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

PDEDEF must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: BNDARY — SUBROUTINE, supplied by the user. *External Procedure*

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

Its specification is:

```
      SUBROUTINE BNDARY(NPDE, T, IBND, NOBC, U, UDOT, NCODE, V, VDOT,
     1                  RES, IRES)
      INTEGER           NPDE, IBND, NOBC, NCODE, IRES
      real              T, U(NPDE), UDOT(NPDE), V(*), VDOT(*), RES(NOBC)
```

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: IBND — INTEGER *Input*

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

4: NOBC — INTEGER *Input*

On entry: NOBC specifies the number of boundary conditions at the boundary specified by IBND.

5: U(NPDE) — ***real*** array *Input*

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

6: UDOT(NPDE) — ***real*** array *Input*

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

7: NCODE — INTEGER *Input*

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

8: V(*) — ***real*** array *Input*

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

9: VDOT(*) — ***real*** array *Input*

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

Note: VDOT(i), for $i = 1, 2, \ldots$, NCODE, may only appear linearly as in (11) and (12).

10: RES(NOBC) — ***real*** array *Output*

On exit: RES(i) must contain the ith component of G^L or G^R, depending on the value of IBND, for $i = 1, 2, \ldots$, NOBC, where G^L is defined as

$$G_i^L = \sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^L \dot{V}_j, \tag{11}$$

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

$$G_i^L = \sum_{j=1}^{\mathrm{NPDE}} E_{i,j}^L \frac{\partial U_j}{\partial t} + \sum_{j=1}^{\mathrm{NCODE}} H_{i,j}^L \dot{V}_j + S_i^L, \tag{12}$$

i.e., all terms in equation (7), and similarly for G_i^R.

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

11: IRES — INTEGER *Input/Output*

On entry: the form of G_i^L (or G_i^R) that must be returned in the array RES. If IRES $= -1$, then equation (11) above must be used. If IRES $= 1$, then equation (12) above must be used.

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 (IFAIL) set to 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 D03PRF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

BNDARY must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: 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 $[a,b]$.

Its specification is:

```
SUBROUTINE UVINIT(NPDE, NPTS, NXI, X, XI, U, NCODE, V)
INTEGER          NPDE, NPTS, NXI, NCODE
real             X(NPTS), XI(*), U(NPDE,NPTS), V(*)
```

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*** array *Input*

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

5: XI(*) — ***real*** array *Input*

On entry: XI(i) contains the ODE/PDE coupling point, ξ_i, for $1,2,\ldots,$NXI.

6: U(NPDE,NPTS) — ***real*** array *Output*

On exit: U(i,j) must contain the value of component $U_i(x_j,t_0)$ for $i = 1,2,\ldots,$NPDE, $j = 1,2,\ldots,$NPTS.

7: NCODE — INTEGER *Input*

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

8: V(*) — ***real*** array *Output*

On exit: V(i) must contain the value of component $V_i(t_0)$ for $i = 1,2,\ldots,$NCODE.

UVINIT must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

7: U(NEQN) — ***real*** array *Output*

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

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/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.

10: NLEFT — INTEGER *Input*

On entry: the number of boundary conditions at the left-hand mesh point X(1).

Constraint: $0 \leq$ NLEFT $\leq$ NPDE.

11: NCODE — INTEGER *Input*

On entry: the number of coupled ODE components.

Constraint: NCODE ≥ 0.

12: ODEDEF — SUBROUTINE, supplied by the user. *External Procedure*

ODEDEF must evaluate the functions F, which define the system of ODEs, as given in (4). If the user wishes to compute the solution of a system of PDEs only (i.e., NCODE = 0), ODEDEF must be the dummy routine D03PEK. (D03PEK 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 ODEDEF(NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,
     1                  UCPT, F, IRES)
      INTEGER           NPDE, NCODE, NXI, IRES
      real              T, V(*), VDOT(*), XI(*), UCP(NPDE,*),
     1                  UCPX(NPDE,*), UCPT(NPDE,*), F(*)
```

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: NCODE — INTEGER *Input*

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

4: V(*) — ***real*** array *Input*

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

5: VDOT(*) — ***real*** array *Input*

On entry: 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(*) — ***real*** array *Input*

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

8: UCP(NPDE,*) — ***real*** array *Input*

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

9: UCPX(NPDE,*) — ***real*** array *Input*

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

10: UCPT(NPDE,*) — ***real*** array *Input*

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

11: F(*) — ***real*** array *Output*

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

$$F = -B\dot{V} - CU_t^*, \tag{13}$$

that is, only terms depending explicitly on time derivatives, or

$$F = A - B\dot{V} - CU_t^*, \tag{14}$$

that is, all terms in equation (4). The definition of F is determined by the input value of IRES.

12: IRES — INTEGER *Input/Output*

On entry: the form of F that must be returned in the array F. If IRES = −1, then equation (13) above must be used. If IRES = 1, then equation (14) above must be used.

On exit: should usually remain unchanged. However, the user 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)program with the error indicator (IFAIL) set to 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 D03PRF returns to the calling (sub)program with the error indicator (IFAIL) set to 4.

ODEDEF must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: NXI — INTEGER *Input*

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

Constraints:

NXI = 0 for NCODE = 0,
NXI ≥ 0 for NCODE > 0.

14: XI(∗) — ***real*** array *Input*

Note: the dimension of the array XI must be at least max(1,NXI).

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

Constraint: X(1) ≤ XI(1) < XI(2) < ... < XI(NXI) ≤ X(NPTS).

15: NEQN — INTEGER *Input*

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

Constraint: NEQN = NPDE × NPTS + NCODE.

16: RTOL(∗) — ***real*** 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) ≥ 0 for all relevant i.

17: ATOL(∗) — ***real*** 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 for all relevant i.
Corresponding elements of RTOL and ATOL cannot both be 0.0.

18: ITOL — INTEGER *Input*

a value to indicate the form of the local error test. ITOL indicates to D03PRF 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:

On entry:

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	$\mathrm{RTOL}(1) \times \lvert \mathrm{U}(i) \rvert + \mathrm{ATOL}(1)$
2	scalar	vector	$\mathrm{RTOL}(1) \times \lvert \mathrm{U}(i) \rvert + \mathrm{ATOL}(i)$
3	vector	scalar	$\mathrm{RTOL}(i) \times \lvert \mathrm{U}(i) \rvert + \mathrm{ATOL}(1)$
4	vector	vector	$\mathrm{RTOL}(i) \times \lvert \mathrm{U}(i) \rvert + \mathrm{ATOL}(i)$

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 parameter NORM, see below.

Constraint: $1 \leq \mathrm{ITOL} \leq 4$.

19: NORM — CHARACTER*1 *Input*

On entry: the type of norm to be used. Two options are available:

'M' – maximum norm.

'A' – averaged L_2 norm.

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

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

while for the maximum norm

$$\mathrm{U_{norm}} = \max_i \lvert \mathrm{U}(i)/w_i \rvert.$$

See the description of the ITOL parameter 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. The possible choices are:

'F' – full matrix routines to be used;

'B' – banded matrix routines to be used;

'S' – sparse matrix routines to be used.

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

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

21: ALGOPT(30) — ***real*** array *Input*

On entry: ALGOPT may be set to control various options available in the integrator. If the user wishes 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) $\leq$ 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 parameter 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 initialise 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, i.e., 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 < 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. ALGOPT(30) must be greater than zero, otherwise the default value is used. 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 D03PRF. Remeshing can be switched off or on at specified times by using appropriate values for the parameters NRMESH and TRMESH at each call.

23: NXFIX — INTEGER *Input*

On entry: the number of fixed mesh points.

Constraint: $0 \leq$ NXFIX $\leq$ 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*** array *Input*

Note: the dimension of the array XFIX must be at least max(1,NXFIX).

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

Constraint: XFIX(i) < XFIX(i+1), $i = 1,2,\ldots$,NXFIX−1, and each fixed mesh point must coincide with a user-supplied initial mesh point, that is XFIX(i) = X(j) for some j, $2 \leq j \leq$ 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. The user should take this into account when choosing the initial mesh distribution.

25: NRMESH — INTEGER *Input*

On entry:

NRMESH < 0
: indicates that a new mesh is adopted according to the parameter DXMESH below. 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 (see below).

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 D03PRF to give greater flexibility over the times of remeshing.

26: DXMESH — ***real*** *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

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

or

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

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

Constraint: DXMESH $\geq$ 0.0.

27: TRMESH — ***real*** *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 D03PRF 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: $0 \leq$ IPMINF ≤ 2.

29: XRATIO — ***real*** *Input*

On entry: input bound on adjacent mesh ratio (greater than 1.0 and typically in the range 1.5 to 3.0). The resmeshing 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: CONST — ***real*** *Input*

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

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

(see Furzeland [6]). CONST gives the user more control over the mesh distribution e.g. decreasing CONST allows more clustering. A typical value is 2/(NPTS − 1), but the user is 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: CONST = 2.0/(NPTS − 1).

Constraint: 0.1/(NPTS − 1) $\leq$ CONST $\leq$ 10.0/(NPTS − 1).

31: MONITF — SUBROUTINE, supplied by the user. *External Procedure*

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

If the user specifies REMESH = .FALSE., i.e., no remeshing, then MONITF will not be called and the dummy routine D03PEL may be used for MONITF. (D03PEL 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 MONITF(T, NPTS, NPDE, X, U, FMON)
INTEGER            NPTS, NPDE
real               T, X(NPTS), U(NPDE,NPTS), FMON(NPTS)
```

1: T — ***real*** *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*** 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*** array *Input*

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

6: FMON(NPTS) — ***real*** array *Output*

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

Constraint: FMON(i) ≥ 0.

MONITF must be declared as EXTERNAL in the (sub)program from which D03PRF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

32: W(NW) — ***real*** array *Workspace*

33: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PRF is called. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',
  NW $\geq$ NEQN $\times$ NEQN + NEQN + NWKRES + LENODE,

LAOPT = 'B',
  NW $\geq$ (2 $\times$ ML + MU + 2) $\times$ NEQN + NWKRES + LENODE,

LAOPT = 'S',
  NW $\geq$ 4 $\times$ NEQN + 11 $\times$ NEQN/2 + 1 + NWKRES + LENODE,

where ML and MU are the lower and upper half bandwidths, given by ML = NPDE + NLEFT − 1, MU = 2 $\times$ NPDE − NLEFT − 1 for problems involving PDEs only, and ML = MU = NEQN − 1, for coupled PDE/ODE problems.

NWKRES = NPDE $\times$ (3 $\times$ NPDE + 6 $\times$ NXI + NPTS + 15) + NXI + NCODE + 7 $\times$ NPTS + NXFIX + 1 when NCODE > 0, and NXI > 0.

$\text{NWKRES} = \text{NPDE} \times (3 \times \text{NPDE} + \text{NPTS} + 21) + \text{NCODE} + 7 \times \text{NPTS} + \text{NXFIX} + 2$

when NCODE > 0, and NXI = 0.

$\text{NWKRES} = \text{NPDE} \times (3 \times \text{NPDE} + \text{NPTS} + 21) + 7 \times \text{NPTS} + \text{NXFIX} + 3$

when NCODE = 0.

LENODE = (6+ int(ALGOPT(2))) ×NEQN + 50, when the BDF method is used and,

LENODE = 9 × NEQN + 50, when the Theta method is used.

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

34: 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.

The rest of the array is used as workspace.

35: NIW — INTEGER *Input*

On entry: the dimension of the array IW as declared in the (sub)program from which D03PRF is called. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',
  NIW ≥ 25 + NXFIX,
LAOPT = 'B',
  NIW ≥ NEQN + 25 + NXFIX,
LAOPT = 'S',
  NIW ≥ 25 × NEQN + 25 + NXFIX.

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

36: 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.

ITASK = 4
normal computation of output values U at t = TOUT but without overshooting $t = t_{crit}$, where t_{crit} is described under the parameter ALGOPT.

ITASK = 5
take one step in the time direction and return, without passing t_{crit}, where t_{crit} is described under the parameter ALGOPT.

Constraint: 1 ≤ ITASK ≤ 5.

37: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03PRF and the underlying ODE solver as follows:

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 = 1, 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.

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

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

Users are advised to set ITRACE = 0, unless they are experienced with the subchapter D02M-N of the NAG Fortran Library.

38: 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 and the remeshing parameters NRMESH, DXMESH, TRMESH, XRATIO and CONST may be reset between calls to D03PRF.

Constraint: 0 ≤ IND ≤ 1.

On exit: IND = 1.

39: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, (TOUT − TS) is too small,
or ITASK ≠ 1, 2, 3, 4 or 5,

or at least one of the coupling points defined in array XI is outside the interval [X(1),X(NPTS)],

or NPTS < 3,

or NPDE < 1,

or NLEFT not in the range 0 to NPDE,

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 incorrectly defined user mesh, i.e., $X(i) \geq X(i+1)$ for some $i = 1, 2, \ldots, \text{NPTS} - 1$,

or NW or NIW are too small,

or NCODE and NXI are incorrectly defined,

or IND = 1 on initial entry to D03PRF,

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 ≤ 1.0,

or CONST not in the range 0.1/(NPTS – 1) to 10/(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. Incorrect positioning 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, BNDARY or ODEDEF, when the residual in the underlying ODE solver was being evaluated. Incorrect positioning of boundary conditions may also result in this error.

IFAIL = 5

In solving the ODE system, a singular Jacobian has been encountered. The user should check their problem formulation.

IFAIL = 6

When evaluating the residual in solving the ODE system, IRES was set to 2 in one of the user-supplied subroutines 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 the user-supplied routines, PDEDEF, BNDARY or ODEDEF, IRES was set to an invalid value.

IFAIL = 9

A serious error has occurred in an internal call to D02NNF. Check problem specification an 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 ATOL and RTOL 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 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 advisory 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) has been taken.

IFAIL = 13

Some error weights w_i became zero during the time integration (see 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 succesful as far as t = TS.

IFAIL = 14

Not applicable.

IFAIL = 15

When using the sparse option, the value of NIW or NW was insufficient (more detailed information may be directed to the current error message unit).

IFAIL = 16

REMESH has been changed between calls to D03PRF.

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 accuracy parameters, ATOL and RTOL.

8 Further Comments

The Keller box scheme can be used to solve higher-order problems which have been reduced to first order by the introduction of new variables (see the example in Section 9). In general, a second-order problem can be solved with slightly greater accuracy using the Keller box scheme instead of a finite-difference scheme (D03PPF for example), but at the expense of increased CPU time due to the larger number of function evaluations required.

It should be noted that the Keller box scheme, in common with other central-difference schemes, may be unsuitable for some hyperbolic first-order problems such as the apparently simple linear advection equation $U_t + aU_x = 0$, where a is a constant, resulting in spurious oscillations due to the lack of dissipation. This type of problem requires a discretisation scheme with upwind weighting (D03PSF for example), or the addition of a second-order artificial dissipation term.

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

9 Example

This example is the first-order system

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

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

for $x \in [0,1]$ and $t \geq 0$.

The initial conditions are

$$U_1(x,0) = e^x,$$

$$U_2(x,0) = x^2 + \sin(2\pi x^2),$$

and the Dirichlet boundary conditions for U_1 at $x = 0$ and U_2 at $x = 1$ are given by the exact solution:

$$U_1(x,t) = \frac{1}{2}\{e^{x+t} + e^{x-3t}\} + \frac{1}{4}\{\sin(2\pi(x-3t)^2) - \sin(2\pi(x+t)^2)\} + 2t^2 - 2xt,$$
$$U_2(x,t) = e^{x-3t} - e^{x+t} + \frac{1}{2}\{\sin(2\pi(x-3t)^2) + \sin(2\pi(x+t)^2)\} + x^2 + 5t^2 - 2xt.$$

9.1 Example 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.

```
*     D03PRF Example Program Text
*     Mark 16 Release. NAG Copyright 1993.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          NPDE, NPTS, NV, NXI, NXFIX, NLEFT, NEQN, NIW,
     +                 NWKRES, LENODE, NW, INTPTS, ITYPE
      PARAMETER        (NPDE=2,NPTS=61,NV=0,NXI=0,NXFIX=0,NLEFT=1,
     +                 NEQN=NPDE*NPTS+NV,NIW=25+NXFIX,
     +                 NWKRES=NPDE*(NPTS+21+3*NPDE)+7*NPTS+NXFIX+3,
     +                 LENODE=11*NEQN+50,NW=NEQN*NEQN+NEQN+NWKRES+
     +                 LENODE,INTPTS=5,ITYPE=1)
*     .. Scalars in Common ..
      real             P
*     .. Local Scalars ..
      real             CONST, DXMESH, TOUT, TRMESH, TS, XRATIO, XX
      INTEGER          I, IFAIL, IND, IPMINF, IT, ITASK, ITOL, ITRACE,
     +                 NRMESH
      LOGICAL          REMESH, THETA
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NPDE,NPTS),
     +                 UE(NPDE,NPTS), UOUT(NPDE,INTPTS,ITYPE), W(NW),
     +                 X(NPTS), XFIX(1), XI(1), XOUT(INTPTS)
      INTEGER          IW(NIW)
```

```
*     .. External Functions ..
      real              X01AAF
      EXTERNAL          X01AAF
*     .. External Subroutines ..
      EXTERNAL          BNDARY, D03PEK, D03PRF, D03PZF, EXACT, MONITF,
     +                  PDEDEF, UVINIT
*     .. Common blocks ..
      COMMON            /PI/P
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PRF Example Program Results'
      P = X01AAF(XX)
      ITRACE = 0
      ITOL = 1
      ATOL(1) = 0.5e-4
      RTOL(1) = ATOL(1)
      WRITE (NOUT,99996) ATOL, NPTS
*
*     Set remesh parameters ..
*
      REMESH = .TRUE.
      NRMESH = 3
      DXMESH = 0.0e0
      CONST = 5.0e0/(NPTS-1.0e0)
      XRATIO = 1.2e0
      IPMINF = 0
      WRITE (NOUT,99999) NRMESH
*
*     Initialise mesh ..
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
      XOUT(1) = 0.0e0
      XOUT(2) = 0.25e0
      XOUT(3) = 0.5e0
      XOUT(4) = 0.75e0
      XOUT(5) = 1.0e0
      WRITE (NOUT,99998) (XOUT(I),I=1,INTPTS)
*
      XI(1) = 0.0e0
      NORM = 'A'
      LAOPT = 'F'
      IND = 0
      ITASK = 1
*
*     Set THETA to .TRUE. if the Theta integrator is required
*
      THETA = .FALSE.
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
      IF (THETA) THEN
         ALGOPT(1) = 2.0e0
         ALGOPT(6) = 2.0e0
         ALGOPT(7) = 1.0e0
      END IF
*
```

```
*     Loop over output value of t
*
      TS = 0.0e0
      TOUT = 0.0e0
*
      DO 60 IT = 1, 5
         TOUT = 0.05e0*IT
         IFAIL = 0
*
         CALL D03PRF(NPDE,TS,TOUT,PDEDEF,BNDARY,UVINIT,U,NPTS,X,NLEFT,
     +               NV,D03PEK,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,
     +               ALGOPT,REMESH,NXFIX,XFIX,NRMESH,DXMESH,TRMESH,
     +               IPMINF,XRATIO,CONST,MONITF,W,NW,IW,NIW,ITASK,
     +               ITRACE,IND,IFAIL)
*
*        Interpolate at output points ..
         CALL D03PZF(NPDE,0,U,NPTS,X,XOUT,INTPTS,ITYPE,UOUT,IFAIL)
*        Check against exact solution ..
         CALL EXACT(TS,NPDE,INTPTS,XOUT,UE)
*
         WRITE (NOUT,99997) TS
         WRITE (NOUT,99994) (UOUT(1,I,1),I=1,INTPTS)
         WRITE (NOUT,99993) (UE(1,I),I=1,INTPTS)
         WRITE (NOUT,99992) (UOUT(2,I,1),I=1,INTPTS)
         WRITE (NOUT,99991) (UE(2,I),I=1,INTPTS)
*
   60 CONTINUE
      WRITE (NOUT,99995) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (' Remeshing every ',I3,' time steps',/)
99998 FORMAT (' X        ',5F10.4,/)
99997 FORMAT (' T = ',F6.3)
99996 FORMAT (//'  Accuracy requirement =',e10.3,' Number of points = ',
     +       I3,/)
99995 FORMAT (' Number of integration steps in time = ',I6,/' Number o',
     +       'f function evaluations = ',I6,/' Number of Jacobian eval',
     +       'uations =',I6,/' Number of iterations = ',I6,/)
99994 FORMAT (' Approx U1',5F10.4)
99993 FORMAT (' Exact  U1',5F10.4)
99992 FORMAT (' Approx U2',5F10.4)
99991 FORMAT (' Exact  U2',5F10.4,/)
      END
*
      SUBROUTINE UVINIT(NPDE,NPTS,NXI,X,XI,U,NV,V)
*     .. Scalar Arguments ..
      INTEGER          NPDE, NPTS, NV, NXI
*     .. Array Arguments ..
      real             U(NPDE,NPTS), V(*), X(NPTS), XI(*)
*     .. Scalars in Common ..
      real             P
*     .. Local Scalars ..
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        EXP, SIN
*     .. Common blocks ..
      COMMON           /PI/P
*     .. Executable Statements ..
```

```
      DO 20 I = 1, NPTS
         U(1,I) = EXP(X(I))
         U(2,I) = X(I)**2 + SIN(2.0e0*P*X(I)**2)
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE PDEDEF(NPDE,T,X,U,UDOT,DUDX,NV,V,VDOT,RES,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NPDE, NV
*     .. Array Arguments ..
      real              DUDX(NPDE), RES(NPDE), U(NPDE), UDOT(NPDE),
     +                  V(*), VDOT(*)
*     .. Executable Statements ..
      IF (IRES.EQ.-1) THEN
         RES(1) = UDOT(1)
         RES(2) = UDOT(2)
      ELSE
         RES(1) = UDOT(1) + DUDX(1) + DUDX(2)
         RES(2) = UDOT(2) + 4.0e0*DUDX(1) + DUDX(2)
      END IF
      RETURN
      END
*
      SUBROUTINE BNDARY(NPDE,T,IBND,NOBC,U,UDOT,NV,V,VDOT,RES,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NOBC, NPDE, NV
*     .. Array Arguments ..
      real              RES(NOBC), U(NPDE), UDOT(NPDE), V(*), VDOT(*)
*     .. Scalars in Common ..
      real              P
*     .. Local Scalars ..
      real              PP
*     .. Intrinsic Functions ..
      INTRINSIC         EXP, SIN
*     .. Common blocks ..
      COMMON            /PI/P
*     .. Executable Statements ..
      PP = 2.0e0*P
      IF (IBND.EQ.0) THEN
         IF (IRES.EQ.-1) THEN
            RES(1) = 0.0e0
         ELSE
            RES(1) = U(1) - 0.5e0*(EXP(T)+EXP(-3.0e0*T)) -
     +               0.25e0*(SIN(PP*9.0e0*T**2)-SIN(PP*T**2)) -
     +               2.0e0*T**2
         END IF
      ELSE
         IF (IRES.EQ.-1) THEN
            RES(1) = 0.0e0
         ELSE
            RES(1) = U(2) - (EXP(1.0e0-3.0e0*T)-EXP(1.0e0+T)
     +               +0.5e0*(SIN(PP*(1.0e0-3.0e0*T)**2)+SIN(PP*(1.0e0+T)
     +               **2))+1.0e0+5.0e0*T**2-2.0e0*T)
         END IF
      END IF
```

```
      RETURN
      END
*
      SUBROUTINE EXACT(T,NPDE,NPTS,X,U)
*     Exact solution (for comparison purposes)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              U(NPDE,NPTS), X(NPTS)
*     .. Scalars in Common ..
      real              P
*     .. Local Scalars ..
      real              PP
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         EXP, SIN
*     .. Common blocks ..
      COMMON            /PI/P
*     .. Executable Statements ..
      PP = 2.0e0*P
      DO 20 I = 1, NPTS
         U(1,I) = 0.5e0*(EXP(X(I)+T)+EXP(X(I)-3.0e0*T)) +
     +            0.25e0*(SIN(PP*(X(I)-3.0e0*T)**2)-SIN(PP*(X(I)+T)**2))
     +            + 2.0e0*T**2 - 2.0e0*X(I)*T
         U(2,I) = EXP(X(I)-3.0e0*T) - EXP(X(I)+T) + 0.5e0*(SIN(PP*(X(I)
     +            -3.0e0*T)**2)+SIN(PP*(X(I)+T)**2)) + X(I)**2 +
     +            5.0e0*T**2 - 2.0e0*X(I)*T
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE MONITF(T,NPTS,NPDE,X,U,FMON)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              FMON(NPTS), U(NPDE,NPTS), X(NPTS)
*     .. Local Scalars ..
      real              D2X1, D2X2, H1, H2, H3
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         ABS, MAX
*     .. Executable Statements ..
      DO 20 I = 2, NPTS - 1
         H1 = X(I) - X(I-1)
         H2 = X(I+1) - X(I)
         H3 = 0.5e0*(X(I+1)-X(I-1))
*        Second derivatives ..
         D2X1 = ABS(((U(1,I+1)-U(1,I))/H2-(U(1,I)-U(1,I-1))/H1)/H3)
         D2X2 = ABS(((U(2,I+1)-U(2,I))/H2-(U(2,I)-U(2,I-1))/H1)/H3)
         FMON(I) = MAX(D2X1,D2X2)
   20 CONTINUE
      FMON(1) = FMON(2)
      FMON(NPTS) = FMON(NPTS-1)
      RETURN
      END
```

9.2 Example Data

None.

9.3 Example Results

```
D03PRF Example Program Results


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

Remeshing every   3 time steps

X             0.0000    0.2500    0.5000    0.7500    1.0000

T =  0.050
Approx U1     0.9923    1.0894    1.4686    2.3388    2.1071
Exact  U1     0.9923    1.0893    1.4686    2.3391    2.1073
Approx U2    -0.0997    0.1057    0.7180    0.0967    0.2021
Exact  U2    -0.0998    0.1046    0.7193    0.0966    0.2022

T =  0.100
Approx U1     1.0613    0.9856    1.3120    2.3084    2.1039
Exact  U1     1.0613    0.9851    1.3113    2.3092    2.1025
Approx U2    -0.0150   -0.0481    0.1075   -0.3240    0.3753
Exact  U2    -0.0150   -0.0495    0.1089   -0.3235    0.3753

T =  0.150
Approx U1     1.1485    0.9763    1.2658    2.0906    2.2027
Exact  U1     1.1485    0.9764    1.2654    2.0911    2.2027
Approx U2     0.1370   -0.0250   -0.4107   -0.8577    0.3096
Exact  U2     0.1366   -0.0266   -0.4100   -0.8567    0.3096

T =  0.200
Approx U1     1.0956    1.0529    1.3407    1.8322    2.2035
Exact  U1     1.0956    1.0515    1.3393    1.8327    2.2050
Approx U2     0.0381    0.1282   -0.7979   -1.1776   -0.4221
Exact  U2     0.0370    0.1247   -0.7961   -1.1784   -0.4221

T =  0.250
Approx U1     0.8119    1.1288    1.5163    1.6076    2.2027
Exact  U1     0.8119    1.1276    1.5142    1.6091    2.2035
Approx U2    -0.4968    0.2123   -1.0259   -1.2149   -1.3938
Exact  U2    -0.4992    0.2078   -1.0257   -1.2183   -1.3938

Number of integration steps in time =     50
Number of function evaluations =    2579
Number of Jacobian evaluations =     20
Number of iterations =     126
```

D03PSF – NAG Fortran Library Routine Document

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

D03PSF integrates a system of linear or nonlinear convection-diffusion equations in one space dimension, with optional source terms and scope for coupled ordinary differential equations (ODEs). The system must be posed in conservative form. This routine also includes the option of automatic adaptive spatial remeshing. 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 partial differential equations (PDEs) to a system of ODEs, and the resulting system is solved using a backward differentiation formula (BDF) method or a Theta method.

2 Specification

```
      SUBROUTINE D03PSF(NPDE, TS, TOUT, PDEDEF, NUMFLX, BNDARY, UVINIT,
     1                  U, NPTS, X, NCODE, ODEDEF, NXI, XI, NEQN, RTOL,
     2                  ATOL, ITOL, NORM, LAOPT, ALGOPT, REMESH, NXFIX,
     3                  XFIX, NRMESH, DXMESH, TRMESH, IPMINF, XRATIO,
     4                  CONST, MONITF, W, NW, IW, NIW, ITASK, ITRACE,
     5                  IND, IFAIL)
      INTEGER           NPDE, NPTS, NCODE, NXI, NEQN, ITOL, NXFIX,
     1                  NRMESH, IPMINF, NW, IW(NIW), NIW, ITASK, ITRACE,
     2                  IND, IFAIL
      real              TS, TOUT, U(NEQN), X(NPTS), XI(*), RTOL(*),
     1                  ATOL(*), ALGOPT(30), XFIX(*), DXMESH, TRMESH,
     2                  XRATIO, CONST, W(NW)
      LOGICAL           REMESH
      CHARACTER*1       NORM, LAOPT
      EXTERNAL          PDEDEF, NUMFLX, BNDARY, UVINIT, ODEDEF, MONITF
```

3 Description

D03PSF 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 PDE solution values

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

The optional coupled ODEs are of the general form

$$R_i(t,V,\dot{V},\xi,U^*,U_x^*,U_t^*) = 0, \quad i = 1,2,...,\text{NCODE}, \tag{3}$$

where the vector V is the set of ODE solution values

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

$\dot{V}$ denotes its derivative with respect to time, and U_x is the spatial derivative of U.

In (2), $P_{i,j}$, F_i and C_i depend on x, t, U and V; D_i depends on x, t, U, U_x and V; and S_i depends on x, t, U, V and **linearly** on $\dot{V}$. Note that $P_{i,j}$, F_i, C_i and S_i must not depend on any space derivatives, and $P_{i,j}$, F_i, C_i and D_i must not depend on any 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.

In (3), 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 PDE spatial mesh points. U^*, U_x^* and U_t^* are the functions U, U_x and U_t evaluated at these coupling points. Each R_i may depend only linearly on time derivatives. Hence (3) may be written more precisely as

$$R = L - M\dot{V} - NU_t^*, \tag{4}$$

where $R = [R_1, \ldots, R_{\mathrm{NCODE}}]^T$, L is a vector of length NCODE, M is an NCODE by NCODE matrix, N is an NCODE by ($n_\xi \times$ NPDE) matrix and the entries in L, M and N may depend on t, ξ, U^*, U_x^* and V. In practice the user only needs to supply a vector of information to define the ODEs and not the matrices L, M and N. (See Section 5 for the specification of the user-supplied procedure ODEDEF).

The integration in time is from t_0 to t_{out}, over the space interval $a \leq x \leq b$, where $a = x_1$ and $b = x_{\mathrm{NPTS}}$ are the leftmost and rightmost points of a user-defined mesh $x_1, x_2, \ldots, x_{\mathrm{NPTS}}$ defined initially by the user and (possibly) adapted automatically during the integration according to user-specified criteria.

The initial ($t = t_0$) values of the functions $U(x,t)$ and $V(t)$ must be specified in a subroutine UVINIT supplied by the user. 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 $a \leq x \leq b$, and not just the initial mesh points.

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, D03PHF and D03PPF, 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, $\hat{F}_i$ 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 $x_{j-\frac{1}{2}} = (x_{j-1} + x_j)/2$ for $j = 2, 3, \ldots$, NPTS. The left and right values are calculated by D03PSF from two adjacent mesh points using a standard upwind technique combined with a Van Leer slope-limiter (see [2]). The physically correct value for $\hat{F}_i$ is derived from the solution of the Riemann problem given by

$$\frac{\partial U_i}{\partial t} + \frac{\partial F_i}{\partial y} = 0, \tag{5}$$

where $y = x - x_{j-\frac{1}{2}}$, i.e., $y = 0$ corresponds to $x = x_{j-\frac{1}{2}}$, 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 [2] and [5]. Roe's scheme [4] 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

$$\hat{F} = \frac{1}{2}(F_L + F_R) - \frac{1}{2}\sum_{k=1}^{\mathrm{NPDE}} \alpha_k |\lambda_k| e_k, \tag{6}$$

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

$$U_R - U_L = \sum_{k=1}^{\mathrm{NPDE}} \alpha_k e_k. \tag{7}$$

Examples are given in the D03PFF and D03PLF documentation.

If the system is nonlinear, Roe's scheme requires that a linearized Jacobian is found (see [4]).

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 D03PLP may be used for PDEDEF (D03PLP is included in the NAG Fortran Library; however, its name may be implementation-dependent: see the Users' Note for your implementation for details). D03PLP sets the matrix with entries $P_{i,j}$ to the identity matrix, and the functions C_i, D_i and S_i to zero.

For second-order problems i.e., diffusion terms present, a boundary condition is required for each PDE at both boundaries for the problem to be well-posed. If there are no diffusion terms present, then the continuous PDE problem generally requires exactly one boundary conditions 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 when using banded matrix algebra the fixed bandwidth means that only linear extrapolation is allowed, i.e., using information at just two interior points adjacent to the boundary). 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. Another method of supplying numerical boundary conditions involves the solution of the characteristic equations associated with the outgoing characteristics. Examples of both methods can be found in the D03PFF and D03PLF documentation.

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

$$G_i^L(x,t,U,V,\dot{V}) = 0 \;\text{ at }\; x = a, \;\; i = 1,2,\ldots,\text{NPDE}, \tag{8}$$

at the left-hand boundary, and

$$G_i^R(x,t,U,V,\dot{V}) = 0 \;\text{ at }\; x = b, \;\; i = 1,2,\ldots,\text{NPDE}, \tag{9}$$

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 algebraic-differential equation system which is defined by the functions R_i must be specified in a subroutine ODEDEF supplied by the user. The user must also specify the coupling points ξ (if any) in the array XI.

In total there are NPDE $\times$ NPTS + NCODE ODEs in the time direction. This system is then integrated forwards in time using a BDF or Theta method, optionally switching between Newton's method and functional iteration (see [5] and the references therein).

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 [6]) 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 the user to supply a subroutine MONITF which specifies in an analytical or numerical form the particular aspect of the solution behaviour the user wishes to track. This so-called monitor function is used by the routines 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.

The user specifies the frequency of mesh updates together with certain other criteria such as adjacent mesh ratios. Remeshing can be expensive and the user is 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. The subroutine 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.

The problem is subject to the following restrictions:

(i) In (1), $\dot{V}_j(t)$, for $j = 1,2,\ldots,$NCODE, may only appear **linearly** in the functions S_i, for $i = 1,2,\ldots,$NPDE, with a similar restriction for G_i^L and G_i^R;

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

(iii) $t_0 < t_{out}$, 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 **fixed** mesh points specified by XFIX;

(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.

For further details of the scheme, see [1] and the references therein.

4 References

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

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

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

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

[5] 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

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

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs to be solved.

Constraint: NPDE $\geq$ 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. $P_{i,j}$ and C_i may depend on x, t, U and V; D_i may depend on x, t, U, U_x and V; and S_i may depend on x, t, U, V and linearly on $\dot{V}$. PDEDEF is called approximately midway between each pair of mesh points in turn by D03PSF. The actual argument D03PLP may be used for PDEDEF for problems in the form (2) (D03PLP 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, NCODE, V, VDOT, P, C, D, S,
     1                  IRES)
      INTEGER           NPDE, NCODE, IRES
      real              T, X, U(NPDE), UX(NPDE), V(*), VDOT(*),
     1                  P(NPDE,NPDE), C(NPDE), 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: NCODE — INTEGER *Input*

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

7: V($*$) — ***real*** array *Input*

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

8: VDOT($*$) — ***real*** 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 S_j, for $j = 1,2,\ldots,$NPDE.

9: P(NPDE,NPDE) — ***real*** array *Output*

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

10: C(NPDE) — ***real*** array *Output*

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

11: D(NPDE) — ***real*** array *Output*

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

12: S(NPDE) — ***real*** array *Output*

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

13: 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 D03PSF 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 D03PSF 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 D03PSF.

Its specification is:

```
SUBROUTINE NUMFLX(NPDE, T, X, NCODE, V, ULEFT, URIGHT, FLUX, IRES)
INTEGER            NPDE, NCODE, IRES
real               T, X, V(*), 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: NCODE — INTEGER *Input*

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

5: V(∗) — ***real*** array *Input*

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

6: ULEFT(NPDE) — ***real*** array *Input*

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

7: URIGHT(NPDE) — ***real*** array *Input*

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

8: FLUX(NPDE) — ***real*** array *Output*

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

9: 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 D03PSF 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 D03PSF 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 (8) and (9).

Its specification is:

```
      SUBROUTINE BNDARY(NPDE, NPTS, T, X, U, NCODE, V, VDOT, IBND, G,
     1                  IRES)
      INTEGER           NPDE, NPTS, NCODE, IBND, IRES
      real              T, X(NPTS), U(NPDE,NPTS), V(*), VDOT(*), 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,NPTS) — ***real*** array *Input*

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

Note. If banded matrix algebra is to be used then the functions G_i^L and G_i^R may depend on the value of $U_i(x, t)$ at the boundary point and the two adjacent points only.

6: NCODE — INTEGER *Input*

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

7: V(*) — ***real*** array *Input*

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

8: VDOT(*) — ***real*** 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 G_j^L and G_j^R, for $j = 1, 2, \ldots,$ NPDE.

9: 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 $x = b$.

10: G(NPDE) — ***real*** array *Output*

On exit: G(i) must contain the ith component of either G_i^L or G_i^R in (8) and (9), depending on the value of 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, 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 D03PSF 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 D03PSF is called. Parameters 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 $a \leq x \leq b$.

Its specification is:

```
SUBROUTINE UVINIT(NPDE, NPTS, NXI, X, XI, U, NCODE, V)
INTEGER            NPDE, NPTS, NXI, NCODE
real               X(NPTS), XI(*), U(NPDE,NPTS), V(*)
```

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*** array *Input*

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

5: XI(*) — ***real*** array *Input*

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

6: U(NPDE,NPTS) — ***real*** array *Output*

On exit: U(i,j) must contain the value of component $U_i(x_j,t_0)$ for $i = 1,2,\ldots,$NPDE; $j = 1,2,\ldots,$NPTS.

7: NCODE — INTEGER *Input*

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

8: V($*$) — ***real*** array *Output*

On exit: V(i) must contain the value of component $V_i(t_0)$ for $i = 1,2,\ldots,$NCODE.

UVINIT must be declared as EXTERNAL in the (sub)program from which D03PSF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: U(NEQN) — ***real*** array *Output*

On exit: U(NPDE $\times$ $(j-1)+i$) contains the computed solution $U_i(x_j,t)$, for $i = 1,2,\ldots,$NPDE; $j = 1,2,\ldots,$NPTS, and U(NPTS$\times$NPDE$+k$) contains $V_k(t)$, for $k = 1,2,\ldots,$NCODE, all 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*** 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: X(1) < X(2) < ... < X(NPTS).

11: NCODE — INTEGER *Input*

On entry: the number of coupled ODE components.

Constraint: NCODE ≥ 0.

12: ODEDEF — SUBROUTINE, supplied by the user. *External Procedure*

ODEDEF must evaluate the functions R, which define the system of ODEs, as given in (4). If the user wishes to compute the solution of a system of PDEs only (i.e., NCODE = 0), ODEDEF must be the dummy routine D03PEK. (D03PEK 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 ODEDEF(NPDE, T, NCODE, V, VDOT, NXI, XI, UCP, UCPX,
     1                  UCPT, R, IRES)
      INTEGER           NPDE, NCODE, NXI, IRES
      real              T, V(*), VDOT(*), XI(*), UCP(NPDE,*),
     1                  UCPX(NPDE,*), UCPT(NPDE,*), R(*)
```

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: NCODE — INTEGER *Input*

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

4: V(*) — ***real*** array *Input*

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

5: VDOT(*) — ***real*** array *Input*

On entry: 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(*) — ***real*** array *Input*

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

8: UCP(NPDE,*) — ***real*** array *Input*

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

9: UCPX(NPDE,*) — ***real*** array *Input*

On entry: UCPX(i, j) contains the value of $\partial U_i(x, t)/\partial x$ at the coupling point $x = \xi_j$, for $i = 1, 2, \ldots,$ NPDE; $j = 1, 2, \ldots,$ NXI.

10: UCPT(NPDE,*) — ***real*** array *Input*

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

11: R(*) — ***real*** array *Output*

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

$$R = L - M\dot{V} - NU_t^*, \tag{10}$$

or

$$R = -M\dot{V} - NU_t^*. \tag{11}$$

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

12: IRES — INTEGER *Input/Output*

On entry: the form of R that must be returned in the array R. If IRES = 1, then the equation (10) above must be used. If IRES = −1, then the equation (11) above must be used.

On exit: should usually remain unchanged. However, the user 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)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 D03PSF returns to the calling (sub)program with the error indicator set to IFAIL = 4.

ODEDEF must be declared as EXTERNAL in the (sub)program from which D03PSF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

13: NXI — INTEGER *Input*

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

Constraints:

NXI = 0 if NCODE = 0,
NXI ≥ 0 if NCODE > 0.

14: XI(∗) — ***real*** array *Input*

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

Constraint: X(1) ≤ XI(1) < XI(2) < ... < XI(NXI) ≤ X(NPTS).

15: NEQN — INTEGER *Input*

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

Constraint: NEQN = NPDE × NPTS + NCODE.

16: RTOL(∗) — ***real*** 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) ≥ 0.0 for all relevant i.

17: ATOL(∗) — ***real*** 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.

Constraint: ATOL(i) ≥ 0.0 for all relevant i.

18: ITOL — INTEGER *Input*

On entry: a value to indicate the form of the local error test. If e_i is the estimated local error for U(i), $i = 1,2,\ldots$,NEQN, and $\|\ \|$ denotes the norm, then the error test to be satisfied is $\|e_i\| < 1.0$. ITOL indicates to D03PSF whether to interpret either or both of RTOL and ATOL as a vector or scalar in the formation of the weights w_i used in the calculation of the norm (see the description of the parameter NORM below):

ITOL	RTOL	ATOL	w_i
1	scalar	scalar	RTOL(1) × |U(i)| + ATOL(1)
2	scalar	vector	RTOL(1) × |U(i)| + ATOL(i)
3	vector	scalar	RTOL(i) × |U(i)| + ATOL(1)
4	vector	vector	RTOL(i) × |U(i)| + ATOL(i)

Constraint: 1 ≤ ITOL ≤ 4.

19: NORM — CHARACTER*1 *Input*

On entry: the type of norm to be used. Two options are available:

'1' – averaged L_1 norm.
'2' – averaged L_2 norm.

If U_{norm} denotes the norm of the vector U of length NEQN, then for the averaged L_1 norm

$$U_{\mathrm{norm}} = \frac{1}{\mathrm{NEQN}} \sum_{i=1}^{\mathrm{NEQN}} U(i)/w_i,$$

and for the averaged L_2 norm

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

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

Constraint: NORM = '1' or '2'.

20: LAOPT — CHARACTER*1 *Input*

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

'F' – full matrix routines to be used;
'B' – banded matrix routines to be used;
'S' – sparse matrix routines to be used.

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

Note. The user is recommended to use the banded option when no coupled ODEs are present (NCODE = 0). Also, the banded option should not be used if the boundary conditions involve solution components at points other than the boundary and the immediately adjacent two points.

21: ALGOPT(30) — ***real*** array *Input*

On entry: ALGOPT may be set to control various options available in the integrator. If the user wishes 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 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 parameter 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 initialise 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, i.e., 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 < ALGOPT(29) < 1.0, with smaller values biasing the algorithm towards maintaining sparsity at the expense of numerical stability. If ALGOPT(29) lies outside the 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 the relative pivot threshold during subsequent Jacobian decompositions (see ALGOPT(29)) below which an internal error is invoked. ALGOPT(30) must be greater than zero, otherwise the default value is used. 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 matrix 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 D03PSF. Remeshing can be switched off or on at specified times by using appropriate values for the parameters NRMESH and TRMESH at each call.

23: NXFIX — INTEGER *Input*

On entry: the number of fixed mesh points.

Constraint: $0 \leq$ NXFIX $\leq$ 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*** array *Input*

Note: the dimension of the array XFIX must be at least max(1,NXFIX).

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

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

Note. The positions of the fixed mesh points in the array X(NPTS) 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. The user 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 parameter DXMESH below. 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 (see below).

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 D03PSF to give greater flexibility over the times of remeshing.

26: DXMESH — ***real*** *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

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

or

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

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

Constraint: DXMESH ≥ 0.0.

27: TRMESH — ***real*** *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 D03PSF 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: $0 \leq$ IPMINF ≤ 2.

29: XRATIO — ***real*** *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

$$(x_i - x_{i-1})/\mathrm{XRATIO} < x_{i+1} - x_i < \mathrm{XRATIO} \times (x_i - x_{i-1})$$

Suggested value: XRATIO = 1.5.

Constraint: XRATIO > 1.0.

30: CONST — ***real*** *Input*

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

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

(see Furzeland [6]). CONST gives the user more control over the mesh distribution e.g. decreasing CONST allows more clustering. A typical value is 2/(NPTS−1), but the user is 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: CONST = 2.0/(NPTS−1).

Constraint: 0.1/(NPTS − 1) $\leq$ CONST $\leq$ 10.0/(NPTS − 1).

31: MONITF — SUBROUTINE, supplied by the user. *External Procedure*

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

If the user specifies REMESH = .FALSE., i.e., no remeshing, then MONITF will not be called and the dummy routine D03PEL may be used for MONITF. (D03PEL 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 MONITF(T, NPTS, NPDE, X, U, FMON)
INTEGER          NPTS, NPDE
real             T, X(NPTS), U(NPDE,NPTS), FMON(NPTS)
```

1: T — ***real*** *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*** 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*** array *Input*

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

6: FMON(NPTS) — ***real*** array *Output*

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

Constraint: FMON(i) ≥ 0.

MONITF must be declared as EXTERNAL in the (sub)program from which D03PSF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

32: W(NW) — ***real*** array *Workspace*

33: NW — INTEGER *Input*

On entry: the dimension of the array W as declared in the (sub)program from which D03PSF is called. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',

$$\text{NW} \geq \text{NEQN} \times \text{NEQN} + \text{NEQN} + \text{NWKRES} + \text{LENODE},$$

LAOPT = 'B',

$$\text{NW} \geq (3 \times \text{MLU} + 1) \times \text{NEQN} + \text{NWKRES} + \text{LENODE},$$

LAOPT = 'S',

$$\text{NW} \geq 4 \times \text{NEQN} + 11 \times \text{NEQN}/2 + 1 + \text{NWKRES} + \text{LENODE}.$$

Where MLU = the lower or upper half bandwidths, and

MLU = $3 \times$ NPDE $- 1$, for PDE problems only, and,

MLU = NEQN $- 1$, for coupled PDE/ODE problems.

$$\text{NWKRES} = \text{NPDE}\times(2\times\text{NPTS}+6\times\text{NXI}+3\times\text{NPDE}+26)+\text{NXI}+\text{NCODE}+7\times\text{NPTS}+\text{NXFIX}+1$$

when NCODE > 0 and NXI > 0;

$$\text{NWKRES} = \text{NPDE} \times (2 \times \text{NPTS} + 3 \times \text{NPDE} + 32) + \text{NCODE} + 7 \times \text{NPTS} + \text{NXFIX} + 2$$

when NCODE > 0 and NXI $= 0$;

NWKRES = NPDE × (2 × NPTS + 3 × NPDE + 32) + 7 × NPTS + NXFIX + 3

when NCODE = 0.

LENODE = (6 + int(ALGOPT(2))) × NEQN + 50, when the BDF method is used and,

LENODE = 9 × NEQN + 50, when the Theta method is used.

Note. When LAOPT = 'S', the value of NW may be too small when supplied to the integrator. An estimate of the minimum size of NW is printed on the current error message unit if ITRACE > 0 and the routine returns with IFAIL = 15.

34: 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 BDF method last used in the time integration, if applicable. When the Theta method is used IW(4) contains no useful information.

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.

35: NIW — INTEGER *Input*

On entry: the dimension of the array IW. Its size depends on the type of matrix algebra selected:

LAOPT = 'F',

NIW ≥ 25,

LAOPT = 'B',

NIW ≥ NEQN + NXFIX + 25,

LAOPT = 'S',

NIW ≥ 25 × NEQN + NXFIX + 25.

Note. When LAOPT = 'S', the value of NIW may be too small when supplied to the integrator. An estimate of the minimum size of NIW is printed on the current error message unit if ITRACE > 0 and the routine returns with IFAIL = 15.

36: 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.

ITASK = 4

normal computation of output values U at t = TOUT but without overshooting $t = t_{\mathrm{crit}}$ where t_{crit} is described under the parameter ALGOPT.

ITASK = 5

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

Constraint: $1 \leq$ ITASK ≤ 5.

37: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03PSF 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 subchapter D02M-N of the NAG Fortran Library.

38: 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, IFAIL, NRMESH and TRMESH may be reset between calls to D03PSF.

Constraint: $0 \leq$ IND ≤ 1.

On exit: IND = 1.

39: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, TS $\geq$ TOUT,
- or TOUT $-$ TS is too small,
- or ITASK $\neq$ 1, 2, 3, 4 or 5,
- or at least one of the coupling points defined in array XI is outside the interval [X(1),X(NPTS)],
- or the coupling points are not in strictly increasing order,
- or NPTS < 3,
- or NPDE < 1,
- or LAOPT $\neq$ 'F' , 'B' or 'S',
- or ITOL $\neq$ 1, 2, 3 or 4,
- 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 NCODE and NXI are incorrectly defined,

or IND = 1 on initial entry to D03PSF,

or NEQN $\neq$ NPDE $\times$ NPTS + NCODE,

or an element of RTOL or ATOL < 0.0,

or corresponding elements of RTOL and ATOL are both 0.0,

or NORM $\neq$ '1' or '2',

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 ≤ 1.0,

or CONST not in the range 0.1/(NPTS $-$ 1) to 10.0/(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. 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, BNDARY or ODEDEF, 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, 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 the user-supplied routines, PDEDEF, NUMFLX, BNDARY or ODEDEF, 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 ATOL and RTOL 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 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 advisory message unit when ITRACE $\geq$ 1). 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) has been taken.

IFAIL = 13

Some error weights w_i became zero during the time integration (see 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

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.

IFAIL = 15

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

IFAIL = 16

REMESH has been changed between calls to D03PSF.

IFAIL = 17

FMON is negative at one or more mesh points, or zero mesh spacing has been obtained due to an inappropriate choice of monitor function.

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 accuracy parameters, ATOL and RTOL.

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 ALGOPT(13). 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 [1].

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

9 Example

For this routine two examples are presented, in Section 9.1 and Section 9.2. In the example programs distributed to sites, there is a single example program for D03PSF, with a main program:

```
*     D03PSF Example Program Text
*     Mark 18 Revised.  NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PSF 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 Section 9.1.1 and Section 9.2.1 respectively.

9.1 Example 1

This example is a simple model of the advection and diffusion of a cloud of material:

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

for $x \in [0,1]$ and $t \leq 0 \leq 0.3$. In this example the constant wind speed $W = 1$ and the diffusion coefficient $C = 0.002$.

The cloud does not reach the boundaries during the time of integration, and so the two (physical) boundary conditions are simply $U(0,t) = U(1,t) = 0.0$, and the initial condition is

$$U(x,0) = \sin\left(\pi\frac{x-a}{b-a}\right), \text{for } a \leq x \leq b,$$

and $U(x,0) = 0$ elsewhere, where $a = 0.2$ and $b = 0.4$.

The numerical flux is simply $\hat{F} = WU_L$.

The monitor function for remeshing is taken to be the absolute value of the second derivative of U.

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, NCODE, NXI, NXFIX, NEQN, NIW, NWKRES,
     +                 LENODE, MLU, NW, INTPTS, ITYPE
      PARAMETER        (NPDE=1,NPTS=61,NCODE=0,NXI=0,NXFIX=0,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=25+NXFIX+NEQN,
     +                 NWKRES=NPDE*(3*NPTS+3*NPDE+32)+7*NPTS+3,
     +                 LENODE=11*NEQN+50,MLU=3*NPDE-1,NW=(3*MLU+1)
     +                 *NEQN+NWKRES+LENODE,INTPTS=7,ITYPE=1)
*     .. Scalars in Common ..
      real             P
*     .. Local Scalars ..
      real             CONST, DXMESH, TOUT, TRMESH, TS, XRATIO, XX
      INTEGER          I, IFAIL, IND, IPMINF, IT, ITASK, ITOL, ITRACE,
     +                 M, NRMESH
      LOGICAL          REMESH
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NPDE,NPTS),
     +                 UOUT(NPDE,INTPTS,ITYPE), W(NW), X(NPTS), XFIX(1),
     +                 XI(1), XOUT(INTPTS)
      INTEGER          IW(NIW)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         BNDRY1, D03PEK, D03PSF, D03PZF, MONIT1, NMFLX1,
     +                 PDEF1, UVIN1
*     .. Common blocks ..
      COMMON           /PI/P
*     .. Save statement ..
      SAVE             /PI/
*     .. Data statements ..
      DATA             XOUT(1)/0.2e+0/, XOUT(2)/0.3e+0/,
     +                 XOUT(3)/0.4e+0/, XOUT(4)/0.5e+0/,
     +                 XOUT(5)/0.6e+0/, XOUT(6)/0.7e+0/, XOUT(7)/0.8e+0/
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Example 1'
      WRITE (NOUT,*)
*
      XX = 0.0e0
      P = X01AAF(XX)
      ITRACE = 0
      ITOL = 1
      NORM = '1'
      ATOL(1) = 0.1e-3
      RTOL(1) = 0.1e-3
      WRITE (NOUT,99998) NPTS, ATOL, RTOL
*
```

```
*     Initialise mesh ..
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
      XFIX(1) = 0.0e0
*
*     Set remesh parameters..
*
      REMESH = .TRUE.
      NRMESH = 3
      DXMESH = 0.0e0
      TRMESH = 0.0e0
      CONST = 2.0e0/(NPTS-1.0e0)
      XRATIO = 1.5e0
      IPMINF = 0
*
      XI(1) = 0.0e0
      LAOPT = 'B'
      IND = 0
      ITASK = 1
*
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
*     b.d.f. integration
      ALGOPT(1) = 1.0e0
      ALGOPT(13) = 0.5e-2
*
*     Loop over output value of t
*
      TS = 0.0e0
      TOUT = 0.0e0
      DO 60 IT = 1, 3
         TOUT = IT*0.1e0
         IFAIL = 0
*
         CALL D03PSF(NPDE,TS,TOUT,PDEF1,NMFLX1,BNDRY1,UVIN1,U,NPTS,X,
     +               NCODE,D03PEK,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,
     +               ALGOPT,REMESH,NXFIX,XFIX,NRMESH,DXMESH,TRMESH,
     +               IPMINF,XRATIO,CONST,MONIT1,W,NW,IW,NIW,ITASK,
     +               ITRACE,IND,IFAIL)
*
         WRITE (NOUT,99999) TS
         WRITE (NOUT,99996) (XOUT(I),I=1,INTPTS)
*         Interpolate at output points ..
         M = 0
         CALL D03PZF(NPDE,M,U,NPTS,X,XOUT,INTPTS,ITYPE,UOUT,IFAIL)
*
         WRITE (NOUT,99995) (UOUT(1,I,1),I=1,INTPTS)
   60 CONTINUE
*
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      RETURN
*
99999 FORMAT (' T = ',F6.3)
99998 FORMAT (/'  NPTS = ',I4,' ATOL = ',e10.3,' RTOL = ',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,'Approx U ',7F9.4,/)
      END
*
      SUBROUTINE UVIN1(NPDE,NPTS,NXI,X,XI,U,NCODE,V)
*     .. Scalar Arguments ..
      INTEGER          NCODE, NPDE, NPTS, NXI
*     .. Array Arguments ..
      real             U(NPDE,NPTS), V(*), X(NPTS), XI(*)
*     .. Scalars in Common ..
      real             P
*     .. Local Scalars ..
      real             TMP
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        SIN
*     .. Common blocks ..
      COMMON           /PI/P
*     .. Save statement ..
      SAVE             /PI/
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         IF (X(I).GT.0.2e0 .AND. X(I).LE.0.4e0) THEN
            TMP = P*(5.0e0*X(I)-1.0e0)
            U(1,I) = SIN(TMP)
         ELSE
            U(1,I) = 0.0e0
         END IF
   20 CONTINUE
      RETURN
      END
*
      SUBROUTINE PDEF1(NPDE,T,X,U,UX,NCODE,V,VDOT,P,C,D,S,IRES)
*     .. Scalar Arguments ..
      real             T, X
      INTEGER          IRES, NCODE, NPDE
*     .. Array Arguments ..
      real             C(NPDE), D(NPDE), P(NPDE,NPDE), S(NPDE), U(NPDE),
     +                 UX(NPDE), V(*), VDOT(*)
*     .. Executable Statements ..
      P(1,1) = 1.0e0
      C(1) = 0.2e-2
      D(1) = UX(1)
      S(1) = 0.0e0
      RETURN
      END
*
      SUBROUTINE BNDRY1(NPDE,NPTS,T,X,U,NCODE,V,VDOT,IBND,G,IRES)
*     .. Scalar Arguments ..
      real             T
      INTEGER          IBND, IRES, NCODE, NPDE, NPTS
*     .. Array Arguments ..
      real             G(NPDE), U(NPDE,NPTS), V(*), VDOT(*), X(NPTS)
*     .. Executable Statements ..
*     Zero solution at both boundaries
      IF (IBND.EQ.0) THEN
```

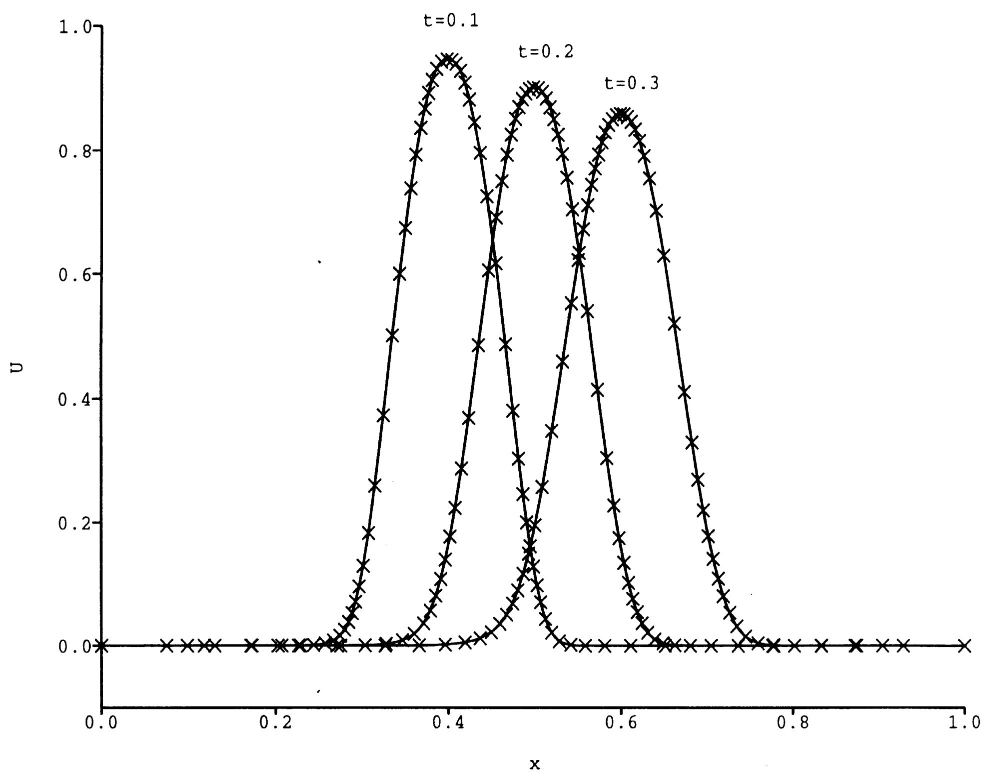
```
         G(1) = U(1,1)
      ELSE
         G(1) = U(1,NPTS)
      END IF
      RETURN
      END
*
      SUBROUTINE MONIT1(T,NPTS,NPDE,X,U,FMON)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              FMON(NPTS), U(NPDE,NPTS), X(NPTS)
*     .. Local Scalars ..
      real              H1, H2, H3
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         ABS
*     .. Executable Statements ..
*     Executable Statements ..
      DO 20 I = 2, NPTS - 1
         H1 = X(I) - X(I-1)
         H2 = X(I+1) - X(I)
         H3 = 0.5e0*(X(I+1)-X(I-1))
*       Second derivatives ..
         FMON(I) = ABS(((U(1,I+1)-U(1,I))/H2-(U(1,I)-U(1,I-1))/H1)/H3)
   20 CONTINUE
      FMON(1) = FMON(2)
      FMON(NPTS) = FMON(NPTS-1)
      RETURN
      END
*
      SUBROUTINE NMFLX1(NPDE,T,X,NCODE,V,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE), V(*)
*     .. Executable Statements ..
      FLUX(1) = ULEFT(1)
      RETURN
      END
```

9.1.2 Program Data

None.

9.1.3 Program Results

```
D03PSF Example Program Results


Example 1


 NPTS =    61 ATOL =  0.100E-03 RTOL =  0.100E-03

T =  0.100
X           0.2000    0.3000    0.4000    0.5000    0.6000    0.7000    0.8000
Approx U    0.0000    0.1198    0.9461    0.1182    0.0000    0.0000    0.0000

T =  0.200
X           0.2000    0.3000    0.4000    0.5000    0.6000    0.7000    0.8000
Approx U    0.0000    0.0007    0.1631    0.9015    0.1629    0.0001    0.0000

T =  0.300
X           0.2000    0.3000    0.4000    0.5000    0.6000    0.7000    0.8000
Approx U    0.0000    0.0000    0.0025    0.1924    0.8596    0.1946    0.0002

Number of integration steps in time =     92
Number of function evaluations =    443
Number of Jacobian evaluations =    39
Number of iterations =    231
```



9.2 Example 2

This example is a linear advection equation with a non-linear source term and discontinuous initial profile:

$$\frac{\partial u}{\partial t} + \frac{\partial u}{\partial x} = -pu(u-1)(u-\frac{1}{2}),$$

for $0 \le x \le 1$ and $t \ge 0$. The discontinuity is modelled by a ramp function of width 0.01 and gradient 100, so that the exact solution at any time $t \ge 0$ is

$$u(x,t) = 1.0 + \max(\min(\delta,0),-1),$$

where $\delta = 100(0.1-x+t)$. The initial profile is given by the exact solution. The characteristic points into the domain at $x=0$ and out of the domain at $x=1$, and so a physical boundary condition $u(0,t)=1$ is imposed at $x=0$, with a numerical boundary condition at $x=1$ which can be specified as $u(1,t)=0$ since the discontinuity does not reach $x=1$ during the time of integration.

The numerical flux is simply $\hat{F} = U_L$ at all times.

The remeshing monitor function (described below) is chosen to create an increasingly fine mesh towards the discontinuity in order to ensure good resolution of the discontinuity, but without loss of efficiency in the surrounding regions. However, refinement must be limited so that the time step required for stability does not become unrealistically small. The region of refinement must also keep up with the discontinuity as it moves across the domain, and hence it cannot be so small that the discontinuity moves out of the refined region between remeshing.

The above requirements mean that the use of the first or second spatial derivative of U for the monitor function is inappropriate; the large relative size of either derivative in the region of the discontinuity leads to extremely small mesh-spacing in a very limited region, and the solution is then far more expensive than for a very fine fixed mesh.

An alternative monitor function based on a cosine function proves very successful. It is only semi-automatic as it requires some knowledge of the solution (for problems without an exact solution an initial approximate solution can be obtained using a coarse fixed mesh). On each call to the user-supplied MONITF subroutine the discontinuity is located by finding the maximum spatial derivative of the solution. On the first call the desired width of the region of non-zero monitor function is set (this can be changed at a later time if desired). Then on each call the monitor function is assigned using a cosine function so that it has a value of one at the discontinuity down to zero at the edges of the predetermined region of refinement, and zero outside the region. Thus the monitor function and the subsequent refinement are limited, and the region is large enough to ensure that there is always sufficient refinement at the discontinuity.

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, NCODE, NXI, NXFIX, NEQN, NIW, NWKRES,
     +                 LENODE, MLU, NW, INTPTS, ITYPE
      PARAMETER        (NPDE=1,NPTS=61,NCODE=0,NXI=0,NXFIX=0,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=25+NXFIX+NEQN,
     +                 NWKRES=NPDE*(2*NPTS+3*NPDE+32)+7*NPTS+3,
     +                 LENODE=11*NEQN+50,MLU=3*NPDE-1,NW=(3*MLU+1)
     +                 *NEQN+NWKRES+LENODE,INTPTS=7,ITYPE=1)
```

```
*     .. Local Scalars ..
      real             CONST, DXMESH, TOUT, TRMESH, TS, XRATIO
      INTEGER          I, IFAIL, IND, IPMINF, IT, ITASK, ITOL, ITRACE,
     +                 M, NRMESH
      LOGICAL          REMESH
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NEQN),
     +                 UE(1,INTPTS), UOUT(1,INTPTS,ITYPE), W(NW),
     +                 X(NPTS), XFIX(1), XI(1), XOUT(INTPTS)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDRY2, D03PEK, D03PSF, D03PZF, EXACT, MONIT2,
     +                 NMFLX2, PDEF2, UVIN2
*     .. Data statements ..
      DATA             XOUT(1)/0.0e+0/, XOUT(2)/0.3e+0/,
     +                 XOUT(3)/0.4e+0/, XOUT(4)/0.5e+0/,
     +                 XOUT(5)/0.6e+0/, XOUT(6)/0.7e+0/, XOUT(7)/1.0e+0/
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Example 2'
      WRITE (NOUT,*)
*
      ITRACE = 0
      ITOL = 1
      NORM = '1'
      ATOL(1) = 0.5e-3
      RTOL(1) = 0.5e-1
      WRITE (NOUT,99998) NPTS, ATOL, RTOL
*
*     Initialise mesh ..
*
      DO 20 I = 1, NPTS
         X(I) = (I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
      XFIX(1) = 0.0e0
*
*     Set remesh parameters..
*
      REMESH = .TRUE.
      NRMESH = 5
      DXMESH = 0.0e0
      CONST = 1.0e0/(NPTS-1.0e0)
      XRATIO = 1.5e0
      IPMINF = 0
*
      XI(1) = 0.0e0
      LAOPT = 'B'
      IND = 0
      ITASK = 1
*
      DO 40 I = 1, 30
         ALGOPT(I) = 0.0e0
   40 CONTINUE
```

```
*     Theta integration ..
      ALGOPT(1) = 2.0e0
      ALGOPT(6) = 2.0e0
      ALGOPT(7) = 2.0e0
*     Max. time step ..
      ALGOPT(13) = 2.5e-3
*
      TS = 0.0e0
      TOUT = 0.0e0
      IFAIL = 0
      DO 80 IT = 1, 2
         TOUT = IT*0.2e0
         CALL D03PSF(NPDE,TS,TOUT,PDEF2,NMFLX2,BNDRY2,UVIN2,U,NPTS,X,
     +               NCODE,D03PEK,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,
     +               ALGOPT,REMESH,NXFIX,XFIX,NRMESH,DXMESH,TRMESH,
     +               IPMINF,XRATIO,CONST,MONIT2,W,NW,IW,NIW,ITASK,
     +               ITRACE,IND,IFAIL)
*
         WRITE (NOUT,99999) TS
         WRITE (NOUT,99996)
*        Interpolate at output points ..
         M = 0
         CALL D03PZF(NPDE,M,U,NPTS,X,XOUT,INTPTS,ITYPE,UOUT,IFAIL)
*
*        Check against exact solution ..
         CALL EXACT(TOUT,UE,XOUT,INTPTS)
         DO 60 I = 1, INTPTS
            WRITE (NOUT,99995) XOUT(I), UOUT(1,I,1), UE(1,I)
   60    CONTINUE
   80 CONTINUE
*
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      RETURN
*
99999 FORMAT (' T = ',F6.3)
99998 FORMAT (/' NPTS = ',I4,' ATOL = ',e10.3,' RTOL = ',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 (8X,'X',8X,'Approx U',4X,'Exact U',/)
99995 FORMAT (3(3X,F9.4))
      END
*
      SUBROUTINE UVIN2(NPDE,NPTS,NXI,X,XI,U,NCODE,V)
*     .. Scalar Arguments ..
      INTEGER           NCODE, NPDE, NPTS, NXI
*     .. Array Arguments ..
      real              U(NPDE,NPTS), V(*), X(NPTS), XI(*)
*     .. Local Scalars ..
      real              T
*     .. External Subroutines ..
      EXTERNAL          EXACT
*     .. Executable Statements ..
      T = 0.0e0
      CALL EXACT(T,U,X,NPTS)
      RETURN
      END
```

```
*
      SUBROUTINE PDEF2(NPDE,T,X,U,UX,NCODE,V,VDOT,P,C,D,S,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              C(NPDE), D(NPDE), P(NPDE,NPDE), S(NPDE), U(NPDE),
     +                  UX(NPDE), V(*), VDOT(*)
*     .. Executable Statements ..
      P(1,1) = 1.0e0
      C(1) = 0.0e0
      D(1) = 0.0e0
      S(1) = -1.0e2*U(1)*(U(1)-1.0e0)*(U(1)-0.5e0)
      RETURN
      END
*
      SUBROUTINE BNDRY2(NPDE,NPTS,T,X,U,NCODE,V,VDOT,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,NPTS), V(*), VDOT(*), X(NPTS)
*     .. Local Arrays ..
      real              UE(1,1)
*     .. External Subroutines ..
      EXTERNAL          EXACT
*     .. Executable Statements ..
*     Solution known to be constant at both boundaries ..
      IF (IBND.EQ.0) THEN
         CALL EXACT(T,UE,X(1),1)
         G(1) = UE(1,1) - U(1,1)
      ELSE
         CALL EXACT(T,UE,X(NPTS),1)
         G(1) = UE(1,1) - U(1,NPTS)
      END IF
      RETURN
      END
*
      SUBROUTINE NMFLX2(NPDE,T,X,NCODE,V,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE), V(*)
*     .. Executable Statements ..
      FLUX(1) = ULEFT(1)
      RETURN
      END
*
      SUBROUTINE MONIT2(T,NPTS,NPDE,X,U,FMON)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              FMON(NPTS), U(NPDE,NPTS), X(NPTS)
```

```
*     .. Local Scalars ..
      real              H1, PI, UX, UXMAX, XL, XLEFT, XMAX, XR, XRIGHT,
     +                  XA, XX
      INTEGER           I, ICOUNT
*     .. External Functions ..
      real              X01AAF
      EXTERNAL          X01AAF
*     .. Intrinsic Functions ..
      INTRINSIC         ABS, COS
*     .. Data statements ..
      DATA              XA/0.0e0/, ICOUNT/0/
*     .. Executable Statements ..
      XX = 0.0e0
      PI = X01AAF(XX)
*     Locate shock ..
      UXMAX = 0.0e0
      XMAX = 0.0e0
      DO 20 I = 2, NPTS - 1
         H1 = X(I) - X(I-1)
         UX = ABS((U(1,I)-U(1,I-1))/H1)
         IF (UX.GT.UXMAX) THEN
            UXMAX = UX
            XMAX = X(I)
         END IF
   20 CONTINUE
*     Assign width (on first call only) ..
      IF (ICOUNT.EQ.0) THEN
         ICOUNT = 1
         XLEFT = XMAX - X(1)
         XRIGHT = X(NPTS) - XMAX
         IF (XLEFT.GT.XRIGHT) THEN
            XA = XRIGHT
         ELSE
            XA = XLEFT
         END IF
      END IF
      XL = XMAX - XA
      XR = XMAX + XA
*     Assign monitor function ..
      DO 40 I = 1, NPTS
         IF (X(I).GT.XL .AND. X(I).LT.XR) THEN
            FMON(I) = 1.0e0 + COS(PI*(X(I)-XMAX)/XA)
         ELSE
            FMON(I) = 0.0e0
         END IF
   40 CONTINUE
      RETURN
      END
*
      SUBROUTINE EXACT(T,U,X,NPTS)
*     Exact solution (for comparison and b.c. purposes)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPTS
*     .. Array Arguments ..
      real              U(1,NPTS), X(*)
```

```
*     .. Local Scalars ..
      real              DEL, PSI, RM, RN, S
      INTEGER           I
*     .. Executable Statements ..
      S = 0.1e0
      DEL = 0.01e0
      RM = -1.0e0/DEL
      RN = 1.0e0 + S/DEL
      DO 20 I = 1, NPTS
         PSI = X(I) - T
         IF (PSI.LT.S) THEN
            U(1,I) = 1.0e0
         ELSE IF (PSI.GT.(DEL+S)) THEN
            U(1,I) = 0.0e0
         ELSE
            U(1,I) = RM*PSI + RN
         END IF
   20 CONTINUE
      RETURN
      END
```

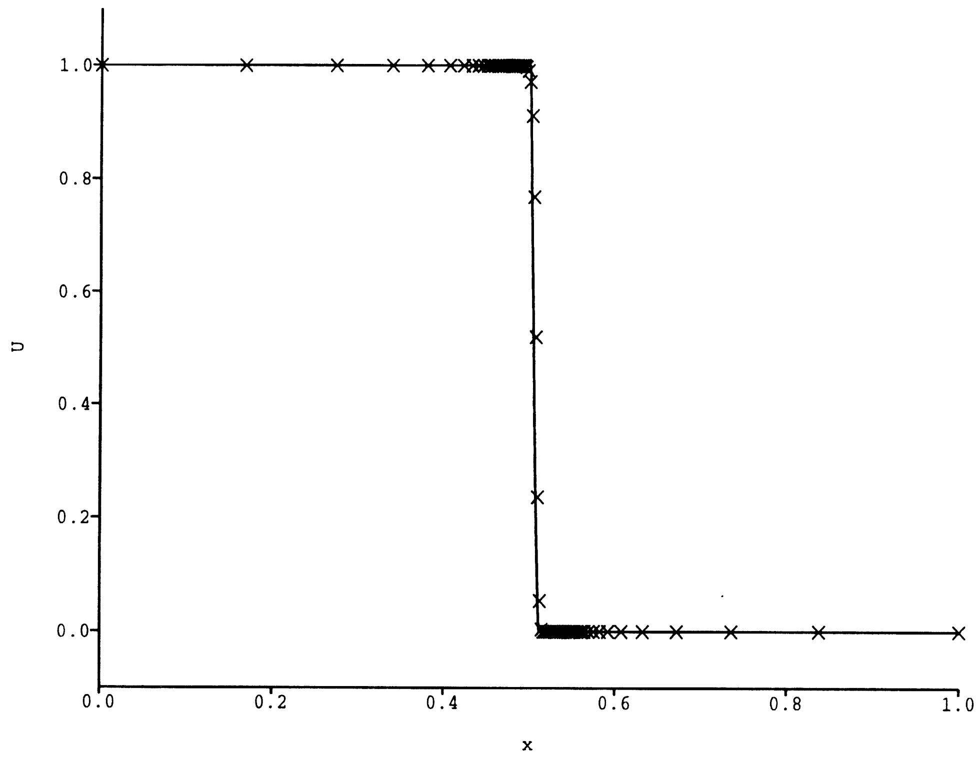
9.2.2 Program Data

None.

9.2.3 Program Results

```
 D03PSF Example Program Results



 Example 2


 NPTS =    61 ATOL =   0.500E-03 RTOL =   0.500E-01

 T =   0.200
          X          Approx U     Exact U

       0.0000        1.0000       1.0000
       0.3000        0.9536       1.0000
       0.4000        0.0000       0.0000
       0.5000        0.0000       0.0000
       0.6000        0.0000       0.0000
       0.7000        0.0000       0.0000
       1.0000        0.0000       0.0000
 T =   0.400
          X          Approx U     Exact U

       0.0000        1.0000       1.0000
       0.3000        1.0000       1.0000
       0.4000        1.0000       1.0000
       0.5000        0.9750       1.0000
       0.6000        0.0000       0.0000
       0.7000        0.0000       0.0000
       1.0000        0.0000       0.0000
```

```
Number of integration steps in time =    672
 Number of function evaluations =   1515
 Number of Jacobian evaluations =     1
 Number of iterations =      2
```



D03PUF – NAG Fortran Library Routine Document

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

D03PUF calculates a numerical flux function using Roe's Approximate Riemann Solver for the Euler equations in conservative form. The routine is designed primarily for use with the upwind discretisation routines D03PFF, D03PLF or D03PSF, but may also be applicable to other conservative upwind schemes requiring numerical flux functions.

2 Specification

```
SUBROUTINE D03PUF(ULEFT, URIGHT, GAMMA, FLUX, IFAIL)
INTEGER          IFAIL
real             ULEFT(3), URIGHT(3), GAMMA, FLUX(3)
```

3 Description

D03PUF calculates a numerical flux function at a single spatial point using Roe's Approximate Riemann Solver [1] for the Euler equations (for a perfect gas) in conservative form. The user must supply the *left* and *right* solution values at the point where the numerical flux is required, i.e. the initial left and right states of the Riemann problem defined below. In the routines D03PFF, D03PLF and D03PSF, the left and right solution values are derived automatically from the solution values at adjacent spatial points and supplied to the subroutine argument NUMFLX from which the user may call D03PUF. The Euler equations for a perfect gas in conservative form are:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0, \tag{1}$$

with

$$U = \begin{bmatrix} \rho \\ m \\ e \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} m \\ \frac{m^2}{\rho} + (\gamma - 1)\left(e - \frac{m^2}{2\rho}\right) \\ \frac{me}{\rho} + \frac{m}{\rho}(\gamma - 1)\left(e - \frac{m^2}{2\rho}\right) \end{bmatrix}, \tag{2}$$

where ρ is the density, m is the momentum, e is the specific total energy, and γ is the (constant) ratio of specific heats. The pressure p is given by

$$p = (\gamma - 1)\left(e - \frac{\rho u^2}{2}\right), \tag{3}$$

where $u = m/\rho$ is the velocity.

The routine calculates the Roe approximation to the numerical flux function $F(U_L, U_R) = F(U^*(U_L, U_R))$, where $U = U_L$ and $U = U_R$ are the left and right solution values, and $U^*(U_L, U_R)$ is the intermediate state $\omega(0)$ arising from the similarity solution $U(y,t) = \omega(y/t)$ of the Riemann problem defined by

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial y} = 0, \tag{4}$$

with U and F as in (2), and initial piecewise constant values $U = U_L$ for $y < 0$ and $U = U_R$ for $y > 0$. The spatial domain is $-\infty < y < \infty$, where $y = 0$ is the point at which the numerical flux is required. This implementation of Roe's scheme for the Euler equations uses the so-called parameter-vector method described in [1].

4 References

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

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

[3] Quirk J J (1994) A contribution to the great Riemann solver debate *Internat. J. Numer. Methods Fluids* **18** 555–574

5 Parameters

1: ULEFT(3) — ***real*** array *Input*

On entry: ULEFT(i) must contain the left value of the component U_i for $i = 1,2,3$. That is, ULEFT(1) must contain the left value of ρ, ULEFT(2) must contain the left value of m and ULEFT(3) must contain the left value of e.

2: URIGHT(3) — ***real*** array *Input*

On entry: URIGHT(i) must contain the right value of the component U_i for $i = 1,2,3$. That is, URIGHT(1) must contain the right value of ρ, URIGHT(2) must contain the right value of m and URIGHT(3) must contain the right value of e.

3: GAMMA — ***real*** *Input*

On entry: the ratio of specific heats γ.

Constraint: GAMMA > 0.0.

4: FLUX(3) — ***real*** array *Output*

On exit: FLUX(i) contains the numerical flux component $\hat{F}_i$ for $i = 1,2,3$.

5: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

Note: if the left and/or right values of ρ or p (from (3)) are found to be negative, then the routine will terminate with an error exit (IFAIL = 2). If the routine is being called from the user-supplied subroutine NUMFLX in D03PFF etc., then a **soft fail** option (IFAIL = 1 or −1) is recommended so that a recalculation of the current time step can be forced using the IRES parameter.

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, GAMMA $\leq$ 0.0.

IFAIL = 2

On entry, the left and/or right density or pressure value is less than 0.0.

7 Accuracy

The routine performs an exact calculation of the Roe numerical flux function, and so the result will be accurate to machine precision.

8 Further Comments

The routine must only be used to calculate the numerical flux for the Euler equations in exactly the form given by (2), with ULEFT(i) and URIGHT(i) containing the left and right values of ρ, m and e for $i = 1, 2, 3$ respectively. It should be noted that Roe's scheme, in common with all Riemann solvers, may be unsuitable for some problems (see Quirk [3] for examples). In particular Roe's scheme does not satisfy an 'entropy condition' which guarantees that the approximate solution of the PDE converges to the correct physical solution, and hence it may admit non-physical solutions such as expansion shocks. The algorithm used in this routine does not detect or correct any entropy violation. The time taken by the routine is independent of the input parameters.

9 Example

See Example 2 in D03PLF.

D03PVF – NAG Fortran Library Routine Document

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

D03PVF calculates a numerical flux function using Osher's Approximate Riemann Solver for the Euler equations in conservative form. The routine is designed primarily for use with the upwind discretisation routines D03PFF, D03PLF or D03PSF, but may also be applicable to other conservative upwind schemes requiring numerical flux functions.

2 Specification

```
SUBROUTINE D03PVF(ULEFT, URIGHT, GAMMA, PATH, FLUX, IFAIL)
INTEGER          IFAIL
real             ULEFT(3), URIGHT(3), GAMMA, FLUX(3)
CHARACTER*1      PATH
```

3 Description

D03PVF calculates a numerical flux function at a single spatial point using Osher's Approximate Riemann Solver ([1], [2]) for the Euler equations (for a perfect gas) in conservative form. The user must supply the *left* and *right* solution values at the point where the numerical flux is required, i.e. the initial left and right states of the Riemann problem defined below. In the routines D03PFF, D03PLF and D03PSF, the left and right solution values are derived automatically from the solution values at adjacent spatial points and supplied to the subroutine argument NUMFLX from which the user may call D03PVF. The Euler equations for a perfect gas in conservative form are:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0, \tag{1}$$

with

$$U = \begin{bmatrix} \rho \\ m \\ e \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} m \\ \dfrac{m^2}{\rho} + (\gamma - 1)\left(e - \dfrac{m^2}{2\rho}\right) \\ \dfrac{me}{\rho} + \dfrac{m}{\rho}(\gamma - 1)\left(e - \dfrac{m^2}{2\rho}\right) \end{bmatrix}, \tag{2}$$

where ρ is the density, m is the momentum, e is the specific total energy, and γ is the (constant) ratio of specific heats. The pressure p is given by

$$p = (\gamma - 1)\left(e - \frac{\rho u^2}{2}\right), \tag{3}$$

where $u = m/\rho$ is the velocity.

The routine calculates the Osher approximation to the numerical flux function $F(U_L, U_R) = F(U^*(U_L, U_R))$, where $U = U_L$ and $U = U_R$ are the left and right solution values, and $U^*(U_L, U_R)$ is the intermediate state $\omega(0)$ arising from the similarity solution $U(y,t) = \omega(y/t)$ of the Riemann problem defined by

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial y} = 0, \tag{4}$$

with U and F as in (2), and initial piecewise constant values $U = U_L$ for $y < 0$ and $U = U_R$ for $y > 0$. The spatial domain is $-\infty < y < \infty$, where $y = 0$ is the point at which the numerical flux is required. Osher's solver carries out an integration along a path in the phase space of U consisting of subpaths which are piecewise parallel to the eigenvectors of the Jacobian of the PDE system. There are two variants of the Osher solver termed O (original) and P (physical), which differ in the order in which the subpaths are taken. The P-variant is generally more efficient, but in some rare cases may fail (see [1] for details). The parameter PATH specifies which variant is to be used. The algorithm for Osher's solver for the Euler equations is given in detail in the Appendix of [2].

4 References

[1] Hemker P W and Spekreijse S P (1986) Multiple grid and Osher's scheme for the efficient solution of the steady Euler equations *Applied Numerical Mathematics* **2** 475–493

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

[3] Quirk J J (1994) A contribution to the great Riemann solver debate *Internat. J. Numer. Methods Fluids* **18** 555–574

5 Parameters

1: ULEFT(3) — ***real*** array *Input*

On entry: ULEFT(i) must contain the left value of the component U_i for $i = 1,2,3$. That is, ULEFT(1) must contain the left value of ρ, ULEFT(2) must contain the left value of m and ULEFT(3) must contain the left value of e.

2: URIGHT(3) — ***real*** array *Input*

On entry: URIGHT(i) must contain the right value of the component U_i for $i = 1,2,3$. That is, URIGHT(1) must contain the right value of ρ, URIGHT(2) must contain the right value of m and URIGHT(3) must contain the right value of e.

3: GAMMA — ***real*** *Input*

On entry: the ratio of specific heats γ.

Constraint: GAMMA > 0.0.

4: PATH — CHARACTER*1 *Input*

On entry: the variant of the Osher scheme. The possible choices are 'O' (original) and 'P' (physical).

Constraint: PATH = 'O' or 'P'.

5: FLUX(3) — ***real*** array *Output*

On exit: FLUX(i) contains the numerical flux component $\hat{F}_i$ for $i = 1,2,3$.

6: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

Note: if the left and/or right values of ρ or p (from (3)) are found to be negative then the routine will terminate with an error exit (IFAIL = 2). If the routine is being called from the user-supplied subroutine NUMFLX in D03PFF etc., then a **soft fail** option (IFAIL = 1 or −1) is recommended so that a recalculation of the current time step can be forced using the IRES parameter.

6 Errors Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, GAMMA ≤ 0.0,
or PATH ≠ 'O' or 'P'.

IFAIL = 2

On entry, the left and/or right density or pressure value is less than 0.0.

7 Accuracy

The routine performs an exact calculation of the Osher numerical flux function, and so the result will be accurate to machine precision.

8 Further Comments

The routine must only be used to calculate the numerical flux for the Euler equations in exactly the form given by (2), with ULEFT(i) and URIGHT(i) containing the left and right values of ρ, m and e for $i = 1, 2, 3$ respectively. It should be noted that Osher's scheme, in common with all Riemann solvers, may be unsuitable for some problems (see Quirk [3] for examples). The time taken by the routine depends on the input parameter PATH and on the left and right solution values, since inclusion of each subpath depends on the signs of the eigenvalues. In general this cannot be determined in advance.

9 Example

See Example 2 in D03PLF.

D03PWF – NAG Fortran Library Routine Document

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

D03PWF calculates a numerical flux function using a modified HLL (Harten-Lax-van Leer) Approximate Riemann Solver for the Euler equations in conservative form. The routine is designed primarily for use with the upwind discretisation routines D03PFF, D03PLF or D03PSF, but may also be applicable to other conservative upwind schemes requiring numerical flux functions.

2 Specification

```
SUBROUTINE D03PWF(ULEFT, URIGHT, GAMMA, FLUX, IFAIL)
INTEGER          IFAIL
real             ULEFT(3), URIGHT(3), GAMMA, FLUX(3)
```

3 Description

D03PWF calculates a numerical flux function at a single spatial point using a modified HLL (Harten-Lax-van Leer) Approximate Riemann Solver (see [1] [2] [3]) for the Euler equations (for a perfect gas) in conservative form. The user must supply the *left* and *right* solution values at the point where the numerical flux is required, i.e., the initial left and right states of the Riemann problem defined below. In the routines D03PFF, D03PLF and D03PSF, the left and right solution values are derived automatically from the solution values at adjacent spatial points and supplied to the subroutine argument NUMFLX from which the user may call D03PWF. The Euler equations for a perfect gas in conservative form are:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0, \tag{1}$$

with

$$U = \begin{bmatrix} \rho \\ m \\ e \end{bmatrix}, \quad \text{and} \quad F = \begin{bmatrix} m \\ \dfrac{m^2}{\rho} + (\gamma - 1)\left(e - \dfrac{m^2}{2\rho}\right) \\ \dfrac{me}{\rho} + \dfrac{m}{\rho}(\gamma - 1)\left(e - \dfrac{m^2}{2\rho}\right) \end{bmatrix}, \tag{2}$$

where ρ is the density, m is the momentum, e is the specific total energy and γ is the (constant) ratio of specific heats. The pressure p is given by

$$p = (\gamma - 1)\left(e - \frac{\rho u^2}{2}\right), \tag{3}$$

where $u = m/\rho$ is the velocity.

The routine calculates an approximation to the numerical flux function $F(U_L, U_R) = F(U^*(U_L, U_R))$, where $U = U_L$ and $U = U_R$ are the left and right solution values, and $U^*(U_L, U_R)$ is the intermediate state $\omega(0)$ arising from the similarity solution $U(y,t) = \omega(y/t)$ of the Riemann problem defined by

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial y} = 0, \tag{4}$$

with U and F as in (2), and initial piecewise constant values $U = U_L$ for $y < 0$ and $U = U_R$ for $y > 0$. The spatial domain is $-\infty < y < \infty$, where $y = 0$ is the point at which the numerical flux is required.

4 References

[1] Toro E F (1996) *Riemann Solvers and Upwind Methods for Fluid Dynamics* Springer-Verlag

[2] Toro E F (1992) The weighted average flux method applied to the Euler equations *Phil. Trans. R. Soc. Lond.* **A341** 499–530

[3] Toro E F, Spruce M and Spears W (1994) Restoration of the contact surface in the HLL Riemann solver *J. Shock Waves* **4** 25–34

5 Parameters

1: ULEFT(3) — ***real*** array *Input*

On entry: ULEFT(i) must contain the left value of the component U_i for i = 1,2,3. That is, ULEFT(1) must contain the left value of ρ, ULEFT(2) must contain the left value of m and ULEFT(3) must contain the left value of e.

2: URIGHT(3) — ***real*** array *Input*

On entry: URIGHT(i) must contain the right value of the component U_i for i = 1,2,3. That is, URIGHT(1) must contain the right value of ρ, URIGHT(2) must contain the right value of m and URIGHT(3) must contain the right value of e.

3: GAMMA — ***real*** *Input*

On entry: the ratio of specific heats γ.

Constraint: GAMMA > 0.0.

4: FLUX(3) — ***real*** array *Output*

On exit: FLUX(i) contains the numerical flux component $\hat{F}_i$ for i = 1,2,3.

5: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

Note. If the left and/or right values of ρ or p (from (3)) are found to be negative, then the routine will terminate with an error exit (IFAIL = 2). If the routine is being called from the user-supplied subroutine NUMFLX in D03PFF etc., then a **soft fail** option (IFAIL = 1 or −1) is recommended so that a recalculation of the current time step can be forced using the IRES parameter.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, GAMMA ≤ 0.0.

IFAIL = 2

On entry, the left and/or right density or derived pressure value is less than 0.0.

7 Accuracy

The routine performs an exact calculation of the HLL numerical flux function, and so the result will be accurate to machine precision.

8 Further Comments

The routine must only be used to calculate the numerical flux for the Euler equations in exactly the form given by (2), with ULEFT(i) and URIGHT(i) containing the left and right values of ρ, m and e for i = 1,2,3 respectively. The time taken by the routine is independent of the input parameters.

9 Example

This example uses D03PLF and D03PWF to solve the Euler equations in the domain $0 \leq x \leq 1$ for $0 < t \leq 0.035$ with initial conditions for the primitive variables $\rho(x,t)$, $u(x,t)$ and $p(x,t)$ given by

$$\rho(x,0) = 5.99924, \quad u(x,0)= \ 19.5975, \quad p(x,0)=460.894, \qquad \text{for } x < 0.5,$$
$$\rho(x,0) = 5.99242, \quad u(x,0)=-6.19633, \quad p(x,0)= \ 46.095, \qquad \text{for } x > 0.5.$$

This test problem is taken from [1] and its solution represents the collision of two strong shocks travelling in opposite directions, consisting of a left facing shock (travelling slowly to the right), a right travelling contact discontinuity and a right travelling shock wave. There is an exact solution to this problem (see [1]) but the calculation is lengthy and has therefore been omitted.

9.1 Program Text

```
*     D03PWF Example Program Text
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          NPDE, NPTS, NCODE, NXI, NEQN, NIW, NWKRES,
     +                 LENODE, MLU, NW
      PARAMETER        (NPDE=3,NPTS=141,NCODE=0,NXI=0,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=NEQN+24,
     +                 NWKRES=NPDE*(2*NPTS+3*NPDE+32)+7*NPTS+4,
     +                 LENODE=9*NEQN+50,MLU=3*NPDE-1,NW=(3*MLU+1)
     +                 *NEQN+NWKRES+LENODE)
*     .. Scalars in Common ..
      real             ELO, ERO, GAMMA, RLO, RRO, ULO, URO
*     .. Local Scalars ..
      real             D, P, TOUT, TS, V
      INTEGER          I, IFAIL, IND, ITASK, ITOL, ITRACE, K
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NPDE,NPTS),
     +                 UE(3,9), W(NW), X(NPTS), XI(1)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PEK, D03PLF, D03PLP, NUMFLX
*     .. Common blocks ..
      COMMON           /INIT/ELO, ERO, RLO, RRO, ULO, URO
      COMMON           /PARAMS/GAMMA
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PWF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*
*     Problem parameters
*
      GAMMA = 1.4e0
      RLO = 5.99924e0
      RRO = 5.99242e0
      ULO = 5.99924e0*19.5975e0
      URO = -5.99242e0*6.19633e0
      ELO = 460.894e0/(GAMMA-1.0e0) + 0.5e0*RLO*19.5975e0**2
      ERO = 46.095e0/(GAMMA-1.0e0) + 0.5e0*RRO*6.19633e0**2
*
```

```
*     Initialise mesh
*
      DO 20 I = 1, NPTS
         X(I) = 1.0e0*(I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
*     Initial values
*
      DO 40 I = 1, NPTS
         IF (X(I).LT.0.5e0) THEN
            U(1,I) = RL0
            U(2,I) = UL0
            U(3,I) = EL0
         ELSE IF (X(I).EQ.0.5e0) THEN
            U(1,I) = 0.5e0*(RL0+RR0)
            U(2,I) = 0.5e0*(UL0+UR0)
            U(3,I) = 0.5e0*(EL0+ER0)
         ELSE
            U(1,I) = RR0
            U(2,I) = UR0
            U(3,I) = ER0
         END IF
   40 CONTINUE
*
      ITRACE = 0
      ITOL = 1
      NORM = '2'
      ATOL(1) = 0.5e-2
      RTOL(1) = 0.5e-3
      XI(1) = 0.0e0
      LAOPT = 'B'
      IND = 0
      ITASK = 1
      DO 60 I = 1, 30
         ALGOPT(I) = 0.0e0
   60 CONTINUE
*
*     Theta integration
*
      ALGOPT(1) = 2.0e0
      ALGOPT(6) = 2.0e0
      ALGOPT(7) = 2.0e0
*
*     Max. time step
*
      ALGOPT(13) = 0.5e-2
*
      TS = 0.0e0
      TOUT = 0.035e0
      IFAIL = 0
*
      CALL D03PLF(NPDE,TS,TOUT,D03PLP,NUMFLX,BNDARY,U,NPTS,X,NCODE,
     +            D03PEK,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,ALGOPT,W,
     +            NW,IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
      WRITE (NOUT,99998) TS
      WRITE (NOUT,99999)
```

```
*
*     Read exact data at output points
*
      DO 80 I = 1, 9
         READ (NIN,*) UE(1,I), UE(2,I), UE(3,I)
   80 CONTINUE
*
*     Calculate density, velocity and pressure
*
      K = 0
      DO 100 I = 15, NPTS - 14, 14
         D = U(1,I)
         V = U(2,I)/D
         P = D*(GAMMA-1.0e0)*(U(3,I)/D-0.5e0*V**2)
         K = K + 1
         WRITE (NOUT,99996) X(I), D, UE(1,K), V, UE(2,K), P, UE(3,K)
  100 CONTINUE
*
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (4X,'X',6X,'APPROX D',3X,'EXACT D',4X,'APPROX V',3X,'EXAC',
     +       'T V',4X,'APPROX P',3X,'EXACT P')
99998 FORMAT (/' T = ',F6.3,/)
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,e8.2,6(1X,e10.4))
      END
*
      SUBROUTINE BNDARY(NPDE,NPTS,T,X,U,NCODE,V,VDOT,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,NPTS), V(*), VDOT(*), X(NPTS)
*     .. Scalars in Common ..
      real              ELO, ERO, RLO, RRO, ULO, URO
*     .. Common blocks ..
      COMMON            /INIT/ELO, ERO, RLO, RRO, ULO, URO
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         G(1) = U(1,1) - RLO
         G(2) = U(2,1) - ULO
         G(3) = U(3,1) - ELO
      ELSE
         G(1) = U(1,NPTS) - RRO
         G(2) = U(2,NPTS) - URO
         G(3) = U(3,NPTS) - ERO
      END IF
      RETURN
      END
*
      SUBROUTINE NUMFLX(NPDE,T,X,NCODE,V,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
```

```
*     .. Array Arguments ..
      real              FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE), V(*)
*     .. Scalars in Common ..
      real              GAMMA
*     .. Local Scalars ..
      INTEGER           IFAIL
*     .. External Subroutines ..
      EXTERNAL          D03PWF
*     .. Common blocks ..
      COMMON            /PARAMS/GAMMA
*     .. Save statement ..
      SAVE              /PARAMS/
*     .. Executable Statements ..
*
      IFAIL = 0
      CALL D03PWF(ULEFT,URIGHT,GAMMA,FLUX,IFAIL)
      RETURN
      END
```

9.2 Program Data

```
D03PWF Example Program Data
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.1428e+02   0.8690e+01   0.1692e+04
 0.1428e+02   0.8690e+01   0.1692e+04
 0.1428e+02   0.8690e+01   0.1692e+04
 0.3104e+02   0.8690e+01   0.1692e+04
```

9.3 Program Results

```
 D03PWF Example Program Results

 T =  0.035

    X       APPROX D   EXACT D    APPROX V   EXACT V    APPROX P   EXACT P
 0.10E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.20E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.30E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.40E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.50E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.60E+00 0.1422E+02 0.1428E+02 0.8658E+01 0.8690E+01 0.1687E+04 0.1692E+04
 0.70E+00 0.1426E+02 0.1428E+02 0.8670E+01 0.8690E+01 0.1688E+04 0.1692E+04
 0.80E+00 0.1944E+02 0.1428E+02 0.8678E+01 0.8690E+01 0.1691E+04 0.1692E+04
 0.90E+00 0.3100E+02 0.3104E+02 0.8676E+01 0.8690E+01 0.1687E+04 0.1692E+04

 Number of integration steps in time =    699
 Number of function evaluations =   1714
 Number of Jacobian evaluations =      1
 Number of iterations =      2
```

D03PXF – NAG Fortran Library Routine Document

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

D03PXF calculates a numerical flux function using an Exact Riemann Solver for the Euler equations in conservative form. The routine is designed primarily for use with the upwind discretisation routines D03PFF, D03PLF or D03PSF, but may also be applicable to other conservative upwind schemes requiring numerical flux functions.

2 Specification

```
SUBROUTINE D03PXF(ULEFT, URIGHT, GAMMA, TOL, NITER, FLUX, IFAIL)
INTEGER            NITER, IFAIL
real               ULEFT(3), URIGHT(3), GAMMA, TOL, FLUX(3)
```

3 Description

D03PXF calculates a numerical flux function at a single spatial point using an Exact Riemann Solver (see [1] and [2]) for the Euler equations (for a perfect gas) in conservative form. The user must supply the *left* and *right* solution values at the point where the numerical flux is required, i.e., the initial left and right states of the Riemann problem defined below. In the routines D03PFF, D03PLF and D03PSF, the left and right solution values are derived automatically from the solution values at adjacent spatial points and supplied to the subroutine argument NUMFLX from which the user may call D03PXF. The Euler equations for a perfect gas in conservative form are:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = 0, \qquad (1)$$

with

$$U = \begin{bmatrix} \rho \\ m \\ e \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} m \\ \dfrac{m^2}{\rho} + (\gamma - 1)\left(e - \dfrac{m^2}{2\rho}\right) \\ \dfrac{me}{\rho} + \dfrac{m}{\rho}(\gamma - 1)\left(e - \dfrac{m^2}{2\rho}\right) \end{bmatrix}, \qquad (2)$$

where ρ is the density, m is the momentum, e is the specific total energy and γ is the (constant) ratio of specific heats. The pressure p is given by

$$p = (\gamma - 1)\left(e - \frac{\rho u^2}{2}\right), \qquad (3)$$

where $u = m/\rho$ is the velocity.

The routine calculates the numerical flux function $F(U_L, U_R) = F(U^*(U_L, U_R))$, where $U = U_L$ and $U = U_R$ are the left and right solution values, and $U^*(U_L, U_R)$ is the intermediate state $\omega(0)$ arising from the similarity solution $U(y,t) = \omega(y/t)$ of the Riemann problem defined by

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial y} = 0, \qquad (4)$$

with U and F as in (2), and initial piecewise constant values $U = U_L$ for $y < 0$ and $U = U_R$ for $y > 0$. The spatial domain is $-\infty < y < \infty$, where $y = 0$ is the point at which the numerical flux is required.

The algorithm is termed an Exact Riemann Solver although it does in fact calculate an approximate solution to a true Riemann problem, as opposed to an Approximate Riemann Solver which involves some form of alternative modelling of the Riemann problem. The approximation part of the Exact Riemann Solver is a Newton-Raphson iterative procedure to calculate the pressure, and the user must supply a tolerance TOL and a maximum number of iterations NITER. Default values for these parameters can be chosen.

A solution can not be found by this routine if there is a vacuum state in the Riemann problem (loosely characterised by zero density), or if such a state is generated by the interaction of two non-vacuum data states. In this case a Riemann solver which can handle vacuum states has to be used (see [1]).

4 References

[1] Toro E F (1996) *Riemann Solvers and Upwind Methods for Fluid Dynamics* Springer-Verlag

[2] Toro E F (1989) A weighted average flux method for hyperbolic conservation laws *Proc. Roy. Soc. Lond.* **A423** 401–418

5 Parameters

1: ULEFT(3) — ***real*** array *Input*

On entry: ULEFT(i) must contain the left value of the component U_i for i = 1,2,3. That is, ULEFT(1) must contain the left value of ρ, ULEFT(2) must contain the left value of m and ULEFT(3) must contain the left value of e.

2: URIGHT(3) — ***real*** array *Input*

On entry: URIGHT(i) must contain the right value of the component U_i for i = 1,2,3. That is, URIGHT(1) must contain the right value of ρ, URIGHT(2) must contain the right value of m and URIGHT(3) must contain the right value of e.

3: GAMMA — ***real*** *Input*

On entry: the ratio of specific heats γ.

Constraint: GAMMA > 0.0.

4: TOL — ***real*** *Input*

On entry: the tolerance to be used in the Newton-Raphson procedure to calculate the pressure. If TOL is set to zero then the default value of 1.0×10^{-6} is used.

Constraint: TOL ≥ 0.0.

5: NITER — INTEGER *Input*

On entry: the maximum number of Newton-Raphson iterations allowed. If NITER is set to zero then the default value of 20 is used.

Constraint: NITER ≥ 0.

6: FLUX(3) — ***real*** array *Output*

On exit: FLUX(i) contains the numerical flux component $\hat{F}_i$ for i = 1,2,3.

7: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

Note. If the left and/or right values of ρ or p (from (3)) are found to be negative, then the routine will terminate with an error exit (IFAIL = 2). If the routine is being called from the user-supplied subroutine NUMFLX in D03PFF etc., then a **soft fail** option (IFAIL = 1 or −1) is recommended so that a recalculation of the current time step can be forced using the IRES parameter.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, GAMMA $\le$ 0.0,
or TOL < 0.0,
or NITER < 0.

IFAIL = 2

On entry, the left and/or right density or derived pressure value is less than 0.0.

IFAIL = 3

A vacuum condition has been detected therefore a solution can not be found using this routine. You are advised to check your problem formulation.

IFAIL = 4

The internal Newton-Raphson iterative procedure used to solve for the pressure has failed to converge. The value of TOL or NITER may be too small, but if the problem persists try an Approximate Riemann Solver (D03PUF, D03PVF or D03PWF).

7 Accuracy

The algorithm is exact apart from the calculation of the pressure which uses a Newton-Raphson iterative procedure, the accuracy of which is controlled by the parameter TOL. In some cases the initial guess for the Newton-Raphson procedure is exact and no further iterations are required.

8 Further Comments

The routine must only be used to calculate the numerical flux for the Euler equations in exactly the form given by (2), with ULEFT(i) and URIGHT(i) containing the left and right values of ρ, m and e for $i = 1, 2, 3$ respectively.

For some problems the routine may fail or be highly inefficient in comparison with an Approximate Riemann Solver (e.g. D03PUF, D03PVF or D03PWF). Hence it is advisable to try more than one Riemann solver and to compare the performance and the results.

The time taken by the routine is independent of all input parameters other than TOL.

9 Example

This example uses D03PLF and D03PXF to solve the Euler equations in the domain $0 \le x \le 1$ for $0 < t \le 0.035$ with initial conditions for the primitive variables $\rho(x,t)$, $u(x,t)$ and $p(x,t)$ given by

$$\begin{array}{llll} \rho(x,0) = 5.99924, & u(x,0) = 19.5975, & p(x,0) = 460.894, & \text{for } x < 0.5, \\ \rho(x,0) = 5.99242, & u(x,0) = -6.19633, & p(x,0) = 46.095, & \text{for } x > 0.5. \end{array}$$

This test problem is taken from [1] and its solution represents the collision of two strong shocks travelling in opposite directions, consisting of a left facing shock (travelling slowly to the right), a right travelling contact discontinuity and a right travelling shock wave. There is an exact solution to this problem (see [1]) but the calculation is lengthy and has therefore been omitted.

9.1 Program Text

```
*     D03PXF Example Program Text
*     Mark 18 Release. NAG Copyright 1997.
*     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
      INTEGER          NPDE, NPTS, NCODE, NXI, NEQN, NIW, NWKRES,
     +                 LENODE, MLU, NW
      PARAMETER        (NPDE=3,NPTS=141,NCODE=0,NXI=0,
     +                 NEQN=NPDE*NPTS+NCODE,NIW=NEQN+24,
     +                 NWKRES=NPDE*(2*NPTS+3*NPDE+32)+7*NPTS+4,
     +                 LENODE=9*NEQN+50,MLU=3*NPDE-1,NW=(3*MLU+1)
     +                 *NEQN+NWKRES+LENODE)
*     .. Scalars in Common ..
      real             EL0, ER0, GAMMA, RL0, RR0, UL0, UR0
*     .. Local Scalars ..
      real             D, P, TOUT, TS, V
      INTEGER          I, IFAIL, IND, ITASK, ITOL, ITRACE, K
      CHARACTER        LAOPT, NORM
*     .. Local Arrays ..
      real             ALGOPT(30), ATOL(1), RTOL(1), U(NPDE,NPTS),
     +                 UE(3,9), W(NW), X(NPTS), XI(1)
      INTEGER          IW(NIW)
*     .. External Subroutines ..
      EXTERNAL         BNDARY, D03PEK, D03PLF, D03PLP, NUMFLX
*     .. Common blocks ..
      COMMON           /INIT/EL0, ER0, RL0, RR0, UL0, UR0
      COMMON           /PARAMS/GAMMA
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03PXF Example Program Results'
*     Skip heading in data file
      READ (NIN,*)
*
*     Problem parameters
*
      GAMMA = 1.4e0
      RL0 = 5.99924e0
      RR0 = 5.99242e0
      UL0 = 5.99924e0*19.5975e0
      UR0 = -5.99242e0*6.19633e0
      EL0 = 460.894e0/(GAMMA-1.0e0) + 0.5e0*RL0*19.5975e0**2
      ER0 = 46.095e0/(GAMMA-1.0e0) + 0.5e0*RR0*6.19633e0**2
*
*     Initialise mesh
*
      DO 20 I = 1, NPTS
         X(I) = 1.0e0*(I-1.0e0)/(NPTS-1.0e0)
   20 CONTINUE
*
*     Initial values
*
      DO 40 I = 1, NPTS
         IF (X(I).LT.0.5e0) THEN
            U(1,I) = RL0
            U(2,I) = UL0
            U(3,I) = EL0
         ELSE IF (X(I).EQ.0.5e0) THEN
```

```
            U(1,I) = 0.5e0*(RLO+RRO)
            U(2,I) = 0.5e0*(ULO+URO)
            U(3,I) = 0.5e0*(ELO+ERO)
         ELSE
            U(1,I) = RRO
            U(2,I) = URO
            U(3,I) = ERO
         END IF
   40 CONTINUE
*
      ITRACE = 0
      ITOL = 1
      NORM = '2'
      ATOL(1) = 0.5e-2
      RTOL(1) = 0.5e-3
      XI(1) = 0.0e0
      LAOPT = 'B'
      IND = 0
      ITASK = 1
      DO 60 I = 1, 30
         ALGOPT(I) = 0.0e0
   60 CONTINUE
*
*     Theta integration
*
      ALGOPT(1) = 2.0e0
      ALGOPT(6) = 2.0e0
      ALGOPT(7) = 2.0e0
*
*     Max. time step
*
      ALGOPT(13) = 0.5e-2
*
      TS = 0.0e0
      TOUT = 0.035e0
      IFAIL = 0
*
      CALL D03PLF(NPDE,TS,TOUT,D03PLP,NUMFLX,BNDARY,U,NPTS,X,NCODE,
     +            D03PEK,NXI,XI,NEQN,RTOL,ATOL,ITOL,NORM,LAOPT,ALGOPT,W,
     +            NW,IW,NIW,ITASK,ITRACE,IND,IFAIL)
*
      WRITE (NOUT,99998) TS
      WRITE (NOUT,99999)
*
*     Read exact data at output points
*
      DO 80 I = 1, 9
         READ (NIN,*) UE(1,I), UE(2,I), UE(3,I)
   80 CONTINUE
*
*     Calculate density, velocity and pressure
*
      K = 0
      DO 100 I = 15, NPTS - 14, 14
         D = U(1,I)
         V = U(2,I)/D
         P = D*(GAMMA-1.0e0)*(U(3,I)/D-0.5e0*V**2)
         K = K + 1
```

```
         WRITE (NOUT,99996) X(I), D, UE(1,K), V, UE(2,K), P, UE(3,K)
  100 CONTINUE
*
      WRITE (NOUT,99997) IW(1), IW(2), IW(3), IW(5)
      STOP
*
99999 FORMAT (4X,'X',6X,'APPROX D',3X,'EXACT D',4X,'APPROX V',3X,'EXAC',
     +       'T V',4X,'APPROX P',3X,'EXACT P')
99998 FORMAT (/' T = ',F6.3,/)
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,e8.2,6(1X,e10.4))
      END
*
      SUBROUTINE BNDARY(NPDE,NPTS,T,X,U,NCODE,V,VDOT,IBND,G,IRES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           IBND, IRES, NCODE, NPDE, NPTS
*     .. Array Arguments ..
      real              G(NPDE), U(NPDE,NPTS), V(*), VDOT(*), X(NPTS)
*     .. Scalars in Common ..
      real              ELO, ERO, RLO, RRO, ULO, URO
*     .. Common blocks ..
      COMMON            /INIT/ELO, ERO, RLO, RRO, ULO, URO
*     .. Executable Statements ..
      IF (IBND.EQ.0) THEN
         G(1) = U(1,1) - RLO
         G(2) = U(2,1) - ULO
         G(3) = U(3,1) - ELO
      ELSE
         G(1) = U(1,NPTS) - RRO
         G(2) = U(2,NPTS) - URO
         G(3) = U(3,NPTS) - ERO
      END IF
      RETURN
      END
*
      SUBROUTINE NUMFLX(NPDE,T,X,NCODE,V,ULEFT,URIGHT,FLUX,IRES)
*     .. Scalar Arguments ..
      real              T, X
      INTEGER           IRES, NCODE, NPDE
*     .. Array Arguments ..
      real              FLUX(NPDE), ULEFT(NPDE), URIGHT(NPDE), V(*)
*     .. Scalars in Common ..
      real              GAMMA
*     .. Local Scalars ..
      real              TOL
      INTEGER           IFAIL, NITER
*     .. External Subroutines ..
      EXTERNAL          D03PXF
*     .. Common blocks ..
      COMMON            /PARAMS/GAMMA
*     .. Save statement ..
      SAVE              /PARAMS/
*     .. Executable Statements ..
*
      IFAIL = 0
```

```
      TOL = 0.0e0
      NITER = 0
      CALL D03PXF(ULEFT,URIGHT,GAMMA,TOL,NITER,FLUX,IFAIL)
      RETURN
      END
```

9.2 Program Data

```
D03PXF Example Program Data
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.5999e+01   0.1960e+02   0.4609e+03
 0.1428e+02   0.8690e+01   0.1692e+04
 0.1428e+02   0.8690e+01   0.1692e+04
 0.1428e+02   0.8690e+01   0.1692e+04
 0.3104e+02   0.8690e+01   0.1692e+04
```

9.3 Program Results

```
 D03PXF Example Program Results

 T =   0.035

     X       APPROX D   EXACT D    APPROX V   EXACT V    APPROX P   EXACT P
 0.10E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.20E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.30E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.40E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.50E+00 0.5999E+01 0.5999E+01 0.1960E+02 0.1960E+02 0.4609E+03 0.4609E+03
 0.60E+00 0.1423E+02 0.1428E+02 0.8660E+01 0.8690E+01 0.1688E+04 0.1692E+04
 0.70E+00 0.1425E+02 0.1428E+02 0.8672E+01 0.8690E+01 0.1688E+04 0.1692E+04
 0.80E+00 0.1921E+02 0.1428E+02 0.8674E+01 0.8690E+01 0.1689E+04 0.1692E+04
 0.90E+00 0.3100E+02 0.3104E+02 0.8675E+01 0.8690E+01 0.1687E+04 0.1692E+04

 Number of integration steps in time =    697
 Number of function evaluations =   1708
 Number of Jacobian evaluations =     1
 Number of iterations =      2
```

D03PYF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

This routine may be used in conjunction with either D03PDF or D03PJF. It computes the solution and its first derivative at user-specified points in the spatial co-ordinate.

2. Specification

```
      SUBROUTINE D03PYF (NPDE, U, NBKPTS, XBKPTS, NPOLY, NPTS, XP, INTPTS,
     1                   ITYPE, UP, W, NW, IFAIL)
      INTEGER            NPDE, NBKPTS, NPOLY, NPTS, INTPTS, ITYPE, NW, IFAIL
      real               U(NPDE,NPTS), XBKPTS(NBKPTS), XP(INTPTS),
     1                   UP(NPDE,INTPTS,ITYPE), W(NW)
```

3. Description

D03PYF is an interpolation routine for evaluating the solution of a system of partial differential equations (PDEs), or the PDE components of a system of PDEs with coupled ordinary differential equations (ODEs), at a set of user-specified points. The solution of a system of equations can be computed using D03PDF or D03PJF on a set of mesh points; D03PYF can then be employed to compute the solution at a set of points other than those originally used in D03PDF or D03PJF. It can also evaluate the first derivative of the solution. Polynomial interpolation is used between each of the break-points XBKPTS(i), for i = 1,2,...,NBKPTS. When the derivative is needed (ITYPE = 2), the array XP(INTPTS) must not contain any of the break-points, as the method, and consequently the interpolation scheme, assumes that only the solution is continuous at these points.

4. References

None.

5. Parameters

Note: the parameters U, NPTS, NPDE, XBKPTS, NBKPTS, W and NW must be supplied unchanged from either D03PDF or D03PJF.

1: NPDE – INTEGER. *Input*

On entry: the number of PDEs.

Constraint: NPDE ≥ 1.

2: U(NPDE,NPTS) – ***real*** array. *Input*

On entry: the PDE part of the original solution returned in the parameter U by the routine D03PDF or D03PJF.

3: NBKPTS – INTEGER. *Input*

On entry: the number of break-points.

Constraint: NBKPTS ≥ 2.

4: XBKPTS(NBKPTS) – ***real*** array. *Input*

On entry: XBKPTS(i), for i = 1,2,...,NBKPTS, must contain the break-points as used by D03PDF or D03PJF.

Constraint: XBKPTS(1) < XBKPTS(2) < ... < XBKPTS(NBKPTS).

5: NPOLY – INTEGER. *Input*

On entry: the degree of the Chebyshev polynomial used for approximation as used by D03PDF or D03PJF.

Constraint: 1 ≤ NPOLY ≤ 49.

6: NPTS – INTEGER. *Input*

On entry: the number of mesh points as used by D03PDF or D03PJF.

Constraint: NPTS = (NBKPTS−1)×NPOLY + 1.

7: XP(INTPTS) – ***real*** array. *Input*

On entry: XP(i), for i = 1,2,...,INTPTS, must contain the spatial interpolation points.

Constraint: XBKPTS(1) ≤ XP(1) < XP(2) < ... < XP(INTPTS) ≤ XBKPTS(NBKPTS).

When ITYPE = 2, XP(i) ≠ XBKPTS(j), for i = 1,2,...,INTPTS; j = 2,3,...,NBKPTS−1.

8: INTPTS – INTEGER. *Input*

On entry: the number of interpolation points.

Constraint: INTPTS ≥ 1.

9: ITYPE – INTEGER. *Input*

On entry: specifies the interpolation to be performed.

If ITYPE = 1, the solution at the interpolation points are computed. If ITYPE = 2, both the solution and the first derivative at the interpolation points are computed.

Constraint: ITYPE = 1 or 2.

10: UP(NPDE,INTPTS,ITYPE) – ***real*** array. *Output*

On exit: if ITYPE = 1, UP(i,j,1), contains the value of the solution $U_i(x_j,t_{out})$, at the interpolation points x_j = XP(j), for j = 1,2,...,INTPTS; i = 1,2,...,NPDE.

If ITYPE = 2, UP(i,j,1) contains $U_i(x_j,t_{out})$ and UP(i,j,2) contains $\frac{\partial U_i}{\partial x}$ at these points.

11: W(NW) – ***real*** array. *Input*

On entry: the array W as returned by D03PDF or D03PJF. The contents of W must not be changed from the call to D03PDF or D03PJF.

12: NW – INTEGER. *Input*

On entry: the size of the workspace W, as in D03PDF or D03PJF.

13: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, ITYPE ≠ 1 or 2,
or NPOLY < 1,
or NPDE < 1,
or NBKPTS < 2,
or INTPTS < 1,

or NPTS ≠ (NBKPTS−1)×NPOLY + 1,
or XBKPTS(i), for i = 1,...,NBKPTS, are not ordered.

IFAIL = 2

On entry, the interpolation points XP(i), for i = 1,...,INTPTS, are not in strictly increasing order, or when ITYPE = 2, at least one of the interpolation points stored in XP is equal to one of the break-points stored in XBKPTS.

IFAIL = 3

The user is attempting extrapolation, that is, one of the interpolation points XP(i), for some i, lies outside the interval [XBKPTS(1),XBKPTS(NBKPTS)]. Extrapolation is not permitted.

7. Accuracy

See the documents for D03PDF or D03PJF.

8. Further Comments

None.

9. Example

See the example program for D03PDF.

D03PZF – NAG Fortran Library Routine Document

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

This routine interpolates in the spatial co-ordinate the solution and derivative of a system of partial differential equations (PDEs). The solution must first be computed using one of the finite difference scheme routines D03PCF, D03PHF or D03PPF, or one of the Keller box scheme routines D03PEF, D03PKF or D03PRF.

2 Specification

```
      SUBROUTINE D03PZF(NPDE, M, U, NPTS, X, XP, INTPTS, ITYPE, UP,
     1                  IFAIL)
      INTEGER           NPDE, M, NPTS, INTPTS, ITYPE, IFAIL
      real              U(NPDE,NPTS), X(NPTS), XP(INTPTS),
     1                  UP(NPDE,INTPTS,ITYPE)
```

3 Description

D03PZF is an interpolation routine for evaluating the solution of a system of partial differential equations (PDEs), at a set of user-specified points. The solution of the system of equations (possibly with coupled ordinary differential equations) must be computed using a finite difference scheme routine or a Keller box scheme routine on a set of mesh points. D03PZF can then be employed to compute the solution at a set of points anywhere in the range of the mesh. It can also evaluate the first spatial derivative of the solution. The routine uses linear interpolation for approximating the solution.

4 References

None.

5 Parameters

Note: the parameters X, M, U, NPTS and NPDE must be supplied unchanged from the PDE routine.

1: NPDE — INTEGER *Input*

On entry: the number of PDEs.

Constraint: NPDE ≥ 1.

2: M — INTEGER *Input*

On entry: the co-ordinate system used. If the call to D03PZF follows one of the finite difference routines then M must be the same parameter M as used by the finite difference routines. For the Keller box scheme routines only Cartesian co-ordinate systems are valid and so M **must** be set to zero. No check will be made by D03PZF in this case.

M = 0
indicates Cartesian co-ordinates

M = 1
indicates cylindrical polar co-ordinates

M = 2
indicates spherical polar co-ordinates

Constraints:

$0 \leq M \leq 2$ following a finite difference routine.

$M = 0$ following a Keller box scheme routine.

3: U(NPDE,NPTS) — ***real*** array *Input*

On entry: the PDE part of the original solution returned in the parameter U by the PDE routine.

Constraint: NPDE ≥ 1.

4: NPTS — INTEGER *Input*

On entry: the number of mesh points.

Constraint: NPTS ≥ 3.

5: X(NPTS) — ***real*** array *Input*

On entry: X(i), for $i = 1, 2, \ldots,$ NPTS, must contain the mesh points as used by the PDE routine.

6: XP(INTPTS) — ***real*** array *Input*

On entry: XP(i), for $i = 1, 2, \ldots,$ INTPTS, must contain the spatial interpolation points.

Constraint: X(1) $\leq$ XP(1) $<$ XP(2) $< \ldots <$ XP(INTPTS) $\leq$ X(NPTS).

7: INTPTS — INTEGER *Input*

On entry: the number of interpolation points.

Constraint: INTPTS ≥ 1.

8: ITYPE — INTEGER *Input*

On entry: specifies the interpolation to be performed.

If ITYPE = 1, the solutions at the interpolation points are computed. If ITYPE = 2, both the solutions and their first derivatives at the interpolation points are computed.

Constraint: ITYPE = 1 or 2.

9: UP(NPDE,INTPTS,ITYPE) — ***real*** array *Output*

On exit: if ITYPE = 1, UP($i,j,1$), contains the value of the solution $U_i(x_j, t_{out})$, at the interpolation points x_j = XP(j), for $j = 1, 2, \ldots,$ INTPTS; $i = 1, 2, \ldots,$ NPDE.

If ITYPE = 2, UP($i,j,1$) contains $U_i(x_j, t_{out})$ and UP($i,j,2$) contains $\frac{\partial U_i}{\partial x}$ at these points.

10: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, ITYPE $\neq$ 1 or 2,
or INTPTS < 1,
or NPDE < 1,
or NPTS < 3,
or M $\neq$ 0, 1 or 2,
or the mesh points X(i), for $i = 1, 2, \ldots,$ NPTS, are not in strictly increasing order.

IFAIL = 2

On entry, the interpolation points XP(i), for $i = 1, 2, \ldots,$ INTPTS, are not in strictly increasing order.

IFAIL = 3

The user is attempting extrapolation, that is, one of the interpolation points XP(i), for some i, lies outside the interval [X(1),X(NPTS)]. Extrapolation is not permitted.

7 Accuracy

See the PDE routine documents.

8 Further Comments

None.

9 Example

See the example programs for D03PCF, D03PPF and D03PRF.

D03RAF – NAG Fortran Library Routine Document

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

D03RAF integrates a system of linear or nonlinear, time-dependent partial differential equations (PDEs) in two space dimensions on a rectangular domain. The method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs) which are solved using a backward differentiation formula (BDF) method. The resulting system of nonlinear equations is solved using a modified Newton method and a Bi-CGSTAB iterative linear solver with ILU preconditioning. Local uniform grid refinement is used to improve the accuracy of the solution. D03RAF originates from the VLUGR2 package [1] [2].

2 Specification

```
      SUBROUTINE D03RAF(NPDE, TS, TOUT, DT, XMIN, XMAX, YMIN, YMAX, NX,
     1                  NY, TOLS,TOLT, PDEDEF, BNDARY, PDEIV, MONITR,
     2                  OPTI,OPTR, RWK, LENRWK, IWK, LENIWK, LWK,LENLWK,
     3                  ITRACE, IND, IFAIL)
      INTEGER           NPDE, NX, NY, OPTI(4), LENRWK, IWK(LENIWK),
     1                  LENIWK, LENLWK, ITRACE, IND, IFAIL
      real              TS, TOUT, DT(3), XMIN, XMAX, YMIN, YMAX, TOLS,
     1                  TOLT, OPTR(3,NPDE), RWK(LENRWK)
      LOGICAL           LWK(LENLWK)
      EXTERNAL          PDEDEF, BNDARY, PDEIV, MONITR
```

3 Description

D03RAF integrates the system of PDEs:

$$F_j(t,x,y,u,u_t,u_x,u_y,u_{xx},u_{xy},u_{yy}) = 0, \quad j = 1,2,\ldots,\mathrm{NPDE}, \tag{1}$$

for x and y in the rectangular domain $x_{min} \leq x \leq x_{max}$, $y_{min} \leq y \leq y_{max}$, and time interval $t_0 \leq t \leq t_{out}$, where the vector u is the set of solution values

$$u(x,y,t) = [u_1(x,y,t),\ldots,u_{\mathrm{NPDE}}(x,y,t)]^T,$$

and u_t denotes partial differentiation with respect to t, and similarly for u_x etc.

The functions F_j must be supplied by the user in a subroutine PDEDEF. Similarly the initial values of the functions $u(x,y,t)$ must be specified at $t = t_0$ in a subroutine PDEIV.

Note that whilst complete generality is offered by the master equations (1), D03RAF is not appropriate for all PDEs. In particular, hyperbolic systems should not be solved using this routine. Also, at least one component of u_t must appear in the system of PDEs.

The boundary conditions must be supplied by the user in a subroutine BNDARY in the form

$$G_j(t,x,y,u,u_t,u_x,u_y) = 0 \ \text{ at } \ x = x_{min}, x_{max}, \ y = y_{min}, y_{max}, \ \text{for } j = 1,2,\ldots,\mathrm{NPDE}. \tag{2}$$

The domain is covered by a uniform coarse base grid of size $n_x \times n_y$ specified by the user, and nested finer uniform subgrids are subsequently created in regions with high spatial activity. The refinement is controlled using a space monitor which is computed from the current solution and a user-supplied space tolerance TOLS. A number of optional parameters, e.g. the maximum number of grid levels at any time, and some weighting factors, can be specified in the arrays OPTI and OPTR. Further details of the refinement strategy can be found in Section 8.

The system of PDEs and the boundary conditions are discretised in space on each grid using a standard second-order finite difference scheme (centred on the internal domain and one-sided at the boundaries), and the resulting system of ODEs is integrated in time using a second-order, two-step, implicit BDF method with variable step size. The time integration is controlled using a time monitor computed at each grid level from the current solution and a user-supplied time tolerance TOLT, and some further optional user-specified weighting factors held in OPTR (see Section 8 for details). The time monitor is used to compute a new step size, subject to restrictions on the size of the change between steps, and (optional) user-specified maximum and minimum step sizes held in DT. The step size is adjusted so that the remaining integration interval is an integer number times Δt. In this way a solution is obtained at $t = t_{out}$.

A modified Newton method is used to solve the nonlinear equations arising from the time integration. The user may specify (in OPTI) the maximum number of Newton iterations to be attempted. A Jacobian matrix is calculated at the beginning of each time step. If the Newton process diverges or the maximum number of iterations is exceeded, a new Jacobian is calculated using the most recent iterates and the Newton process is restarted. If convergence is not achieved after the (optional) user-specified maximum number of new Jacobian evaluations, the time step is retried with $\Delta t = \Delta t/4$. The linear systems arising from the Newton iteration are solved using a Bi-CGSTAB iterative method, in combination with ILU preconditioning. The maximum number of iterations can be specified by the user in OPTI.

The solution at all grid levels is stored in the workspace arrays, along with other information needed for a restart (i.e., a continuation call). It is not intended that the user extracts the solution from these arrays, indeed the necessary information regarding these arrays is not included. The user-supplied monitor routine MONITR should be used to obtain the solution at particular levels and times. MONITR is called at the end of every time step, with the last step being identified via the input argument TLAST.

Within the user-specified subroutines PDEIV, PDEDEF, BNDARY and MONITR the data structure is as follows. Each point on a particular grid is given an index (ranging from 1 to the total number of points on the grid) and all coordinate or solution information is stored in arrays according to this index, e.g. X(i) and Y(i) contain the x- and y-coordinate of point i, and U(i,j) contains the jth solution component u_j at point i.

Further details of the underlying algorithm can be found in Section 8 and in [1] [2] and the references therein.

4 References

[1] Blom J G and Verwer J G (1993) VLUGR2: A vectorized local uniform grid refinement code for PDEs in 2D *Report NM-R9306* CWI, Amsterdam

[2] Blom J G, Trompert R A and Verwer J G (1996) Algorithm 758. VLUGR2: A vectorizable adaptive grid solver for PDEs in 2D *Trans. Math. Software* **22** 302–328

[3] Trompert R A and Verwer J G (1993) Analysis of the implicit Euler local uniform grid refinement method *SIAM J. Sci. Comput.* **14** 259–278

[4] Trompert R A (1993) Local uniform grid refinement and systems of coupled partial differential equations *Appl. Numer. Maths* **12** 331–355

[5] Adjerid S and Flaherty J E (1988) A local refinement finite element method for two dimensional parabolic systems *SIAM J. Sci. Statist. Comput.* **9** 792–811

[6] Brown P N, Hindmarsh A C and Petzold L R (1994) Using Krylov methods in the solution of large scale differential-algebraic systems *SIAM J. Sci. Statist. Comput.* **15** 1467–1488

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system.

Constraint: NPDE $\geq$ 1.

2: TS — ***real*** *Input/Output*

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

On exit: the value of t which has been reached. 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: DT(3) — ***real*** array *Input/Output*

On entry: the initial, minimum and maximum time step sizes respectively. DT(1) specifies the initial time step size to be used on the first entry, i.e., when IND = 0. If DT(1) = 0.0 then the default value DT(1) = 0.01 × (TOUT−TS) is used. On subsequent entries (IND = 1), the value of DT(1) is not referenced.

DT(2) specifies the minimum time step size to be attempted by the integrator. If DT(2) = 0.0 the default value DT(2) = 10.0 × ***machine precision*** is used.

DT(3) specifies the maximum time step size to be attempted by the integrator. If DT(3) = 0.0 the default value DT(3) = TOUT − TS is used.

On exit: DT(1) contains the time step size for the next time step. DT(2) and DT(3) are unchanged or set to their default values if zero on entry.

Constraints: if IND = 1 then DT(1) is unconstrained. Otherwise DT(1) ≥ 0 and if DT(1) > 0.0 then it must satisfy the constraints:

10.0 × ***machine precision*** × max(|TS|,|TOUT|) ≤ DT(1) ≤ TOUT − TS
DT(2) ≤ DT(1) ≤ DT(3)

where the values of DT(2) and DT(3) will have been reset to their default values if zero on entry.

DT(2) and DT(3) must satisfy DT(i) ≥ 0, i = 2,3 and DT(2) ≤ DT(3) for IND = 0 and IND = 1.

5: XMIN — ***real*** *Input*

6: XMAX — ***real*** *Input*

On entry: the extents of the rectangular domain in the x-direction, i.e., the x-coordinates of the left and right boundaries respectively.

Constraint: XMIN < XMAX and XMAX must be sufficiently distinguishable from XMIN for the precision of the machine being used.

7: YMIN — ***real*** *Input*

8: YMAX — ***real*** *Input*

On entry: the extents of the rectangular domain in the y-direction, i.e., the y-coordinates of the lower and upper boundaries respectively.

Constraint: YMIN < YMAX and YMAX must be sufficiently distinguishable from YMIN for the precision of the machine being used.

9: NX — INTEGER *Input*

On entry: the number of grid points in the x-direction (including the boundary points).

Constraint: NX ≥ 4.

10: NY — INTEGER *Input*

On entry: the number of grid points in the y-direction (including the boundary points).

Constraint: NY ≥ 4.

11: TOLS — ***real*** *Input*

On entry: the space tolerance used in the grid refinement strategy (σ in equation (4)). See Section 8.2.

Constraint: TOLS > 0.0.

12: TOLT — ***real*** *Input*

On entry: the time tolerance used to determine the time step size (τ in equation (7)). See Section 8.3.

Constraint: TOLT > 0.0.

13: PDEDEF — SUBROUTINE, supplied by the user. *External Procedure*

PDEDEF must evaluate the functions F_j, $j = 1,2,..,$ NPDE, in equation (1) which define the system of PDEs (i.e., the residuals of the resulting ODE system) at all interior points of the domain. Values at points on the boundaries of the domain are ignored and will be overwritten by the subroutine BNDARY. PDEDEF is called for each subgrid in turn.

Its specification is:

```
      SUBROUTINE PDEDEF(NPTS, NPDE, T, X, Y, U, UT, UX, UY, UXX, UXY,
     1                  UYY, RES)
      INTEGER           NPTS, NPDE
      real              T, X(NPTS), Y(NPTS), U(NPTS,NPDE),
     1                  UT(NPTS,NPDE), UX(NPTS,NPDE), UY(NPTS,NPDE),
     2                  UXX(NPTS,NPDE), UXY(NPTS,NPDE), UYY(NPTS,NPDE),
     3                  RES(NPTS,NPDE)
```

1: NPTS — INTEGER *Input*

On entry: the number of grid points in the current grid.

2: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system.

3: T — ***real*** *Input*

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

4: X(NPTS) — ***real*** array *Input*

On entry: X(i) contains the x-coordinate of the ith grid point, for $i = 1,2,\ldots,$NPTS.

5: Y(NPTS) — ***real*** array *Input*

On entry: Y(i) contains the y-coordinate of the ith grid point, for $i = 1,2,\ldots,$NPTS.

6: U(NPTS,NPDE) — ***real*** array *Input*

On entry: U(i,j) contains the value of the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

7: UT(NPTS,NPDE) — ***real*** array *Input*

On entry: UT(i,j) contains the value of $\partial u/\partial t$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

8: UX(NPTS,NPDE) — ***real*** array *Input*

On entry: UX(i,j) contains the value of $\partial u/\partial x$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

9: UY(NPTS,NPDE) — ***real*** array *Input*

On entry: UY(i,j) contains the value of $\partial u/\partial y$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

10: UXX(NPTS,NPDE) — ***real*** array *Input*

On entry: UXX(i,j) contains the value of $\partial^2 u/\partial x^2$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

step, indicated by the parameter TLAST. The input arguments contain information about the grid and solution at all grid levels used.

MONITR can also be used to force an immediate tidy termination of the solution process and return to the calling program.

Its specification is:

```
      SUBROUTINE MONITR(NPDE, T, DT, DTNEW, TLAST, NLEV, NGPTS, XPTS,
     1                  YPTS, LSOL, SOL, IERR)
      INTEGER           NPDE, NLEV, NGPTS(NLEV), LSOL(NLEV), IERR
      real              T, DT, DTNEW, XPTS(*), YPTS(*), SOL(*)
      LOGICAL           TLAST
```

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, i.e., the time at the end of the integration step just completed.

3: DT — ***real*** *Input*

On entry: the current time step size DT, i.e., the time step size used for the integration step just completed.

4: DTNEW — ***real*** *Input*

On entry: the step size that will be used for the next time step.

5: TLAST — LOGICAL *Input*

On entry: indicates if intermediate or final time step. TLAST = .FALSE. for an intermediate step, TLAST = .TRUE. for the last call to MONITR before returning to the user's program.

6: NLEV — INTEGER *Input*

On entry: the number of grid levels used at time T.

7: NGPTS(NLEV) — INTEGER array *Input*

On entry: NGPTS(l) contains the number of grid points at level l, for $l = 1,2,\ldots$,NLEV.

8: XPTS(*) — ***real*** array *Input*

On entry: contains the x-coordinates of the grid points in each level in turn, i.e., X(i), for $i = 1,2,\ldots$,NGPTS(l), $l = 1,2,\ldots$,NLEV.

So for level l, X(i) = XPTS($k+i$), where k = NGPTS(1) + NGPTS(2) + $\cdots$+ NGPTS(l−1), for $i = 1,2,\ldots$,NGPTS(l), $l = 1,2,\ldots$,NLEV.

9: YPTS(*) — ***real*** array *Input*

On entry: contains the y-coordinates of the grid points in each level in turn, i.e., Y(i), for $i = 1,2,\ldots$,NGPTS(l), $l = 1,2,\ldots$,NLEV.

So for level l, Y(i) = YPTS($k+i$), where k = NGPTS(1) + NGPTS(2) + $\cdots$+ NGPTS(l−1), for $i = 1,2,\ldots$,NGPTS(l), $l = 1,2,\ldots$,NLEV.

10: LSOL(NLEV) — INTEGER array *Input*

On entry: LSOL(l) contains the pointer to the solution in SOL at grid level l and time T. (LSOL(l) actually contains the array index immediately preceding the start of the solution in SOL. See below.)

11: SOL(*) — ***real*** array *Input*

On entry: SOL contains the solution U(NGPTS(l),NPDE) at time T for each grid level l in turn, positioned according to LSOL i.e., for level l,

$$U(i,j) = SOL(LSOL(l) + (j-1) \times NGPTS(l) + i),$$

for $i = 1,\ldots$,NGPTS(l), $j = 1,\ldots$,NPDE, $l = 1,\ldots$,NLEV.

12: IERR — INTEGER *Output*

On exit: IERR should be set to 1 to force a tidy termination and an immediate return to the calling program with IFAIL set to 4. IERR should remain unchanged otherwise.

MONITR must be declared as EXTERNAL in the (sub)program from which D03RAF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

17: OPTI(4) — INTEGER array *Input*

On entry: OPTI may be set to control various options available in the integrator. If OPTI(1) = 0 then **all** the default options are employed.

If OPTI(1) > 0 then the default value of OPTI(i) for i = 2,3,4, can be obtained by setting OPTI(i) = 0.

OPTI(1) specifies the maximum number of grid levels allowed (including the base grid). OPTI(1) $\geq$ 0. The default value is OPTI(1) = 3.

OPTI(2) specifies the maximum number of Jacobian evaluations allowed during each nonlinear equations solution. OPTI(2) $\geq$ 0. The default value is OPTI(2) = 2.

OPTI(3) specifies the maximum number of Newton iterations in each nonlinear equations solution. OPTI(3) $\geq$ 0. The default value is OPTI(3) = 10.

OPTI(4) specifies the maximum number of iterations in each linear equations solution. OPTI(4) $\geq$ 0. The default value is OPTI(4) = 100.

Constraints: if OPTI(1) $\geq$ 0 and OPTI(1) > 0 then OPTI(i) ≥ 0, $i = 2, 3, 4$.

18: OPTR(3,NPDE) — ***real*** array *Input*

On entry: OPTR may be used to specify the optional vectors u^{max}, w^s and w^t in the space and time monitors (see Section 8).

If an optional vector is not required then all its components should be set to 1.0.

OPTR(1,j), for $j = 1, 2, \ldots$,NPDE, specifies u_j^{max}, the approximate maximum absolute value of the jth component of u, as used in (4) and (7). OPTR(1,j) > 0.0 for $j = 1, 2, \ldots$,NPDE.

OPTR(2,j), for $j = 1, 2, \ldots$,NPDE, specifies w_j^s, the weighting factors used in the space monitor (see (4)) to indicate the relative importance of the jth component of u on the space monitor. OPTR(2,j) $\geq$ 0.0 for $j = 1, 2, \ldots$,NPDE.

OPTR(3,j), for $j = 1, 2, \ldots$,NPDE, specifies w_j^t, the weighting factors used in the time monitor (see (6)) to indicate the relative importance of the jth component of u on the time monitor. OPTR(3,j) $\geq$ 0.0 for $j = 1, 2, \ldots$,NPDE.

Constraints:

OPTR(1,j) > 0.0 for $j = 1, 2, \ldots$,NPDE and
OPTR(i,j) $\geq$ 0.0 for $i = 2, 3$ and $j = 1, 2, \ldots$,NPDE.

19: RWK(LENRWK) — ***real*** array *Workspace*

20: LENRWK — INTEGER *Input*

On entry: the dimension of the array RWK as declared in the (sub)program from which D03RAF is called.

The required value of LENRWK can not be determined exactly in advance, but a suggested value is

LENRWK = MAXPTS × NPDE × (5×l+18×NPDE+9) + 2 × MAXPTS,

where l = OPTI(1) if OPTI(1) $\neq$ 0 and l = 3 otherwise, and MAXPTS is the expected maximum number of grid points at any one level. If during the execution the supplied value is found to be too small then the routine returns with IFAIL = 3 and an estimated required size is printed on the current error message unit (see X04AAF).

Constraint: LENRWK $\geq$ NX × NY × NPDE × (14+18×NPDE) + 2 × NX × NY (the required size for the initial grid).

21: IWK(LENIWK) — INTEGER array *Output*

On entry: if IND = 0, IWK need not be set. Otherwise IWK must remain unchanged from a previous call to D03RAF.

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

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

IWK(2) contains the number of rejected time steps.

IWK(2+l) contains the total number of residual evaluations performed (i.e., the number of times PDEDEF was called) at grid level l;

IWK(2+m+l) contains the total number of Jacobian evaluations performed at grid level l;

IWK(2+2×m+l) contains the total number of Newton iterations performed at grid level l;

IWK(2+3×m+l) contains the total number of linear solver iterations performed at grid level l;

IWK(2+4×m+l) contains the maximum number of Newton iterations performed at any one time step at grid level l;

IWK(2+5×m+l) contains the maximum number of linear solver iterations performed at any one time step at grid level l;

for $l = 1, 2, \ldots, nl$, where nl is the number of levels used and m = OPTI(1) if OPTI(1) > 0 and m = 3 otherwise.

Note. The total and maximum numbers are cumulative over all calls to D03RAF. If the specified maximum number of Newton or linear solver iterations is exceeded at any stage, then the maximums above are set to the specified maximum plus one.

22: LENIWK — INTEGER *Input*

On entry: the dimension of the array IWK as declared in the (sub)program from which D03RAF is called.

The required value of LENIWK can not be determined exactly in advance, but a suggested value is LENIWK = MAXPTS × (14+5×m) + 7 × m + 2, where MAXPTS is the expected maximum number of grid points at any one level and m = OPTI(1) if OPTI(1) > 0 and m = 3 otherwise. If during the execution the supplied value is found to be too small then the routine returns with IFAIL = 3 and an estimated required size is printed on the current error message unit (see X04AAF).

Constraint: LENIWK $\geq$ 19 × NX × NY + 9 (the required size for the initial grid).

23: LWK(LENLWK) — LOGICAL array *Workspace*

24: LENLWK — INTEGER *Input*

On entry: the dimension of the array LWK as declared in the (sub)program from which D03RAF is called.

The required value of LENLWK can not be determined exactly in advanced, but a suggested value is LENLWK = MAXPTS + 1, where MAXPTS is the expected maximum number of grid points at any one level. If during the execution the supplied value is found to be too small then the routine returns with IFAIL = 3 and an estimated required size is printed on the current error message unit (see X04AAF).

Constraint: LENLWK $\geq$ NX $\times$ NY + 1 (the required size for the initial grid).

25: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03RAF. 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 are printed, and if ITRACE > 0, then output from the underlying solver is printed on the current advisory message unit (see X04ABF). This output contains details of the time integration, the nonlinear iteration and the linear solver. The advisory messages are given in greater detail as ITRACE increases. Setting ITRACE = 1 allows the user to monitor the progress of the integration without possibly excessive information.

26: IND — INTEGER *Input/Output*

On entry: IND must be set to 0 or 1.

IND = 0

starts the integration in time.

IND = 1

continues the integration after an earlier exit from the routine. In this case, only the following parameters may be reset between calls to D03RAF: TOUT, DT(2), DT(3), TOLS, TOLT, OPTI, OPTR, ITRACE and IFAIL.

Constraint: $0 \leq$ IND ≤ 1.

On exit: IND = 1.

27: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL= 1

On entry, NPDE < 1,

or TOUT $\leq$ TS,

or TOUT is too close to TS,

or IND = 0 and DT(1) < 0.0,

or DT(i) < 0.0 for $i = 2$ or 3

or DT(2) $>$ DT(3),

or IND = 0.0 and $0.0 <$ DT(1) $< 10 \times$ ***machine precision*** $\times$ max(|TS|,|TOUT|),

or IND = 0.0 and DT(1) > TOUT − TS,
or IND = 0.0 and DT(1) < DT(2) or DT(1) > DT(3),
or XMIN ≥ XMAX,
or XMAX too close to XMIN,
or YMIN ≥ YMAX,
or YMAX too close to YMIN,
or NX or NY < 4,
or TOLS or TOLT ≤ 0.0,
or OPTI(1) < 0,
or OPTI(1) > 0 and OPTI(j) < 0 for j = 2, 3 or 4,
or OPTR(1,j) ≤ 0.0 for some $j = 1, 2, \ldots$,NPDE,
or OPTR(2,j) < 0.0 for some $j = 1, 2, \ldots$,NPDE,
or OPTR(3,j) < 0.0 for some $j = 1, 2, \ldots$,NPDE,
or LENRWK, LENIWK or LENLWK too small for initial grid level,
or IND ≠ 0 or 1,
or IND = 1 on initial entry to D03RAF,

IFAIL = 2

The time step size to be attempted is less than the specified minimum size. This may occur following time step failures and subsequent step size reductions caused by one or more of the following:

the requested accuracy could not be achieved, i.e., TOLT is too small,

the maximum number of linear solver iterations, Newton iterations or Jacobian evaluations is too small,

ILU decomposition of the Jacobian matrix could not be performed, possibly due to singularity of the Jacobian.

Setting ITRACE to a higher value may provide further information.

In the latter two cases the user is advised to check their problem formulation in PDEDEF and/or BNDARY, and the initial values in PDEIV if appropriate.

IFAIL = 3

One or more of the workspace arrays is too small for the required number of grid points. An estimate of the required sizes for the current stage is output, but more space may be required at a later stage.

IFAIL = 4

IERR was set to 1 in the user-supplied subroutine MONITR, forcing control to be passed back to calling program. Integration was successful as far as T = TS.

IFAIL = 5

The integration has been completed but the maximum number of levels specified in OPTI(1) was insufficient at one or more time steps, meaning that the requested space accuracy could not be achieved. To avoid this warning either increase the value of OPTI(1) or decrease the value of TOLS.

7 Accuracy

There are three sources of error in the algorithm: space and time discretisation, and interpolation (linear) between grid levels. The space and time discretisation errors are controlled separately using the parameters TOLS and TOLT described in the following section, and the user should test the effects of varying these parameters. Interpolation errors are generally implicitly controlled by the refinement criterion since in areas where interpolation errors are potentially large, the space monitor will also be large. It can be shown that the global spatial accuracy is comparable to that which would be obtained on a uniform grid of the finest grid size. A full error analysis can be found in [3].

8 Further Comments

8.1 Algorithm Outline

The local uniform grid refinement method is summarised as follows

(1) Initialise the course base grid, an initial solution and an initial time step,
(2) Solve the system of PDEs on the current grid with the current time step,
(3) If the required accuracy in space and the maximum number of grid levels have not yet been reached:
  (a) Determine new finer grid at forward time level,
  (b) Get solution values at previous time level(s) on new grid,
  (c) Interpolate internal boundary values from old grid at forward time,
  (d) Get initial values for the Newton process at forward time,
  (e) Goto 2,
(4) Update the coarser grid solution using the finer grid values,
(5) Estimate error in time integration. If time error is acceptable advance time level,
(6) Determine new step size then goto 2 with coarse base as current grid.

8.2 Refinement Strategy

For each grid point i a space monitor μ_i^s is determined by

$$\mu_i^s = \max_{j=1,\mathrm{NPDE}} \{\gamma_j(|\,\Delta x^2 \frac{\partial^2}{\partial x^2} u_j(x_i,y_i,t)\,| + |\,\Delta y^2 \frac{\partial^2}{\partial y^2} u_j(x_i,y_i,t)\,|)\}, \qquad (3)$$

where Δx and Δy are the grid widths in the x and y directions; and x_i, y_i are the x and y co-ordinates at grid point i. The parameter γ_j is obtained from

$$\gamma_j = \frac{w_j^s}{u_j^{max}\,\sigma}, \qquad (4)$$

where σ is the user-supplied space tolerance; w_j^s is a weighting factor for the relative importance of the jth PDE component on the space monitor; and u_j^{max} is the approximate maximum absolute value of the jth component. A value for σ must be supplied by the user. Values for w_j^s and u_j^{max} must also be supplied but may be set to the value 1.0 if little information about the solution is known.

A new level of refinement is created if

$$\max_i\{\mu_i^s\} > 0.9 \text{ or } 1.0, \qquad (5)$$

depending on the grid level at the previous step in order to avoid fluctuations in the number of grid levels between time steps. If (5) is satisfied then all grid points for which $\mu_i^s > 0.25$ are flagged and surrounding cells are quartered in size.

No derefinement takes place as such, since at each time step the solution on the base grid is computed first and new finer grids are then created based on the new solution. Hence derefinement occurs implicitly. See Section 8.1.

8.3 Time Integration

The time integration is controlled using a time monitor calculated at each level l up to the maximum level used, given by

$$\mu_l^t = \sqrt{\frac{1}{\mathrm{N}} \sum_{j=1}^{\mathrm{NPDE}} w_j^t \sum_{i=1}^{\mathrm{NGPTS}(l)} \left(\frac{\Delta t}{\alpha_{ij}} u_t(x_i,y_i,t)\right)^2} \qquad (6)$$

where NGPTS(l) is the total number of points on grid level l; N = NGPTS(l) × NPDE; Δt is the current time step; u_t is the time derivative of u which is approximated by first-order finite differences; w_j^t is the time equivalent of the space weighting factor w_j^s; and α_{ij} is given by

$$\alpha_{ij} = \tau\left(\frac{u_j^{max}}{100} + |\,u(x_i,y_i,t)\,|\right) \qquad (7)$$

where u_j^{max} is as before, and τ is the user-specified time tolerance.

An integration step is rejected and retried at all levels if

$$\max_l \{\mu_l^t\} > 1.0. \quad (8)$$

9 Example

For this routine two examples are presented, in Section 9.1 and Section 9.2. In the example programs distributed to sites, there is a single example program for D03RAF, with a main program:

```
*     D03RAF Example Program Text
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. External Subroutines ..
      EXTERNAL         EX1, EX2
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03RAF 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 Section 9.1.1 and Section 9.2.1 respectively.

9.1 Example 1

This example stems from combustion theory and is a model for a single, one-step reaction of a mixture of two chemicals [5]. The PDE for the temperature of the mixture u is

$$\frac{\partial u}{\partial t} = d\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + D(1+\alpha-u)\exp\left(-\frac{\delta}{u}|\right)$$

for $0 \leq x, y \leq 1$ and $t \geq 0$, with initial conditions $u(x,y,0) = 1$ for $0 \leq x, y \leq 1$, and boundary conditions

$$u_x(0,y,t) = 0, u(1,y,t) = 1 \;\text{ for }\; 0 \leq y \leq 1,$$

$$u_y(x,0,t) = 0, u(x,1,t) = 1 \;\text{ for }\; 0 \leq x \leq 1.$$

The heat release parameter $\alpha = 1$, the Damkohler number $D = R\exp(\delta)/(\alpha\delta)$, the activation energy $\delta = 20$, the reaction rate $R = 5$, and the diffusion parameter $d = 0.1$.

For small times the temperature gradually increases in a circular region about the origin, and at about $t = 0.24$ 'ignition' occurs causing the temperature to suddenly jump from near unity to $1 + \alpha$, and a reaction front forms and propagates outwards, becoming steeper. Thus during the solution, just one grid level is used up to the ignition point, then two levels, and then three as the reaction front steepens.

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          MXLEV, NPDE, NPTS
      PARAMETER        (MXLEV=3,NPDE=1,NPTS=2000)
      INTEGER          LENIWK, LENRWK, LENLWK
      PARAMETER        (LENIWK=NPTS*(5*MXLEV+14)+2+7*MXLEV,
     +                 LENRWK=NPTS*NPDE*(5*MXLEV+9+18*NPDE)+NPTS*2,
     +                 LENLWK=NPTS+1)
*     .. Scalars in Common ..
      real             ALPHA, D, DELTA, DIFF, REAC
      INTEGER          IOUT
*     .. Arrays in Common ..
      real             TWANT(2)
*     .. Local Scalars ..
      real             TOLS, TOLT, TOUT, TS, XMAX, XMIN, YMAX, YMIN
      INTEGER          I, IFAIL, IND, ITRACE, J, MAXLEV, NX, NY
*     .. Local Arrays ..
      real             DT(3), OPTR(3,NPDE), RWK(LENRWK)
      INTEGER          IWK(LENIWK), OPTI(4)
      LOGICAL          LWK(LENLWK)
*     .. External Subroutines ..
      EXTERNAL         BNDRY1, D03RAF, MONIT1, PDEF1, PDEIV1
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Common blocks ..
      COMMON           /OTIME1/TWANT, IOUT
      COMMON           /PARAM1/ALPHA, DELTA, REAC, DIFF, D
*     .. Save statement ..
      SAVE             /OTIME1/, /PARAM1/
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Example 1'
      WRITE (NOUT,*)
*
*     Problem Parameters
*
      ALPHA = 1.0e0
      DELTA = 20.0e0
      REAC = 5.0e0
      DIFF = 0.1e0
      D = REAC*EXP(DELTA)/(ALPHA*DELTA)
*
      IND = 0
      ITRACE = 0
      TS = 0.0e0
      DT(1) = 0.1e-2
      DT(2) = 0.0e0
      DT(3) = 0.0e0
      TOUT = 0.24e0
      TWANT(1) = 0.24e0
      TWANT(2) = 0.25e0
      XMIN = 0.0e0
      XMAX = 1.0e0
      YMIN = 0.0e0
      YMAX = 1.0e0
      NX = 21
      NY = 21
      TOLS = 0.5e0
      TOLT = 0.01e0
```

```
      DO 20 I = 1, 4
         OPTI(I) = 0
   20 CONTINUE
      DO 60 J = 1, NPDE
         DO 40 I = 1, 3
            OPTR(I,J) = 1.0e0
   40    CONTINUE
   60 CONTINUE
*
      DO 120 IOUT = 1, 2
         IFAIL = -1
         TOUT = TWANT(IOUT)
         CALL D03RAF(NPDE,TS,TOUT,DT,XMIN,XMAX,YMIN,YMAX,NX,NY,TOLS,
     +               TOLT,PDEF1,BNDRY1,PDEIV1,MONIT1,OPTI,OPTR,RWK,
     +               LENRWK,IWK,LENIWK,LWK,LENLWK,ITRACE,IND,IFAIL)
*
*        Print statistics
*
         WRITE (NOUT,'('' Statistics:'')')
         WRITE (NOUT,'('' Time = '',F8.4)') TS
         WRITE (NOUT,'('' Total number of accepted timesteps ='', I5)')
     +     IWK(1)
         WRITE (NOUT,'('' Total number of rejected timesteps ='', I5)')
     +     IWK(2)
         WRITE (NOUT,*)
         WRITE (NOUT,
     +     '(''              T o t a l   n u m b e r   o f      '')')
         WRITE (NOUT,
     + '(''            Residual  Jacobian    Newton ''    , ''  Lin sys'')'
     +     )
         WRITE (NOUT,
     + '(''               evals     evals     iters ''    , ''    iters'')'
     +     )
         WRITE (NOUT,'('' At level '')')
         MAXLEV = 3
         DO 80 J = 1, MAXLEV
            IF (IWK(J+2).NE.0) WRITE (NOUT,'(I8,4I10)') J, IWK(J+2),
     +          IWK(J+2+MAXLEV), IWK(J+2+2*MAXLEV), IWK(J+2+3*MAXLEV)
*
   80    CONTINUE
         WRITE (NOUT,*)
         WRITE (NOUT,
     +     '(''              M a x i m u m   n u m b e r '', '' o f'')')
         WRITE (NOUT,
     +     '(''              Newton iters    Lin sys iters '')')
         WRITE (NOUT,'('' At level '')')
         DO 100 J = 1, MAXLEV
            IF (IWK(J+2).NE.0) WRITE (NOUT,'(I8,2I14)') J,
     +          IWK(J+2+4*MAXLEV), IWK(J+2+5*MAXLEV)
  100    CONTINUE
         WRITE (NOUT,*)
*
  120 CONTINUE
*
      RETURN
      END
*
```

```
      SUBROUTINE PDEIV1(NPTS,NPDE,T,X,Y,U)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              U(NPTS,NPDE), X(NPTS), Y(NPTS)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
*
      DO 20 I = 1, NPTS
         U(I,1) = 1.0e0
   20 CONTINUE
*
      RETURN
      END
*
      SUBROUTINE PDEF1(NPTS,NPDE,T,X,Y,U,UT,UX,UY,UXX,UXY,UYY,RES)
*     .. Scalar Arguments ..
      real             T
      INTEGER          NPDE, NPTS
*     .. Array Arguments ..
      real             RES(NPTS,NPDE), U(NPTS,NPDE), UT(NPTS,NPDE),
     +                 UX(NPTS,NPDE), UXX(NPTS,NPDE), UXY(NPTS,NPDE),
     +                 UY(NPTS,NPDE), UYY(NPTS,NPDE), X(NPTS), Y(NPTS)
*     .. Scalars in Common ..
      real             ALPHA, D, DELTA, DIFF, REAC
*     .. Local Scalars ..
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Common blocks ..
      COMMON           /PARAM1/ALPHA, DELTA, REAC, DIFF, D
*     .. Save statement ..
      SAVE             /PARAM1/
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         RES(I,1) = UT(I,1) - DIFF*(UXX(I,1)+UYY(I,1)) -
     +              D*(1.0e0+ALPHA-U(I,1))*EXP(-DELTA/U(I,1))
   20 CONTINUE
*
      RETURN
      END
*
      SUBROUTINE BNDRY1(NPTS,NPDE,T,X,Y,U,UT,UX,UY,NBPTS,LBND,RES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NBPTS, NPDE, NPTS
*     .. Array Arguments ..
      real              RES(NPTS,NPDE), U(NPTS,NPDE), UT(NPTS,NPDE),
     +                  UX(NPTS,NPDE), UY(NPTS,NPDE), X(NPTS), Y(NPTS)
      INTEGER           LBND(NBPTS)
*     .. Local Scalars ..
      real              TOL
      INTEGER           I, J
```

```
*     .. External Functions ..
      real              X02AJF
      EXTERNAL          X02AJF
*     .. Intrinsic Functions ..
      INTRINSIC         ABS
*     .. Executable Statements ..
*
      TOL = 10.e0*X02AJF()
*
      DO 20 I = 1, NBPTS
         J = LBND(I)
         IF (ABS(X(J)).LE.TOL) THEN
            RES(J,1) = UX(J,1)
         ELSE IF (ABS(X(J)-1.0e0).LE.TOL) THEN
            RES(J,1) = U(J,1) - 1.0e0
         ELSE IF (ABS(Y(J)).LE.TOL) THEN
            RES(J,1) = UY(J,1)
         ELSE IF (ABS(Y(J)-1.0e0).LE.TOL) THEN
            RES(J,1) = U(J,1) - 1.0e0
         END IF
   20 CONTINUE
*
      RETURN
      END
*
      SUBROUTINE MONIT1(NPDE,T,DT,DTNEW,TLAST,NLEV,NGPTS,XPTS,YPTS,LSOL,
     +                  SOL,IERR)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              DT, DTNEW, T
      INTEGER           IERR, NLEV, NPDE
      LOGICAL           TLAST
*     .. Array Arguments ..
      real              SOL(*), XPTS(*), YPTS(*)
      INTEGER           LSOL(NLEV), NGPTS(NLEV)
*     .. Scalars in Common ..
      INTEGER           IOUT
*     .. Arrays in Common ..
      real              TWANT(2)
*     .. Local Scalars ..
      INTEGER           I, IPSOL, IPT, LEVEL, NPTS
*     .. Common blocks ..
      COMMON            /OTIME1/TWANT, IOUT
*     .. Save statement ..
      SAVE              /OTIME1/
*     .. Executable Statements ..
*
      IF (TLAST) THEN
*
*        Print solution
*
         IF (IOUT.EQ.2) THEN
            WRITE (NOUT,
     +'('' Solution at every 4th grid point '',    ''in level 1 at time
     +'', F8.4,'':'')') T
```

```
            WRITE (NOUT,*)
            WRITE (NOUT,'(7X,''x'',10X,''y'',8X,''approx u'')')
            WRITE (NOUT,*)
            LEVEL = 1
            NPTS = NGPTS(LEVEL)
            IPSOL = LSOL(LEVEL)
            IPT = 1
            DO 20 I = 1, NPTS, 4
               WRITE (NOUT,'(3(1X,D11.4))') XPTS(IPT+I-1),
     +           YPTS(IPT+I-1), SOL(IPSOL+I)
   20       CONTINUE
            WRITE (NOUT,*)
         END IF
      END IF
*
      RETURN
      END
*
```

9.1.2 Program Data

None.

9.1.3 Program Results

```
D03RAF Example Program Results


Example 1

Statistics:
Time =   0.2400
Total number of accepted timesteps =   75
Total number of rejected timesteps =    0

           T o t a l  n u m b e r  o f
        Residual  Jacobian    Newton   Lin sys
           evals     evals     iters     iters
At level
     1       600        75       150       159

           M a x i m u m  n u m b e r  o f
            Newton iters    Lin sys iters
At level
     1              2              2

Solution at every 4th grid point in level 1 at time   0.2500:

      x           y         approx u

 0.0000E+00  0.0000E+00  0.2000E+01
 0.2000E+00  0.0000E+00  0.2000E+01
 0.4000E+00  0.0000E+00  0.2000E+01
 0.6000E+00  0.0000E+00  0.2000E+01
 0.8000E+00  0.0000E+00  0.1240E+01
 0.1000E+01  0.0000E+00  0.1000E+01
 0.1500E+00  0.5000E-01  0.2000E+01
 0.3500E+00  0.5000E-01  0.2000E+01
```

0.5500E+00	0.5000E-01	0.2000E+01
0.7500E+00	0.5000E-01	0.1645E+01
0.9500E+00	0.5000E-01	0.1048E+01
0.1000E+00	0.1000E+00	0.2000E+01
0.3000E+00	0.1000E+00	0.2000E+01
0.5000E+00	0.1000E+00	0.2000E+01
0.7000E+00	0.1000E+00	0.1999E+01
0.9000E+00	0.1000E+00	0.1097E+01
0.5000E-01	0.1500E+00	0.2000E+01
0.2500E+00	0.1500E+00	0.2000E+01
0.4500E+00	0.1500E+00	0.2000E+01
0.6500E+00	0.1500E+00	0.2000E+01
0.8500E+00	0.1500E+00	0.1154E+01
0.0000E+00	0.2000E+00	0.2000E+01
0.2000E+00	0.2000E+00	0.2000E+01
0.4000E+00	0.2000E+00	0.2000E+01
0.6000E+00	0.2000E+00	0.2000E+01
0.8000E+00	0.2000E+00	0.1240E+01
0.1000E+01	0.2000E+00	0.1000E+01
0.1500E+00	0.2500E+00	0.2000E+01
0.3500E+00	0.2500E+00	0.2000E+01
0.5500E+00	0.2500E+00	0.2000E+01
0.7500E+00	0.2500E+00	0.1635E+01
0.9500E+00	0.2500E+00	0.1048E+01
0.1000E+00	0.3000E+00	0.2000E+01
0.3000E+00	0.3000E+00	0.2000E+01
0.5000E+00	0.3000E+00	0.2000E+01
0.7000E+00	0.3000E+00	0.1999E+01
0.9000E+00	0.3000E+00	0.1097E+01
0.5000E-01	0.3500E+00	0.2000E+01
0.2500E+00	0.3500E+00	0.2000E+01
0.4500E+00	0.3500E+00	0.2000E+01
0.6500E+00	0.3500E+00	0.2000E+01
0.8500E+00	0.3500E+00	0.1153E+01
0.0000E+00	0.4000E+00	0.2000E+01
0.2000E+00	0.4000E+00	0.2000E+01
0.4000E+00	0.4000E+00	0.2000E+01
0.6000E+00	0.4000E+00	0.2000E+01
0.8000E+00	0.4000E+00	0.1234E+01
0.1000E+01	0.4000E+00	0.1000E+01
0.1500E+00	0.4500E+00	0.2000E+01
0.3500E+00	0.4500E+00	0.2000E+01
0.5500E+00	0.4500E+00	0.2000E+01
0.7500E+00	0.4500E+00	0.1508E+01
0.9500E+00	0.4500E+00	0.1048E+01
0.1000E+00	0.5000E+00	0.2000E+01
0.3000E+00	0.5000E+00	0.2000E+01
0.5000E+00	0.5000E+00	0.2000E+01
0.7000E+00	0.5000E+00	0.1993E+01
0.9000E+00	0.5000E+00	0.1095E+01
0.5000E-01	0.5500E+00	0.2000E+01
0.2500E+00	0.5500E+00	0.2000E+01
0.4500E+00	0.5500E+00	0.2000E+01
0.6500E+00	0.5500E+00	0.2000E+01
0.8500E+00	0.5500E+00	0.1145E+01
0.0000E+00	0.6000E+00	0.2000E+01
0.2000E+00	0.6000E+00	0.2000E+01
0.4000E+00	0.6000E+00	0.2000E+01

```
 0.6000E+00  0.6000E+00  0.2000E+01
 0.8000E+00  0.6000E+00  0.1200E+01
 0.1000E+01  0.6000E+00  0.1000E+01
 0.1500E+00  0.6500E+00  0.2000E+01
 0.3500E+00  0.6500E+00  0.2000E+01
 0.5500E+00  0.6500E+00  0.2000E+01
 0.7500E+00  0.6500E+00  0.1253E+01
 0.9500E+00  0.6500E+00  0.1044E+01
 0.1000E+00  0.7000E+00  0.1999E+01
 0.3000E+00  0.7000E+00  0.1999E+01
 0.5000E+00  0.7000E+00  0.1993E+01
 0.7000E+00  0.7000E+00  0.1279E+01
 0.9000E+00  0.7000E+00  0.1082E+01
 0.5000E-01  0.7500E+00  0.1645E+01
 0.2500E+00  0.7500E+00  0.1635E+01
 0.4500E+00  0.7500E+00  0.1508E+01
 0.6500E+00  0.7500E+00  0.1253E+01
 0.8500E+00  0.7500E+00  0.1109E+01
 0.0000E+00  0.8000E+00  0.1240E+01
 0.2000E+00  0.8000E+00  0.1240E+01
 0.4000E+00  0.8000E+00  0.1234E+01
 0.6000E+00  0.8000E+00  0.1200E+01
 0.8000E+00  0.8000E+00  0.1119E+01
 0.1000E+01  0.8000E+00  0.1000E+01
 0.1500E+00  0.8500E+00  0.1154E+01
 0.3500E+00  0.8500E+00  0.1153E+01
 0.5500E+00  0.8500E+00  0.1145E+01
 0.7500E+00  0.8500E+00  0.1109E+01
 0.9500E+00  0.8500E+00  0.1029E+01
 0.1000E+00  0.9000E+00  0.1097E+01
 0.3000E+00  0.9000E+00  0.1097E+01
 0.5000E+00  0.9000E+00  0.1095E+01
 0.7000E+00  0.9000E+00  0.1082E+01
 0.9000E+00  0.9000E+00  0.1039E+01
 0.5000E-01  0.9500E+00  0.1048E+01
 0.2500E+00  0.9500E+00  0.1048E+01
 0.4500E+00  0.9500E+00  0.1048E+01
 0.6500E+00  0.9500E+00  0.1044E+01
 0.8500E+00  0.9500E+00  0.1029E+01
 0.0000E+00  0.1000E+01  0.1000E+01
 0.2000E+00  0.1000E+01  0.1000E+01
 0.4000E+00  0.1000E+01  0.1000E+01
 0.6000E+00  0.1000E+01  0.1000E+01
 0.8000E+00  0.1000E+01  0.1000E+01
 0.1000E+01  0.1000E+01  0.1000E+01

Statistics:
Time =   0.2500
Total number of accepted timesteps =  180
Total number of rejected timesteps =    1

           T o t a l   n u m b e r   o f
        Residual  Jacobian    Newton   Lin sys
           evals     evals     iters     iters
At level
      1     1468       181       382       391
      2      662        82       170       170
      3      176        22        44        44
```

```
                 M a x i m u m   n u m b e r   o f
                  Newton iters     Lin sys iters
     At level
          1            4                 2
          2            4                 1
          3            2                 1
```

9.2 Example 2

This example is taken from a multispecies food web model, in which predator-prey relationships in a spatial domain are simulated [6]. In this example there is just one species each of prey and predator, and the two PDEs for the concentrations c_1 and c_2 of the prey and the predator respectively are

$$\frac{\partial c_1}{\partial t} = c_1(b_1 + a_{11}c_1 + a_{12}c_2) + d_1\left(\frac{\partial^2 c_1}{\partial x^2} + \frac{\partial^2 c_1}{\partial y^2}\right),$$

$$0 = c_2(b_2 + a_{21}c_1 + a_{22}c_2) + d_2\left(\frac{\partial^2 c_2}{\partial x^2} + \frac{\partial^2 c_2}{\partial y^2}\right),$$

with $a_{11} = a_{22} = -1$, $a_{12} = -0.5 \times 10^{-6}$, and $a_{21} = 10^4$, and

$$b_1 = 1 + \alpha xy + \beta \sin(4\pi x)\sin(4\pi y),$$

where $\alpha = 50$ and $\beta = 300$, and $b_2 = -b_1$.

The initial conditions are taken to be simple peaked functions which satisfy the boundary conditions and very nearly satisfy the PDEs:

$$c_1 = 10 + (16x(1-x)y(1-y))^2,$$

$$c_2 = b_2 + a_{21}c_1,$$

and the boundary conditions are of Neumann type, i.e., zero normal derivatives everywhere.

During the solution a number of peaks and troughs develop across the domain, and so the number of levels required increases with time. Since the solution varies rapidly in space across the whole of the domain, refinement at intermediate levels tends to occur at all points of the domain.

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          MXLEV, NPDE, NPTS
      PARAMETER        (MXLEV=4,NPDE=2,NPTS=8000)
      INTEGER          LENIWK, LENRWK, LENLWK
      PARAMETER        (LENIWK=NPTS*(5*MXLEV+14)+2+7*MXLEV,
     +                 LENRWK=NPTS*NPDE*(5*MXLEV+9+18*NPDE)+NPTS*2,
     +                 LENLWK=NPTS+1)
*     .. Scalars in Common ..
      real             ALPHA, BETA, PI
      INTEGER          IOUT
*     .. Arrays in Common ..
      real             TWANT(2)
```

```
*     .. Local Scalars ..
      real             TOLS, TOLT, TOUT, TS, XMAX, XMIN, XX, YMAX, YMIN
      INTEGER          I, IFAIL, IND, ITRACE, J, MAXLEV, NX, NY
*     .. Local Arrays ..
      real             DT(3), OPTR(3,NPDE), RWK(LENRWK)
      INTEGER          IWK(LENIWK), OPTI(4)
      LOGICAL          LWK(LENLWK)
*     .. External Functions ..
      real             X01AAF
      EXTERNAL         X01AAF
*     .. External Subroutines ..
      EXTERNAL         BNDRY2, D03RAF, MONIT2, PDEF2, PDEIV2
*     .. Common blocks ..
      COMMON           /OTIME2/TWANT, IOUT
      COMMON           /PARAM2/ALPHA, BETA, PI
*     .. Save statement ..
      SAVE             /OTIME2/, /PARAM2/
*     .. Executable Statements ..
      WRITE (NOUT,*)
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Example 2'
      WRITE (NOUT,*)
*
      XX = 0.0e0
      PI = X01AAF(XX)
      ALPHA = 50.0e0
      BETA = 300.0e0
*
      IND = 0
      ITRACE = 0
      TS = 0.0e0
      TWANT(1) = 0.01e0
      TWANT(2) = 0.025e0
      DT(1) = 0.5e-3
      DT(2) = 1.0e-6
      DT(3) = 0.0e0
      XMIN = 0.0e0
      XMAX = 1.0e0
      YMIN = 0.0e0
      YMAX = 1.0e0
      TOLS = 0.075e0
      TOLT = 0.1e0
      NX = 11
      NY = 11
      OPTI(1) = 4
      DO 20 I = 2, 4
         OPTI(I) = 0
   20 CONTINUE
      OPTR(1,1) = 250.0e0
      OPTR(1,2) = 1.5e6
      DO 60 J = 1, NPDE
         DO 40 I = 2, 3
            OPTR(I,J) = 1.0e0
   40    CONTINUE
   60 CONTINUE
*
      DO 120 IOUT = 1, 2
```

```
      IFAIL = -1
      TOUT = TWANT(IOUT)
      CALL D03RAF(NPDE,TS,TOUT,DT,XMIN,XMAX,YMIN,YMAX,NX,NY,TOLS,
     +            TOLT,PDEF2,BNDRY2,PDEIV2,MONIT2,OPTI,OPTR,RWK,
     +            LENRWK,IWK,LENIWK,LWK,LENLWK,ITRACE,IND,IFAIL)
*
*     Print statistics
*
      MAXLEV = OPTI(1)
      WRITE (NOUT,'('' Statistics:'')')
      WRITE (NOUT,'('' Time = '',F8.4)') TS
      WRITE (NOUT,'('' Total number of accepted timesteps ='', I5)')
     +  IWK(1)
      WRITE (NOUT,'('' Total number of rejected timesteps ='', I5)')
     +  IWK(2)
      WRITE (NOUT,*)
      WRITE (NOUT,
     +  '(''          T o t a l   n u m b e r   o f     '')')
      WRITE (NOUT,
     + '(''        Residual  Jacobian    Newton ''   , ''  Lin sys'')'
     +  )
      WRITE (NOUT,
     + '(''            evals     evals     iters ''   , ''    iters'')'
     +  )
      WRITE (NOUT,'('' At level '')')
      MAXLEV = OPTI(1)
      DO 80 J = 1, MAXLEV
         IF (IWK(J+2).NE.0) WRITE (NOUT,'(I6,4I10)') J, IWK(J+2),
     +       IWK(J+2+MAXLEV), IWK(J+2+2*MAXLEV), IWK(J+2+3*MAXLEV)
*
   80 CONTINUE
      WRITE (NOUT,*)
      WRITE (NOUT,
     +  '(''          M a x i m u m   n u m b e r '', '' o f'')')
      WRITE (NOUT,
     +  '(''            Newton iters    Lin sys iters '')')
      WRITE (NOUT,'('' At level '')')
      DO 100 J = 1, MAXLEV
         IF (IWK(J+2).NE.0) WRITE (NOUT,'(I6,2I14)') J,
     +       IWK(J+2+4*MAXLEV), IWK(J+2+5*MAXLEV)
  100 CONTINUE
      WRITE (NOUT,*)
*
  120 CONTINUE
*
      RETURN
      END
*
      SUBROUTINE PDEIV2(NPTS,NPDE,T,X,Y,U)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              U(NPTS,NPDE), X(NPTS), Y(NPTS)
*     .. Scalars in Common ..
      real              ALPHA, BETA, PI
```

```
*     .. Local Scalars ..
      real             B2, FP
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        SIN
*     .. Common blocks ..
      COMMON           /PARAM2/ALPHA, BETA, PI
*     .. Save statement ..
      SAVE             /PARAM2/
*     .. Executable Statements ..
*
      FP = 4.0e0*PI
*
      DO 20 I = 1, NPTS
         B2 = -1.0e0 - ALPHA*X(I)*Y(I) - BETA*SIN(FP*X(I))*SIN(FP*Y(I))
         U(I,1) = 1.0e1 + (16.0e0*X(I)*(1.0e0-X(I))*Y(I)*(1.0e0-Y(I)))
     +            **2
         U(I,2) = B2 + 1.0e4*U(I,1)
   20 CONTINUE
*
      RETURN
      END
*
      SUBROUTINE PDEF2(NPTS,NPDE,T,X,Y,U,UT,UX,UY,UXX,UXY,UYY,RES)
*     .. Scalar Arguments ..
      real             T
      INTEGER          NPDE, NPTS
*     .. Array Arguments ..
      real             RES(NPTS,NPDE), U(NPTS,NPDE), UT(NPTS,NPDE),
     +                 UX(NPTS,NPDE), UXX(NPTS,NPDE), UXY(NPTS,NPDE),
     +                 UY(NPTS,NPDE), UYY(NPTS,NPDE), X(NPTS), Y(NPTS)
*     .. Scalars in Common ..
      real             ALPHA, BETA, PI
*     .. Local Scalars ..
      real             B1, B2, FP
      INTEGER          I
*     .. Intrinsic Functions ..
      INTRINSIC        SIN
*     .. Common blocks ..
      COMMON           /PARAM2/ALPHA, BETA, PI
*     .. Save statement ..
      SAVE             /PARAM2/
*     .. Executable Statements ..
      FP = 4.0e0*PI
*
      DO 20 I = 1, NPTS
         B1 = 1.0e0 + ALPHA*X(I)*Y(I) + BETA*SIN(FP*X(I))*SIN(FP*Y(I))
         B2 = -B1
         RES(I,1) = UT(I,1) - (UXX(I,1)+UYY(I,1)) - U(I,1)*(B1-U(I,1)
     +              -0.5e-6*U(I,2))
         RES(I,2) = -0.05e0*(UXX(I,2)+UYY(I,2)) - U(I,2)
     +              *(B2+1.0e4*U(I,1)-U(I,2))
   20 CONTINUE
*
      RETURN
      END
*
```

```
      SUBROUTINE BNDRY2(NPTS,NPDE,T,X,Y,U,UT,UX,UY,NBPTS,LBND,RES)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NBPTS, NPDE, NPTS
*     .. Array Arguments ..
      real              RES(NPTS,NPDE), U(NPTS,NPDE), UT(NPTS,NPDE),
     +                  UX(NPTS,NPDE), UY(NPTS,NPDE), X(NPTS), Y(NPTS)
      INTEGER           LBND(NBPTS)
*     .. Local Scalars ..
      real              TOL
      INTEGER           I, J
*     .. External Functions ..
      real              X02AJF
      EXTERNAL          X02AJF
*     .. Intrinsic Functions ..
      INTRINSIC         ABS
*     .. Executable Statements ..
*
      TOL = 10.e0*X02AJF()
*
      DO 20 I = 1, NBPTS
         J = LBND(I)
         IF (ABS(X(J)).LE.TOL .OR. ABS(X(J)-1.0e0).LE.TOL) THEN
            RES(J,1) = UX(J,1)
            RES(J,2) = UX(J,2)
         ELSE IF (ABS(Y(J)).LE.TOL .OR. ABS(Y(J)-1.0e0).LE.TOL) THEN
            RES(J,1) = UY(J,1)
            RES(J,2) = UY(J,2)
         END IF
   20 CONTINUE
*
      RETURN
      END
*
      SUBROUTINE MONIT2(NPDE,T,DT,DTNEW,TLAST,NLEV,NGPTS,XPTS,YPTS,LSOL,
     +                  SOL,IERR)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              DT, DTNEW, T
      INTEGER           IERR, NLEV, NPDE
      LOGICAL           TLAST
*     .. Array Arguments ..
      real              SOL(*), XPTS(*), YPTS(*)
      INTEGER           LSOL(NLEV), NGPTS(NLEV)
*     .. Scalars in Common ..
      INTEGER           IOUT
*     .. Arrays in Common ..
      real              TWANT(2)
*     .. Local Scalars ..
      INTEGER           I, IPSOL, IPT, LEVEL, NPTS
*     .. Common blocks ..
      COMMON            /OTIME2/TWANT, IOUT
*     .. Save statement ..
      SAVE              /OTIME2/
*     .. Executable Statements ..
*
```

```
      IF (TLAST) THEN
*
*        Print solution
*
         IF (IOUT.EQ.2) THEN
            WRITE (NOUT,
     +'('' Solution at every 2nd grid point '',    ''in level 1 at time
     +'', F8.4,'':'')') T
            WRITE (NOUT,*)
            WRITE (NOUT,
     +        '(7X,''x'',10X,''y'',9X,''approx c1'',3X,''approx c2'')')
            WRITE (NOUT,*)
            LEVEL = 1
            NPTS = NGPTS(LEVEL)
            IPSOL = LSOL(LEVEL)
            IPT = 1
            DO 20 I = 1, NPTS, 2
               WRITE (NOUT,'(2(1X,D11.4),2X,D11.4,2X,D11.4)')
     +           XPTS(IPT+I-1), YPTS(IPT+I-1), SOL(IPSOL+I),
     +           SOL(IPSOL+NPTS+I)
   20       CONTINUE
            WRITE (NOUT,*)
         END IF
      END IF
*
      RETURN
      END
```

9.2.2 Program Data

None.

9.2.3 Program Results

```
 D03RAF Example Program Results


 Example 2

 Statistics:
 Time =   0.0100
 Total number of accepted timesteps =   14
 Total number of rejected timesteps =    0

             T o t a l  n u m b e r  o f
          Residual  Jacobian   Newton  Lin sys
             evals     evals    iters    iters
 At level
     1       196        14       28       42
     2       168        12       24       34
     3        70         5       10       16

             M a x i m u m  n u m b e r  o f
              Newton iters   Lin sys iters
 At level
     1             2              2
     2             2              2
     3             2              3
```

```
Solution at every 2nd grid point in level 1 at time   0.0250:

      x           y          approx c1    approx c2

 0.0000E+00  0.0000E+00   0.6615E+02   0.6615E+06
 0.2000E+00  0.0000E+00   0.5138E+02   0.5137E+06
 0.4000E+00  0.0000E+00   0.1274E+02   0.1275E+06
 0.6000E+00  0.0000E+00   0.5217E+02   0.5217E+06
 0.8000E+00  0.0000E+00   0.1684E+02   0.1684E+06
 0.1000E+01  0.0000E+00   0.4618E+01   0.4619E+05
 0.1000E+00  0.1000E+00   0.8832E+02   0.8829E+06
 0.3000E+00  0.1000E+00   0.1897E+02   0.1898E+06
 0.5000E+00  0.1000E+00   0.3109E+02   0.3109E+06
 0.7000E+00  0.1000E+00   0.5115E+02   0.5114E+06
 0.9000E+00  0.1000E+00   0.6498E+01   0.6526E+05
 0.0000E+00  0.2000E+00   0.5138E+02   0.5137E+06
 0.2000E+00  0.2000E+00   0.4480E+02   0.4479E+06
 0.4000E+00  0.2000E+00   0.1763E+02   0.1764E+06
 0.6000E+00  0.2000E+00   0.4849E+02   0.4848E+06
 0.8000E+00  0.2000E+00   0.2308E+02   0.2309E+06
 0.1000E+01  0.2000E+00   0.1998E+02   0.1998E+06
 0.1000E+00  0.3000E+00   0.1897E+02   0.1898E+06
 0.3000E+00  0.3000E+00   0.3745E+02   0.3744E+06
 0.5000E+00  0.3000E+00   0.2815E+02   0.2815E+06
 0.7000E+00  0.3000E+00   0.2379E+02   0.2380E+06
 0.9000E+00  0.3000E+00   0.6076E+02   0.6074E+06
 0.0000E+00  0.4000E+00   0.1274E+02   0.1275E+06
 0.2000E+00  0.4000E+00   0.1763E+02   0.1764E+06
 0.4000E+00  0.4000E+00   0.5816E+02   0.5813E+06
 0.6000E+00  0.4000E+00   0.1425E+02   0.1428E+06
 0.8000E+00  0.4000E+00   0.5783E+02   0.5782E+06
 0.1000E+01  0.4000E+00   0.6492E+02   0.6492E+06
 0.1000E+00  0.5000E+00   0.3109E+02   0.3109E+06
 0.3000E+00  0.5000E+00   0.2815E+02   0.2815E+06
 0.5000E+00  0.5000E+00   0.2966E+02   0.2966E+06
 0.7000E+00  0.5000E+00   0.3422E+02   0.3422E+06
 0.9000E+00  0.5000E+00   0.4004E+02   0.4003E+06
 0.0000E+00  0.6000E+00   0.5217E+02   0.5217E+06
 0.2000E+00  0.6000E+00   0.4849E+02   0.4848E+06
 0.4000E+00  0.6000E+00   0.1425E+02   0.1428E+06
 0.6000E+00  0.6000E+00   0.7001E+02   0.6998E+06
 0.8000E+00  0.6000E+00   0.2397E+02   0.2398E+06
 0.1000E+01  0.6000E+00   0.1981E+02   0.1981E+06
 0.1000E+00  0.7000E+00   0.5115E+02   0.5114E+06
 0.3000E+00  0.7000E+00   0.2379E+02   0.2380E+06
 0.5000E+00  0.7000E+00   0.3422E+02   0.3422E+06
 0.7000E+00  0.7000E+00   0.5069E+02   0.5067E+06
 0.9000E+00  0.7000E+00   0.3143E+02   0.3145E+06
 0.0000E+00  0.8000E+00   0.1684E+02   0.1684E+06
 0.2000E+00  0.8000E+00   0.2308E+02   0.2309E+06
 0.4000E+00  0.8000E+00   0.5783E+02   0.5781E+06
 0.6000E+00  0.8000E+00   0.2397E+02   0.2398E+06
 0.8000E+00  0.8000E+00   0.7164E+02   0.7162E+06
 0.1000E+01  0.8000E+00   0.8397E+02   0.8397E+06
 0.1000E+00  0.9000E+00   0.6498E+01   0.6526E+05
 0.3000E+00  0.9000E+00   0.6076E+02   0.6074E+06
 0.5000E+00  0.9000E+00   0.4004E+02   0.4003E+06
```

```
 0.7000E+00  0.9000E+00   0.3143E+02   0.3145E+06
 0.9000E+00  0.9000E+00   0.1403E+03   0.1403E+07
 0.0000E+00  0.1000E+01   0.4618E+01   0.4619E+05
 0.2000E+00  0.1000E+01   0.1998E+02   0.1998E+06
 0.4000E+00  0.1000E+01   0.6492E+02   0.6491E+06
 0.6000E+00  0.1000E+01   0.1980E+02   0.1980E+06
 0.8000E+00  0.1000E+01   0.8397E+02   0.8396E+06
 0.1000E+01  0.1000E+01   0.1075E+03   0.1075E+07

Statistics:
Time =    0.0250
Total number of accepted timesteps =   29
Total number of rejected timesteps =    0

            T o t a l   n u m b e r   o f
         Residual  Jacobian    Newton   Lin sys
            evals     evals     iters     iters
At level
    1        406        29        58        87
    2        378        27        54        79
    3        280        20        40        61
    4         98         7        14        27

            M a x i m u m   n u m b e r   o f
             Newton iters    Lin sys iters
At level
    1             2             2
    2             2             2
    3             2             3
    4             2             3
```

D03RBF – NAG Fortran Library Routine Document

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

D03RBF integrates a system of linear or nonlinear, time-dependent partial differential equations (PDEs) in two space dimensions on a rectilinear domain. The method of lines is employed to reduce the PDEs to a system of ordinary differential equations (ODEs) which are solved using a backward differentiation formula (BDF) method. The resulting system of nonlinear equations is solved using a modified Newton method and a Bi-CGSTAB iterative linear solver with ILU preconditioning. Local uniform grid refinement is used to improve the accuracy of the solution. D03RBF originates from the VLUGR2 package [1] [2].

2 Specification

```
      SUBROUTINE D03RBF(NPDE, TS, TOUT, DT, TOLS, TOLT, INIDOM, PDEDEF,
     1                  BNDARY, PDEIV, MONITR, OPTI, OPTR, RWK, LENRWK,
     2                  IWK, LENIWK, LWK, LENLWK, ITRACE, IND, IFAIL)
      INTEGER           NPDE, OPTI(4), LENRWK, IWK(LENIWK), LENIWK,
     1                  LENLWK, ITRACE, IND, IFAIL
      real              TS, TOUT, DT(3), TOLS, TOLT, OPTR(3,NPDE),
     1                  RWK(LENRWK)
      LOGICAL           LWK(LENLWK)
      EXTERNAL          INIDOM, PDEDEF, BNDARY, PDEIV, MONITR
```

3 Description

D03RBF integrates the system of PDEs:

$$F_j(t,x,y,u,u_t,u_x,u_y,u_{xx},u_{xy},u_{yy}) = 0, \quad j = 1,2,\ldots,\text{NPDE}, \quad (x,y) \in \Omega, \quad t_0 \leq t \leq t_{\text{out}}, \qquad (1)$$

where Ω is an arbitrary rectilinear domain, i.e., a domain bounded by perpendicular straight lines. If the domain is rectangular then it is recommended that D03RAF is used.

The vector u is the set of solution values

$$u(x,y,t) = [u_1(x,y,t),\ldots,u_{\text{NPDE}}(x,y,t)]^T,$$

and u_t denotes partial differentiation with respect to t, and similarly for u_x etc.

The functions F_j must be supplied by the user in a subroutine PDEDEF. Similarly the initial values of the functions $u(x,y,t)$ for $(x,y) \in \Omega$ must be specified at $t = t_0$ in a subroutine PDEIV.

Note that whilst complete generality is offered by the master equations (1), D03RBF is not appropriate for all PDEs. In particular, hyperbolic systems should not be solved using this routine. Also, at least one component of u_t must appear in the system of PDEs.

The boundary conditions must be supplied by the user in a subroutine BNDARY in the form

$$G_j(t,x,y,u,u_t,u_x,u_y) = 0 \quad j = 1,2,\ldots,\text{NPDE}, \quad (x,y) \in \partial\Omega, \quad t_0 \leq t \leq t_{\text{out}}. \qquad (2)$$

The domain is covered by a uniform coarse base grid specified by the user, and nested finer uniform subgrids are subsequently created in regions with high spatial activity. The refinement is controlled using a space monitor which is computed from the current solution and a user-supplied space tolerance TOLS. A number of optional parameters, e.g., the maximum number of grid levels at any time, and some weighting factors, can be specified in the arrays OPTI and OPTR. Further details of the refinement strategy can be found in Section 8.

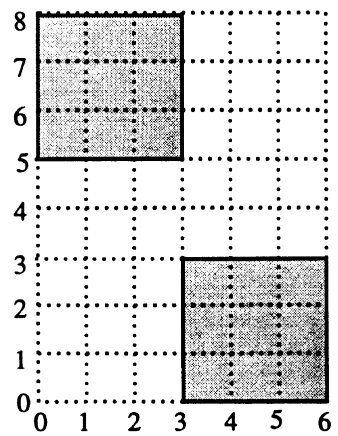
The system of PDEs and the boundary conditions are discretised in space on each grid using a standard second-order finite difference scheme (centred on the internal domain and one-sided at the boundaries), and the resulting system of ODEs is integrated in time using a second-order, two-step, implicit BDF method with variable step size. The time integration is controlled using a time monitor computed at each grid level from the current solution and a user-supplied time tolerance TOLT, and some further optional user-specified weighting factors held in OPTR (see Section 8 for details). The time monitor is used to compute a new step size, subject to restrictions on the size of the change between steps, and (optional) user-specified maximum and minimum step sizes held in DT. The step size is adjusted so that the remaining integration interval is an integer number times Δt. In this way a solution is obtained at $t = t_{\text{out}}$.

A modified Newton method is used to solve the nonlinear equations arising from the time integration. The user may specify (in OPTI) the maximum number of Newton iterations to be attempted. A Jacobian matrix is calculated at the beginning of each time step. If the Newton process diverges or the maximum number of iterations is exceeded, a new Jacobian is calculated using the most recent iterates and the Newton process is restarted. If convergence is not achieved after the (optional) user-specified maximum number of new Jacobian evaluations, the time step is retried with $\Delta t = \Delta t/4$. The linear systems arising from the Newton iteration are solved using a Bi-CGSTAB iterative method, in combination with ILU preconditioning. The maximum number of iterations can be specified by the user in OPTI.

In order to define the base grid the user must first specify a virtual uniform rectangular grid which contains the entire base grid. The position of the virtual grid in physical (x, y) space is given by the (x, y) co-ordinates of its boundaries. The number of points n_x and n_y in the x and y directions must also be given, corresponding to the number of columns and rows respectively. This is sufficient to determine precisely the (x, y) co-ordinates of all virtual grid points. Each virtual grid point is then referred to by integer co-ordinates (v_x, v_y), where $(0,0)$ corresponds to the lower-left corner and $(n_x - 1, n_y - 1)$ corresponds to the upper-right corner. v_x and v_y are also referred to as the virtual column and row indices respectively.

The base grid is then specified with respect to the virtual grid, with each base grid point coinciding with a virtual grid point. Each base grid point must be given an index, starting from 1, and incrementing row-wise from the leftmost point of the lowest row. Also, each base grid row must be numbered consecutively from the lowest row in the grid, so that row 1 contains grid point 1.

As an example, consider the domain consisting of the two separate squares shown in Figure 1. The left-hand diagram shows the virtual grid and its integer co-ordinates (i.e., its column and row indices), and the right-hand diagram shows the base grid point indices and the base row indices (in brackets).



(8)	29	30	31	32	.	.	.
(7)	25	26	27	28	.	.	.
(6)	21	22	23	24	.	.	.
(5)	17	18	19	20	.	.	.
	.	.	.	.	.	.	.
(4)	.	.	.	13	14	15	16
(3)	.	.	.	9	10	11	12
(2)	.	.	.	5	6	7	8
(1)	.	.	.	1	2	3	4

Figure 1

Hence the base grid point with index 6 say is in base row 2, virtual column 4, and virtual row 1, i.e., virtual grid integer co-ordinates (4,1); and the base grid point with index 19 say is in base row 5, virtual column 2, and virtual row 5, i.e., virtual grid integer co-ordinates (2,5).

The base grid must then be defined in the subroutine INIDOM by specifying the number of base grid rows, the number of base grid points, the number of boundaries, the number of boundary points, and the following integer arrays:

LROW contains the base grid indices of the starting points of the base grid rows.

IROW contains the virtual row numbers v_y of the base grid rows.

ICOL contains the virtual column numbers v_x of the base grid points.

LBND contains the grid indices of the boundary edges (without corners) and corner points.

LLBND contains the starting elements of the boundaries and corners in LBND.

Finally, ILBND contains the types of the boundaries and corners, as follows:

Boundaries:

- **1** – lower boundary
- **2** – left boundary
- **3** – upper boundary
- **4** – right boundary

External corners (90°):

- **12** – lower-left corner
- **23** – upper-left corner
- **34** – upper-right corner
- **41** – lower-right corner

Internal corners (270°):

- **21** – lower-left corner
- **32** – upper-left corner
- **43** – upper-right corner
- **14** – lower-right corner

Figure 2 shows the boundary types of a domain with a hole. Notice the logic behind the labelling of the corners: each one includes the types of the two adjacent boundary edges, in a clockwise fashion (outside the domain).

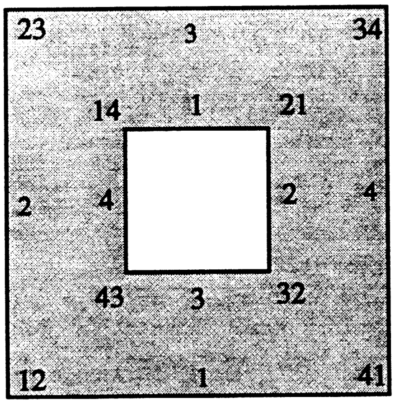


Figure 2

As an example, consider the domain shown in Figure 3. The left-hand diagram shows the physical domain and the right-hand diagram shows the base and virtual grids. The numbers outside the base grid are the indices of the left and rightmost base grid points, and the numbers inside the base grid are the boundary or corner numbers, indicating the order in which the boundaries are stored in LBND.

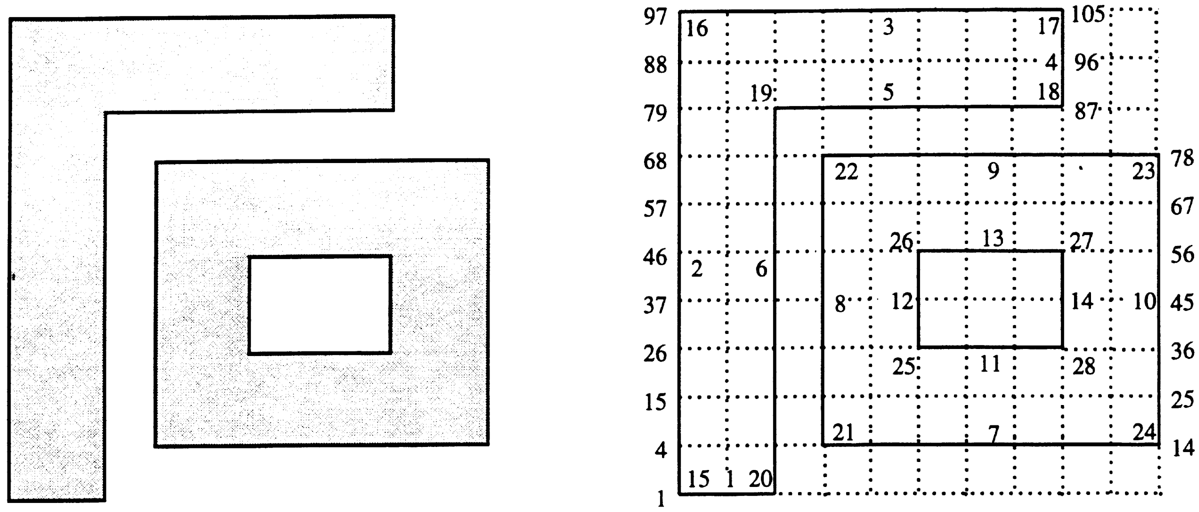


Figure 3

For this example we have

```
NROWS = 11
NPTS = 105
NBNDS = 28
NBPTS = 72

LROW = (1,4,15,26,37,46,57,68,79,88,97)

IROW = (0,1,2,3,4,5,6,7,8,9,10)

ICOL = (0,1,2,
        0,1,2,3,4,5,6,7,8,9,10,
        0,1,2,3,4,5,6,7,8,9,10,
        0,1,2,3,4,5,6,7,8,9,10,
        0,1,2,3,4,5,8,9,10,
        0,1,2,3,4,5,6,7,8,9,10,
        0,1,2,3,4,5,6,7,8,9,10,
        0,1,2,3,4,5,6,7,8,9,10,
        0,1,2,3,4,5,6,7,8,
        0,1,2,3,4,5,6,7,8,
        0,1,2,3,4,5,6,7,8)

LBND = (2,
        4,15,26,37,46,57,68,79,88,
        98,99,100,101,102,103,104,
        96,
        86,85,84,83,82,
        70,59,48,39,28,17,6,
        8,9,10,11,12,13,
        18,29,40,49,60,
        72,73,74,75,76,77,
        67,56,45,36,25,
        33,32,
        42,
        52,53,
        43,
        1,97,105,87,81,3,7,71,78,14,31,51,54,34)
```

```
LLBND = (1,2,11,18,19,24,31,37,42,48,53,55,56,58,59,60,
         61,62,63,64,65,66,67,68,69,70,71,72)

ILBND = (1,2,3,4,1,4,1,2,3,4,3,4,1,2,12,23,34,41,14,41,12,23,34,41,43,14,21,32)
```

This particular domain is used in the example in Section 9, and data statements are used to define the above arrays in that example program. For less complicated domains it is simpler to assign the values of the arrays in do-loops. This also allows flexibility in the number of base grid points.

The routine D03RYF can be called from INIDOM to obtain a simple graphical representation of the base grid, and to verify the data that the user has specified in INIDOM.

Subgrids are stored internally using the same data structure, and solution information is communicated to the user in the subroutines PDEIV, PDEDEF and BNDARY in arrays according to the grid index on the particular level, e.g., X(i) and Y(i) contain the (x, y) co-ordinates of grid point i, and U(i,j) contains the jth solution component u_j at grid point i.

The grid data and the solutions at all grid levels are stored in the workspace arrays, along with other information needed for a restart (i.e., a continuation call). It is not intended that the user extracts the solution from these arrays, indeed the necessary information regarding these arrays is not provided. The user-supplied monitor routine MONITR should be used to obtain the solution at particular levels and times. MONITR is called at the end of every time step, with the last step being identified via the input argument TLAST. The routine D03RZF should be called from MONITR to obtain grid information at a particular level.

Further details of the underlying algorithm can be found in Section 8 and in [1] and [2] and the references therein.

4 References

[1] Blom J G and Verwer J G (1993) VLUGR2: A vectorized local uniform grid refinement code for PDEs in 2D *Report NM-R9306* CWI, Amsterdam

[2] Blom J G, Trompert R A and Verwer J G (1996) Algorithm 758. VLUGR2: A vectorizable adaptive grid solver for PDEs in 2D *Trans. Math. Software* **22** 302–328

[3] Trompert R A and Verwer J G (1993) Analysis of the implicit Euler local uniform grid refinement method *SIAM J. Sci. Comput.* **14** 259–278

[4] Trompert R A (1993) Local uniform grid refinement and systems of coupled partial differential equations *Appl. Numer. Maths* **12** 331–355

5 Parameters

1: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system.

Constraint: NPDE ≥ 1.

2: TS — ***real*** *Input/Output*

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

On exit: the value of t which has been reached. 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: DT(3) — ***real*** array *Input/Output*

On entry: the initial, minimum and maximum time step sizes respectively. DT(1) specifies the initial time step size to be used on the first entry, i.e., when IND = 0. If DT(1) = 0.0 then the default value DT(1) = 0.01 × (TOUT−TS) is used. On subsequent entries (IND = 1), the value of DT(1) is not referenced.

DT(2) specifies the minimum time step size to be attempted by the integrator. If DT(2) = 0.0 the default value DT(2) = 10.0 × ***machine precision*** is used.

DT(3) specifies the maximum time step size to be attempted by the integrator. If DT(3) = 0.0 the default value DT(3) = TOUT − TS is used.

On exit: DT(1) contains the time step size for the next time step. DT(2) and DT(3) are unchanged or set to their default values if zero on entry.

Constraints: if IND = 1 then DT(1) is unconstrained. Otherwise DT(1) ≥ 0 and if DT(1) > 0.0 then it must satisfy the constraints:

$$10.0 \times \textit{machine precision} \times \max(|TS|,|TOUT|) \leq DT(1) \leq TOUT - TS$$
$$DT(2) \leq DT(1) \leq DT(3)$$

where the values of DT(2) and DT(3) will have been reset to their default values if zero on entry.

DT(2) and DT(3) must satisfy DT(i) ≥ 0, i = 2,3 and DT(2) ≤ DT(3) for IND = 0 and IND = 1.

5: TOLS — ***real*** *Input*

On entry: the space tolerance used in the grid refinement strategy (σ in equation (4)). See Section 8.2.

Constraint: TOLS > 0.0.

6: TOLT — ***real*** *Input*

On entry: the time tolerance used to determine the time step size (τ in equation (7)). See Section 8.3.

Constraint: TOLT > 0.0.

7: INIDOM — SUBROUTINE, supplied by the user. *External Procedure*

INIDOM must specify the base grid in terms of the data structure described in Section 3. INIDOM is not referenced if, on entry, IND = 1. D03RYF can be called from INIDOM to obtain a simple graphical representation of the base grid, and to verify the data that the user has specified in INIDOM. D03RBF also checks the validity of the data, but the user is strongly advised to call D03RYF to ensure that the base grid is exactly as required.

Note. The boundaries of the base grid should consist of as many points as are necessary to employ second-order space discretization, i.e.,, a boundary enclosing the internal part of the domain must include at least 3 grid points including the corners. If Neumann boundary conditions are to be applied the minimum is 4.

Its specification is:

```
      SUBROUTINE INIDOM(MAXPTS, XMIN, XMAX, YMIN, YMAX, NX, NY, NPTS,
     1                  NROWS, NBNDS, NBPTS, LROW, IROW, ICOL, LLBND,
     2                  ILBND, LBND, IERR)
      INTEGER           MAXPTS, NX, NY, NPTS, NROWS, NBNDS, NBPTS,
     1                  LROW(*), IROW(*), ICOL(*), LLBND(*), ILBND(*),
     2                  LBND(*), IERR
      real              XMIN, XMAX, YMIN, YMAX
```

1: MAXPTS — INTEGER *Input*

On entry: the maximum number of base grid points allowed by the available workspace.

2: XMIN — ***real*** *Output*

3: XMAX — ***real*** *Output*

On exit: the extents of the virtual grid in the x-direction, i.e., the x co-ordinates of the left and right boundaries respectively.

Constraints: XMIN < XMAX and XMAX must be sufficiently distinguishable from XMIN for the precision of the machine being used.

4: YMIN — ***real*** *Output*

5: YMAX — ***real*** *Output*

On exit: the extents of the virtual grid in the y-direction, i.e., the y co-ordinates of the left and right boundaries respectively.

Constraints: YMIN < YMAX and YMAX must be sufficiently distinguishable from YMIN for the precision of the machine being used.

6: NX — INTEGER *Output*

7: NY — INTEGER *Output*

On exit: the number of virtual grid points in the x- and y-direction respectively (including the boundary points).

Constraints: NX and NY ≥ 4.

8: NPTS — INTEGER *Output*

On exit: the total number of points in the base grid. If the required number of points is greater than MAXPTS then INIDOM must be exited immediately with IERR set to -1 to avoid overwriting memory.

Constraints: NPTS $\leq$ NX $\times$ NY and if IERR $\neq -1$ on exit, NPTS $\leq$ MAXPTS.

9: NROWS — INTEGER *Output*

On exit: the total number of rows of the virtual grid that contain base grid points. This is the maximum base row index.

Constraint: $4 \leq$ NROWS $\leq$ NY.

10: NBNDS — INTEGER *Output*

On exit: the total number of physical boundaries and corners in the base grid.

Constraint: NBNDS ≥ 8.

11: NBPTS — INTEGER *Output*

On exit: the total number of boundary points in the base grid.

Constraint: $12 \leq$ NBPTS < NPTS.

12: LROW(*) — INTEGER array *Output*

On exit: LROW(i) for $i = 1, 2, \ldots,$NROWS must contain the base grid index of the first grid point in base grid row i.

Constraints: $1 \leq$ LROW(i) $\leq$ NPTS for $i = 1, 2, \ldots,$NROWS,
LROW($i-1$) < LROW(i) for $i = 2, 3, \ldots,$NROWS.

13: IROW(*) — INTEGER array *Output*

On exit: IROW(i) for $i = 1, 2, \ldots,$NROWS must contain the virtual row number v_y that corresponds to base grid row i.

Constraints: $0 \leq$ IROW(i) $\leq$ NY for $i = 1, 2, \ldots,$NROWS,
IROW($i-1$) < IROW(i) for $i = 2, 3, \ldots,$NROWS.

14: ICOL(*) — INTEGER array *Output*

On exit: ICOL(i) for $i = 1, 2, \ldots,$NPTS must contain the virtual column number v_x that contains base grid point i.

Constraint: $0 \leq$ ICOL(i) $\leq$ NX for $i = 1, 2, \ldots,$NPTS.

15: LLBND(*) — INTEGER array *Output*

On exit: LLBND(i) for $i = 1, 2, \ldots$,NBNDS must contain the element of LBND corresponding to the start of the ith boundary or corner.

Note. The order of the boundaries and corners in LLBND must be first all the boundaries and then all the corners. The end points of a boundary (i.e., the adjacent corner points) must **not** be included in the list of points on that boundary. Also, if a corner is shared by two pairs of physical boundaries then it has two types and must therefore be treated as two corners.

Constraints: $1 \leq$ LLBND(i) $\leq$ NBPTS for $i = 1, 2, \ldots$,NBNDS,
LLBND($i-1$) $<$ LLBND(i) for $i = 2, 3, \ldots$,NBNDS.

16: ILBND(*) — INTEGER array *Output*

On exit: ILBND(i) for $i = 1, 2, \ldots$,NBNDS must contain the type of the ith boundary (or corner), as given in Section 3.

Constraint: ILBND(i) must be equal to one of the following: 1, 2, 3, 4, 12, 23, 34, 41, 21, 32, 43 or 14, for $i = 1, 2, \ldots$,NBNDS.

17: LBND(*) — INTEGER array *Output*

On exit: LBND(i) for $i = 1, 2, \ldots$,NBPTS must contain the grid index of the ith boundary point. The order of the boundaries is as specified in LLBND, but within this restriction the order of the points in LBND is arbitrary.

Constraint: $1 \leq$ LBND(i) $\leq$ NPTS for $i = 1, 2, \ldots$,NBPTS.

18: IERR — INTEGER *Output*

On exit: if the required number of grid points is larger than MAXPTS, IERR must be set to -1 to force a termination of the integration and an immediate return to the calling program with IFAIL set to 3. Otherwise, IERR should remain unchanged.

INIDOM must be declared as EXTERNAL in the (sub)program from which D03RBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

8: PDEDEF — SUBROUTINE, supplied by the user. *External Procedure*

PDEDEF must evaluate the functions F_j, $j = 1, 2, \ldots$,NPDE, in equation (1) which define the system of PDEs (i.e., the residuals of the resulting ODE system) at all interior points of the domain. Values at points on the boundaries of the domain are ignored and will be overwritten by the subroutine BNDARY. PDEDEF is called for each subgrid in turn.

Its specification is:

```
      SUBROUTINE PDEDEF(NPTS, NPDE, T, X, Y, U, UT, UX, UY, UXX, UXY,
     1                  UYY, RES)
      INTEGER           NPTS, NPDE
      real              T, X(NPTS), Y(NPTS), U(NPTS,NPDE),
     1                  UT(NPTS,NPDE), UX(NPTS,NPDE), UY(NPTS,NPDE),
     2                  UXX(NPTS,NPDE), UXY(NPTS,NPDE), UYY(NPTS,NPDE),
     3                  RES(NPTS,NPDE)
```

1: NPTS — INTEGER *Input*

On entry: the number of grid points in the current grid.

2: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system.

3: T — ***real*** *Input*

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

4: X(NPTS) — ***real*** array *Input*

On entry: X(i) contains the x co-ordinate of the ith grid point, for $i = 1,2,\ldots,$NPTS.

5: Y(NPTS) — ***real*** array *Input*

On entry: Y(i) contains the y co-ordinate of the ith grid point, for $i = 1,2,\ldots,$NPTS.

6: U(NPTS,NPDE) — ***real*** array *Input*

On entry: U(i,j) contains the value of the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

7: UT(NPTS,NPDE) — ***real*** array *Input*

On entry: UT(i,j) contains the value of $\partial u/\partial t$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

8: UX(NPTS,NPDE) — ***real*** array *Input*

On entry: UX(i,j) contains the value of $\partial u/\partial x$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

9: UY(NPTS,NPDE) — ***real*** array *Input*

On entry: UY(i,j) contains the value of $\partial u/\partial y$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

10: UXX(NPTS,NPDE) — ***real*** array *Input*

On entry: UXX(i,j) contains the value of $\partial^2 u/\partial x^2$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

11: UXY(NPTS,NPDE) — ***real*** array *Input*

On entry: UXY(i,j) contains the value of $\partial^2 u/\partial x \partial y$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

12: UYY(NPTS,NPDE) — ***real*** array *Input*

On entry: UYY(i,j) contains the value of $\partial^2 u/\partial y^2$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

13: RES(NPTS,NPDE) — ***real*** array *Output*

On exit: RES(i,j) must contain the value of F_j for $j = 1,2,\ldots,$NPDE, at the ith grid point for $i = 1,2,\ldots,$NPTS, although the residuals at boundary points will be ignored (and overwritten later on) and so they need not be specified here.

PDEDEF must be declared as EXTERNAL in the (sub)program from which D03RBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

9: BNDARY — SUBROUTINE, supplied by the user. *External Procedure*

BNDARY must evaluate the functions G_j, $j = 1,2,\ldots,$NPDE, in equation (2) which define the boundary conditions at all boundary points of the domain. Residuals at interior points must **not** be altered by this subroutine.

Its specification is:

```
      SUBROUTINE BNDARY(NPTS, NPDE, T, X, Y, U, UT, UX, UY, NBNDS,
     1                  NBPTS, LLBND, ILBND, LBND, RES)
      INTEGER           NPTS, NPDE, NBNDS, NBPTS, LLBND(NBNDS),
     1                  ILBND(NBNDS), LBND(NBPTS)
      real              T, X(NPTS), Y(NPTS), U(NPTS,NPDE),
     1                  UT(NPTS,NPDE), UX(NPTS,NPDE), UY(NPTS,NPDE),
     2                  RES(NPTS,NPDE)
```

1: NPTS — INTEGER *Input*

On entry: the number of grid points in the current grid.

2: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system.

3: T — ***real*** *Input*

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

4: X(NPTS) — ***real*** array *Input*

On entry: X(i) contains the x co-ordinate of the ith grid point, for $i = 1,2,\ldots,$NPTS.

5: Y(NPTS) — ***real*** array *Input*

On entry: Y(i) contains the y co-ordinate of the ith grid point, for $i = 1,2,\ldots,$NPTS.

6: U(NPTS,NPDE) — ***real*** array *Input*

On entry: U(i,j) contains the value of the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

7: UT(NPTS,NPDE) — ***real*** array *Input*

On entry: UT(i,j) contains the value of $\partial u/\partial t$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

8: UX(NPTS,NPDE) — ***real*** array *Input*

On entry: UX(i,j) contains the value of $\partial u/\partial x$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

9: UY(NPTS,NPDE) — ***real*** array *Input*

On entry: UY(i,j) contains the value of $\partial u/\partial y$ for the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

10: NBNDS — INTEGER *Input*

On entry: the total number of physical boundaries and corners in the grid.

11: NBPTS — INTEGER *Input*

On entry: the total number of boundary points in the grid.

12: LLBND(NBNDS) — INTEGER array *Input*

On entry: LLBND(i) for $i = 1,2,\ldots,$NBNDS contains the element of LBND corresponding to the start of the ith boundary (or corner).

13: ILBND(NBNDS) — INTEGER array *Input*

On entry: ILBND(i) for $i = 1,2,\ldots,$NBNDS contains the type of the ith boundary, as given in Section 3.

14: LBND(NBPTS) — INTEGER array *Input*

On entry: LBND(i) for $i = 1,2,\ldots,$NBPTS contains the grid index of the ith boundary point, where the order of the boundaries is as specified in LLBND. Hence the ith boundary point has co-ordinates X(LBND(i)) and Y(LBND(i)), and the corresponding solution values are U(LBND(i),j), $j = 1,2,\ldots,$NPDE.

15: RES(NPTS,NPDE) — ***real*** array *Output*

On exit: RES(LBND(i),j) must contain the value of G_j for $j = 1,2,\ldots,$NPDE, at the ith boundary point for $i = 1,2,\ldots,$NBPTS.

Note. Elements of RES corresponding to interior points, i.e., points not included in LBND, must **not** be altered.

BNDARY must be declared as EXTERNAL in the (sub)program from which D03RBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

10: PDEIV — SUBROUTINE, supplied by the user. *External Procedure*

PDEIV must specify the initial values of the PDE components u at all points in the base grid. PDEIV is not referenced if, on entry, IND = 1.

Its specification is:

```
      SUBROUTINE PDEIV(NPTS, NPDE, T, X, Y, U)
      INTEGER          NPTS, NPDE
      real             T, X(NPTS), Y(NPTS), U(NPTS,NPDE)
```

1: NPTS — INTEGER *Input*

On entry: the number of grid points in the base grid.

2: NPDE — INTEGER *Input*

On entry: the number of PDEs in the system.

3: T — ***real*** *Input*

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

4: X(NPTS) — ***real*** array *Input*

On entry: X(i) contains the x co-ordinate of the ith grid point, for $i = 1,2,\ldots,$NPTS.

5: Y(NPTS) — ***real*** array *Input*

On entry: Y(i) contains the y co-ordinate of the ith grid point, for $i = 1,2,\ldots,$NPTS.

6: U(NPTS,NPDE) — ***real*** array *Output*

On exit: U(i,j) must contain the value of the jth PDE component at the ith grid point, for $i = 1,2,\ldots,$NPTS, $j = 1,2,\ldots,$NPDE.

PDEIV must be declared as EXTERNAL in the (sub)program from which D03RBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

11: MONITR — SUBROUTINE, supplied by the user. *External Procedure*

MONITR is called by D03RBF at the end of every successful time step, and may be used to examine or print the solution or perform other tasks such as error calculations, particularly at the final time step, indicated by the parameter TLAST.

The input arguments contain information about the grid and solution at all grid levels used. D03RZF should be called from MONITR in order to extract the number of points and their (x, y) co-ordinates on a particular grid.

MONITR can also be used to force an immediate tidy termination of the solution process and return to the calling program.

Its specification is:

```
      SUBROUTINE MONITR(NPDE, T, DT, DTNEW, TLAST, NLEV, XMIN, YMIN,
     1                  DXB, DYB, LGRID, ISTRUC, LSOL, SOL, IERR)
      INTEGER           NPDE, NLEV, LGRID(*), ISTRUC(*), LSOL(NLEV), IERR
      real              T, DT, DTNEW, XMIN, YMIN, DXB, DYB, SOL(*)
      LOGICAL           TLAST
```

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, i.e., the time at the end of the integration step just completed.

3: DT — ***real*** *Input*

On entry: the current time step size DT, i.e., the time step size used for the integration step just completed.

4: DTNEW — ***real*** *Input*

On entry: the time step size that will be used for the next time step.

5: TLAST — LOGICAL *Input*

On entry: indicates if intermediate or final time step. TLAST = .FALSE. for an intermediate step, TLAST = .TRUE. for the last call to MONITR before returning to the user's program.

6: NLEV — INTEGER *Input*

On entry: the number of grid levels used at time T.

7: XMIN — ***real*** *Input*
8: YMIN — ***real*** *Input*

On entry: the (x, y) co-ordinates of the lower-left corner of the virtual grid.

9: DXB — ***real*** *Input*
10: DYB — ***real*** *Input*

On entry: the sizes of the base grid spacing in the x- and y-direction respectively.

11: LGRID(*) — INTEGER array *Input*

On entry: LGRID contains pointers to the start of the grid structures in ISTRUC, and must be passed unchanged to D03RZF in order to extract the grid information.

12: ISTRUC(*) — INTEGER array *Input*

On entry: ISTRUC contains the grid structures for each grid level and must be passed unchanged to D03RZF in order to extract the grid information.

13: LSOL(NLEV) — INTEGER array *Input*

On entry: LSOL(l) contains the pointer to the solution in SOL at grid level l and time T. (LSOL(l) actually contains the array index immediately preceding the start of the solution in SOL. See below.)

14: SOL(*) — ***real*** array *Input*

On entry: SOL contains the solution u at time T for each grid level l in turn, positioned according to LSOL. More precisely

$$\mathrm{U}(i,j) = \mathrm{SOL}(\mathrm{LSOL}(l) + (j-1) \times n_l + i)$$

represents the jth component of the solution at the ith grid point in the lth level, for $i = 1,\ldots,n_l$, $j = 1,\ldots,\mathrm{NPDE}$, $l = 1,\ldots,\mathrm{NLEV}$, where n_l is the number of grid points at level l (obtainable by a call to D03RZF).

15: IERR — INTEGER *Output*

On exit: IERR should be set to 1 to force a termination of the integration and an immediate return to the calling program with IFAIL set to 4. IERR should remain unchanged otherwise.

MONITR must be declared as EXTERNAL in the (sub)program from which D03RBF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

12: OPTI(4) — INTEGER array *Input*

On entry: OPTI may be set to control various options available in the integrator. If OPTI(1) = 0 then **all** the default options are employed.

If OPTI(1) > 0 then the default value of OPTI(i) for $i = 2, 3, 4$, can be obtained by setting OPTI(i) = 0.

OPTI(1) specifies the maximum number of grid levels allowed (including the base grid). OPTI(1) $\geq$ 0. The default value is OPTI(1) = 3.

OPTI(2) specifies the maximum number of Jacobian evaluations allowed during each nonlinear equations solution. OPTI(2) $\geq$ 0. The default value is OPTI(2) = 2.

OPTI(3) specifies the maximum number of Newton iterations in each nonlinear equations solution. OPTI(3) $\geq$ 0. The default value is OPTI(3) = 10.

OPTI(4) specifies the maximum number of iterations in each linear equations solution. OPTI(4) $\geq$ 0. The default value is OPTI(4) = 100.

Constraint: OPTI(1) $\geq$ 0 and if OPTI(1) > 0 then OPTI(i) $\geq$ 0 for i = 2,3,4.

13: OPTR(3,NPDE) — ***real*** array *Input*

On entry: OPTR may be used to specify the optional vectors u^{max}, w^s and w^t in the space and time monitors (see Section 8).

If an optional vector is not required then all its components should be set to 1.0.

OPTR(1,j), for $j = 1, 2, \ldots$,NPDE, specifies u_j^{max}, the approximate maximum absolute value of the jth component of u, as used in (4) and (7). OPTR(1,j) > 0.0 for $j = 1, 2, \ldots$,NPDE.

OPTR(2,j), for $j = 1, 2, \ldots$,NPDE, specifies w_j^s, the weighting factors used in the space monitor (see (4)) to indicate the relative importance of the jth component of u on the space monitor. OPTR(2,j) $\geq$ 0.0 for $j = 1, 2, \ldots$,NPDE.

OPTR(3,j), for $j = 1, 2, \ldots$,NPDE, specifies w_j^t, the weighting factors used in the time monitor (see (6)) to indicate the relative importance of the jth component of u on the time monitor. OPTR(3,j) $\geq$ 0.0 for $j = 1, 2, \ldots$,NPDE.

Constraint: OPTR(1,j) > 0.0 for $j = 1, 2, \ldots$,NPDE and OPTR(i, j) $\geq$ 0.0 for $i = 2, 3$ and $j = 1, 2, \ldots$,NPDE.

14: RWK(LENRWK) — ***real*** array *Workspace*

15: LENRWK — INTEGER *Input*

On entry: the dimension of the array RWK as declared in the (sub)program from which D03RBF is called.

The required value of LENRWK can not be determined exactly in advance, but a suggested value is

LENRWK = MAXPTS × NPDE × (5×l+18×NPDE+9) + 2 × MAXPTS,

where l = OPTI(1) if OPTI(1) $\neq$ 0 and l = 3 otherwise, and MAXPTS is the expected maximum number of grid points at any one level. If during the execution the supplied value is found to be too small then the routine returns with IFAIL = 3 and an estimated required size is printed on the current error message unit (see X04AAF).

Note. The size of LENRWK can not be checked upon initial entry to D03RBF since the number of grid points on the base grid is not known.

16: IWK(LENIWK) — INTEGER array *Input/Output*

On entry: if IND = 0, IWK need not be set. Otherwise IWK must remain unchanged from a previous call to D03RBF.

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

IWK(1) contains the number of steps taken in time;

IWK(2) contains the number of rejected time steps;

IWK(2+l) contains the total number of residual evaluations performed (i.e., the number of times PDEDEF was called) at grid level l;

IWK(2+$m+l$) contains the total number of Jacobian evaluations performed at grid level l;

IWK(2+2$\times m+l$) contains the total number of Newton iterations performed at grid level l;

IWK(2+3$\times m+l$) contains the total number of linear solver iterations performed at grid level l;

IWK(2+4$\times m+l$) contains the maximum number of Newton iterations performed at any one time step at grid level l;

IWK(2+5$\times m+l$) contains the maximum number of linear solver iterations performed at any one time step at grid level l;

for $l = 1, 2, \ldots, nl$, where nl is the number of levels used and m = OPTI(1) if OPTI(1) > 0 and m = 3 otherwise.

Note. The total and maximum numbers are cumulative over all calls to D03RBF. If the specified maximum number of Newton or linear solver iterations is exceeded at any stage, then the maximums above are set to the specified maximum plus one.

17: LENIWK — INTEGER *Input*

On entry: the dimension of the array IWK as declared in the (sub)program from which D03RBF is called.

The required value of LENIWK can not be determined exactly in advance, but a suggested value is

LENIWK = MAXPTS $\times$ (14+5$\times m$) + 7 $\times$ m + 2,

where MAXPTS is the expected maximum number of grid points at any one level and m =OPTI(1) if OPTI(1) > 0 and m = 3 otherwise. If during the execution the supplied value is found to be too small then the routine returns with IFAIL = 3 and an estimated required size is printed on the current error message unit (see X04AAF).

Note. The size of LENIWK can not be checked upon initial entry to D03RBF since the number of grid points on the base grid is not known.

18: LWK(LENLWK) — LOGICAL array *Workspace*

19: LENLWK — INTEGER *Input*

On entry: the dimension of the array LWK as declared in the (sub)program from which D03RBF is called.

The required value of LENLWK can not be determined exactly in advance, but a suggested value is

LENLWK = MAXPTS + 1,

where MAXPTS is the expected maximum number of grid points at any one level. If during the execution the supplied value is found to be too small then the routine returns with IFAIL = 3 and an estimated required size is printed on the current error message unit (see X04AAF).

Note. The size of LENLWK can not be checked upon initial entry to D03RBF since the number of grid points on the base grid is not known.

20: ITRACE — INTEGER *Input*

On entry: the level of trace information required from D03RBF. 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 are printed, and if ITRACE > 0, then output from the underlying solver is printed on the current advisory message unit (see X04ABF). This output contains details of the time integration, the nonlinear iteration and the linear solver. The advisory messages are given in greater detail as ITRACE increases. Setting ITRACE = 1 allows the user to monitor the progress of the integration without possibly excessive information.

21: IND — INTEGER *Input/Output*

On entry: IND must be set to 0 or 1.

IND = 0

starts the integration in time.

IND = 1

continues the integration after an earlier exit from the routine. In this case, only the following parameters may be reset between calls to D03RBF: TOUT, DT(2), DT(3), TOLS, TOLT, OPTI, OPTR, ITRACE and IFAIL.

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

On exit: IND = 1.

22: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NPDE < 1,
or TOUT ≤ TS,
or TOUT is too close to TS,
or IND = 0 and DT(1) < 0.0,
or DT(i) < 0.0 for i = 2 or 3,
or DT(2) > DT(3),
or IND = 0 and 0.0 < DT(1) < 10 × ***machine precision*** × max(|TS|,|TOUT|),
or IND = 0 and DT(1) > TOUT − TS,
or IND = 0 and DT(1) < DT(2) or DT(1) > DT(3),
or TOLS or TOLT ≤ 0.0,
or OPTI(1) < 0,
or OPTI(1) > 0 and OPTI(j) < 0 for j = 2, 3 or 4,
or OPTR(1,j) ≤ 0.0 for some $j = 1,2,\ldots$,NPDE,
or OPTR(2,j) < 0.0 for some $j = 1,2,\ldots$,NPDE,
or OPTR(3,j) < 0.0 for some $j = 1,2,\ldots$,NPDE,
or IND ≠ 0 or 1,

or IND = 1 on initial entry to D03RBF.

IFAIL = 2

The time step size to be attempted is less than the specified minimum size. This may occur following time step failures and subsequent step size reductions caused by one or more of the following:

the requested accuracy could not be achieved, i.e., TOLT is too small,

the maximum number of linear solver iterations, Newton iterations or Jacobian evaluations is too small,

ILU decomposition of the Jacobian matrix could not be performed, possibly due to singularity of the Jacobian.

Setting ITRACE to a higher value may provide further information.

In the latter two cases the user is advised to check their problem formulation in PDEDEF and/or BNDARY, and the initial values in PDEIV if appropriate.

IFAIL = 3

One or more of the workspace arrays is too small for the required number of grid points. At the initial time step this error may result from either the user setting IERR to −1 in INIDOM, or the internal check on the number of grid points following the call to INIDOM. An estimate of the required sizes for the current stage is output, but more space may be required at a later stage.

IFAIL = 4

IERR was set to 1 in the user-supplied subroutine MONITR, forcing control to be passed back to calling program. Integration was successful as far as T = TS.

IFAIL = 5

The integration has been completed but the maximum number of levels specified in OPTI(1) was insufficient at one or more time steps, meaning that the requested space accuracy could not be achieved. To avoid this warning either increase the value of OPTI(1) or decrease the value of TOLS.

IFAIL = 6

One or more of the output arguments of the user-suppled subroutine INIDOM was incorrectly specified, i.e.,

XMIN $\geq$ XMAX,
or XMAX too close to XMIN,
or YMIN $\geq$ YMAX,
or YMAX too close to YMIN,
or NX or NY $<$ 4,
or NROWS $<$ 4,
or NROWS $>$ NY,
or NPTS $>$ NX $\times$ NY,
or NBNDS $<$ 8,
or NBPTS $<$ 12,
or NBPTS $\geq$ NPTS,
or LROW(i) $<$ 1 or LROW(i) $>$ NPTS for some $i = 1, 2, \ldots$,NROWS,
or LROW(i) $\leq$ LROW($i-1$) for some $i = 2, 3, \ldots$,NROWS,
or IROW(i) $<$ 0 or IROW(i) $>$ NY for some $i = 1, 2, \ldots$,NROWS,
or IROW(i) $\leq$ IROW($i-1$) for some $i = 2, 3, \ldots$,NROWS,
or ICOL(i) $<$ 0 or ICOL(i) $>$ NX for some $i = 1, 2, \ldots$,NPTS,
or LLBND(i) $<$ 1 or LLBND(i) $>$ NBPTS for some $i = 1, 2, \ldots$,NBNDS,
or LLBND(i) $\leq$ LLBND($i-1$) for some $i = 2, 3, \ldots$,NBNDS,
or ILBND(i) $\neq$ 1, 2, 3, 4, 12, 23, 34, 41, 21, 32, 43 or 14, for some $i = 1, 2, \ldots$,NBNDS,
or LBND(i) $<$ 1 or LBND(i) $>$ NPTS for some $i = 1, 2, \ldots$,NBPTS.

7 Accuracy

There are three sources of error in the algorithm: space and time discretisation, and interpolation (linear) between grid levels. The space and time discretisation errors are controlled separately using the parameters TOLS and TOLT described in the following section, and the user should test the effects of varying these parameters. Interpolation errors are generally implicitly controlled by the refinement criterion since in areas where interpolation errors are potentially large, the space monitor will also be large. It can be shown that the global spatial accuracy is comparable to that which would be obtained on a uniform grid of the finest grid size. A full error analysis can be found in [3].

8 Further Comments

8.1 Algorithm Outline

The local uniform grid refinement method is summarised as follows

(1) Initialise the course base grid, an initial solution and an initial time step,
(2) Solve the system of PDEs on the current grid with the current time step,
(3) If the required accuracy in space and the maximum number of grid levels have not yet been reached:
   (a) Determine new finer grid at forward time level,
   (b) Get solution values at previous time level(s) on new grid,
   (c) Interpolate internal boundary values from old grid at forward time,
   (d) Get initial values for the Newton process at forward time,
   (e) Goto 2,
(4) Update the coarser grid solution using the finer grid values,
(5) Estimate error in time integration. If time error is acceptable advance time level,
(6) Determine new step size then goto 2 with coarse base as current grid.

8.2 Refinement Strategy

For each grid point i a space monitor μ_i^s is determined by

$$\mu_i^s = \max_{j=1,\mathrm{NPDE}}\{\gamma_j(\mid \Delta x^2 \frac{\partial^2}{\partial x^2} u_j(x_i, y_i, t) \mid + \mid \Delta y^2 \frac{\partial^2}{\partial y^2} u_j(x_i, y_i, t) \mid)\}, \quad (3)$$

where Δx and Δy are the grid widths in the x and y directions; and x_i, y_i are the (x, y) co-ordinates at grid point i. The parameter γ_j is obtained from

$$\gamma_j = \frac{w_j^s}{u_j^{max}\ \sigma}, \quad (4)$$

where σ is the user-supplied space tolerance; w_j^s is a weighting factor for the relative importance of the jth PDE component on the space monitor; and u_j^{max} is the approximate maximum absolute value of the jth component. A value for σ must be supplied by the user. Values for w_j^s and u_j^{max} must also be supplied but may be set to the values 1.0 if little information about the solution is known.

A new level of refinement is created if

$$\max_i\{\mu_i^s\} > 0.9 \text{ or } 1.0, \quad (5)$$

depending on the grid level at the previous step in order to avoid fluctuations in the number of grid levels between time steps. If (5) is satisfied then all grid points for which $\mu_i^s > 0.25$ are flagged and surrounding cells are quartered in size.

No derefinement takes place as such, since at each time step the solution on the base grid is computed first and new finer grids are then created based on the new solution. Hence derefinement occurs implicitly. See Section 8.1.

8.3 Time Integration

The time integration is controlled using a time monitor calculated at each level l up to the maximum level used, given by

$$\mu_l^t = \sqrt{\frac{1}{\mathrm{N}}\sum_{j=1}^{\mathrm{NPDE}} w_j^t \sum_{i=1}^{\mathrm{NGPTS}(l)} \left(\frac{\Delta t}{\alpha_{ij}} u_t(x_i, y_i, t)\right)^2} \tag{6}$$

where NGPTS(l) is the total number of points on grid level l; N = NGPTS(l) × NPDE; Δt is the current time step; u_t is the time derivative of u which is approximated by first-order finite differences; w_j^t is the time equivalent of the space weighting factor w_j^s; and α_{ij} is given by

$$\alpha_{ij} = \tau\left(\frac{u_j^{max}}{100} + \mid u(x_i, y_i, t) \mid\right) \tag{7}$$

where u_j^{max} is as before, and τ is the user-specified time tolerance.

An integration step is rejected and retried at all levels if

$$\max_l \{\mu_l^t\} > 1.0. \tag{8}$$

9 Example

This example is taken from [1] and is the two dimensional Burgers' system

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - v\frac{\partial u}{\partial y} + \epsilon\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$

$$\frac{\partial v}{\partial t} = -u\frac{\partial v}{\partial x} - v\frac{\partial v}{\partial y} + \epsilon\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right),$$

with $\epsilon = 10^{-3}$ on the domain given in Figure 3. Dirichlet boundary conditions are used on all boundaries using the exact solution

$$u = \frac{3}{4} - \frac{1}{4(1 + \exp\left((-4x + 4y - t)/(32\epsilon)\right))},$$

$$v = \frac{3}{4} + \frac{1}{4(1 + \exp\left((-4x + 4y - t)/(32\epsilon)\right))}.$$

The solution contains a wave front at $y = x + 0.25t$ which propagates in a direction perpendicular to the front with speed $\sqrt{2}/8$.

9.1 Program Text

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.

```
*     D03RBF Example Program Text
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          MXLEV, NPDE, NPTS
      PARAMETER        (MXLEV=5,NPDE=2,NPTS=3000)
      INTEGER          LENIWK, LENRWK, LENLWK
      PARAMETER        (LENIWK=NPTS*(5*MXLEV+14)+2+7*MXLEV,
     +                 LENRWK=NPTS*NPDE*(5*MXLEV+9+18*NPDE)+2*NPTS,
     +                 LENLWK=2*NPTS)
*     .. Scalars in Common ..
      INTEGER          IOUT
*     .. Arrays in Common ..
      real             TWANT(2)
*     .. Local Scalars ..
```

```
      real             TOLS, TOLT, TOUT, TS
      INTEGER          I, IFAIL, IND, ITRACE, J, MAXLEV
*     .. Local Arrays ..
      real             DT(3), OPTR(3,NPDE), RWK(LENRWK)
      INTEGER          IWK(LENIWK), OPTI(4)
      LOGICAL          LWK(LENLWK)
*     .. External Subroutines ..
      EXTERNAL         BNDRY, D03RBF, INIDM, MONIT, PDEF, PDEIV
*     .. Common blocks ..
      COMMON           /OTIME/TWANT, IOUT
*     .. Save statement ..
      SAVE             /OTIME/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03RBF Example Program Results'
*
      IND = 0
      ITRACE = 0
      TS = 0.0e0
      TWANT(1) = 0.25e0
      TWANT(2) = 1.0e0
      DT(1) = 0.001e0
      DT(2) = 1.0e-7
      DT(3) = 0.0e0
      TOLS = 0.1e0
      TOLT = 0.05e0
      OPTI(1) = 5
      MAXLEV = OPTI(1)
      DO 20 I = 2, 4
         OPTI(I) = 0
   20 CONTINUE
      DO 60 J = 1, NPDE
         DO 40 I = 1, 3
            OPTR(I,J) = 1.0e0
   40    CONTINUE
   60 CONTINUE
*
*     Call main routine
*
      DO 120 IOUT = 1, 2
         IFAIL = -1
         TOUT = TWANT(IOUT)
         CALL D03RBF(NPDE,TS,TOUT,DT,TOLS,TOLT,INIDM,PDEF,BNDRY,PDEIV,
     +               MONIT,OPTI,OPTR,RWK,LENRWK,IWK,LENIWK,LWK,LENLWK,
     +               ITRACE,IND,IFAIL)
*
*        Print statistics
*
         WRITE (NOUT,'('' Statistics:'')')
         WRITE (NOUT,'('' Time = '',F8.4)') TS
         WRITE (NOUT,'('' Total number of accepted timesteps ='', I5)')
     +     IWK(1)
         WRITE (NOUT,'('' Total number of rejected timesteps ='', I5)')
     +     IWK(2)
         WRITE (NOUT,*)
         WRITE (NOUT,
     +     '(''           T o t a l   n u m b e r   o f     '')')
         WRITE (NOUT,
     + '(''          Residual  Jacobian    Newton ''   , ''  Lin sys'')'
```

```
     +      )
        WRITE (NOUT,
     + '(''            evals     evals     iters ''   , ''    iters'')'
     +      )
        WRITE (NOUT,'('' At level '')')
        MAXLEV = OPTI(1)
        DO 80 J = 1, MAXLEV
           IF (IWK(J+2).NE.0) WRITE (NOUT,'(I6,4I10)') J, IWK(J+2),
     +         IWK(J+2+MAXLEV), IWK(J+2+2*MAXLEV), IWK(J+2+3*MAXLEV)
*
   80   CONTINUE
        WRITE (NOUT,*)
        WRITE (NOUT,
     +     '(''            M a x i m u m  n u m b e r '', '' o f'')')
        WRITE (NOUT,
     +     '(''             Newton iters    Lin sys iters '')')
        WRITE (NOUT,'('' At level '')')
        DO 100 J = 1, MAXLEV
           IF (IWK(J+2).NE.0) WRITE (NOUT,'(I6,2I14)') J,
     +         IWK(J+2+4*MAXLEV), IWK(J+2+5*MAXLEV)
  100   CONTINUE
        WRITE (NOUT,*)
*
  120 CONTINUE
*
      STOP
      END
*
      SUBROUTINE INIDM(MAXPTS,XMIN,XMAX,YMIN,YMAX,NX,NY,NPTS,NROWS,
     +                 NBNDS,NBPTS,LROW,IROW,ICOL,LLBND,ILBND,LBND,IERR)
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalar Arguments ..
      real             XMAX, XMIN, YMAX, YMIN
      INTEGER          IERR, MAXPTS, NBNDS, NBPTS, NPTS, NROWS, NX, NY
*     .. Array Arguments ..
      INTEGER          ICOL(*), ILBND(*), IROW(*), LBND(*), LLBND(*),
     +                 LROW(*)
*     .. Local Scalars ..
      INTEGER          I, IFAIL, J, LENIWK
*     .. Local Arrays ..
      INTEGER          ICOLD(105), ILBNDD(28), IROWD(11), IWK(122),
     +                 LBNDD(72), LLBNDD(28), LROWD(11)
      CHARACTER*33     PGRID(11)
*     .. External Subroutines ..
      EXTERNAL         D03RYF
*     .. Data statements ..
      DATA             ICOLD/0, 1, 2, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
     +                 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 1, 2, 3, 4,
     +                 5, 6, 7, 8, 9, 10, 0, 1, 2, 3, 4, 5, 8, 9, 10, 0,
     +                 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 1, 2, 3, 4, 5,
     +                 6, 7, 8, 9, 10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
     +                 0, 1, 2, 3, 4, 5, 6, 7, 8, 0, 1, 2, 3, 4, 5, 6,
     +                 7, 8, 0, 1, 2, 3, 4, 5, 6, 7, 8/
      DATA             ILBNDD/1, 2, 3, 4, 1, 4, 1, 2, 3, 4, 3, 4, 1, 2,
     +                 12, 23, 34, 41, 14, 41, 12, 23, 34, 41, 43, 14,
     +                 21, 32/
```

```
      DATA              IROWD/0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10/
      DATA              LBNDD/2, 4, 15, 26, 37, 46, 57, 68, 79, 88, 98,
     +                  99, 100, 101, 102, 103, 104, 96, 86, 85, 84, 83,
     +                  82, 70, 59, 48, 39, 28, 17, 6, 8, 9, 10, 11, 12,
     +                  13, 18, 29, 40, 49, 60, 72, 73, 74, 75, 76, 77,
     +                  67, 56, 45, 36, 25, 33, 32, 42, 52, 53, 43, 1,
     +                  97, 105, 87, 81, 3, 7, 71, 78, 14, 31, 51, 54,
     +                  34/
      DATA              LLBNDD/1, 2, 11, 18, 19, 24, 31, 37, 42, 48, 53,
     +                  55, 56, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
     +                  68, 69, 70, 71, 72/
      DATA              LROWD/1, 4, 15, 26, 37, 46, 57, 68, 79, 88, 97/
*     .. Executable Statements ..
      NX = 11
      NY = 11
*
*     Check MAXPTS against rough estimate of NPTS
*
      NPTS = NX*NY
      IF (MAXPTS.LT.NPTS) THEN
         IERR = -1
         RETURN
      END IF
*
      XMIN = 0.0e0
      YMIN = 0.0e0
      XMAX = 1.0e0
      YMAX = 1.0e0
*
      NROWS = 11
      NPTS = 105
      NBNDS = 28
      NBPTS = 72
*
      DO 20 I = 1, NROWS
         LROW(I) = LROWD(I)
         IROW(I) = IROWD(I)
   20 CONTINUE
*
      DO 40 I = 1, NBNDS
         LLBND(I) = LLBNDD(I)
         ILBND(I) = ILBNDD(I)
   40 CONTINUE
*
      DO 60 I = 1, NBPTS
         LBND(I) = LBNDD(I)
   60 CONTINUE
*
      DO 80 I = 1, NPTS
         ICOL(I) = ICOLD(I)
   80 CONTINUE
*
      WRITE (NOUT,*) 'Base grid:'
      WRITE (NOUT,*)
      LENIWK = 122
      IFAIL = -1
*
      CALL D03RYF(NX,NY,NPTS,NROWS,NBNDS,NBPTS,LROW,IROW,ICOL,LLBND,
```

```
     +           ILBND,LBND,IWK,LENIWK,PGRID,IFAIL)
*
      IF (IFAIL.EQ.0) THEN
         WRITE (NOUT,*) ' '
         DO 100 J = 1, NY
            WRITE (NOUT,*) PGRID(J)
            WRITE (NOUT,*) ' '
  100    CONTINUE
         WRITE (NOUT,*) ' '
      END IF
*
      RETURN
      END
*
      SUBROUTINE PDEIV(NPTS,NPDE,T,X,Y,U)
*     .. Parameters ..
      real              EPS
      PARAMETER         (EPS=1e-3)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              U(NPTS,NPDE), X(NPTS), Y(NPTS)
*     .. Local Scalars ..
      real              A
      INTEGER           I
*     .. Intrinsic Functions ..
      INTRINSIC         EXP
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
         A = (-4.0e0*X(I)+4.0e0*Y(I)-T)/(32.0e0*EPS)
         IF (A.LE.0.0e0) THEN
            U(I,1) = 0.75e0 - 0.25e0/(1.0e0+EXP(A))
            U(I,2) = 0.75e0 + 0.25e0/(1.0e0+EXP(A))
         ELSE
            U(I,1) = 0.75e0 - 0.25e0*EXP(-A)/(EXP(-A)+1.0e0)
            U(I,2) = 0.75e0 + 0.25e0*EXP(-A)/(EXP(-A)+1.0e0)
         END IF
   20 CONTINUE
*
      RETURN
      END
*
      SUBROUTINE PDEF(NPTS,NPDE,T,X,Y,U,UT,UX,UY,UXX,UXY,UYY,RES)
*     .. Parameters ..
      real              EPS
      PARAMETER         (EPS=1e-3)
*     .. Scalar Arguments ..
      real              T
      INTEGER           NPDE, NPTS
*     .. Array Arguments ..
      real              RES(NPTS,NPDE), U(NPTS,NPDE), UT(NPTS,NPDE),
     +                  UX(NPTS,NPDE), UXX(NPTS,NPDE), UXY(NPTS,NPDE),
     +                  UY(NPTS,NPDE), UYY(NPTS,NPDE), X(NPTS), Y(NPTS)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      DO 20 I = 1, NPTS
```

```
         RES(I,1) = UT(I,1) - (-U(I,1)*UX(I,1)-U(I,2)*UY(I,1)
     +              +EPS*(UXX(I,1)+UYY(I,1)))
         RES(I,2) = UT(I,2) - (-U(I,1)*UX(I,2)-U(I,2)*UY(I,2)
     +              +EPS*(UXX(I,2)+UYY(I,2)))
   20 CONTINUE

      RETURN
      END
      SUBROUTINE BNDRY(NPTS,NPDE,T,X,Y,U,UT,UX,UY,NBNDS,NBPTS,LLBND,
     +                 ILBND,LBND,RES)
*     .. Parameters ..
      real             EPS
      PARAMETER        (EPS=1e-3)
*     .. Scalar Arguments ..
      real             T
      INTEGER          NBNDS, NBPTS, NPDE, NPTS
*     .. Array Arguments ..
      real             RES(NPTS,NPDE), U(NPTS,NPDE), UT(NPTS,NPDE),
     +                 UX(NPTS,NPDE), UY(NPTS,NPDE), X(NPTS), Y(NPTS)
      INTEGER          ILBND(NBNDS), LBND(NBPTS), LLBND(NBNDS)
*     .. Local Scalars ..
      real             A
      INTEGER          I, K
*     .. Intrinsic Functions ..
      INTRINSIC        EXP
*     .. Executable Statements ..
      DO 20 K = LLBND(1), NBPTS
         I = LBND(K)
         A = (-4.0e0*X(I)+4.0e0*Y(I)-T)/(32.0e0*EPS)
         IF (A.LE.0.0e0) THEN
            RES(I,1) = U(I,1) - (0.75e0-0.25e0/(1.0e0+EXP(A)))
            RES(I,2) = U(I,2) - (0.75e0+0.25e0/(1.0e0+EXP(A)))
         ELSE
            RES(I,1) = U(I,1) - (0.75e0-0.25e0*EXP(-A)/(EXP(-A)+1.0e0))
            RES(I,2) = U(I,2) - (0.75e0+0.25e0*EXP(-A)/(EXP(-A)+1.0e0))
         END IF
   20 CONTINUE
*
      RETURN
      END
*
      SUBROUTINE MONIT(NPDE,T,DT,DTNEW,TLAST,NLEV,XMIN,YMIN,DXB,DYB,
     +                 LGRID,ISTRUC,LSOL,SOL,IERR)
*     .. Parameters ..
      INTEGER          MAXPTS, NOUT
      PARAMETER        (MAXPTS=2500,NOUT=6)
*     .. Scalar Arguments ..
      real             DT, DTNEW, DXB, DYB, T, XMIN, YMIN
      INTEGER          IERR, NLEV, NPDE
      LOGICAL          TLAST
*     .. Array Arguments ..
      real             SOL(*)
      INTEGER          ISTRUC(*), LGRID(*), LSOL(NLEV)
*     .. Scalars in Common ..
      INTEGER          IOUT
*     .. Arrays in Common ..
      real             TWANT(2)
*     .. Local Scalars ..
```

```
      INTEGER          IFAIL, IPSOL, IPT, LEVEL, NPTS
*     .. Local Arrays ..
      real             UEX(105,2), X(MAXPTS), Y(MAXPTS)
*     .. External Subroutines ..
      EXTERNAL         D03RZF, PDEIV
*     .. Common blocks ..
      COMMON           /OTIME/TWANT, IOUT
*     .. Save statement ..
      SAVE             /OTIME/
*     .. Executable Statements ..
*
      IFAIL = -1
      IF (TLAST) THEN
         DO 40 LEVEL = 1, NLEV
            IPSOL = LSOL(LEVEL)
*
*           Get grid information
*
            CALL D03RZF(LEVEL,NLEV,XMIN,YMIN,DXB,DYB,LGRID,ISTRUC,NPTS,
     +                  X,Y,MAXPTS,IFAIL)
            IF (IFAIL.NE.0) THEN
               IERR = 1
               RETURN
            END IF
*
            IF (IOUT.EQ.2 .AND. LEVEL.EQ.1) THEN
*
*              Get exact solution
*
               CALL PDEIV(NPTS,NPDE,T,X,Y,UEX)
               WRITE (NOUT,*)
               WRITE (NOUT,
     +'('' Solution at every 2nd grid point '', ''in level 1 at time '',
     + F8.4,'':'')') T
               WRITE (NOUT,*)
               WRITE (NOUT,
     +'(7X,''x'',10X,''y'',8X,''approx u'',5X,''exact u'',4X,
     + ''approx v'',4X,''exact v'')')
               WRITE (NOUT,*)
               IPSOL = LSOL(LEVEL)
               DO 20 IPT = 1, NPTS, 2
                  WRITE (NOUT,'(6(1X,D11.4))') X(IPT), Y(IPT),
     +              SOL(IPSOL+IPT), UEX(IPT,1), SOL(IPSOL+NPTS+IPT),
     +              UEX(IPT,2)
   20          CONTINUE
               WRITE (NOUT,*)
            END IF
*
   40    CONTINUE
      END IF
*
      RETURN
      END
```

9.2 Program Data

None.

9.3 Program Results

```
D03RBF Example Program Results
Base grid:


 23  3  3  3  3  3  3  3 34 XX XX

  2 .. .. .. .. .. .. ..  4 XX XX

  2 .. 14  1  1  1  1  1 41 XX XX

  2 ..  4 23  3  3  3  3  3  3 34

  2 ..  4  2 .. .. .. .. .. ..  4

  2 ..  4  2 .. 14  1  1 21 ..  4

  2 ..  4  2 ..  4 XX XX  2 ..  4

  2 ..  4  2 .. 43  3  3 32 ..  4

  2 ..  4  2 .. .. .. .. .. ..  4

  2 ..  4 12  1  1  1  1  1  1 41

 12  1 41 XX XX XX XX XX XX XX XX


Statistics:
Time =   0.2500
Total number of accepted timesteps =   14
Total number of rejected timesteps =    0

          T o t a l  n u m b e r  o f
       Residual  Jacobian   Newton   Lin sys
          evals     evals    iters     iters
At level
    1      196        14       28        14
    2      196        14       28        22
    3      196        14       28        25
    4      196        14       28        31
    5      141        10       21        29

          M a x i m u m  n u m b e r  o f
           Newton iters    Lin sys iters
At level
    1           2             1
    2           2             1
    3           2             1
    4           2             2
    5           3             2


Solution at every 2nd grid point in level 1 at time   1.0000:

     x          y        approx u     exact u    approx v    exact v
```

```
0.0000E+00  0.0000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.2000E+00  0.0000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.1000E+00  0.1000E+00  0.5002E+00  0.5000E+00  0.9998E+00  0.1000E+01
0.3000E+00  0.1000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.5000E+00  0.1000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.7000E+00  0.1000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.9000E+00  0.1000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.0000E+00  0.2000E+00  0.5005E+00  0.5005E+00  0.9995E+00  0.9995E+00
0.2000E+00  0.2000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.4000E+00  0.2000E+00  0.5001E+00  0.5000E+00  0.9999E+00  0.1000E+01
0.6000E+00  0.2000E+00  0.4999E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.8000E+00  0.2000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.1000E+01  0.2000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.1000E+00  0.3000E+00  0.5000E+00  0.5005E+00  0.1000E+01  0.9995E+00
0.3000E+00  0.3000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.5000E+00  0.3000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.7000E+00  0.3000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.9000E+00  0.3000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.0000E+00  0.4000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.2000E+00  0.4000E+00  0.5005E+00  0.5005E+00  0.9995E+00  0.9995E+00
0.4000E+00  0.4000E+00  0.5002E+00  0.5000E+00  0.9998E+00  0.1000E+01
0.8000E+00  0.4000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.1000E+01  0.4000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.1000E+00  0.5000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.3000E+00  0.5000E+00  0.5005E+00  0.5005E+00  0.9995E+00  0.9995E+00
0.5000E+00  0.5000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.7000E+00  0.5000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.9000E+00  0.5000E+00  0.5001E+00  0.5000E+00  0.9999E+00  0.1000E+01
0.0000E+00  0.6000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.2000E+00  0.6000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.4000E+00  0.6000E+00  0.5000E+00  0.5005E+00  0.1000E+01  0.9995E+00
0.6000E+00  0.6000E+00  0.4999E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.8000E+00  0.6000E+00  0.4998E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.1000E+01  0.6000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.1000E+00  0.7000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.3000E+00  0.7000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.5000E+00  0.7000E+00  0.5005E+00  0.5005E+00  0.9995E+00  0.9995E+00
0.7000E+00  0.7000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.9000E+00  0.7000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.0000E+00  0.8000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.2000E+00  0.8000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.4000E+00  0.8000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.6000E+00  0.8000E+00  0.5005E+00  0.5005E+00  0.9995E+00  0.9995E+00
0.8000E+00  0.8000E+00  0.5000E+00  0.5000E+00  0.1000E+01  0.1000E+01
0.1000E+00  0.9000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.3000E+00  0.9000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.5000E+00  0.9000E+00  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.7000E+00  0.9000E+00  0.4999E+00  0.5005E+00  0.1000E+01  0.9995E+00
0.0000E+00  0.1000E+01  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.2000E+00  0.1000E+01  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.4000E+00  0.1000E+01  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.6000E+00  0.1000E+01  0.7500E+00  0.7500E+00  0.7500E+00  0.7500E+00
0.8000E+00  0.1000E+01  0.5005E+00  0.5005E+00  0.9995E+00  0.9995E+00

Statistics:
Time =    1.0000
Total number of accepted timesteps =    45
Total number of rejected timesteps =     0
```

```
           T o t a l   n u m b e r   o f
        Residual  Jacobian    Newton   Lin sys
           evals     evals     iters     iters
At level
     1       630        45        90        45
     2       630        45        90        78
     3       630        45        90        87
     4       630        45        90       124
     5       575        41        83       122

           M a x i m u m   n u m b e r   o f
            Newton iters    Lin sys iters
At level
     1             2              1
     2             2              1
     3             2              1
     4             2              2
     5             3              2
```

D03RYF – NAG Fortran Library Routine Document

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

D03RYF is designed to be used in conjunction with D03RBF. It can be called from the user-supplied subroutine INIDOM to check the user-specified initial grid data and to obtain a simple graphical representation of the initial grid.

2 Specification

```
      SUBROUTINE D03RYF(NX, NY, NPTS, NROWS, NBNDS, NBPTS, LROW, IROW,
     1                  ICOL, LLBND, ILBND, LBND, IWK, LENIWK, PGRID,
     2                  IFAIL)
      INTEGER           NX, NY, NPTS, NROWS, NBNDS, NBPTS, LROW(NROWS),
     1                  IROW(NROWS), ICOL(NPTS), LLBND(NBNDS),
     2                  ILBND(NBNDS), LBND(NBPTS), IWK(LENIWK), LENIWK,
     3                  IFAIL
      CHARACTER*(*)     PGRID(NY)
```

3 Description

D03RYF outputs a character array which can be printed to provide a simple graphical representation of the virtual and base grids supplied to D03RBF. It must be called only from within the user-supplied subroutine INIDOM after all output parameters of INIDOM (other than IERR) have been set. D03RYF also checks the validity of the grid data specified in INIDOM.

The user is strongly advised to call D03RYF during the initial call of D03RBF (at least) and to print the resulting character array in order to check that the base grid is exactly as required.

D03RYF writes a representation of each point in the virtual and base grids to the character array PGRID as follows:

Internal base grid points are written as two dots (..);

Boundary base grid points are written as the ILBND value (i.e., the type) of the boundary;

Points external to the base grid are written as XX.

As an example, consider a rectangular domain with a rectangular hole in which the virtual domain extends by one base grid point beyond the actual domain in all directions. The output when each row of PGRID is printed consecutively is as follows:

```
XX XX XX XX XX XX XX XX XX XX XX XX XX XX
XX 23  3  3  3  3  3  3  3  3  3  3 34 XX
XX  2 .. .. .. .. .. .. .. .. .. ..  4 XX
XX  2 .. .. .. .. .. .. .. .. .. ..  4 XX
XX  2 .. .. 14  1  1  1  1 21 .. ..  4 XX
XX  2 .. ..  4 XX XX XX XX  2 .. ..  4 XX
XX  2 .. ..  4 XX XX XX XX  2 .. ..  4 XX
XX  2 .. ..  4 XX XX XX XX  2 .. ..  4 XX
XX  2 .. ..  4 XX XX XX XX  2 .. ..  4 XX
XX  2 .. ..  4 XX XX XX XX  2 .. ..  4 XX
XX  2 .. .. 43  3  3  3  3 32 .. ..  4 XX
XX  2 .. .. .. .. .. .. .. .. .. ..  4 XX
XX  2 .. .. .. .. .. .. .. .. .. ..  4 XX
XX 12  1  1  1  1  1  1  1  1  1  1 41 XX
XX XX XX XX XX XX XX XX XX XX XX XX XX XX
```

4 References

None.

5 Parameters

1: NX — INTEGER *Input*

2: NY — INTEGER *Input*

On entry: the number of virtual grid points in the x- and y-direction respectively (including the boundary points).

Constraints: NX and NY ≥ 4.

3: NPTS — INTEGER *Input*

On entry: the total number of points in the base grid.

Constraint: NPTS $\leq$ NX $\times$ NY.

4: NROWS — INTEGER *Input*

On entry: the total number of rows of the virtual grid that contain base grid points.

Constraint: $4 \leq$ NROWS $\leq$ NY.

5: NBNDS — INTEGER *Input*

On entry: the total number of physical boundaries and corners in the base grid.

Constraint: NBNDS ≥ 8.

6: NBPTS — INTEGER *Input*

On entry: the total number of boundary points in the base grid.

Constraint: $12 \leq$ NBPTS $<$ NPTS.

7: LROW(NROWS) — INTEGER array *Input*

On entry: LROW(i) for $i = 1,2,\ldots$,NROWS contains the base grid index of the first grid point in base grid row i

Constraints:

$1 \leq$ LROW(i) $\leq$ NPTS for $i = 1,2,\ldots$,NROWS,
LROW(i−1) $<$ LROW(i), $i = 2,3,\ldots$,NROWS.

8: IROW(NROWS) — INTEGER array *Input*

On entry: IROW(i) for $i = 1,2,\ldots$,NROWS contains the virtual grid row number that corresponds to base grid row i.

Constraints:

$0 \leq$ IROW(i) $\leq$ NY for $i = 1,2,\ldots$,NROWS,
IROW(i−1) $<$ IROW(i), $i = 2,3,\ldots$,NROWS.

9: ICOL(NPTS) — INTEGER array *Input*

On entry: ICOL(i) for $i = 1,2,\ldots$,NPTS contains the virtual grid column number that contains base grid point i.

Constraint: $0 \leq$ ICOL(i) $\leq$ NX for $i = 1,2,\ldots$,NPTS.

10: LLBND(NBNDS) — INTEGER array *Input*

On entry: LLBND(i) for $i = 1,2,\ldots$,NBNDS contains the element of LBND corresponding to the start of the ith boundary (or corner).

Constraints:

$1 \leq$ LLBND(i) $\leq$ NBPTS for $i = 1,2,\ldots$,NBNDS,
LLBND(i–1) < LLBND(i), $i = 2,3,\ldots$,NBNDS.

11: ILBND(NBNDS) — INTEGER array *Input*

On entry: ILBND(i) for $i = 1,2,\ldots$,NBNDS contains the type of the ith boundary (or corner), as defined in D03RBF.

Constraint: ILBND(i) must be equal to one of the following: 1, 2, 3, 4, 12, 23, 34, 41, 21, 32, 43 or 14, for $i = 1,2,\ldots$,NBNDS.

12: LBND(NBPTS) — INTEGER array *Input*

On entry: LBND(i) for $i = 1,2,\ldots$,NBPTS contains the grid index of the ith boundary point.

Constraint: $1 \leq$ LBND(i) $\leq$ NPTS for $i = 1,2,\ldots$,NBPTS.

13: IWK(LENIWK) — INTEGER array *Workspace*
14: LENIWK — INTEGER *Input*

On entry: the dimension of the array IWK as declared in the (sub)program from which D03RYF is called.

Constraint: LENIWK $\geq$ NX $\times$ NY + 1.

15: PGRID(NY) — CHARACTER*(*) *Output*

On exit: PGRID(i) for $i = 1,2,\ldots$,NY contains a graphical representation of row NY–i + 1 of the virtual grid (see Section 3).

Constraint: LEN(PGRID(1)) $\geq 3 \times$ NX.

16: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, NX or NY < 4,
or NPTS > NX $\times$ NY,
or NROWS < 4,
or NROWS > NY,
or NBNDS < 8,
or NBPTS < 12,
or NBPTS $\geq$ NPTS,
or LROW(i) < 1 for some $i = 1,2,\ldots$,NROWS,
or LROW(i) > NPTS for some $i = 1,2,\ldots$,NROWS,
or LROW(i) $\leq$ LROW(i–1) for some $i = 2,3,\ldots$,NROWS,

or IROW(i) $<$ 0 for some $i = 1, 2, \ldots$,NROWS,

or IROW(i) $>$ NY for some $i = 1, 2, \ldots$,NROWS,

or IROW(i) $\leq$ IROW($i-1$) for some $i = 2, 3, \ldots$,NROWS,

or ICOL(i) $<$ 0 for some $i = 1, 2, \ldots$,NPTS,

or ICOL(i) $>$ NX for some $i = 1, 2, \ldots$,NPTS,

or LLBND(i) $<$ 1 for some $i = 1, 2, \ldots$,NBNDS,

or LLBND(i) $>$ NBPTS for some $i = 1, 2, \ldots$,NBNDS,

or LLBND(i) $\leq$ LLBND($i-1$) for some $i = 2, 3, \ldots$,NBPTS,

or ILBND(i) $\neq$ 1, 2, 3, 4, 12, 23, 34, 41, 21, 32, 43 or 14, for some $i = 1, 2, \ldots$,NBNDS,

or LBND(i) $<$ 1 for some $i = 1, 2, \ldots$,NBPTS,

or LBND(i) $>$ NPTS for some $i = 1, 2, \ldots$,NBPTS,

or LENIWK $<$ NX $\times$ NY + 1,

or LEN(PGRID(1)) $<$ 3 $\times$ NX.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

See Section 9 of the document for D03RBF.

D03RZF – NAG Fortran Library Routine Document

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

D03RZF is designed to be used in conjunction with D03RBF. It can be called from the user-supplied MONITR subroutine to obtain the number of grid points and their (x, y) co-ordinates on a solution grid.

2 Specification

```
      SUBROUTINE D03RZF(LEVEL, NLEV, XMIN, YMIN, DXB, DYB, LGRID,
     1                  ISTRUC, NPTS, X, Y, LENXY, IFAIL)
      INTEGER           LEVEL, NLEV, LGRID(*), ISTRUC(*), NPTS, LENXY,
     1                  IFAIL
      real              XMIN, YMIN, DXB, DYB, X(LENXY), Y(LENXY)
```

3 Description

D03RZF extracts the number of grid points and their (x, y) co-ordinates on a specific solution grid produced by D03RBF. It must be called only from within the user-supplied subroutine MONITR. The parameters NLEV, XMIN, YMIN, DXB, DYB, LGRID and ISTRUC to MONITR must be passed unchanged to D03RZF.

4 References

None.

5 Parameters

1: LEVEL — INTEGER *Input*

On entry: the grid level at which the co-ordinates are required.

Constraint: $1 \leq$ LEVEL $\leq$ NLEV.

2: NLEV — INTEGER *Input*

3: XMIN — ***real*** *Input*

4: YMIN — ***real*** *Input*

5: DXB — ***real*** *Input*

6: DYB — ***real*** *Input*

7: LGRID(*) — INTEGER array *Input*

8: ISTRUC(*) — INTEGER array *Input*

On entry: NLEV, XMIN, YMIN, DXB, DYB, LGRID and ISTRUC as supplied to MONITR must be passed unchanged to D03RZF.

9: NPTS — INTEGER *Output*

On exit: the number of grid points in the grid level LEVEL.

10: X(LENXY) — ***real*** array *Output*

11: Y(LENXY) — ***real*** array *Output*

On exit: X(i) and Y(i) contain the (x, y) co-ordinates respectively of the ith grid point, for $i = 1, 2, \ldots,$NPTS.

12: LENXY — INTEGER *Input*

On entry: the dimension of the arrays X and Y as declared in MONITR.

Constraint: LENXY $\geq$ NPTS.

13: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, LEVEL < 1,
or LEVEL > NLEV.

IFAIL = 2

The dimension of the arrays X and Y is too small for the requested grid level, i.e., LENXY < NPTS.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

See Section 9 of the document for D03RBF.

D03UAF – NAG Fortran Library Routine Document

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. The routine name may be precision-dependent.

1. Purpose

D03UAF performs at each call one iteration of the Strongly Implicit Procedure. It is used to calculate on successive calls a sequence of approximate corrections to the current estimate of the solution when solving a system of simultaneous algebraic equations for which the iterative up-date matrix is of five-point molecule form on a two-dimensional topologically-rectangular mesh. ('Topological' means that a polar grid, for example (r,θ), can be used, being equivalent to a rectangular box.)

2. Specification

```
      SUBROUTINE D03UAF (N1, N2, N1M, A, B, C, D, E, APARAM, IT, R,
     1                   WRKSP1, WRKSP2, IFAIL)
      INTEGER            N1, N2, N1M, IT, IFAIL
      real               A(N1M,N2), B(N1M,N2), C(N1M,N2), D(N1M,N2),
     1                   E(N1M,N2), APARAM, R(N1M,N2), WRKSP1(N1M,N2),
     2                   WRKSP2(N1M,N2)
```

3. Description

Given a set of simultaneous equations

$$Mt = q \qquad (1)$$

(which could be nonlinear) derived, for example, from a finite difference representation of a two-dimensional elliptic partial differential equation and its boundary conditions, the solution t may be obtained iteratively from a starting approximation $t^{(1)}$ by the formulae

$$\begin{aligned} r^{(n)} &= q - Mt^{(n)} \\ Ms^{(n)} &= r^{(n)} \\ t^{(n+1)} &= t^{(n)} + s^{(n)}. \end{aligned}$$

Thus $r^{(n)}$ is the residual of the nth approximate solution $t^{(n)}$, and $s^{(n)}$ is the update change vector.

D03UAF determines the approximate change vector s corresponding to a given residual r, i.e. it determines an approximate solution to a set of equations

$$Ms = r \qquad (2)$$

where r is a known vector of length $n_1 \times n_2$, and M is a square $(n_1 \times n_2)$ by $(n_1 \times n_2)$ matrix. The system (2) must be of five-diagonal form

$$a_{ij}s_{i,j-1} + b_{ij}s_{i-1,j} + c_{ij}s_{ij} + d_{ij}s_{i+1,j} + e_{ij}s_{i,j+1} = r_{ij}$$

for $i = 1,2,\ldots,n_1$; $j = 1,2,\ldots,n_2$, provided that $c_{ij} \neq 0.0$. Indeed, if $c_{ij} = 0.0$, then the equation is assumed to be

$$s_{ij} = r_{ij}.$$

For example, if $n_1 = 3$ and $n_2 = 2$, the equations take the form

$$\begin{bmatrix} c_{11} & d_{11} & & e_{11} & & \\ b_{21} & c_{21} & d_{21} & & e_{21} & \\ & b_{31} & c_{31} & & & e_{31} \\ a_{12} & & & c_{12} & d_{12} & \\ & a_{22} & & b_{22} & c_{22} & d_{22} \\ & & a_{32} & & b_{32} & c_{32} \end{bmatrix} \begin{bmatrix} s_{11} \\ s_{21} \\ s_{31} \\ s_{12} \\ s_{22} \\ s_{32} \end{bmatrix} = \begin{bmatrix} r_{11} \\ r_{21} \\ r_{31} \\ r_{12} \\ r_{22} \\ r_{32} \end{bmatrix}$$

The calling program supplies the current residual r at each iteration and the coefficients of the five-point molecule system of equations on which the up-date procedure is based. The routine performs one iteration, using the approximate *LU* factorization of the Strongly Implicit Procedure with the necessary acceleration parameter adjustment, to calculate the approximate solution s of the system (2). The change s overwrites the residual array for return to the calling program. The calling program must combine this change stored in r with the old approximation to obtain the new approximate solution for t. It must then recalculate the residuals and, if the accuracy requirements have not been satisfied, commence the next iterative cycle.

Clearly there is no requirement that the iterative up-date matrix passed in the form of the five-diagonal element arrays A, B, C, D, E is the same as that used to calculate the residuals, and therefore the one governing the problem. However the convergence may be impaired if they are not equal. Indeed, if the system of equations (1) is not precisely of the five-diagonal form illustrated above but has a few additional terms, then the methods of deferred or defect correction can be employed. The residual is calculated by the calling program using the full system of equations, but the up-date formula is based on a five-diagonal system (2) of the form given above. For example, the solution of a system of nine-diagonal equations each involving the combination of terms with $t_{i\pm1,j\pm1}$, $t_{i\pm1,j}$, $t_{i,j\pm1}$ and t_{ij} could use the five-diagonal coefficients on which to base the up-date, provided these incorporate the major features of the equations.

Problems in topologically non-rectangular regions can be solved using the routine, by surrounding the region with a circumscribing topological rectangle. The equations for the nodal values external to the region of interest are set to zero (i.e. $c_{ij} = r_{ij} = 0$) and the boundary conditions are incorporated into the equations for the appropriate nodes.

If there is no better initial approximation when starting the iterative cycle, one can use an array of all zeros as the initial approximation from which the first set of residuals are determined.

The routine can be used to solve linear elliptic equations in which case the arrays A, B, C, D, E and Q will be unchanged during the iterative cycles, or for solving nonlinear elliptic equations in which case some or all of these arrays may require updating as each new approximate solution is derived. Depending on the nonlinearity, some under-relaxation of the coefficients and/or source terms may be needed during their recalculation using the new estimates of the solution (see Jacobs [1]).

The routine can also be used to solve each step of a time-dependent parabolic equation in two-space dimensions. The solution at each time step can be expressed in terms of an elliptic equation if the Crank-Nicolson or other form of implicit time integration is used.

Neither diagonal dominance, nor positive definiteness, of the matrix M and the up-date matrix formed from the arrays A, B, C, D, E is necessary to ensure convergence.

For problems in which the solution is not unique, in the sense that an arbitrary constant can be added to the solution, (for example Laplace's equation with all Neumann boundary conditions), the calling program should subtract a typical nodal value from the whole solution t at every iteration to keep rounding errors to a minimum.

4. References

[1] AMES, W.F.
Nonlinear Partial Differential Equations in Engineering.
Academic Press, London, 1965.

[2] JACOBS, D.A.H.
The strongly implicit procedure for the numerical solution of parabolic and elliptic partial differential equations.
Central Electricity Research Laboratory Note RD/L/N66/72, 1972.

[3] STONE, H.L.
Iterative solution of implicit approximations of multidimensional partial differential equations.
SIAM J. Numer. Anal., 5, pp. 530-558, 1968.

5. Parameters

1: N1 – INTEGER. *Input*

On entry: the number of nodes in the first co-ordinate direction, n_1.

Constraint: N1 > 1.

2: N2 – INTEGER. *Input*

On entry: the number of nodes in the second co-ordinate direction, n_2.

Constraint: N2 > 1.

3: N1M – INTEGER. *Input*

On entry: the first dimension of the arrays A, B, C, D, E, R, WRKSP1 and WRKSP2, as declared in the (sub)program from which D03UAF is called.

Constraint: N1M ≥ N1.

4: A(N1M,N2) – ***real*** array. *Input*

On entry: A(i,j) must contain the coefficient of the 'southerly' term involving $s_{i,j-1}$ in the (i,j)th equation of the system (2), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of A for $j = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

5: B(N1M,N2) – ***real*** array. *Input*

On entry: B(i,j) must contain the coefficient of the 'westerly' term involving $s_{i-1,j}$ in the (i,j)th equation of the system (2), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of B for $i = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

6: C(N1M,N2) – ***real*** array. *Input*

On entry: C(i,j) must contain the coefficient of the 'central' term involving s_{ij} in the (i,j)th equation of the system (2), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of C are checked to ensure that they are non-zero. If any element is found to be zero, the corresponding algebraic equation is assumed to be $s_{ij} = r_{ij}$. This feature can be used to define the equations for nodes at which, for example, Dirichlet boundary conditions are applied, or for nodes external to the problem of interest, by setting C(i,j) = 0.0 at appropriate points. The corresponding value of R(i,j) is set equal to the appropriate value, namely the difference between the prescribed value of t_{ij} and the current value of t_{ij} in the Dirichlet case, or zero at an external point.

7: D(N1M,N2) – ***real*** array. *Input*

On entry: D(i,j) must contain the coefficient of the 'easterly' term involving $s_{i+1,j}$ in the (i,j)th equation of the system (2), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of D for i = N1 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

8: E(N1M,N2) – ***real*** array. *Input*

On entry: E(i,j) must contain the coefficient of the 'northerly' term involving $s_{i,j+1}$ in the (i,j)th equation of the system (2), for i = 1,2,...,N1; j = 1,2,...,N2. The elements of E for j = N2 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the rectangle.

9: APARAM – ***real***. *Input*

On entry: the iteration acceleration factor. A value of 1.0 is adequate for most typical problems. However, if convergence is slow, the value can be reduced, typically to 0.2 or 0.1. If divergence is obtained, the value can be increased, typically to 2.0, 5.0 or 10.0.

Constraint: $0.0 < \text{APARAM} \le ((\text{N1}-1)^2+(\text{N2}-1)^2)/2.0$.

10: IT – INTEGER. *Input*

On entry: the iteration number. It must be initialised, but not necessarily to 1, before the first call, and must be incremented by one in the calling program for each subsequent call. The routine uses the counter to select the appropriate acceleration parameter from a sequence of nine, each one being used twice in succession. (Note that the acceleration parameter depends on the value of APARAM.)

11: R(N1M,N2) – ***real*** array. *Input/Output*

On entry: R(i,j) must contain the current residual r_{ij} on the right-hand side of the (i,j)th equation of the system (2), for $i = 1,2,...,\text{N1}$; $j = 1,2,...,\text{N2}$.

On exit: these residuals are overwritten by the corresponding components of solution s to the system (2), i.e. the changes to be made to the vector t to reduce the residuals supplied.

12: WRKSP1(N1M,N2) – ***real*** array. *Workspace*
13: WRKSP2(N1M,N2) – ***real*** array. *Workspace*

14: IFAIL – INTEGER. *Input/Output*

On entry: IFAIL must be set to 0, –1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6. Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, N1 < 2,
or N2 < 2.

IFAIL = 2

On entry, N1M < N1.

IFAIL = 3

On entry, APARAM ≤ 0.0.

IFAIL = 4

On entry, $\text{APARAM} > ((\text{N1}-1)^2+(\text{N2}-1)^2)/2.0$.

7. Accuracy

The improvement in accuracy for each iteration, i.e. on each call, depends on the size of the system and on the condition of the up-date matrix characterised by the five-diagonal coefficient arrays. The ultimate accuracy obtainable depends on the above factors and on the ***machine precision***. However, since the routine works with residuals and the up-date vector, the calling program can, in most cases where at each iteration all the residuals are usually of about the same size, calculate the residuals from extended precision values of the function, source term and equation coefficients if greater accuracy is required. The rate of convergence obtained with the Strongly Implicit Procedure is not always smooth because of the cyclic use of nine acceleration parameters. The convergence may become slow with very large problems, for example

N1 = N2 = 60. The final accuracy obtained can be judged approximately from the rate of convergence determined from the changes to the dependent variable T and in particular the change on the last iteration.

8. Further Comments

The time taken by the routine is approximately proportional to N1×N2 for each call.

When used with deferred or defect correction, the residual is calculated in the calling program from a different system of equations to those represented by the five-point molecule coefficients used by the routine as the basis of the iterative up-date procedure. When using deferred correction the overall rate of convergence depends not only on the items detailed in Section 7 but also on the difference between the two coefficient matrices used.

Convergence may not always be obtained when the problem is very large and/or the coefficients of the equations have widely disparate values. The latter case is often associated with a near ill-conditioned matrix.

9. Example

To solve Laplace's equation in a rectangle with a non-uniform grid spacing in the x and y co-ordinate directions and with Dirichlet boundary conditions specifying the function on the perimeter of the rectangle equal to $e^{(1.0+x)/y(n_2)}\times\cos(y/y(n_2))$.

9.1. 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.

```
*     D03UAF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          N1, N2, N1M, NITS
      PARAMETER        (N1=6,N2=10,N1M=N1,NITS=10)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ADEL, APARAM, ARES, DELMAX, DELMN, RESMAX, RESMN
      INTEGER          I, IFAIL, IT, J
*     .. Local Arrays ..
      real             A(N1M,N2), B(N1M,N2), C(N1M,N2), D(N1M,N2),
     +                 E(N1M,N2), Q(N1M,N2), R(N1M,N2), T(N1M,N2),
     +                 WRKSP1(N1M,N2), WRKSP2(N1M,N2), X(N1), Y(N2)
*     .. External Subroutines ..
      EXTERNAL         D03UAF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, COS, EXP, MAX, real
*     .. Data statements ..
      DATA             X(1), X(2), X(3), X(4), X(5), X(6)/0.0e0, 1.0e0,
     +                 3.0e0, 6.0e0, 10.0e0, 15.0e0/
      DATA             Y(1), Y(2), Y(3), Y(4), Y(5), Y(6), Y(7), Y(8),
     +                 Y(9), Y(10)/0.0e0, 1.0e0, 3.0e0, 6.0e0, 10.0e0,
     +                 15.0e0, 21.0e0, 28.0e0, 36.0e0, 45.0e0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03UAF Example Program Results'
      WRITE (NOUT,*)
      APARAM = 1.0e0
*     Set up difference equation coefficients, source terms and
*     initial S
      DO 40 J = 1, N2
         DO 20 I = 1, N1
            IF ((I.NE.1) .AND. (I.NE.N1) .AND. (J.NE.1) .AND. (J.NE.N2))
     +          THEN
```

```
*                Specification for internal nodes
                 A(I,J) = 2.0e0/((Y(J)-Y(J-1))*(Y(J+1)-Y(J-1)))
                 E(I,J) = 2.0e0/((Y(J+1)-Y(J))*(Y(J+1)-Y(J-1)))
                 B(I,J) = 2.0e0/((X(I)-X(I-1))*(X(I+1)-X(I-1)))
                 D(I,J) = 2.0e0/((X(I+1)-X(I))*(X(I+1)-X(I-1)))
                 C(I,J) = -A(I,J) - B(I,J) - D(I,J) - E(I,J)
                 Q(I,J) = 0.0e0
                 T(I,J) = 0.0e0
              ELSE
*                Specification for boundary nodes
                 A(I,J) = 0.0e0
                 B(I,J) = 0.0e0
                 C(I,J) = 0.0e0
                 D(I,J) = 0.0e0
                 E(I,J) = 0.0e0
                 Q(I,J) = EXP((X(I)+1.0e0)/Y(N2))*COS(Y(J)/Y(N2))
                 T(I,J) = 0.0e0
              END IF
   20      CONTINUE
   40 CONTINUE
*     Iterative loop
      WRITE (NOUT,*) 'Iteration      Residual                   Change'
      WRITE (NOUT,*)
     + '  No        Max.         Mean            Max.        Mean'
      WRITE (NOUT,*)
      DO 140 IT = 1, NITS
*        Calculate the residuals
         RESMAX = 0.0e0
         RESMN = 0.0e0
         DO 80 J = 1, N2
            DO 60 I = 1, N1
               IF (C(I,J).NE.0.0e0) THEN
*                 Five point molecule formula
                  R(I,J) = Q(I,J) - A(I,J)*T(I,J-1) - B(I,J)*T(I-1,J) -
     +                     C(I,J)*T(I,J) - D(I,J)*T(I+1,J) - E(I,J)*T(I,
     +                     J+1)
               ELSE
*                 Explicit equation
                  R(I,J) = Q(I,J) - T(I,J)
               END IF
               ARES = ABS(R(I,J))
               RESMAX = MAX(RESMAX,ARES)
               RESMN = RESMN + ARES
   60       CONTINUE
   80    CONTINUE
         RESMN = RESMN/(real(N1*N2))
         IFAIL = 0
*
         CALL D03UAF(N1,N2,N1M,A,B,C,D,E,APARAM,IT,R,WRKSP1,WRKSP2,
     +               IFAIL)
*
*        Update the dependent variable
         DELMAX = 0.0e0
         DELMN = 0.0e0
         DO 120 J = 1, N2
            DO 100 I = 1, N1
               T(I,J) = T(I,J) + R(I,J)
               ADEL = ABS(R(I,J))
               DELMAX = MAX(DELMAX,ADEL)
               DELMN = DELMN + ADEL
  100       CONTINUE
  120    CONTINUE
         DELMN = DELMN/(real(N1*N2))
         WRITE (NOUT,99999) IT, RESMAX, RESMN, DELMAX, DELMN
*        Convergence tests here if required
  140 CONTINUE
```

```
*     End of iterative loop
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Table of calculated function values'
      WRITE (NOUT,*)
      WRITE (NOUT,*)
     +'   I     1           2           3           4           5           6'
      WRITE (NOUT,*) ' J'
      DO 160 J = 1, N2
         WRITE (NOUT,99998) J, (T(I,J),I=1,N1)
  160 CONTINUE
      STOP
*
99999 FORMAT (1X,I3,4(2X,e11.4))
99998 FORMAT (1X,I2,1X,6(F9.3,2X))
      END
```

9.2. Program Data

None.

9.3. Program Results

```
 D03UAF Example Program Results

 Iteration       Residual                      Change
   No         Max.         Mean          Max.        Mean

    1   0.1427E+01   0.4790E+00   0.1427E+01   0.1031E+01
    2   0.1098E-02   0.3871E-03   0.2176E-01   0.6158E-02
    3   0.7364E-03   0.5926E-04   0.1621E-02   0.2475E-03
    4   0.2036E-04   0.2914E-05   0.1810E-03   0.2259E-04
    5   0.6946E-05   0.6214E-06   0.1199E-04   0.2347E-05
    6   0.2267E-06   0.4215E-07   0.1245E-05   0.2270E-06
    7   0.5625E-07   0.4500E-08   0.1081E-06   0.1761E-07
    8   0.2305E-08   0.3998E-09   0.1289E-07   0.1794E-08
    9   0.4733E-09   0.7397E-10   0.1422E-08   0.1841E-09
   10   0.7109E-10   0.8598E-11   0.3214E-09   0.2791E-10

 Table of calculated function values

    I     1           2           3           4           5           6
  J
  1       1.022       1.045       1.093       1.168       1.277       1.427
  2       1.022       1.045       1.093       1.168       1.277       1.427
  3       1.020       1.043       1.091       1.166       1.274       1.424
  4       1.013       1.036       1.083       1.158       1.266       1.414
  5       0.997       1.020       1.066       1.140       1.246       1.392
  6       0.966       0.988       1.033       1.104       1.207       1.348
  7       0.913       0.934       0.976       1.044       1.141       1.274
  8       0.831       0.850       0.888       0.950       1.038       1.160
  9       0.712       0.728       0.762       0.814       0.890       0.994
 10       0.552       0.565       0.591       0.631       0.690       0.771
```

D03UBF – NAG Fortran Library Routine Document

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

D03UBF performs at each call one iteration of the Strongly Implicit Procedure. It is used to calculate on successive calls the sequence of approximate corrections to the current estimate of the solution when solving a system of simultaneous algebraic equations for which the iterative up-date matrix is of seven-point molecule form on a three-dimensional topologically-rectangular mesh. ('Topological' means that a polar grid, for example, can be used if it is equivalent to a rectangular box.)

2 Specification

```
 SUBROUTINE D03UBF(N1, N2, N3, N1M, N2M, A, B, C, D, E, F, G,
1                  APARAM, IT, R, WRKSP1, WRKSP2, WRKSP3, IFAIL)
 INTEGER           N1, N2, N3, N1M, N2M, IT, IFAIL
 real              A(N1M,N2M,N3), B(N1M,N2M,N3), C(N1M,N2M,N3),
1                  D(N1M,N2M,N3), E(N1M,N2M,N3), F(N1M,N2M,N3),
2                  G(N1M,N2M,N3), APARAM, R(N1M,N2M,N3),
3                  WRKSP1(N1M,N2M,N3), WRKSP2(N1M,N2M,N3),
4                  WRKSP3(N1M,N2M,N3)
```

3 Description

Given a set of simultaneous equations

$$Mt = q \qquad (1)$$

(which could be nonlinear) derived, for example, from a finite difference representation of a three-dimensional elliptic partial differential equation and its boundary conditions, the solution t may be obtained iteratively from a starting approximation $t^{(1)}$ by the formulae

$$r^{(n)} = q - Mt^{(n)}$$

$$Ms^{(n)} = r^{(n)}$$

$$t^{(n+1)} = t^{(n)} + s^{(n)}.$$

Thus $r^{(n)}$ is the residual of the nth approximate solution $t^{(n)}$, and $s^{(n)}$ is the update change vector.

D03UBF determines the approximate change vector s corresponding to a given residual r, i.e., it determines an approximate solution to a set of equations

$$Ms = r \qquad (2)$$

where M is a square $(n_1 \times n_2 \times n_3)$ by $(n_1 \times n_2 \times n_3)$ matrix and r is a known vector of length $(n_1 \times n_2 \times n_3)$. The equations (2) must be of seven-diagonal form:

$$a_{ijk}s_{ij,k-1} + b_{ijk}s_{i,j-1,k} + c_{ijk}s_{i-1,jk} + d_{ijk}s_{ijk} + e_{ijk}s_{i+1,jk} + f_{ijk}s_{i,j+1,k} + g_{ijk}s_{ij,k+1} = r_{ijk}$$

with $i = 1, 2, \ldots, n_1$; $j = 1, 2, \ldots, n_2$ and $k = 1, 2, \ldots, n_3$, provided that $d_{ijk} \neq 0.0$. Indeed, if $d_{ijk} = 0.0$, then the equation is assumed to be:

$$s_{ijk} = r_{ijk}.$$

The calling program supplies the current residual r at each iteration and the coefficients of the seven-point molecule system of equations on which the up-date procedure is based. The routine performs one iteration, using the approximate LU factorization of the Strongly Implicit Procedure with the necessary acceleration parameter adjustment, to calculate the approximate solution s of the system (2). The change s overwrites the residual array, for return to the calling program. The calling program must combine this change stored in r with the old approximation to obtain the new approximate solution for t. It must then recalculate the residuals and, if the accuracy requirements have not been satisfied, commence the next iterative cycle.

Clearly there is no requirement that the iterative up-date matrix passed in the form of the seven-diagonal element arrays A, B, C, D, E, F, G is the same as that used to calculate the residuals, and therefore the one governing the problem. However, the convergence may be impaired if they are not equal. Indeed, if the system of equations (1) is not precisely of the seven-diagonal form illustrated above but has a few additional terms, then the methods of deferred or defect correction can be employed. The residual is calculated by the calling program using the full system of equations, but the up-date formula is based on a seven-diagonal system (2) of the form given above. For example, the solution of a system of eleven-diagonal equations, each involving the combination of terms with $t_{i\pm1,j\pm1,k}$, $t_{i\pm1,j,k}$, $t_{i,j\pm1,k}$, $t_{i,j,k\pm1}$ and t_{ijk} could use the seven-diagonal coefficients on which to base the up-date, provided these incorporate the major features of the equations.

Problems in topologically non-rectangular-box-shaped regions can be solved using the routine by surrounding the region by a circumscribing topologically rectangular box. The equations for the nodal values external to the region of interest are set to zero (i.e., $d_{ijk} = r_{ijk} = 0$) and the boundary conditions are incorporated into the equations for the appropriate nodes.

If there is no better initial approximation when starting the iterative cycle, one can use an array of all zeros as the initial approximation from which the first set of residuals are determined.

The routine can be used to solve linear elliptic equations in which case Q and the arrays A, B, C, D, E, F and G will be unchanged during the iterative cycles. It can also be used for solving nonlinear elliptic equations in which case some or all of these arrays may require updating as each new approximate solution is derived. Depending on the nonlinearity, some under-relaxation of the coefficient and/or source terms may be needed during their recalculations using the new estimates of the solution (see Ames [1]).

The routine can also be used to solve each step of a time-dependent parabolic equation in three space dimensions. The solution at each time step can be expressed in terms of an elliptic equation if the Crank–Nicolson or other form of implicit time integration is used.

Neither diagonal dominance, nor positive-definiteness, of the matrix M or of the up-date matrix formed from the arrays A, B, C, D, E, F, G is necessary to ensure convergence.

For problems in which the solution is not unique, in the sense that an arbitrary constant can be added to the solution, for example Poisson's equation with all Neumann boundary conditions, the calling program should subtract a typical nodal value from the whole solution t at every iteration to keep rounding errors to a minimum for those cases when convergence is slow. For such problems there is generally an associated compatibility condition. For the example mentioned this compatibility condition equates the total net source within the region (i.e., the source integrated over the region) with the total net outflow across the boundaries defined by the Neumann conditions (i.e., the normal derivative integrated along the whole boundary). It is very important that the algebraic equations derived to model such a problem accurately implement the compatibility condition. If they do not, a net source or sink is very likely to be represented by the set of algebraic equations and no steady-state solution of the equations exists.

4 References

[1] Ames W F (1977) *Nonlinear Partial Differential Equations in Engineering* Academic Press (2nd Edition)

[2] Jacobs D A H (1972) The strongly implicit procedure for the numerical solution of parabolic and elliptic partial differential equations *Note RD/L/N66/72* Central Electricity Research Laboratory

[3] Stone H L (1968) Iterative solution of implicit approximations of multi-dimensional partial differential equations *SIAM J. Numer. Anal.* **5** 530–558

[4] Weinstein H G, Stone H L and Kwan T V (1969) Iterative procedure for solution of systems of parabolic and elliptic equations in three dimensions *Industrial and Engineering Chemistry Fundamentals* **8** 281–287

5 Parameters

1: N1 — INTEGER *Input*

On entry: the number of nodes in the first co-ordinate direction, n_1.

Constraint: N1 > 1.

2: N2 — INTEGER *Input*

On entry: the number of nodes in the second co-ordinate direction, n_2.

Constraint: N2 > 1.

3: N3 — INTEGER *Input*

On entry: the number of nodes in the third co-ordinate direction, n_3.

Constraint: N3 > 1.

4: N1M — INTEGER *Input*

On entry: the first dimension of all the three-dimensional arrays, as declared in the (sub)program from which D03UBF is called.

Constraint: N1M $\geq$ N1.

5: N2M — INTEGER *Input*

On entry: the second dimension of all the three-dimensional arrays, as declared in the (sub)program from which D03UBF is called.

Constraint: N2M $\geq$ N2.

6: A(N1M,N2M,N3) — ***real*** array *Input*

On entry: A(i,j,k) must contain the coefficient of $s_{ij,k-1}$ in the (i,j,k)th equation of the system (2) for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of A for $k = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

7: B(N1M,N2M,N3) — ***real*** array *Input*

On entry: B(i,j,k) must contain the coefficient of $s_{i,j-1,k}$ in the (i,j,k)th equation of the system (2) for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of B for $j = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

8: C(N1M,N2M,N3) — ***real*** array *Input*

On entry: C(i,j,k) must contain the coefficient of $s_{i-1,j,k}$ in the (i,j,k)th equation of the system (2), for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of C for $i = 1$ must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

9: D(N1M,N2M,N3) — ***real*** array *Input*

On entry: D(i,j,k) must contain the coefficient of s_{ijk}, the 'central' term, in the (i,j,k)th equation of the system (2), for $i = 1,2,\ldots,$N1; $j = 1,2,\ldots,$N2 and $k = 1,2,\ldots,$N3. The elements of D are checked to ensure that they are non-zero. If any element is found to be zero, the corresponding algebraic equation is assumed to be $s_{ijk} = r_{ijk}$. This feature can be used to define the equations for nodes at which, for example, Dirichlet boundary conditions are applied, or for nodes external to the problem of interest, by setting D$(i,j,k) = 0.0$ at appropriate points. The corresponding value of r_{ijk} is set equal to the appropriate value, namely the difference between the prescribed value of t_{ijk} and the current value in the Dirichlet case, or zero at an external point.

10: E(N1M,N2M,N3) — ***real*** array *Input*

On entry: E(i,j,k) must contain the coefficient of $s_{i+1,j,k}$ in the (i,j,k)th equation of the system (2), for $i = 1,2,\ldots$,N1; $j = 1,2,\ldots$,N2 and $k = 1,2,\ldots$,N3. The elements of E for i = N1 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

11: F(N1M,N2M,N3) — ***real*** array *Input*

On entry: F(i,j,k) must contain the coefficient of $s_{i,j+1,k}$ in the (i,j,k)th equation of the system (2), for $i = 1,2,\ldots$,N1; $j = 1,2,\ldots$,N2 and $k = 1,2,\ldots$,N3. The elements of F for j = N2 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

12: G(N1M,N2M,N3) — ***real*** array *Input*

On entry: G(i,j,k) must contain the coefficient of $s_{i,j,k+1}$ in the (i,j,k)th equation of the system (2), for $i = 1,2,\ldots$,N1; $j = 1,2,\ldots$,N2 and $k = 1,2,\ldots$,N3. The elements of G for k = N3 must be zero after incorporating the boundary conditions, since they involve nodal values from outside the box.

13: APARAM — ***real*** *Input*

On entry: the iteration acceleration factor. A value of 1.0 is adequate for most typical problems. However, if convergence is slow, the value can be reduced, typically to 0.2 or 0.1. If divergence is obtained, the value can be increased, typically to 2.0, 5.0 or 10.0.

Constraint: 0.0 < APARAM $\leq$ ((N1−1)^2 + (N2−1)^2 + (N3−1)^2)/3.0.

14: IT — INTEGER *Input*

On entry: the iteration number. It must be initialised, but not necessarily to 1, before the first call, and should be incremented by one in the calling program for each subsequent call. The routine uses this counter to select the appropriate acceleration parameter from a sequence of nine, each one being used twice in succession. (Note that the acceleration parameter depends on the value of APARAM).

15: R(N1M,N2M,N3) — ***real*** array *Input/Output*

On entry: the current residual r_{ijk} on the right-hand side of the (i,j,k)th equation of the system (2), $i = 1,2,\ldots$,N1; $j = 1,2,\ldots$,N2 and $k = 1,2,\ldots$,N3.

On exit: these residuals are overwritten by the corresponding components of the solution s of the system (2), i.e., the changes to be made to the vector T to reduce the residuals supplied.

16: WRKSP1(N1M,N2M,N3) — ***real*** array *Workspace*

17: WRKSP2(N1M,N2M,N3) — ***real*** array *Workspace*

18: WRKSP3(N1M,N2M,N3) — ***real*** array *Workspace*

19: IFAIL — INTEGER *Input/Output*

On entry: IFAIL must be set to 0, −1 or 1. For users not familiar with this parameter (described in Chapter P01) the recommended value is 0.

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

6 Error Indicators and Warnings

Errors detected by the routine:

IFAIL = 1

On entry, N1 < 2,
or N2 < 2,
or N3 < 2.

IFAIL = 2

On entry, N1M < N1,
or N2M < N2.

IFAIL = 3

On entry, APARAM $\leq$ 0.0.

IFAIL = 4

On entry, APARAM > $((\mathrm{N1}-1)^2 + (\mathrm{N2}-1)^2 + (\mathrm{N3}-1)^2)/3.0$.

7 Accuracy

The improvement in accuracy for each iteration, i.e., on each call, depends on the size of the system and on the condition of the up-date matrix characterised by the seven-diagonal coefficient arrays. The ultimate accuracy obtainable depends on the above factors and on the ***machine precision***. However, since the routine works with residuals and the up-date vector, the calling program can calculate the residuals from extended precision values of the function, source term and equation coefficients if greater accuracy is required. The rate of convergence obtained with the Strongly Implicit Procedure is not always smooth because of the cyclic use of nine acceleration parameters. The convergence may become slow with very large problems. The final accuracy obtained can be judged approximately from the rate of convergence determined from the changes to the dependent variable T and in particular the change on the last iteration.

8 Further Comments

The time taken by the routine is approximately proportional to N1 $\times$ N2 $\times$ N3 for each call.

When used with deferred or defect correction, the residual is calculated in the calling program from a different system of equations to those represented by the seven-point molecule coefficients used by the routine as the basis of the iterative up-date procedure. When using deferred correction the overall rate of convergence depends not only on the items detailed in Section 7 but also on the difference between the two coefficient matrices used.

Convergence may not always be obtained when the problem is very large and/or the coefficients of the equations have widely disparate values. The latter case may be associated with a nearly ill-conditioned matrix.

9 Example

To solve Laplace's equation in a rectangular box with a non-uniform grid spacing in the x, y, and z co-ordinate directions and with Dirichlet boundary conditions specifying the function on the surfaces of the box equal to

$$e^{(1.0+x)/y(n_2)} \times \cos(\sqrt{2}y/y(n_2)) \times e^{(-1.0-z)/y(n_2)}.$$

Note that this is the same problem as that solved in the example for D03ECF. The differences in the maximum residuals obtained at each iteration between the two test runs are explained by the fact that in D03ECF the residual at each node is normalised by dividing by the central coefficient, whereas this normalisation has not been used in the example program for D03UBF.

9.1 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.

```
*     D03UBF Example Program Text
*     Mark 19 Revised. NAG Copyright 1999.
*     .. Parameters ..
      INTEGER          N1, N2, N3, N1M, N2M, NITS
      PARAMETER        (N1=4,N2=5,N3=6,N1M=N1,N2M=N2,NITS=10)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Local Scalars ..
      real             ADEL, APARAM, ARES, DELMAX, DELMN, RESMAX, RESMN,
     +                 ROOT2
      INTEGER          I, IFAIL, IT, J, K
*     .. Local Arrays ..
      real             A(N1M,N2M,N3), B(N1M,N2M,N3), C(N1M,N2M,N3),
     +                 D(N1M,N2M,N3), E(N1M,N2M,N3), F(N1M,N2M,N3),
     +                 G(N1M,N2M,N3), Q(N1M,N2M,N3), R(N1M,N2M,N3),
     +                 T(N1M,N2M,N3), WRKSP1(N1M,N2M,N3),
     +                 WRKSP2(N1M,N2M,N3), WRKSP3(N1M,N2M,N3), X(N1),
     +                 Y(N2), Z(N3)
*     .. External Subroutines ..
      EXTERNAL         D03UBF
*     .. Intrinsic Functions ..
      INTRINSIC        ABS, COS, EXP, MAX, real, SQRT
*     .. Data statements ..
      DATA             X(1), X(2), X(3), X(4)/0.0e0, 1.0e0, 3.0e0,
     +                 6.0e0/
      DATA             Y(1), Y(2), Y(3), Y(4), Y(5)/0.0e0, 1.0e0, 3.0e0,
     +                 6.0e0, 10.0e0/
      DATA             Z(1), Z(2), Z(3), Z(4), Z(5), Z(6)/0.0e0, 1.0e0,
     +                 3.0e0, 6.0e0, 10.0e0, 15.0e0/
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D03UBF Example Program Results'
      WRITE (NOUT,*)
      ROOT2 = SQRT(2.0e0)
      APARAM = 1.0e0
*     Set up difference equation coefficients, source terms and
*     initial approximation
      DO 60 K = 1, N3
         DO 40 J = 1, N2
            DO 20 I = 1, N1
               IF ((I.NE.1) .AND. (I.NE.N1) .AND. (J.NE.1)
     +             .AND. (J.NE.N2) .AND. (K.NE.1) .AND. (K.NE.N3)) THEN
*                 Specification for internal nodes
                  A(I,J,K) = 2.0e0/((Z(K)-Z(K-1))*(Z(K+1)-Z(K-1)))
                  G(I,J,K) = 2.0e0/((Z(K+1)-Z(K))*(Z(K+1)-Z(K-1)))
                  B(I,J,K) = 2.0e0/((Y(J)-Y(J-1))*(Y(J+1)-Y(J-1)))
                  F(I,J,K) = 2.0e0/((Y(J+1)-Y(J))*(Y(J+1)-Y(J-1)))
                  C(I,J,K) = 2.0e0/((X(I)-X(I-1))*(X(I+1)-X(I-1)))
                  E(I,J,K) = 2.0e0/((X(I+1)-X(I))*(X(I+1)-X(I-1)))
                  D(I,J,K) = -A(I,J,K) - B(I,J,K) - C(I,J,K) - E(I,J,K)
     +                       - F(I,J,K) - G(I,J,K)
                  Q(I,J,K) = 0.0e0
                  T(I,J,K) = 0.0e0
               ELSE
*                 Specification for boundary nodes
```

```
                  A(I,J,K) = 0.0e0
                  B(I,J,K) = 0.0e0
                  C(I,J,K) = 0.0e0
                  E(I,J,K) = 0.0e0
                  F(I,J,K) = 0.0e0
                  G(I,J,K) = 0.0e0
                  D(I,J,K) = 0.0e0
                  Q(I,J,K) = EXP((X(I)+1.0e0)/Y(N2))*COS(ROOT2*Y(J)
     +                       /Y(N2))*EXP((-Z(K)-1.0e0)/Y(N2))
                  T(I,J,K) = 0.0e0
               END IF
   20       CONTINUE
   40    CONTINUE
   60 CONTINUE
*     Iterative loop
      WRITE (NOUT,*) 'Iteration     Residual                  Change'
      WRITE (NOUT,*)
     + '  No      Max.        Mean          Max.       Mean'
      WRITE (NOUT,*)
      DO 200 IT = 1, NITS
         RESMAX = 0.0e0
         RESMN = 0.0e0
         DO 120 K = 1, N3
            DO 100 J = 1, N2
               DO 80 I = 1, N1
                  IF (D(I,J,K).NE.0.0e0) THEN
*                    Seven point molecule formula
                     R(I,J,K) = Q(I,J,K) - A(I,J,K)*T(I,J,K-1) - B(I,J,
     +                          K)*T(I,J-1,K) - C(I,J,K)*T(I-1,J,K) -
     +                          D(I,J,K)*T(I,J,K) - E(I,J,K)*T(I+1,J,K)
     +                          - F(I,J,K)*T(I,J+1,K) - G(I,J,K)*T(I,J,
     +                          K+1)
                  ELSE
*                    Explicit equation
                     R(I,J,K) = Q(I,J,K) - T(I,J,K)
                  END IF
                  ARES = ABS(R(I,J,K))
                  RESMAX = MAX(RESMAX,ARES)
                  RESMN = RESMN + ARES
   80          CONTINUE
  100       CONTINUE
  120    CONTINUE
         RESMN = RESMN/(real(N1*N2*N3))
         IFAIL = 0
*
         CALL D03UBF(N1,N2,N3,N1M,N2M,A,B,C,D,E,F,G,APARAM,IT,R,WRKSP1,
     +               WRKSP2,WRKSP3,IFAIL)
*
*        Update the dependent variable
         DELMAX = 0.0e0
         DELMN = 0.0e0
         DO 180 K = 1, N3
            DO 160 J = 1, N2
               DO 140 I = 1, N1
                  T(I,J,K) = T(I,J,K) + R(I,J,K)
                  ADEL = ABS(R(I,J,K))
                  DELMAX = MAX(DELMAX,ADEL)
                  DELMN = DELMN + ADEL
```

```
  140           CONTINUE
  160         CONTINUE
  180       CONTINUE
            DELMN = DELMN/(real(N1*N2*N3))
            WRITE (NOUT,99999) IT, RESMAX, RESMN, DELMAX, DELMN
*           Convergence tests here if required
  200 CONTINUE
*     End of iterative loop
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Table of calculated function values'
      WRITE (NOUT,*)
     +  'K  J   (I       T    ) (I       T    ) (I       T    ) (I       T    )'
      WRITE (NOUT,*)
      WRITE (NOUT,99998) ((K,J,(I,T(I,J,K),I=1,N1),J=1,N2),K=1,N3)
      STOP
*
99999 FORMAT (1X,I5,4(2X,e11.4))
99998 FORMAT ((1X,I1,I3,1X,4(1X,I3,2X,F8.3)))
      END
```

9.2 Program Data

None.

9.3 Program Results

```
D03UBF Example Program Results

Iteration      Residual                        Change
   No       Max.          Mean            Max.          Mean

    1   0.1822E+01   0.4847E+00   0.1822E+01   0.6173E+00
    2   0.8585E-02   0.9369E-03   0.1970E-01   0.1895E-02
    3   0.3168E-02   0.7783E-04   0.1496E-02   0.5819E-04
    4   0.4085E-04   0.2179E-05   0.3848E-04   0.1931E-05
    5   0.7820E-05   0.3999E-06   0.5481E-05   0.2312E-06
    6   0.2246E-06   0.1524E-07   0.2333E-06   0.1093E-07
    7   0.2219E-07   0.1669E-08   0.2222E-07   0.9131E-09
    8   0.2841E-08   0.1820E-09   0.1969E-08   0.9337E-10
    9   0.6696E-09   0.4762E-10   0.5873E-09   0.2450E-10
   10   0.7848E-10   0.4908E-11   0.5863E-10   0.2671E-11

Table of calculated function values
K  J   (I       T    ) (I       T    ) (I       T    ) (I       T    )

1  1    1     1.000    2     1.105    3     1.350    4     1.822
1  2    1     0.990    2     1.094    3     1.336    4     1.804
1  3    1     0.911    2     1.007    3     1.230    4     1.661
1  4    1     0.661    2     0.731    3     0.892    4     1.205
1  5    1     0.156    2     0.172    3     0.211    4     0.284
2  1    1     0.905    2     1.000    3     1.221    4     1.649
2  2    1     0.896    2     0.990    3     1.210    4     1.632
2  3    1     0.825    2     0.912    3     1.114    4     1.503
2  4    1     0.598    2     0.662    3     0.809    4     1.090
2  5    1     0.141    2     0.156    3     0.190    4     0.257
3  1    1     0.741    2     0.819    3     1.000    4     1.350
3  2    1     0.733    2     0.811    3     0.991    4     1.336
3  3    1     0.675    2     0.747    3     0.913    4     1.230
```

3	4	1	0.490	2	0.543	3	0.664	4	0.892
3	5	1	0.116	2	0.128	3	0.156	4	0.211
4	1	1	0.549	2	0.607	3	0.741	4	1.000
4	2	1	0.543	2	0.601	3	0.734	4	0.990
4	3	1	0.500	2	0.554	3	0.677	4	0.911
4	4	1	0.363	2	0.402	3	0.492	4	0.661
4	5	1	0.086	2	0.095	3	0.116	4	0.156
5	1	1	0.368	2	0.407	3	0.497	4	0.670
5	2	1	0.364	2	0.403	3	0.492	4	0.664
5	3	1	0.335	2	0.371	3	0.454	4	0.611
5	4	1	0.243	2	0.270	3	0.330	4	0.443
5	5	1	0.057	2	0.063	3	0.077	4	0.105
6	1	1	0.223	2	0.247	3	0.301	4	0.407
6	2	1	0.221	2	0.244	3	0.298	4	0.403
6	3	1	0.203	2	0.225	3	0.274	4	0.371
6	4	1	0.148	2	0.163	3	0.199	4	0.269
6	5	1	0.035	2	0.038	3	0.047	4	0.063