NAG Fortran Library Routine Document

D02KDF

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

D02KDF finds a specified eigenvalue of a regular or singular second-order Sturm–Liouville system on a finite or infinite interval, using a Pruefer transformation and a shooting method. Provision is made for discontinuities in the coefficient functions or their derivatives.

2 Specification

```
 SUBROUTINE D02KDF(XPOINT, M, COEFFN, BDYVAL, K, TOL, ELAM, DELAM, HMAX,
1                  MAXIT, MAXFUN, MONIT, IFAIL)
 INTEGER           M, K, MAXIT, MAXFUN, IFAIL
 real              XPOINT(M), TOL, ELAM, DELAM, HMAX(2,M)
 EXTERNAL          COEFFN, BDYVAL, MONIT
```

3 Description

D02KDF finds a specified eigenvalue $\tilde{\lambda}$ of a Sturm–Liouville system defined by a self-adjoint differential equation of the second-order

$$(p(x)y')' + q(x;\lambda)y = 0, \quad a < x < b,$$

together with appropriate boundary conditions at the two, finite or infinite, end-points a and b. The functions p and q, which are real-valued, must be defined by a subroutine COEFFN. The boundary conditions must be defined by a subroutine BDYVAL, and in the case of a singularity at a or b take the form of an asymptotic formula for the solution near the relevant end-point.

For the theoretical basis of the numerical method to be valid, the following conditions should hold on the coefficient functions:

(a) $p(x)$ must be non-zero and of one sign throughout the interval (a,b); and

(b) $\frac{\partial q}{\partial \lambda}$ must be of one sign throughout (a,b) for all relevant values of λ, and must not be identically zero as x varies for any λ.

Points of discontinuity in the functions p and q or their derivatives are allowed, and should be included as 'break-points' in the array XPOINT.

The eigenvalue $\tilde{\lambda}$ is determined by a shooting method based on the Scaled Pruefer form of the differential equation as described in Pryce (1981), with certain modifications. The Pruefer equations are integrated by a special internal routine using Merson's Runge–Kutta formula with automatic control of local error. Providing certain assumptions (see Section 8.1) are met, the computed value of $\tilde{\lambda}$ will have a mixed absolute/relative error, estimated by the user-supplied value TOL.

A good account of the theory of Sturm–Liouville systems, with some description of Pruefer transformations, is given in Chapter X of Birkhoff and Rota (1962). An introduction to the user of Pruefer transformations for the numerical solution of eigenvalue problems arising from physics and chemistry is in Bailey (1966).

The scaled Pruefer method is fairly recent, and is described in a short note in Pryce and Hargrave (1977) and in Pryce (1981).

4 References

Abramowitz M and Stegun I A (1972) *Handbook of Mathematical Functions* (3rd Edition) Dover Publications

Bailey P B (1966) Sturm–Liouville eigenvalues via a phase function *SIAM J. Appl. Math.* **14** 242–249

Banks D O and Kurowski I (1968) Computation of eigenvalues of singular Sturm–Liouville Systems *Math. Comput.* **22** 304–310

Birkhoff G and Rota G C (1962) *Ordinary Differential Equations* Ginn & Co., Boston and New York

Pryce J D (1981) Two codes for Sturm–Liouville problems *Technical Report CS-81-01* Department of Computer Science, Bristol University

Pryce J D and Hargrave B A (1977) The scaled Prüfer method for one-parameter and multi-parameter eigenvalue problems in ODEs *IMA Numerical Analysis Newsletter* **1 (3)**

5 Parameters

1: XPOINT(M) – ***real*** array *Input*

On entry: the points where the boundary conditions computed by BDYVAL are to be imposed, and also any break-points, i.e., XPOINT(1) to XPOINT(m) must contain values $x_1, \ldots, x_m$ such that

$$x_1 \leq x_2 < x_3 < \cdots < x_{m-1} \leq x_m$$

with the following meanings:

(a) x_1 and x_m are left and right end-points, a and b, of the domain of definition of the Sturm–Liouville system if these are finite. If either of a or b is infinite, the corresponding value x_1 or x_m may be a more-or-less arbitrary 'large' number of appropriate sign.

(b) x_2 and x_{m-1} are the Boundary Matching Points (BMP), that is the points at which the left and right boundary conditions computed in BDYVAL are imposed.

If the left-hand end-point is a regular point then the user should set $x_2 = x_1$ $(= a)$, while if it is a singular point the user must set $x_2 > x_1$. Similarly $x_{m-1} = x_m$ $(= b)$ if the right-hand end-point is regular, and $x_{m-1} < x_m$ if it is singular.

(c) The remaining $m - 4$ points $x_3, \ldots, x_{m-2}$, if any, define 'break-points' which divide the interval $[x_2, x_{m-1}]$ into $m - 3$ sub-intervals

$$i_1 = [x_2, x_3], \ldots, i_{m-3} = [x_{m-2}, x_{m-1}].$$

Numerical integration of the differential equation is stopped and restarted at each break-point. In simple cases no break-points are needed. However, if $p(x)$ or $q(x;\lambda)$ are given by different formulae in different parts of the interval, then integration is more efficient if the range is broken up by break-points in the appropriate way. Similarly points where any jumps occur in $p(x)$ or $q(x;\lambda)$, or in their derivatives up to the fifth-order, should appear as break-points.

Examples are given in Section 8 and Section 9. XPOINT determines the position of the Shooting Matching Point (SMP), as explained in Section 8.3.

Constraint: XPOINT$(1) \leq$ XPOINT$(2) < \cdots <$ XPOINT$(\mathrm{M}-1) \leq$ XPOINT(M).

2: M – INTEGER *Input*

On entry: the number of points in the array XPOINT.

Constraint: M ≥ 4.

3: COEFFN – SUBROUTINE, supplied by the user. *External Procedure*

COEFFN must compute the values of the coefficient functions $p(x)$ and $q(x;\lambda)$ for given values of x and λ. Section 3 states conditions which p and q must satisfy.

Its specification is:

```
SUBROUTINE COEFFN(P, Q, DQDL, X, ELAM, JINT)
INTEGER         JINT
real            P, Q, DQDL, X, ELAM
```

1: P – ***real*** *Output*

On exit: the value of $p(x)$ for the current value of x.

2: Q – ***real*** *Output*

On exit: the value of $q(x;\lambda)$ for the current value of x and the current trial value of λ.

3: DQDL – ***real*** *Output*

On exit: the value of $\frac{\partial q}{\partial \lambda}$ for the current value of x and the current trial value of λ. However DQDL is only used in error estimation and an approximation (say to within 20 per cent) will suffice.

4: X – ***real*** *Input*

On entry: the current value of x.

5: ELAM – ***real*** *Input*

On entry: the current trial value of the eigenvalue parameter λ.

6: JINT – INTEGER *Input*

On entry: the index j of the sub-interval i_j (see specification of XPOINT) in which x lies.

COEFFN must be declared as EXTERNAL in the (sub)program from which D02KDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

See Section 8.4 and Section 9 for examples.

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

BDYVAL must define the boundary conditions. For each end-point, BDYVAL must return (in YL or YR) values of $y(x)$ and $p(x)y'(x)$ which are consistent with the boundary conditions at the end-points; only the ratio of the values matters. Here x is a given point (XL or XR) equal to, or close to, the end-point.

For a **regular** end-point (a, say), $x = a$, a boundary condition of the form

$$c_1 y(a) + c_2 y'(a) = 0$$

can be handled by returning constant values in YL, e.g., $\mathrm{YL}(1) = c_2$ and $\mathrm{YL}(2) = -c_1 p(a)$.

For a **singular** end-point however, YL(1) and YL(2) will in general be functions of XL and ELAM, and YR(1) and YR(2) functions of XR and ELAM, usually derived analytically from a power-series or asymptotic expansion. Examples are given in Section 8.5 and Section 9.

Its specification is:

```
SUBROUTINE BDYVAL(XL, XR, ELAM, YL, YR)
real            XL, XR, ELAM, YL(3), YR(3)
```

1: XL – ***real*** *Input*

On entry: if a is a regular end-point of the system (so that $a = x_1 = x_2$), then XL contains a. If a is a singular point (so that $a \le x_1 < x_2$), then XL contains a point x such that $x_1 < x \le x_2$.

2: XR – ***real*** *Input*

On entry: if b is a regular end-point of the system (so that $x_{m-1} = x_m = b$), then XR contains b. If b is a singular point (so that $x_{m-1} < x_m \leq b$), then XR contains a point x such that $x_{m-1} \leq x < x_m$.

3: ELAM – ***real*** *Input*

On entry: the current trial value of λ.

4: YL(3) – ***real*** array *Output*

On exit: YL(1) and YL(2) should contain values of $y(x)$ and $p(x)y'(x)$ respectively (not both zero) which are consistent with the boundary condition at the left-hand end-point, given by x = XL. YL(3) should not be set.

5: YR(3) – ***real*** array *Output*

On exit: YR(1) and YR(2) should contain values of $y(x)$ and $p(x)y'(x)$ respectively (not both zero) which are consistent with the boundary condition at the right-hand end-point, given by x = XR. YR(3) should not be set.

BDYVAL must be declared as EXTERNAL in the (sub)program from which D02KDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: K – INTEGER *Input*

On entry: the index k of the required eigenvalue when the eigenvalues are ordered

$$\lambda_0 < \lambda_1 < \lambda_2 < \cdots < \lambda_k < \cdots$$

Constraint: K ≥ 0.

6: TOL – ***real*** *Input*

On entry: the tolerance parameter which determines the accuracy of the computed eigenvalue. The error estimate held in DELAM on exit satisfies the mixed absolute/relative error test

$$\text{DELAM} \leq \text{TOL} \times \max(1.0, |\text{ELAM}|) \quad (1)$$

where ELAM is the final estimate of the eigenvalue. DELAM is usually somewhat smaller than the right-hand side of (1) but not several orders of magnitude smaller.

Constraint: TOL > 0.0.

7: ELAM – ***real*** *Input/Output*

On entry: an initial estimate of the eigenvalue $\tilde{\lambda}$.

On exit: the final computed estimate, whether or not an error occurred.

8: DELAM – ***real*** *Input/Output*

On entry: an indication of the scale of the problem in the λ-direction. DELAM holds the initial 'search step' (positive or negative). Its value is not critical but the first two trial evaluations are made at ELAM and ELAM+ EDELAM, so the routine will work most efficiently if the eigenvalue lies between these values. A reasonable choice (if a closer bound is not known) is half the distance between adjacent eigenvalues in the neighbourhood of the one sought. In practice, there will often be a problem, similar to the one in hand but with known eigenvalues, which will help one to choose initial values for ELAM and DELAM.

If DELAM $= 0.0$ on entry, it is given the default value of $0.25 \times \max(1.0, |\text{ELAM}|)$.

On exit: with IFAIL=0, DELAM holds an estimate of the absolute error in the computed eigenvalue, that is $|\tilde{\lambda} - \text{ELAM}| \approx \text{DELAM}$ (In Section 8.2 we discuss the assumptions under which this is true.) The true error is rarely more than twice, or less than a tenth, of the estimated error.

With IFAIL $\neq 0$, DELAM may hold an estimate of the error, or its initial value, depending on the value of IFAIL. See Section 6 for further details.

9: HMAX(2,M) – ***real*** array *Input/Output*

On entry: HMAX$(1,j)$ should contain a maximum step size to be used by the differential equation code in the jth sub-interval i_j (as described in the specification of parameter XPOINT) for $j = 1, 2, \ldots, m - 3$. If it is zero the routine generates a maximum step size internally.

It is recommended that HMAX$(1,j)$ be set to zero unless the coefficient functions p and q have features (such as a narrow peak) within the jth sub-interval that could be 'missed' if a long step were taken. In such a case HMAX$(1,j)$ should be set to about half the distance over which the feature should be observed. Too small a value will increase the computing time for the routine. See Section 8 for further suggestions.

The rest of the array is used as workspace.

On exit: HMAX$(1, m-1)$ and HMAX$(1, m)$ contain the sensitivity coefficients σ_l, σ_r, described in Section 8.6. Other entries also contain diagnostic output in case of an error exit (see Section 6 for details).

10: MAXIT – INTEGER *Input/Output*

On entry: a bound on n_r, the number of rootfinding iterations allowed, that is the number of trial values of λ that are used; if MAXIT ≤ 0, no such bound is assumed.

Suggested value: MAXIT $= 0$. (See also under MAXFUN.)

On exit: MAXIT will have been decreased by the number of iterations actually performed, whether or not it was positive on entry.

11: MAXFUN – INTEGER *Input*

On entry: a bound on n_f, the number of calls to COEFFN made in any one rootfinding iteration. If MAXFUN ≤ 0, no such bound is assumed.

Suggested value: MAXFUN $= 0$.

MAXFUN and MAXIT may be used to limit the computational cost of a call to D02KDF, which is roughly proportional to $n_r \times n_f$.

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

MONIT is called by D02KDF at the end of each rootfinding iteration and allows the user to monitor the course of the computation by printing out the parameters (see Section 9 for an example).

If no monitoring is required, the dummy subroutine D02KAY may be used. (D02KAY is included in the NAG Fortran Library. In some implementations of the Library the name is changed to KAYD02: refer to the Users' Note for your implementation.)

Its specification is:

```
SUBROUTINE MONIT(MAXIT, IFLAG, ELAM, FINFO)
INTEGER          MAXIT, IFLAG
real             ELAM, FINFO(15)
```

1: MAXIT – INTEGER *Input*

On entry: the current value of the parameter MAXIT of D02KDF, which is decreased by one at each iteration.

2: IFLAG – INTEGER *Input*

On entry: IFLAG describes what phase the computation is in.

IFLAG < 0

An error occurred in the computation of the 'miss-distance' at this iteration; an error exit from D02KDF with IFAIL $= -$IFLAG will follow.

IFLAG $= 1$

The routine is trying to bracket the eigenvalue $\tilde{\lambda}$.

IFLAG $= 2$

The routine is converging to the eigenvalue $\tilde{\lambda}$ (having already bracketed it).

3: ELAM – ***real*** *Input*

On entry: the current trial value of λ.

4: FINFO(15) – ***real*** array *Input*

On entry: information about the behaviour of the shooting method, and diagnostic information in the case of errors. It should not normally be printed in full if no error has occurred (that is, if IFLAG > 0), though the first few components may be of interest to the user. In case of an error (IFLAG < 0) all the components of FINFO should be printed.

The contents of FINFO are as follows:

FINFO(1), the current value of the 'miss-distance' or 'residual' function $f(\lambda)$ on which the shooting method is based. FINFO(1) is set to zero if FLAG < 0.

FINFO(2), an estimate of the quantity $\delta\lambda$ defined as follows. Consider the perturbation in the miss-distance $f(\lambda)$ that would result if the local error, in the solution of the differential equation, were always positive and equal to its maximum permitted value. Then $\delta\lambda$ is the perturbation in λ that would have the same effect on $f(\lambda)$. Thus, at the zero of $f(\lambda)$, $|\delta\lambda|$ is an approximate bound on the perturbation of the zero (that is the eigenvalue) caused by errors in numerical solution. If $\delta\lambda$ is very large then it is possible that there has been a programming error in COEFFN such that q is independent of λ. If this is the case, an error exit with IFAIL=5 should follow. FINFO(2) is set to zero if IFLAG < 0.

FINFO(3), the number of internal iterations, using the same value of λ and tighter accuracy tolerances, needed to bring the accuracy (that is the value of $\delta\lambda$) to an acceptable value. Its value should normally be 1.0, and should almost never exceed 2.0.

FINFO(4), the number of calls to COEFFN at this iteration.

FINFO(5), the number of successful steps taken by the internal differential equation solver at this iteration.

FINFO(6), the number of unsuccessful steps used by the internal integrator at this iteration.

FINFO(7), the number of successful steps at the maximum step size taken by the internal integrator at this iteration.

FINFO(8), is not used.

FINFO(9) to FINFO(15), set to zero, unless IFLAG < 0 in which case they hold the following values describing the point of failure:

FINFO(9), the index of the sub-interval where failure occurred, in the range 1 to $m-3$. In case of an error in BDYVAL, it is set to 0 or $m-2$ depending on whether the left or right boundary condition caused the error.

FINFO(10), the value of the independent variable x, the point at which the error occurred. In case of an error in BDYVAL, it is set to the value of XL or XR as appropriate (see the specification of BDYVAL).

FINFO(11), FINFO(12), FINFO(13), the current value of the Pruefer dependent variables β, ϕ and ρ respectively. These are set to zero in case of an error in BDYVAL. (See D02KEF for a description of these variables.)

FINFO(14), the local-error tolerance being used by the internal integrator at the point of failure. This is set to zero in the case of an error in BDYVAL.

FINFO(15), the last integration mesh point. This is set to zero in the case of an error in BDYVAL.

MONIT must be declared as EXTERNAL in the (sub)program from which D02KDF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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 (see Section 6).

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, for users not familiar with this parameter the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

6 Error Indicators and Warnings

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

Errors or warnings detected by the routine:

IFAIL = 1

A parameter error. All parameters (except IFAIL) are left unchanged. The reason for the error is shown by the value of HMAX(2,1) as follows:

HMAX(2,1) = 1: M < 4;

HMAX(2,1) = 2: K < 0;

HMAX(2,1) = 3: TOL $\leq$ 0.0;

HMAX(2,1) = 4: XPOINT(1) to XPOINT(m) are not in ascending order. HMAX(2,2) gives the position i in XPOINT where this was detected.

IFAIL = 2

At some call to BDYVAL, invalid values were returned, that is, either $\mathrm{YL}(1) = \mathrm{YL}(2) = 0.0$, or $\mathrm{YR}(1) = \mathrm{YR}(2) = 0.0$ (a programming error in BDYVAL). See the last call of MONIT for details.

This error exit will also occur if $p(x)$ is zero at the point where the boundary condition is imposed. Probably BDYVAL was called with XL equal to a singular end-point a or with XR equal to a singular end-point b.

IFAIL = 3

At some point between XL and XR the value of $p(x)$ computed by COEFFN became zero or changed sign. See the last call of MONIT for details.

IFAIL = 4

MAXIT > 0 on entry, and after MAXIT iterations the eigenvalue had not been found to the required accuracy.

IFAIL $= 5$

The 'bracketing' phase (with parameter IFLAG of MONIT equal to 1) failed to bracket the eigenvalue within ten iterations. This is caused by an error in formulating the problem (for example, q is independent of λ), or by very poor initial estimates of ELAM, DELAM.

On exit, ELAM and ELAM + DELAM give the end-points of the interval within which no eigenvalue was located by the routine.

IFAIL $= 6$

MAXFUN > 0 on entry, and the last iteration was terminated because more than MAXFUN calls to COEFFN were used. See the last call of MONIT for details.

IFAIL $= 7$

To obtain the desired accuracy the local error tolerance was set so small at the start of some sub-interval that the differential equation solver could not choose an initial step size large enough to make significant progress. See the last call of MONIT for diagnostics.

IFAIL $= 8$

At some point inside a sub-interval the step size in the differential equation solver was reduced to a value too small to make significant progress (for the same reasons as with IFAIL=7). This could be due to pathological behaviour of $p(x)$ and $q(x;\lambda)$ or to an unreasonable accuracy requirement or to the current value of λ making the equations 'stiff'. See the last call of MONIT for details.

IFAIL $= 9$

TOL is too small for the problem being solved and the ***machine precision*** being used. The final value of ELAM should be a very good approximation to the eigenvalue.

IFAIL $= 10$

C05AZF, called by D02KDF, has terminated with the error exit corresponding to a pole of the residual function $f(\lambda)$. This error exit should not occur, but if it does, try solving the problem again with a smaller TOL.

IFAIL $= 11$ (D02KDY)
IFAIL $= 12$ (C05AZF)

A serious error has occurred in the specified routine. Check all subroutine calls and array dimensions. Seek expert help.

HMAX(2,1) holds the failure exit number from the routine where the failure occurred. In the case of a failure in C05AZF, HMAX(2,2) holds the value of parameter IND of C05AZF.

Note: error exits with IFAIL $= 2$, 3, 6, 7, 8, 11 are caused by being unable to set up or solve the differential equation at some iteration, and will be immediately preceded by a call of MONIT giving diagnostic information. For other errors, diagnostic information is contained in HMAX$(2,j)$, for $j = 1,2,\ldots,m$, where appropriate.

7 Accuracy

See the discussion in Section 8.2.

8 Further Comments

8.1 Timing

This depends on the complexity of the coefficient functions, whether they or their derivatives are rapidly changing, the tolerance demanded, and how many iterations are needed to obtain convergence. The amount of work per iteration is roughly doubled when TOL is divided by 16. To make economical use of the routine, one should try to obtain good initial values for ELAM and DELAM, and where appropriate good asymptotic formulae. Also the boundary matching points should not be set unnecessarily close to singular points.

8.2 General Description of the Algorithm

A shooting method, for differential equation problems containing unknown parameters, relies on the construction of a 'miss-distance function', which for given trial values of the parameters measures how far the conditions of the problem are from being met. The problem is then reduced to one of finding the values of the parameters for which the miss-distance function is zero, that is to a root-finding process. Shooting methods differ mainly in how the miss-distance is defined.

This routine defines a miss-distance $f(\lambda)$ based on the rotation round the origin of the point $\mathrm{P}(x) = (p(x)y'(x), y(x))$ in the Phase Plane as the solution proceeds from a to b. The **boundary conditions** define the ray (i.e., two-sided line through the origin) on which $p(x)$ should start, and the ray on which it should finish. The **eigenvalue index** k defines the total number of half-turns it should make. Numerical solution is actually done by 'shooting forward' from $x = a$ and 'shooting backward' from $x = b$ to a matching point $x = c$. Then $f(\lambda)$ is taken as the angle between the rays to the two resulting points $P_a(c)$ and $P_b(c)$. A relative scaling of the py' and y axes, based on the behaviour of the coefficient functions p and q, is used to improve the numerical behaviour.

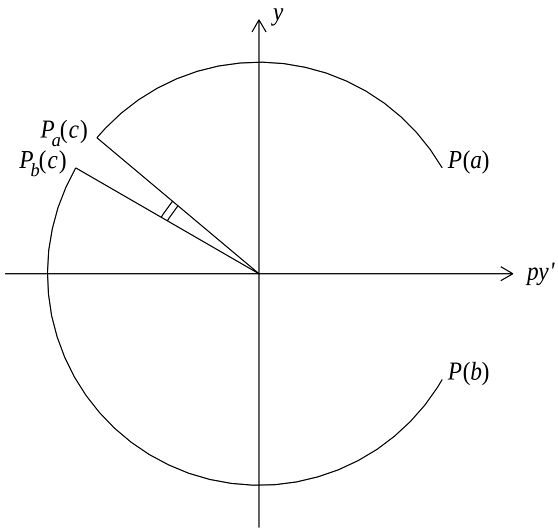

Figure 1

The resulting function $f(\lambda)$ is monotonic over $-\infty < \lambda < \infty$, increasing if $\frac{\partial q}{\partial \lambda} > 0$ and decreasing if $\frac{\partial q}{\partial \lambda} < 0$, with a unique zero at the desired eigenvalue $\tilde{\lambda}$. The routine measures $f(\lambda)$ in units of a half-turn. This means that as λ increases, $f(\lambda)$ varies by about 1 as each eigenvalue is passed. (This feature implies that the values of $f(\lambda)$ at successive iterations – especially in the early stages of the iterative process – can be used with suitable extrapolation or interpolation to help the choice of initial estimates for eigenvalues near to the one currently being found.)

The routine actually computes a value for $f(\lambda)$ with errors, arising from the local errors of the differential equation code and from the asymptotic formulae provided by the user if singular points are involved. However, the error estimate output in DELAM is usually fairly realistic, in that the actual error $|\tilde{\lambda} - \mathrm{ELAM}|$ is within an order of magnitude of DELAM.

8.3 The Position of the Shooting Matching Point c

This point is always one of the values x_i in array XPOINT. It is chosen to be the value of that x_i, $2 \leq i \leq m-1$, that lies closest to the middle of the interval $[x_2, x_{m-1}]$. If there is a tie, the rightmost candidate is chosen. In particular if there are no break-points, then $c = x_{m-1}$ $(= x_3)$; that is, the shooting is from left to right in this case. A break-point may be inserted purely to move c to an interior point of the interval, even though the form of the equations does not require it. This often speeds up convergence especially with singular problems.

8.4 Examples of Coding the COEFFN Routine

Coding COEFFN is straightforward except when break-points are needed. The examples below show:

(a) a simple case,

(b) a case where discontinuities in the coefficient functions or their derivatives necessitate break-points, and

(c) a case where break-points together with the HMAX parameter are an efficient way to deal with a coefficient function that is well-behaved except over one short interval.

(Some of these cases are among the examples in Section 9.)

Example A

The modified Bessel equation

$$x(xy')' + (\lambda x^2 - \nu^2)y = 0.$$

Assuming the interval of solution does not contain the origin and dividing through by x, we have $p(x) = x, q(x; \lambda) = \lambda x - \nu^2/x$. The code for COEFFN could be:

```
SUBROUTINE COEFFN (P, Q, DQDL, X, ELAM, JINT)
...
P = X
Q = ELAM*X - NU*NU/X
DQDL = X
RETURN
END
```

where NU (standing for ν) is a ***real*** variable that might be defined in a DATA statement, or might be in user-declared COMMON so that its value could be set in the main program.

Example B

A Schroedinger equation

$$y'' + (\lambda + q(x))y = 0,$$

where

$$q(x) = \begin{cases} x^2 - 10 & (|x| \leq 4) \\ \dfrac{6}{|x|} & (|x| > 4) \end{cases}$$

over some interval 'approximating to $(-\infty, \infty)$', say $[-20, 20]$. Here we need break-points at ± 4, forming three sub-intervals $i_1 = [-20, -4]$, $i_2 = [-4, 4]$, $i_3 = [4, 20]$. The code for COEFFN could be:

```
SUBROUTINE COEFFN (P, Q, DQDL, X, ELAM, JINT)
...
IF (JINT.EQ.2) THEN
  Q = ELAM + X*X - 10.0E0
ELSE
  Q = ELAM + 6.0E0/ABS(X)
ENDIF
P = 1.0E0
DQDL = 1.0
RETURN
END
```

The array XPOINT would contain the values x_1, -20.0, -4.0, $+4.0$, $+20.0$, x_6 and m would be 6. The choice of appropriate values for x_1 and x_6 depends on the form of the asymptotic formula computed by BDYVAL and the technique is discussed in the next sub-section.

Example C

$$y'' + \lambda(1 - 2e^{-100x^2})y = 0, \quad -10 \leq x \leq 10.$$

Here $q(x;\lambda)$ is nearly constant over the range except for a sharp inverted spike over approximately $-0.1 \leq x \leq 0.1$. There is a danger that the routine will build up to a large step size and 'step over' the spike without noticing it. By using break-points – say ± 0.5 – one can restrict the step size near the spike without impairing the efficiency elsewhere.

The code for COEFFN could be:

```
SUBROUTINE COEFFN (P, Q, DQDL, X, ELAM, JINT)
...
P = 1.0
DQDL = 1.0 - 2.0 * EXP(-100.0*X*X)
Q = ELAM * DQDL
RETURN
END
```

XPOINT might contain -10.0, -10.0, -0.5, 0.5, 10.0, 10.0 (assuming ± 10, are regular points) and m would be 6. HMAX$(1, j)$, $j = 1, 2, 3$ might contain 0.0, 0.1 and 0.0.

8.5 Examples of Boundary Conditions at Singular Points

Quoting from page 243 of Bailey (1966): 'Usually ... the differential equation has two essentially different types of solutions near a singular point, and the boundary condition there merely serves to distinguish one kind from the other. This is the case in all the standard examples of mathematical physics'.

In most cases the behaviour of the ratio $p(x)y'/y$ near the point is quite different for the two types of solution. Essentially what the user provides through the BDYVAL routine is an approximation to this ratio, valid as x tends to the singular point (SP).

The user must decide (a) how accurate to make this approximation or asymptotic formula, for example how many terms of a series to use, and (b) where to place the boundary matching point (BMP) at which the numerical solution of the differential equation takes over from the asymptotic formula. Taking the BMP closer to the SP will generally improve the accuracy of the asymptotic formula, but will make the computation more expensive as the Pruefer differential equations generally become progressively more ill-behaved as the SP is approached. The user is strongly recommended to experiment with placing the BMPs. In many singular problems quite crude asymptotic formulae will do. To help the user avoid needlessly accurate formulae, D02KDF outputs two 'sensitivity coefficients' σ_l, σ_r which estimate how much the errors at the BMPs affect the computed eigenvalue. They are described in detail below, see Section 8.6.

Example of coding BDYVAL:

The example below illustrates typical situations:

$$y'' + \left(\lambda - x - \frac{2}{x^2}\right)y = 0, \quad \text{on} \quad 0 < x < \infty$$

the boundary conditions being that y should remain bounded as x tends to 0 and x tends to ∞.

At the end $x = 0$ there is one solution that behaves like x^2 and another that behaves like x^{-1}. For the first of these solutions $p(x)y'/y$ is asymptotically $2/x$ while for the second it is asymptotically $-1/x$. Thus the desired ratio is specified by setting

$$\text{YL}(1) = x \quad \text{and} \quad \text{YL}(2) = 2.0.$$

At the end $x = \infty$ the equation behaves like Airy's equation shifted through λ, i.e., like $y'' - ty = 0$ where $t = x - \lambda$, so again there are two types of solutions. The solution we require behaves as

$$\exp(-\frac{2}{3}t^{\frac{3}{2}})/\sqrt[4]{t}$$

and the other as

$$\exp(+\frac{2}{3}t^{\frac{3}{2}})/\sqrt[4]{t}.$$

Hence, the desired solution has $p(x)y'/y \sim -\sqrt{t}$ so that we could set $\mathrm{YL}(1) = 1.0$ and $\mathrm{YL}(2) = -\sqrt{x-\lambda}$. The complete subroutine might thus be

```
SUBROUTINE BDYVAL (XL, XR, ELAM, YL, YR)
real XL, XR, ELAM, YL(3), YR(3)
YL(1) = XL
YL(2) = 2.0
YR(1) = 1.0
YR(2) = -SQRT(XR-ELAM)
RETURN
END
```

Clearly for this problem it is essential that any value given by D02KDF to XR is well to the right of the value of ELAM, so that the user must vary the right-hand BMP with the eigenvalue index k. One would expect λ_k to be near the kth zero of the Airy function $\mathrm{Ai}(x)$, so there is no problem estimating ELAM.

More accurate asymptotic formulae are easily found: near $x = 0$ by the standard Frobenius method, and near $x = \infty$ by using standard asymptotics for $\mathrm{Ai}(x)$, $\mathrm{Ai}'(x)$, e.g., see page 448 of Abramowitz and Stegun (1972).

For example by the Frobenius method the solution near x = 0 has the expansion

$$y = x^2(c_0 + c_1x + c_2x^2 + \ldots)$$

with

$$c_0 = 1,\ c_1 = 0,\ c_2 = \frac{-\lambda}{10},\ c_3 = \tfrac{1}{18},\ \ldots,\ c_n = \frac{c_{n-3} - \lambda c_{n-2}}{n(n+3)}.$$

This yields

$$\frac{p(x)y'}{y} = \frac{2 - \frac{2}{5}\lambda x^2 + \cdots}{x\left(1 - \frac{\lambda}{10}x^2 + \cdots\right)}.$$

8.6 The Sensitivity Parameters \sigma_{l} and \sigma_{r}

The sensitivity parameters σ_l, σ_r (held in $\mathrm{HMAX}(1, m-1)$ and $\mathrm{HMAX}(1, m)$ on output) estimate the effect of errors in the boundary conditions. For sufficiently small errors Δy, $\Delta py'$ in y and py' respectively, the relations

$$\Delta\lambda \simeq (y.\Delta py' - py'.\Delta y)_l\sigma_l$$
$$\Delta\lambda \simeq (y.\Delta py' - py'.\Delta y)_r\sigma_r$$

are satisfied, where the subscripts l, r denote errors committed at the left- and right-hand BMPs respectively, and $\Delta\lambda$ denotes the consequent error in the computed eigenvalue.

8.7 Missed Zeros

This is a pitfall to beware of at a singular point. If the BMP is chosen so far from the SP that a zero of the desired eigenfunction lies in between them, then the routine will fail to 'notice' this zero. Since the index of k of an eigenvalue is the number of zeros of its eigenfunction, the result will be that

(a) the wrong eigenvalue will be computed for the given index k – in fact some $\lambda_{k+k'}$ will be found where $k' \geq 1$;

(b) the same index k can cause convergence to any of several eigenvalues depending on the initial values of ELAM and DELAM.

It is up to the user to take suitable precautions – for instance by varying the position of the BMPs in the light of knowledge of the asymptotic behaviour of the eigenfunction at different eigenvalues.

9 Example

We find the 11th eigenvalue of the example of Section 8.5 of the documents for D02KDF, using the simple asymptotic formulae for the boundary conditions. The results exhibit slow convergence, mainly because XPOINT is set so that the shooting matching point c is at the right-hand end $x = 30.0$. The example results for D02KEF show that much faster convergence is obtained if XPOINT is set to contain an additional break-point at or near the maximum of the coefficient function $q(x;\lambda)$, which in this case is at $x = \sqrt[3]{4}$.

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.

```
*     D02KDF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
      INTEGER          M
      PARAMETER        (M=4)
*     .. Local Scalars ..
      real             DELAM, ELAM, TOL
      INTEGER          IFAIL, IFLAG, K, MAXIT
*     .. Local Arrays ..
      real             HMAX(2,M), XPOINT(M)
*     .. External Subroutines ..
      EXTERNAL         BDYVL, COEFF, D02KAY, D02KDF
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D02KDF Example Program Results'
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'A singular problem'
      TOL = 1.0e-4
      XPOINT(1) = 0.0e0
      XPOINT(2) = 0.1e0
      XPOINT(3) = 30.0e0
      XPOINT(4) = 30.0e0
      HMAX(1,1) = 0.0e0
      MAXIT = 0
      K = 11
      ELAM = 14.0e0
      DELAM = 1.0e0
      IFLAG = 0
      IFAIL = 0
*
*     * To obtain monitoring information from the supplied
*     subroutine MONIT replace the name D02KAY by MONIT in
*     the next statement, and declare MONIT as external *
*
      CALL D02KDF(XPOINT,M,COEFF,BDYVL,K,TOL,ELAM,DELAM,HMAX,MAXIT,
     +            IFLAG,D02KAY,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,*) 'Final results'
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'K =', K, '  ELAM =', ELAM, '  DELAM =', DELAM
      WRITE (NOUT,99998) 'HMAX(1,M-1) =', HMAX(1,M-1),
     +  '    HMAX(1,M) =', HMAX(1,M)
      STOP
*
99999 FORMAT (1X,A,I3,A,F12.3,A,e12.2)
99998 FORMAT (1X,A,F10.3,A,F10.3)
      END
*
      SUBROUTINE COEFF(P,Q,DQDL,X,ELAM,JINT)
*     .. Scalar Arguments ..
      real             DQDL, ELAM, P, Q, X
      INTEGER          JINT
```

```
*     .. Executable Statements ..
      P = 1.0e0
      Q = ELAM - X - 2.0e0/(X*X)
      DQDL = 1.0e0
      RETURN
      END
*
      SUBROUTINE BDYVL(XL,XR,ELAM,YL,YR)
*     .. Scalar Arguments ..
      real              ELAM, XL, XR
*     .. Array Arguments ..
      real              YL(3), YR(3)
*     .. Intrinsic Functions ..
      INTRINSIC         SQRT
*     .. Executable Statements ..
      YL(1) = XL
      YL(2) = 2.0e0
      YR(1) = 1.0e0
      YR(2) = -SQRT(XR-ELAM)
      RETURN
      END
*
      SUBROUTINE MONIT(MAXIT,IFLAG,ELAM,FINFO)
*     .. Parameters ..
      INTEGER           NOUT
      PARAMETER         (NOUT=6)
*     .. Scalar Arguments ..
      real              ELAM
      INTEGER           IFLAG, MAXIT
*     .. Array Arguments ..
      real              FINFO(15)
*     .. Local Scalars ..
      INTEGER           I
*     .. Executable Statements ..
      IF (MAXIT.EQ.-1) THEN
         WRITE (NOUT,*)
         WRITE (NOUT,*) 'Output from MONIT'
      END IF
      WRITE (NOUT,99999) MAXIT, IFLAG, ELAM, (FINFO(I),I=1,4)
      RETURN
*
99999 FORMAT (1X,2I4,F10.3,2e12.2,2F8.1)
      END
```

9.2 Program Data

None.

9.3 Program Results

```
 D02KDF Example Program Results

 A singular problem

 Final results

 K = 11  ELAM =      14.947  DELAM =    0.86E-03
 HMAX(1,M-1) =    -0.000    HMAX(1,M) =     5.456
```