NAG Library Routine Document

D02KEF

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

D02KEF 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. It also reports values of the eigenfunction and its derivatives. Provision is made for discontinuities in the coefficient functions or their derivatives.

2 Specification

```
SUBROUTINE D02KEF (XPOINT, M, MATCH, COEFFN, BDYVAL, K, TOL, ELAM, DELAM,       &
                   HMAX, MAXIT, MAXFUN, MONIT, REPORT, IFAIL)
INTEGER             M, MATCH, K, MAXIT, MAXFUN, IFAIL
REAL (KIND=nag_wp) XPOINT(M), TOL, ELAM, DELAM, HMAX(2,M)
EXTERNAL            COEFFN, BDYVAL, MONIT, REPORT
```

3 Description

D02KEF has essentially the same purpose as D02KDF with minor modifications to enable values of the eigenfunction to be obtained after convergence to the eigenvalue has been achieved.

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

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

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, are defined by COEFFN. The boundary conditions must be defined by 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.

When the final estimate $\lambda = \tilde{\lambda}$ of the eigenvalue has been found, the routine integrates the differential equation once more with that value of λ, and with initial conditions chosen so that the integral

$$S = \int_a^b y(x)^2 \frac{\partial q}{\partial \lambda}(x;\lambda)\,dx$$

is approximately one. When $q(x;\lambda)$ is of the form $\lambda w(x) + q(x)$, which is the most common case, S represents the square of the norm of y induced by the inner product

$$\langle f, g\rangle = \int_a^b f(x)g(x)w(x)\,dx,$$

with respect to which the eigenfunctions are mutually orthogonal. This normalization of y is only approximate, but experience shows that S generally differs from unity by only one or two per cent.

During this final integration the REPORT is called at each integration mesh point x. Sufficient information is returned to permit you to compute $y(x)$ and $y'(x)$ for printing or plotting. For reasons described in Section 8.2, D02KEF passes across to REPORT, not y and y', but the Pruefer variables β, ϕ and ρ on which the numerical method is based. Their relationship to y and y' is given by the equations

$$p(x)y' = \sqrt{\beta}\exp\left(\frac{\rho}{2}\right)\cos\left(\frac{\phi}{2}\right), \qquad y = \frac{1}{\sqrt{\beta}}\exp\left(\frac{\rho}{2}\right)\sin\left(\frac{\phi}{2}\right).$$

A specimen REPORT is given in Section 9 below.

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 nonzero and must not change sign throughout the interval (a, b); and,

(b) $\dfrac{\partial q}{\partial \lambda}$ must not change sign throughout the interval (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.

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 use of Pruefer transformations for the numerical solution of eigenvalue problems arising from physics and chemistry is given in Bailey (1966).

The scaled Pruefer method is described in a short note by Pryce and Hargrave (1977) and in some detail in the technical report by 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 (KIND=nag_wp) 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 the left- and right-hand end points, a and b, of the domain of definition of the Sturm–Liouville system if these are finite. If either a or b is infinite, the corresponding value x_1 or x_m may be a more-or-less arbitrarily 'large' number of appropriate sign.

(b) x_2 and x_{m-1} are the Boundary Matching Points (BMPs), 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 you should set $x_2 = x_1$ $(= a)$, while if it is a singular point you 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 Sections 8 and 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(M $-$ 1) $\leq$ XPOINT(M).

2: M – INTEGER *Input*

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

Constraint: M ≥ 4.

3: MATCH – INTEGER *Input/Output*

On entry: must be set to the index of the 'break point' to be used as the matching point (see Section 8.3). If MATCH is set to a value outside the range $[2, m-1]$ then a default value is taken, corresponding to the break point nearest the centre of the interval $[\text{XPOINT}(2), \text{XPOINT}(m-1)]$.

On exit: the index of the break point actually used as the matching point.

4: 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 the conditions which p and q must satisfy. See Sections 8.4 and 9 for examples.

The specification of COEFFN is:

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

INTEGER            JINT
REAL (KIND=nag_wp) P, Q, DQDL, X, ELAM
```

1: P – REAL (KIND=nag_wp) *Output*

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

2: Q – REAL (KIND=nag_wp) *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 (KIND=nag_wp) *Output*

On exit: the value of $\dfrac{\partial q}{\partial \lambda}(x;\lambda)$ for the current value of x and the current trial value of λ. However DQDL is only used in error estimation and, in the rare cases where it may be difficult to evaluate, an approximation (say to within 20%) will suffice.

4: X – REAL (KIND=nag_wp) *Input*

On entry: the current value of x.

5: ELAM – REAL (KIND=nag_wp) *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 either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02KEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

5: 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, $\mathrm{YL}(1)$ and $\mathrm{YL}(2)$ will in general be functions of XL and ELAM, and $\mathrm{YR}(1)$ and $\mathrm{YR}(2)$ functions of XR and ELAM, usually derived analytically from a power-series or asymptotic expansion. Examples are given in Sections 8.5 and 9.

The specification of BDYVAL is:

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

REAL (KIND=nag_wp) XL, XR, ELAM, YL(3), YR(3)
```

1: XL – REAL (KIND=nag_wp) *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 $a \leq x_1 < x_2$), then XL contains a point x such that $x_1 < x \leq x_2$.

2: XR – REAL (KIND=nag_wp) *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 (KIND=nag_wp) *Input*

On entry: the current trial value of λ.

4: YL(3) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{YL}(1)$ and $\mathrm{YL}(2)$ 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 $x = \mathrm{XL}$. $\mathrm{YL}(3)$ should not be set.

5: YR(3) – REAL (KIND=nag_wp) array *Output*

On exit: $\mathrm{YR}(1)$ and $\mathrm{YR}(2)$ 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 $x = \mathrm{XR}$. $\mathrm{YR}(3)$ should not be set.

BDYVAL must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02KEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

6: K – INTEGER *Input*

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

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

Constraint: K ≥ 0.

7: TOL – REAL (KIND=nag_wp) *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

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

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.

8: ELAM – REAL (KIND=nag_wp) *Input/Output*

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

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

9: DELAM – REAL (KIND=nag_wp) *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 + DELAM, 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: if IFAIL $= 0$, DELAM holds an estimate of the absolute error in the computed eigenvalue, that is $\left|\tilde{\lambda} - \text{ELAM}\right| \simeq$ 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.

If 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.

10: HMAX(2,M) – REAL (KIND=nag_wp) 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 contain diagnostic output in the case of an error exit (see Section 6).

11: MAXIT – INTEGER *Input/Output*

On entry: a bound on n_r, the number of root-finding iterations allowed, that is the number of trial values of λ that are used. If MAXIT ≤ 0, no such bound is assumed. (See also MAXFUN.)

Suggested value: MAXIT $= 0$.

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

12: MAXFUN – INTEGER *Input*

On entry: a bound on n_f, the number of calls to COEFFN made in any one root-finding 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 D02KEF, which is roughly proportional to $n_r \times n_f$.

13: MONIT – SUBROUTINE, supplied by the NAG Library or the user. *External Procedure*

MONIT is called by D02KEF at the end of each root-finding iteration and allows you 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 (sub)program D02KAY may be used. (D02KAY is included in the NAG Library.)

The specification of MONIT is:

```
SUBROUTINE MONIT (NIT, IFLAG, ELAM, FINFO)

INTEGER            NIT, IFLAG
REAL (KIND=nag_wp) ELAM, FINFO(15)
```

1: NIT – INTEGER *Input*

On entry: the current value of the parameter MAXIT of D02KEF, this is decreased by one at each iteration.

2: IFLAG – INTEGER *Input*

On entry: describes what phase the computation is in.

IFLAG < 0

An error occurred in the computation at this iteration; an error exit from D02KEF 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 (KIND=nag_wp) *Input*

On entry: the current trial value of λ.

4: FINFO(15) – REAL (KIND=nag_wp) 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 you. 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. (See Section 8.2 for further information.) FINFO(1) is set to zero if IFLAG < 0.

FINFO(2)

An estimate of the quantity $\partial\lambda$ 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 $\partial\lambda$ is the perturbation in λ that would have the same effect on $f(\lambda)$. Thus, at the zero of $f(\lambda), |\partial\lambda|$ is an approximate bound on the perturbation of the zero (that is the eigenvalue) caused by errors in numerical solution. If $\partial\lambda$ 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 $\partial\lambda$) 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. A step is successful if it is used to advance the integration.

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)

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 values of the Pruefer dependent variables β, ϕ and ρ respectively. These are set to zero in case of an error in BDYVAL.

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 either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02KEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

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

REPORT provides the means by which you may compute the eigenfunction $y(x)$ and its derivative at each integration mesh point x. (See Section 8 for an example.)

The specification of REPORT is:

```
SUBROUTINE REPORT (X, V, JINT)

INTEGER            JINT
REAL (KIND=nag_wp) X, V(3)
```

1: X – REAL (KIND=nag_wp) *Input*

On entry: the current value of the independent variable x. See Section 8.3 for the order in which values of x are supplied.

2: V(3) – REAL (KIND=nag_wp) array *Input*

On entry: V(1), V(2), V(3) hold the current values of the Pruefer variables β, ϕ, ρ respectively.

3: JINT – INTEGER *Input*

On entry: indicates the sub-interval between break points in which X lies exactly as for COEFFN, **except** that at the extreme left-hand end point (when $x = \text{XPOINT}(2)$) JINT is set to 0 and at the extreme right-hand end point (when $x = x_r = \text{XPOINT}(m-1)$) JINT is set to $m-2$.

REPORT must either be a module subprogram USEd by, or declared as EXTERNAL in, the (sub)program from which D02KEF is called. Parameters denoted as *Input* must **not** be changed by this procedure.

15: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this parameter you should refer to Section 3.3 in the Essential Introduction for details.

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

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (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 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 ≤ 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 YL(1) = YL$(2) = 0.0$, or YR(1) = 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 the 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 and 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*** is being used. The final value of ELAM should be a very good approximation to the eigenvalue.

IFAIL $= 10$

C05AZF, called by D02KEF, 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 value for TOL.

IFAIL $= 11$
IFAIL $= 12$

A serious error has occurred in an internal call. Check all (sub)program calls and array dimensions. Seek expert help.

Note: error exits with IFAIL $= 2$, 3, 6, 7, 8 or 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

The time taken by D02KEF 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 the most 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. The extra time needed to compute the eigenfunction is principally the cost of one additional integration once the eigenvalue has been found.

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.

D02KEF defines a miss-distance $f(\lambda)$ based on the rotation about the origin of the point $\mathrm{P}(x) = \big(p(x)y'(x), y(x)\big)$ 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** 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.

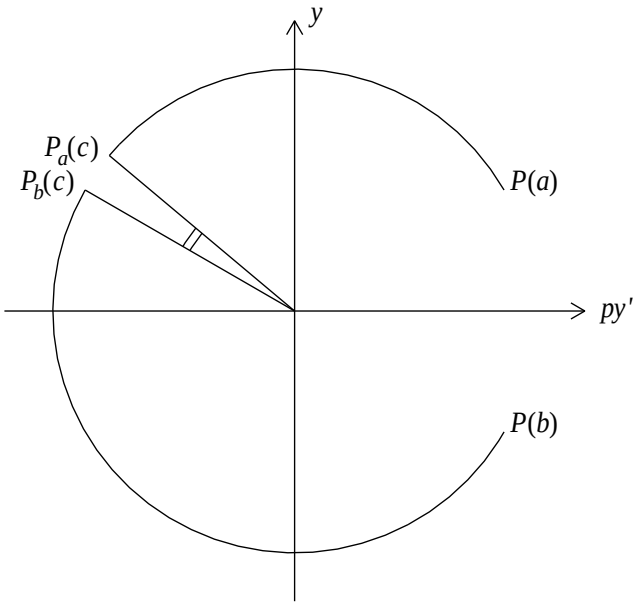

Figure 1

The resulting function $f(\lambda)$ is monotonic over $-\infty < \lambda < \infty$, increasing if $\dfrac{\partial q}{\partial \lambda} > 0$ and decreasing if $\dfrac{\partial q}{\partial \lambda} < 0$, 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 you if singular points are involved. However, the error estimate output in DELAM is usually fairly realistic, in that the actual error $\left|\tilde{\lambda} - \text{ELAM}\right|$ is within an order of magnitude of DELAM.

We pass the values of β, ϕ, ρ across through REPORT rather than converting them to values of y, y' inside D02KEF, for the following reasons. First, there may be cases where auxiliary quantities can be more accurately computed from the Pruefer variables than from y and y'. Second, in singular problems on an infinite interval y and y' may underflow towards the end of the range, whereas the Pruefer variables remain well-behaved. Third, with high-order eigenvalues (and therefore highly oscillatory eigenfunctions) the eigenfunction may have a complete oscillation (or more than one oscillation) between two mesh points, so that values of y and y' at mesh points give a very poor representation of the curve. The probable behaviour of the Pruefer variables in this case is that β and ρ vary slowly whilst ϕ increases quickly: for all three Pruefer variables linear interpolation between the values at adjacent mesh points is probably sufficiently accurate to yield acceptable intermediate values of β, ϕ, ρ (and hence of y, y') for graphical purposes.

Similar considerations apply to the exponentially decaying 'tails' of the eigenfunctions that often occur in singular problems. Here ϕ has approximately constant value whilst ρ increases rapidly in the direction of integration, though the step length is generally fairly small over such a range.

If the solution is output through REPORT at x values which are too widely spaced, the step length can be controlled by choosing HMAX suitably, or, preferably, by reducing TOL. Both these choices will lead to more accurate eigenvalues and eigenfunctions but at some computational cost.

8.3 The Position of the Shooting Matching Point *c*

This point is always one of the values x_i in array XPOINT. It may be specified using the parameter MATCH. The default value 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.

Note that the shooting method used by the code integrates first from the left-hand end x_l, then from the right-hand end x_r, to meet at the matching point c in the middle. This will of course be reflected in printed or graphical output. The diagram shows a possible sequence of nine mesh points τ_1 through τ_9 in the order in which they appear, assuming there are just two sub-intervals (so $m = 5$).

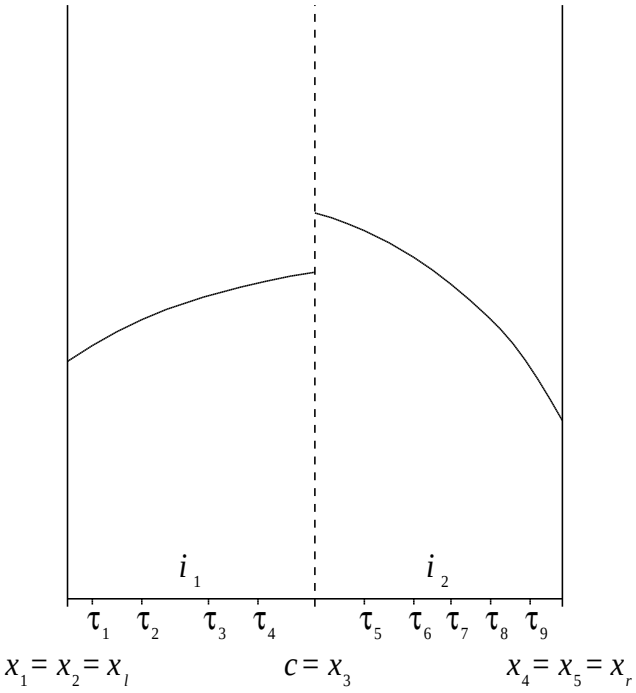

Figure 2

Since the shooting method usually fails to match up the two 'legs' of the curve exactly, there is bound to be a jump in y, or in $p(x)y'$ or both, at the matching point c. The code in fact 'shares' the discrepancy out so that both y and $p(x)y'$ have a jump. A large jump does **not** imply an inaccurate eigenvalue, but implies either

(a) a badly chosen matching point: if $q(x;\lambda)$ has a 'humped' shape, c should be chosen near the maximum value of q, especially if q is negative at the ends of the interval;

(b) an inherently ill-conditioned problem, typically one where another eigenvalue is pathologically close to the one being sought. In this case it is extremely difficult to obtain an accurate eigenfunction.

In Section 9, we find the 11th eigenvalue and corresponding eigenfunction of the equation

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

the boundary conditions being that y should remain bounded as x tends to 0 and x tends to ∞. The coding of this problem is discussed in detail in Section 8.5.

The choice of matching point c is open. If we choose $c = 30.0$ as in D02KDF example program we find that the exponentially increasing component of the solution dominates and we get extremely inaccurate values for the eigenfunction (though the eigenvalue is determined accurately). The values of the

eigenfunction calculated with $c = 30.0$ are given schematically in Figure 3.

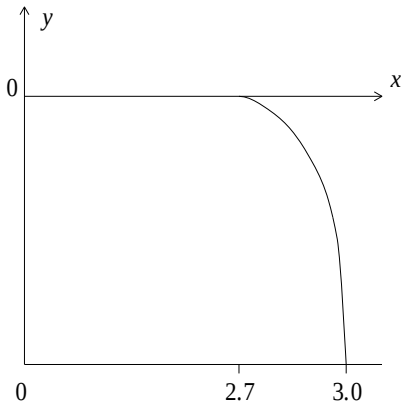

Figure 3

If we choose c as the maximum of the hump in $q(x;\lambda)$ (see item (a) above) we instead obtain the accurate results given in Figure 4

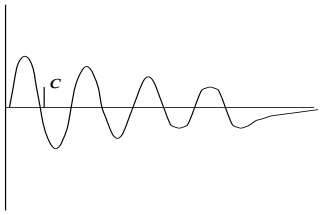

Figure 4

8.4 Examples of Coding the COEFFN

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

(a) a simple case,

(b) a case in which 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

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

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

The 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 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 = 1.0E0
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 Section 8.5.

Example C

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

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 at ± 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 = 1.0E0
DQDL = 1.0E0 - 2.0E0*EXP(-100.0E0*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$(1, j)$, for $j = 1, 2, 3$, 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 solution 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 you provide through the BDYVAL is an approximation to this ratio, valid as x tends to the singular point (SP).

You 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. You are strongly recommended to experiment with placing the BMPs. In many singular problems quite crude asymptotic formulae will do. To help you avoid needlessly accurate formulae, D02KEF 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 in Section 8.6.

Example of coding BDYVAL:

The example below illustrates typical situations:

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

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

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

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 solution. The solution we require behaves as

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

and the other as

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

Hence, the desired solution has $p(x)y'/y \sim -\sqrt{t}$ so that we could set $\mathrm{YR}(1) = 1.0$ and $\mathrm{YR}(2) = -\sqrt{x - \lambda}$. 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) = 2.0E0
YR(1) = 1.0E0
YR(2) = -SQRT(XR-ELAM)
RETURN
END
```

Clearly for this problem it is essential that any value given by D02KEF to XR is well to the right of the value of ELAM, so that you 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 $\mathrm{Ai}(x)$, $\mathrm{Ai}'(x)$, (see page 448 of Abramowitz and Stegun (1972)).

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

$$y = x^2\left(c_0 + c_1 x + c_2 x^2 + \cdots\right)$$

with

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

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 σ_l and σ_r

The sensitivity parameters σ_l, σ_r (held in HMAX$(1, m-1)$ and 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

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

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 you 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

This example finds the 11th eigenvalue and eigenfunction of the example of Section 8.5, using the simple asymptotic formulae for the boundary conditions.

Comparison of the results from this example program with the corresponding results from D02KDF example program shows that similar output is produced from MONIT, followed by the eigenfunction values from REPORT, and then a further line of information from MONIT (corresponding to the integration to find the eigenfunction). Final information is printed within the example program exactly as with D02KDF.

Note the discrepancy at the matching point c ($= \sqrt[3]{4}$, the maximum of $q(x;\lambda)$, in this case) between the solutions obtained by integrations from left- and right-hand end points.

9.1 Program Text

```
!   D02KEF Example Program Text
!   Mark 24 Release. NAG Copyright 2012.

    Module d02kefe_mod

!     Data for D02KEF example program

!     .. Use Statements ..
      Use nag_library, Only: nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Real (Kind=nag_wp), Parameter        :: one = 1.0_nag_wp
      Real (Kind=nag_wp), Parameter        :: two = 2.0_nag_wp
      Real (Kind=nag_wp), Parameter        :: zero = 0.0_nag_wp
      Integer, Parameter                   :: nin = 5, nout = 6
    Contains
      Subroutine coeffn(p,q,dqdl,x,elam,jint)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: dqdl, p, q
        Real (Kind=nag_wp), Intent (In)      :: elam, x
        Integer, Intent (In)                 :: jint
```

```
!       .. Executable Statements ..
        p = one
        q = elam - x - two/(x*x)
        dqdl = one
        Return
      End Subroutine coeffn
      Subroutine bdyval(xl,xr,elam,yl,yr)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: elam, xl, xr
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (Out)     :: yl(3), yr(3)
!       .. Intrinsic Procedures ..
        Intrinsic                            :: sqrt
!       .. Executable Statements ..
        yl(1) = xl
        yl(2) = two
        yr(1) = one
        yr(2) = -sqrt(xr-elam)
        Return
      End Subroutine bdyval
      Subroutine report(x,v,jint)

!       .. Use Statements ..
        Use nag_library, Only: x02amf
!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: x
        Integer, Intent (In)                 :: jint
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: v(3)
!       .. Local Scalars ..
        Real (Kind=nag_wp)                   :: pyp, r, sqrtb, y
!       .. Intrinsic Procedures ..
        Intrinsic                            :: cos, exp, log, sin, sqrt
!       .. Executable Statements ..
        If (jint==0) Then
          Write (nout,*)
          Write (nout,*) ' Eigenfunction values'
          Write (nout,*) '       X          Y           PYP'
        End If
        sqrtb = sqrt(v(1))
!       Avoid underflow in call of EXP
        If (0.5_nag_wp*v(3)>=log(x02amf())) Then
          r = exp(0.5_nag_wp*v(3))
        Else
          r = zero
        End If
        pyp = r*sqrtb*cos(0.5_nag_wp*v(2))
        y = r/sqrtb*sin(0.5_nag_wp*v(2))
        Write (nout,99999) x, y, pyp
        Return

99999   Format (1X,F10.3,1P,2F12.4)
      End Subroutine report
      Subroutine monit(nit,iflag,elam,finfo)

!       .. Scalar Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: elam
        Integer, Intent (In)                 :: iflag, nit
!       .. Array Arguments ..
        Real (Kind=nag_wp), Intent (In)      :: finfo(15)
!       .. Executable Statements ..
        If (nit==-1) Then
          Write (nout,*)
          Write (nout,*) 'Output from MONIT'
        End If
        Write (nout,99999) nit, iflag, elam, finfo(1:4)
        Return

99999   Format (1X,2I4,F10.3,2E12.2,2F8.1)
      End Subroutine monit
```

```
    End Module d02kefe_mod
    Program d02kefe

!     D02KEF Example Main Program

!     .. Use Statements ..
      Use nag_library, Only: d02kay, d02kef, nag_wp
      Use d02kefe_mod, Only: bdyval, coeffn, nin, nout, report
!     .. Implicit None Statement ..
      Implicit None
!     .. Local Scalars ..
      Real (Kind=nag_wp)                  :: delam, elam, tol
      Integer                             :: ifail, k, m, match, maxfun, maxit
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable     :: hmax(:,:), xpoint(:)
!     .. Executable Statements ..
      Write (nout,*) 'D02KEF Example Program Results'
      Write (nout,*)
      Write (nout,*) 'A singular problem'
!     Skip heading in data file
      Read (nin,*)
!     m: number of points in xpoint
      Read (nin,*) m
      Allocate (hmax(2,m),xpoint(m))
!     xpoint: points where the boundary conditions are to be imposed
!             and any break points,
!     tol: tolerance parameter which determines the accuracy of the
!          computed eigenvalue,
!     k: index of the required eigenvalue,  hmax: maximum step size,
!     elam: initial estimate of the eigenvalue,  delam: initial search step,
!     maxit: number of root-finding iterations allowed,
!     maxfun: number of calls to coeffn in any one root-finding iteration,
!     match: index of the break point.
      Read (nin,*) xpoint(1:m)
      Read (nin,*) tol
      Read (nin,*) k
      Read (nin,*) elam, delam
      Read (nin,*) hmax(1,1:m-3)
      Read (nin,*) maxit, maxfun, match

!     ifail: behaviour on error exit
!            =0 for hard exit, =1 for quiet-soft, =-1 for noisy-soft
      ifail = 0

!     * To obtain monitoring information from the supplied
!     subroutine MONIT replace the name D02KAY by MONIT in
!     the next statement and USE MONIT from d02kefe_mod *

      Call d02kef(xpoint,m,match,coeffn,bdyval,k,tol,elam,delam,hmax,maxit, &
        maxfun,d02kay,report,ifail)

      Write (nout,*)
      Write (nout,*) 'Final results'
      Write (nout,*)
      Write (nout,99999) k, elam, delam
      Write (nout,99998) hmax(1,m-1), hmax(1,m)

99999 Format (1X,'K =',I3,'  ELAM =',F12.3,'  DELAM =',E12.2)
99998 Format (1X,'HMAX(1,M-1) =',F10.3,'    HMAX(1,M) =',F10.3)
    End Program d02kefe
```

9.2 Program Data

```
D02KEF Example Program Data
   5                                                          : m
   0.0  0.1  1.58740105196819947475  30.0  30.0               : xpoint
   1.0E-4                                                     : tol
   11                                                         : k
   14.0  1.0                                                  : elam, delam
   0.0  0.0                                                   : hmax
   0 0 0                                                      : maxit, maxfun, match
```

9.3 Program Results

```
 D02KEF Example Program Results

 A singular problem

  Eigenfunction values
        X          Y          PYP
      0.100      0.1233      2.4656
      0.168      0.3413      3.9058
      0.216      0.5500      4.7425
      0.312      1.0652      5.8104
      0.407      1.6307      5.9191
      0.578      2.4933      3.7209
      0.724      2.7787     -0.0262
      0.909      2.2745     -5.3609
      1.137      0.5046     -9.3563
      1.453     -2.1525     -5.7225
      1.587     -2.6541     -1.5714
     30.000     -0.0000      0.0000
     29.096     -0.0000      0.0000
     28.629     -0.0000      0.0000
     28.356     -0.0000      0.0000
     28.062     -0.0000      0.0000
     27.713     -0.0000      0.0000
     27.262     -0.0000      0.0000
     26.855     -0.0000      0.0000
     26.432     -0.0000      0.0000
     26.062     -0.0000      0.0000
     25.686     -0.0000      0.0000
     25.301     -0.0000      0.0000
     24.891     -0.0000      0.0000
     24.574     -0.0000      0.0000
     24.249     -0.0000      0.0000
     23.855     -0.0000      0.0000
     23.530     -0.0000      0.0000
     23.157     -0.0000      0.0000
     22.843     -0.0000      0.0000
     22.467     -0.0000      0.0000
     22.140     -0.0000      0.0000
     21.743     -0.0000      0.0000
     21.397     -0.0000      0.0001
     20.979     -0.0001      0.0002
     20.614     -0.0002      0.0005
     20.173     -0.0006      0.0013
     19.786     -0.0014      0.0031
     19.453     -0.0029      0.0063
     19.016     -0.0073      0.0151
     18.601     -0.0169      0.0334
     18.224     -0.0349      0.0658
     17.865     -0.0676      0.1208
     17.503     -0.1268      0.2139
     17.124     -0.2352      0.3708
     16.716     -0.4367      0.6328
     16.248     -0.8293      1.0702
     15.732     -1.5356      1.6814
     15.411     -2.1372      2.0482
     15.079     -2.8634      2.2954
     14.785     -3.5418      2.2855
```

```
    14.484     -4.1869      1.9267
    14.237     -4.5916      1.2983
    13.895     -4.8116     -0.1128
    13.519     -4.3778     -2.2666
    13.124     -3.0023     -4.6509
    12.559      0.2491     -6.2859
    12.070      2.9403     -4.0725
    11.605      3.7229      1.0225
    11.133      1.9542      6.0941
    10.652     -1.3713      6.6399
    10.199     -3.3599      1.3740
     9.773     -2.4822     -5.2726
     9.325      0.6805     -7.6006
     8.860      3.1382     -1.7426
     8.441      2.1843      5.9550
     8.005     -1.1293      7.6227
     7.569     -3.0761      0.1991
     7.157     -1.3896     -7.5651
     6.721      2.0523     -6.1828
     6.313      2.7637      3.1900
     5.891     -0.1324      8.7656
     5.480     -2.7833      2.3579
     5.072     -1.5819     -7.5149
     4.649      1.9381     -6.5648
     4.248      2.4987      4.2347
     3.837     -0.6976      8.9149
     3.418     -2.7508     -0.8295
     3.016     -0.3336     -9.3122
     2.607      2.6120     -2.5897
     2.173      0.8858      8.9896
     1.722     -2.5601      2.9245
     1.587     -2.6533     -1.5665

 Final results

 K = 11  ELAM =      14.946  DELAM =    0.96E-03
 HMAX(1,M-1) =    -0.015    HMAX(1,M) =     0.000
```