NAG Toolbox for MATLAB

d02kd

1 Purpose

d02kd 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 Syntax

```
[elam, delam, hmax, maxit, ifail] = d02kd(xpoint, coeffn, bdyval, k,
tol, elam, delam, hmax, monit, 'm', m, 'maxit', maxit, 'maxfun', maxfun)
```

3 Description

d02kd finds a specified eigenvalue $\tilde{\lambda}$ of a Sturm–Liouville system defined by a self-adjoint differential equation of the second-order

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

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, must be defined by a user-supplied (sub)program **coeffn**. The boundary conditions must be defined by a user-supplied (sub)program **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 nonzero and of one sign throughout the interval (a,b); and

(b) $\dfrac{\partial q}{\partial \lambda}$ 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 function 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 **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 use 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

5.1 Compulsory Input Parameters

1: **xpoint(m) – double array**

The points where the boundary conditions computed by user-supplied (sub)program **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-hand 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 user-supplied (sub)program **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**$(\mathbf{m} - 1) \leq$ **xpoint**$(\mathbf{m})$.

2: **coeffn – string containing name of m-file**

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. See Sections 8.4 and 9 for examples.

Its specification is:

```
[p, q, dqdl] = coeffn(x, elam, jint)
```

Input Parameters

1: **x – double scalar**

The current value of x.

2: **elam – double scalar**

The current trial value of the eigenvalue parameter λ.

3: **jint – int32 scalar**

The index j of the sub-interval i_j (see specification of **xpoint**) in which x lies.

Output Parameters

1: **p – double scalar**

The value of $p(x)$ for the current value of x.

2: **q – double scalar**

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

3: **dqdl – double scalar**

The value of $\dfrac{\partial q}{\partial \lambda}$ 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%) will suffice.

3: **bdyval – string containing name of m-file**

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., $\mathbf{yl}(1) = c_2$ and $\mathbf{yl}(2) = -c_1 p(a)$.

For a **singular** end point however, $\mathbf{yl}(1)$ and $\mathbf{yl}(2)$ will in general be functions of **xl** and **elam**, and $\mathbf{yr}(1)$ and $\mathbf{yr}(2)$ functions of **xr** and **elam**, usually derived analytically from a power-series or asymptotic expansion. Examples are given in Sections 8.5 and 9.

Its specification is:

```
[yl, yr] = bdyval(xl, xr, elam)
```

Input Parameters

1: **xl – double scalar**

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 – double scalar**

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 – double scalar**

The current trial value of λ.

Output Parameters

1: **yl**(**3**) – **double array**

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 $x = \mathbf{xl}$. **yl**(3) should not be set.

2: **yr**(**3**) – **double array**

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 $x = \mathbf{xr}$. **yr**(3) should not be set.

4: **k** – **int32 scalar**

The index k of the required eigenvalue when the eigenvalues are ordered

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

Constraint: $\mathbf{k} \geq 0$.

5: **tol** – **double scalar**

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

$$\mathbf{delam} \leq \mathbf{tol} \times \max(1.0, |\mathbf{elam}|) \tag{1}$$

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: $\mathbf{tol} > 0.0$.

6: **elam** – **double scalar**

An initial estimate of the eigenvalue $\tilde{\lambda}$.

7: **delam** – **double scalar**

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 function 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 $\mathbf{delam} = 0.0$ on entry, it is given the default value of $0.25 \times \max(1.0, |\mathbf{elam}|)$.

8: **hmax**(**2,m**) – **double array**

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 function 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 function. See Section 8 for further suggestions.

The rest of the array is used as workspace.

9: **monit – string containing name of m-file**

monit is called by d02kd at the end of each rootfinding 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 Fortran Library.)

Its specification is:

```
[] = monit(maxit, iflag, elam, finfo)
```

Input Parameters

1: **maxit – int32 scalar**

The current value of the parameter **maxit** of d02kd, which is decreased by one at each iteration.

2: **iflag – int32 scalar**

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 d02kd with **ifail** $= -$**iflag** will follow.

iflag $= 1$

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

iflag $= 2$

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

3: **elam – double scalar**

The current trial value of λ.

4: **finfo**$(\mathbf{15})$ **– double array**

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. **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 user-supplied (sub)program **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 user-supplied (sub)program **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 user-supplied (sub)program **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 user-supplied (sub)program **bdyval**, it is set to the value of **xl** or **xr** as appropriate (see the specification of the **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 user-supplied (sub)program **bdyval**. (See d02ke 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 user-supplied (sub)program **bdyval**.

finfo(15), the last integration mesh point. This is set to zero in the case of an error in user-supplied (sub)program **bdyval**.

Output Parameters

5.2 Optional Input Parameters

1: **m – int32 scalar**

Default: The dimension of the arrays **xpoint**, **hmax**. (An error is raised if these dimensions are not equal.)

the number of points in the array **xpoint**.

Constraint: **m** ≥ 4.

2: **maxit – int32 scalar**

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**.)

Default: 0

3: **maxfun – int32 scalar**

A bound on n_f, the number of calls to user-supplied (sub)program **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 d02kd, which is roughly proportional to $n_r \times n_f$.

Default: 0

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **elam – double scalar**

The final computed estimate, whether or not an error occurred.

2: **delam – double scalar**

With **ifail** $= 0$, **delam** holds an estimate of the absolute error in the computed eigenvalue, that is $\left|\tilde{\lambda} - \textbf{elam}\right| \approx$ **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.

3: **hmax(2,m) – double array**

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).

4: **maxit – int32 scalar**

Suggested value: **maxit** $= 0$. (See also under **maxfun**.)

Default: 0

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

5: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

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 user-supplied (sub)program **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 (sub)program **monit** for details.

This error exit will also occur if $p(x)$ is zero at the point where the boundary condition is imposed. Probably user-supplied (sub)program **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 user-supplied (sub)program **coeffn** became zero or changed sign. See the last call of (sub)program **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 **iflag** of the (sub)program **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 function.

ifail $= 6$

maxfun > 0 on entry, and the last iteration was terminated because more than **maxfun** calls to user-supplied (sub)program **coeffn** were used. See the last call of (sub)program **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 (sub)program **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 (sub)program **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 $= 10$

c05az, called by d02kd, 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 $= 12$ (c05az)

A serious error has occurred in the specified function. Check all (sub)program calls and array dimensions. Seek expert help.

hmax$(2,1)$ holds the failure exit number from the function where the failure occurred. In the case of a failure in c05az, **hmax**$(2,2)$ holds the value of parameter **ind** of c05az.

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 (sub)program **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 function, 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.

d02kd defines a miss-distance $f(\lambda)$ based on the rotation round the origin of the point $\mathbf{p}(x) = \big(p(x)y'(x), y(x)\big)$ 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 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.

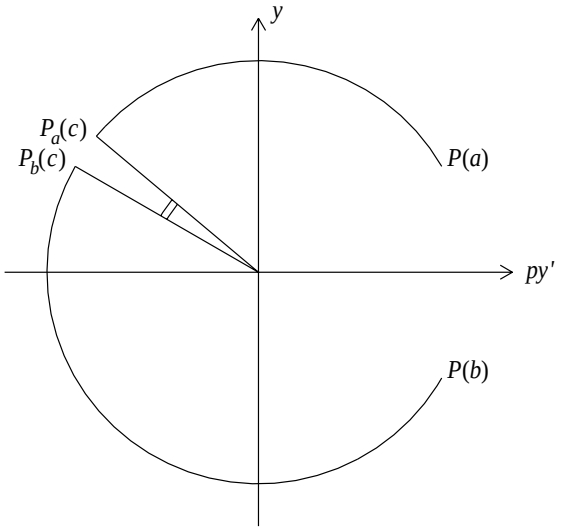

Figure 1

The resulting function $f(\lambda)$ is monotonic over $-\infty < \lambda < \infty$, increasing if $\dfrac{\partial q}{\partial \lambda} > 0$ and decreasing if $\dfrac{\partial q}{\partial \lambda} < 0$, with a unique zero at the desired eigenvalue $\tilde{\lambda}$. The function 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 function 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 $\left|\tilde{\lambda} - \mathbf{elam}\right|$ 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 User-supplied (Sub)program coeffn

Coding user-supplied (sub)program **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 user-supplied (sub)program **coeffn** could be:

```
function [p, q, dqdl] = coeffn(x, elam, jint)
  global nu;
  ...
  p = x;
  q = elam*x - nu*nu/x
  dqdl = x;
```

where nu (standing for ν) is a double global variable declared in the calling 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 user-supplied (sub)program **coeffn** could be:

```
function [p, q, dqdl] = coeffn(x, elam, jint)
```

```
...
p = 1;
dqdl = 1;
if (jint == 2)
  q = elam + x*x - 10
else
  q = elam + 6/abs(x)
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 user-supplied (sub)program **bdyval** and the technique is discussed in the next sub-section.

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 function 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 user-supplied (sub)program **coeffn** could be:

```
function [p, q, dqdl] = coeffn(x, elam, jint)
...
p = 1;
dqdl = 1 - 2*exp(-100*x*x);
q = elam * dqdl;
```

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 you provide through the user-supplied (sub)program **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, d02kd 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 user-supplied (sub)program bdyval:

The example below illustrates typical situations:

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

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

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

$$\mathbf{yl}(1) = x \qquad \text{and} \qquad \mathbf{yl}(2) = 2.0.$$

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

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

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 $\mathbf{yl}(1) = 1.0$ and $\mathbf{yl}(2) = -\sqrt{x-\lambda}$. The complete (sub)program might thus be

```
function [yl, yr] = bdyval(xl, xr, elam)
  yl(1) = xl;
  yl(2) = 2;
  yr(1) = 1;
  yr(2) = -sqrt(xr-elam);
```

Clearly for this problem it is essential that any value given by d02kd 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 $\mathrm{Ai}(x)$, $\mathrm{Ai}'(x)$, e.g., see page 448 of Abramowitz and Stegun 1972.

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

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

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 σ_l and σ_r

The sensitivity parameters σ_l, σ_r (held in $\mathbf{hmax}(1, m-1)$ and $\mathbf{hmax}(1, m)$ 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 function 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

d02kd_bdyval.m

```
function [yl, yr] = bdyval(xl, xr, elam)
  yl(1) = xl;
  yl(2) = 2;
  yr(1) = 1;
  yr(2) = -sqrt(xr-elam);
```

d02kd_coeffn.m

```
function [p, q, dqdl] = coeffn(x, elam, jint)
  p = 1;
  dqdl = 1;
  q = elam - x - 2/x/x;
```

d02kd_monit.m

```
function monit(maxit, iflag, elam, finfo)
  if (maxit == -1)
    fprintf('\nOutput from Monit\n');
  end

  fprintf('%2d   %d   %6.3f %+6.3f %+6.3g %+6.3f %+6.3f\n', ...
     maxit, iflag, elam, finfo(1), finfo(2), finfo(3), finfo(4));
```

```
xpoint = [0;
     0.1;
     30;
     30];
k = int32(11);
tol = 0.0001;
elam = 14;
delam = 1;
hmax = [0, 0, -0.7991583347320557, 0;
        0, 0, 0, 0];
[elamOut, delamOut, hmaxOut, maxit, ifail] = ...
    d02kd(xpoint, 'd02kd_coeffn', 'd02kd_bdyval', k, tol, elam, delam,
hmax, ...
   'd02kd_monit')
```

```
Output from Monit
-1   1   14.000 -1.499 -0.000197 +1.000 +679.000
-2   1   15.000 +0.501 -0.000359 +1.000 +627.000
-3   2   14.750 -0.499 -0.000495 +1.000 +632.000
-4   2   14.875 -0.499 -0.000244 +1.000 +570.000
-5   2   14.937 -0.499 -0.000664 +1.000 +471.000
-6   2   14.969 +0.501 -0.00027 +1.000 +441.000
-7   2   14.953 +0.501 -0.000412 +1.000 +431.000
-8   2   14.945 -0.499 -0.000408 +1.000 +431.000
-9   2   14.949 +0.501 -0.000208 +1.000 +421.000
-10   2   14.947 +0.501 -0.000408 +1.000 +417.000
```

```
-11   2   14.946 -0.499 -0.000673 +1.000 +413.000
-12   2   14.947 -0.499 -0.000371 +1.000 +417.000
elamOut =
   14.9466
delamOut =
   8.6040e-04
hmaxOut =
         0         0   -0.0000    5.4563
         0         0         0         0
maxit =
        -12
ifail =
          0
```