NAG Toolbox for MATLAB

d02tk

1 Purpose

d02tk solves a general two point boundary-value problem for a nonlinear mixed order system of ordinary differential equations.

2 Syntax

```
[work, iwork, ifail] = d02tk(ffun, fjac, gafun, gbfun, gajac, gbjac,
guess, work, iwork)
```

3 Description

d02tk and its associated functions (d02tv, d02tx, d02ty and d02tz) solve the two point boundary-value problem for a nonlinear mixed order system of ordinary differential equations

$$\begin{aligned}
y_1^{(m_1)}(x) &= f_1\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}\right) \\
y_2^{(m_2)}(x) &= f_2\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}\right) \\
&\vdots \\
y_n^{(m_n)}(x) &= f_n\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots y_n^{(m_n-1)}\right)
\end{aligned}$$

over an interval $[a, b]$ subject to p (>0) nonlinear boundary conditions at a and q (>0) nonlinear boundary conditions at b, where $p + q = \sum_1^n m_i$. Note that $y_i^{(m)}(x)$ is the mth derivative of the ith solution component. Hence $y_i^{(0)}(x) = y_i(x)$. The left boundary conditions at a are defined as

$$g_i(z(y(a))) = 0, \qquad i = 1, 2, \ldots, p,$$

and the right boundary conditions at b as

$$\bar{g}_j(z(y(b))) = 0, \qquad j = 1, 2, \ldots, q,$$

where $y = (y_1, y_2, \ldots, y_n)$ and

$$z(y(x)) = \left(y_1(x), y_1^{(1)}(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)\right).$$

First, d02tv must be called to specify the initial mesh, error requirements and other details. Note that the error requirements apply only to the solution components $y_1, y_2, \ldots, y_n$ and that no error control is applied to derivatives of solution components. (If error control is required on derivatives then the system must be reduced in order by introducing the derivatives whose error is to be controlled as new variables. See Section 8 of the document for d02tv.) Then, d02tk can be used to solve the boundary-value problem. After successful computation, d02tz can be used to ascertain details about the final mesh and other details of the solution procedure, and d02ty can be used to compute the approximate solution anywhere on the interval $[a, b]$.

A description of the numerical technique used in d02tk is given in Section 3 of the document for d02tv.

d02tk can also be used in the solution of a series of problems, for example in performing continuation, when the mesh used to compute the solution of one problem is to be used as the initial mesh for the solution of the next related problem. d02tx should be used in between calls to d02tk in this context.

See Section 8 of the document for d02tv for details of how to solve boundary-value problems of a more general nature.

The functions are based on modified versions of the codes COLSYS and COLNEW (see Ascher *et al.* 1979 and Ascher and Bader 1987). A comprehensive treatment of the numerical solution of boundary-value problems can be found in Ascher *et al.* 1988 and Keller 1992.

4 References

Ascher U M and Bader G 1987 A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

Ascher U M, Christiansen J and Russell R D 1979 A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

Ascher U M, Mattheij R M M and Russell R D 1988 *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice–Hall

Keller H B 1992 *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

5 Parameters

5.1 Compulsory Input Parameters

1: **ffun – string containing name of m-file**

ffun must evaluate the functions f_i for given values $x, z(y(x))$.

Its specification is:

```
[f] = ffun(x, y, neq, m)
```

Input Parameters

1: **x – double scalar**

x, the independent variable.

2: **y(neq,0 : $*$) – double array**

The first dimension of the array **y** must be at least

The second dimension of the array must be at least $maxdeg$

$\mathbf{y}(i,j)$ contains $y_i^{(j)}(x)$, for $i = 1, 2, \ldots, \mathbf{neq}, j = 0, 1, \ldots, \mathbf{m}(i) - 1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

3: **neq – int32 scalar**

the number of differential equations.

4: **m(neq) – int32 array**

The order, m_i, of the ith differential equation, for $i = 1, 2, \ldots, \mathbf{neq}$.

Output Parameters

1: **f(neq) – double array**

The values of f_i, for $i = 1, 2, \ldots, \mathbf{neq}$.

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

fjac must evaluate the partial derivatives of f_i with respect to the elements of $z(y(x))$ $\left(= \left(y_1(x), y_1^1(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)\right)\right)$.

Its specification is:

```
[dfdy] = fjac(x, y, neq, m)
```

Input Parameters

1: **x – double scalar**

x, the independent variable.

2: **y(neq,0 : $*$) – double array**

The first dimension of the array **y** must be at least

The second dimension of the array must be at least $maxdeg$

y(i,j) contains $y_i^{(j)}(x)$, for $i = 1, 2, \ldots,$ **neq**,$j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

3: **neq – int32 scalar**

the number of differential equations.

4: **m(neq) – int32 array**

The order, m_i, of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

Output Parameters

1: **dfdy(neq,neq,0 : $*$) – double array**

Note: the last dimension of the array **dfdy** must be at least $maxdeg$.

dfdy(i,j,k) must contain the partial derivative of f_i with respect to $y_j^{(k)}$, for $i,j = 1, 2, \ldots,$ **neq**, $k = 0, 1, \ldots,$ **m**$(j) - 1$. Only nonzero partial derivatives need be set.

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

gafun must evaluate the boundary conditions at the left-hand end of the range, that is functions $g_i(z(y(a)))$ for given values of $z(y(a))$.

Its specification is:

```
[ga] = gafun(ya, neq, m, nlbc)
```

Input Parameters

1: **ya(neq,0 : $*$) – double array**

The first dimension of the array **ya** must be at least

The second dimension of the array must be at least $maxdeg$

ya(i,j) contains $y_i^{(j)}(a)$, for $i = 1, 2, \ldots,$ **neq**, $j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(a) = y_i(a)$.

2: **neq – int32 scalar**

the number of differential equations.

3: **m(neq) – int32 array**

The order, m_i, of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

4: **nlbc – int32 scalar**

The number of boundary conditions at a.

Output Parameters

1: **ga(nlbc) – double array**

The values of $g_i(z(y(a)))$, for $i = 1, 2, \ldots,$ **nlbc**.

4: **gbfun – string containing name of m-file**

gbfun must evaluate the boundary conditions at the right-hand end of the range, that is functions $\bar{g}_i(z(y(b)))$ for given values of $z(y(b))$.

Its specification is:

```
[gb] = gbfun(yb, neq, m, nrbc)
```

Input Parameters

1: **yb(neq,0 : ∗) – double array**

The first dimension of the array **yb** must be at least

The second dimension of the array must be at least $maxdeg$

yb(i,j) contains $y_i^{(j)}(b)$, for $i = 1, 2, \ldots,$ **neq**, $j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(b) = y_i(b)$.

2: **neq – int32 scalar**

the number of differential equations.

3: **m(neq) – int32 array**

The order, m_i, of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

4: **nrbc – int32 scalar**

The number of boundary conditions at b.

Output Parameters

1: **gb(nrbc) – double array**

The values of $\bar{g}_i(z(y(b)))$, for $i = 1, 2, \ldots,$ **nrbc**.

5: **gajac – string containing name of m-file**

gajac must evaluate the partial derivatives of $g_i(z(y(a)))$ with respect to the elements of $z(y(a))$ $\left(= \left(y_1(a), y_1^1(a), \ldots, y_1^{(m_1-1)}(a), y_2(a), \ldots, y_n^{(m_n-1)}(a)\right)\right)$.

Its specification is:

```
[dgady] = gajac(ya, neq, m, nlbc)
```

Input Parameters

1: **ya(neq,0 : ∗) – double array**

The first dimension of the array **ya** must be at least

The second dimension of the array must be at least $maxdeg$

ya(i,j) contains $y_i^{(j)}(a)$, for $i = 1, 2, \ldots,$ **neq**, $j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(a) = y_i(a)$.

2: **neq – int32 scalar**

the number of differential equations.

3: **m(neq) – int32 array**

The order, m_i, of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

4: **nlbc – int32 scalar**

The number of boundary conditions at a.

Output Parameters

1: **dgady(nlbc,neq,0 : $*$) – double array**

Note: the last dimension of the array **dgady** must be at least $maxdeg$.

dgady(i,j,k) must contain the partial derivative of $g_i(z(y(a)))$ with respect to $y_j^{(k)}(a)$, for $i = 1, 2, \ldots,$ **nlbc**, $j = 1, 2, \ldots,$ **neq**, $k = 0, 1, \ldots,$ **m**$(j) - 1$. Only nonzero partial derivatives need be set.

6: **gbjac – string containing name of m-file**

gbjac must evaluate the partial derivatives of $\bar{g}_i(z(y(b)))$ with respect to the elements of $z(y(b))$ $\left(= \left(y_1(b), y_1^1(b), \ldots, y_1^{(m_1-1)}(b), y_2(b), \ldots, y_n^{(m_n-1)}(b)\right)\right)$.

Its specification is:

```
[dgbdy] = gbjac(yb, neq, m, nrbc)
```

Input Parameters

1: **yb(neq,0 : $*$) – double array**

The first dimension of the array **yb** must be at least

The second dimension of the array must be at least $maxdeg$

yb(i,j) contains $y_i^{(j)}(b)$, for $i = 1, 2, \ldots,$ **neq**, $j = 0, 1, \ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(b) = y_i(b)$.

2: **neq – int32 scalar**

the number of differential equations.

3: **m(neq) – int32 array**

The order, m_i, of the ith differential equation, for $i = 1, 2, \ldots,$ **neq**.

4: **nrbc – int32 scalar**

The number of boundary conditions at a.

Output Parameters

1: **dgbdy(nrbc,neq,0 : $*$) – double array**

Note: the last dimension of the array **dgbdy** must be at least $maxdeg$.

dgbdy(i,j,k) must contain the partial derivative of $\bar{g}_i(z(y(b)))$ with respect to $y_j^{(k)}(b)$, for $i = 1,2,\ldots,$ **nrbc**, $j = 1,2,\ldots,$ **neq**, $k = 0,1,\ldots,$ **m**$(j) - 1$. Only nonzero partial derivatives need be set.

7: **guess – string containing name of m-file**

guess must return initial approximations for the solution components $y_i^{(j)}$ and the derivatives $y_i^{(m_i)}$, for $i = 1,2,\ldots,$ **neq**, $j = 0,1,\ldots,$ **m**$(i) - 1$. Try to compute each derivative $y_i^{(m_i)}$ such that it corresponds to your approximations to $y_i^{(j)}$, for $j = 0,1,\ldots,$ **m**$(i) - 1$. You should **not** call user-supplied (sub)program **ffun** to compute $y_i^{(m_i)}$.

If d02tk is being used in conjunction with d02tx as part of a continuation process, then **guess** is not called by d02tk after the call to d02tx.

Its specification is:

```
[y, dym] = guess(x, neq, m)
```

Input Parameters

1: **x – double scalar**

The independent variable, x; $x \in [a,b]$.

2: **neq – int32 scalar**

the number of differential equations.

3: **m(neq) – int32 array**

The order, m_i, of the ith differential equation, for $i = 1,2,\ldots,$ **neq**.

Output Parameters

1: **y(neq,0 : $*$) – double array**

The first dimension of the array **y** must be at least

The second dimension of the array must be at least $maxdeg$

y(i,j) must contain $y_i^{(j)}(x)$, for $i = 1,2,\ldots,$ **neq**, $j = 0,1,\ldots,$ **m**$(i) - 1$.
Note: $y_i^{(0)}(x) = y_i(x)$.

2: **dym(neq) – double array**

dym(i) must contain $y_i^{(m_i)}(x)$, for $i = 1,2,\ldots,$ **neq**.

8: **work($*$) – double array**

Note: the dimension of the array **work** must be at least **lrwork** (see d02tv).

This must be the same array as supplied to d02tv and **must** remain unchanged between calls.

9: **iwork**(*) **– int32 array**

Note: the dimension of the array **iwork** must be at least **liwork** (see d02tv).

This must be the same array as supplied to d02tv and **must** remain unchanged between calls.

5.2 Optional Input Parameters

None.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **work**(*) **– double array**

Note: the dimension of the array **work** must be at least **lrwork** (see d02tv).

Contains information about the solution for use on subsequent calls to associated functions.

2: **iwork**(*) **– int32 array**

Note: the dimension of the array **iwork** must be at least **liwork** (see d02tv).

Contains information about the solution for use on subsequent calls to associated functions.

3: **ifail – int32 scalar**

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

6 Error Indicators and Warnings

Note: d02tk may return useful information for one or more of the following detected errors or warnings.

ifail = 1

On entry, an invalid call was made to d02tk, for example, without a previous call to the setup function d02tv.

ifail = 2

Numerical singularity has been detected in the Jacobian used in the underlying Newton iteration. No meaningful results have been computed. You should check carefully how you have coded user-supplied (sub)programs **fjac**, **gajac** and **gbjac**. If the user-supplied (sub)programs have been coded correctly then supplying a different initial approximation to the solution in **guess** might be appropriate. See also Section 8.

ifail = 3

The nonlinear iteration has failed to converge. At no time during the computation was convergence obtained and no meaningful results have been computed. You should check carefully how you have coded procedures user-supplied (sub)program **fjac**, user-supplied (sub)program **gajac** and user-supplied (sub)program **gbjac**. If the procedures have been coded correctly then supplying a better initial approximation to the solution in user-supplied (sub)program **guess** might be appropriate. See also Section 8.

ifail = 4

The nonlinear iteration has failed to converge. At some earlier time during the computation convergence was obtained and the corresponding results have been returned for diagnostic purposes and may be inspected by a call to d02tz. Nothing can be said regarding the suitability of these results for use in any subsequent computation for the same problem. You should try to provide a better mesh and initial approximation to the solution in user-supplied (sub)program **guess**. See also Section 8.

ifail = 5

The expected number of sub-intervals required exceeds the maximum number specified by the argument **mxmesh** in the setup function d02tv. Results for the last mesh on which convergence was obtained have been returned. Nothing can be said regarding the suitability of these results for use in any subsequent computation for the same problem. An indication of the error in the solution on the last mesh where convergence was obtained can be obtained by calling d02tz. The error requirements may need to be relaxed and/or the maximum number of mesh points may need to be increased. See also Section 8.

7 Accuracy

The accuracy of the solution is determined by the parameter **tols** in the prior call to d02tv (see Sections 3 and 8 of the document for d02tv for details and advice). Note that error control is applied only to solution components (variables) and not to any derivatives of the solution. An estimate of the maximum error in the computed solution is available by calling d02tz.

8 Further Comments

If d02tk returns with **ifail** = 2, 3, 4 or 5 and the call to d02tk was a part of some continuation procedure for which successful calls to d02tk have already been made, then it is possible that the adjustment(s) to the continuation parameter(s) between calls to d02tk is (are) too large for the problem under consideration. More conservative adjustment(s) to the continuation parameter(s) might be appropriate.

9 Example

d02tx_ffun.m

```
function [f] = ffun(x, y, neq, m)
  global el;
  global en;
  global s;
  f = zeros(neq, 1);

      f(1)   =   el^3*(1-y(2,1)^2)   +   el^2*s*y(1,2)   -   el*(0.5*(3-
en)*y(1,1)*y(1,3)+en*y(1,2)^2);
    f(2)  =  el^2*s*(y(2,1)-1)  -  el*(0.5*(3-en)*y(1,1)*y(2,2)  +(en-
1)*y(1,2)*y(2,1));
```

d02tx_fjac.m

```
function [dfdy] = fjac(x, y, neq, m)
  global el;
  global en;
  global s;
  dfdy = zeros(neq, neq, 3);

  dfdy(1,2,1) = -2.0*el^3*y(2,1);
  dfdy(1,1,1) = -el*0.5*(3.0-en)*y(1,3);
  dfdy(1,1,2) = el^2*s - el*2.0*en*y(1,2);
  dfdy(1,1,3) = -el*0.5*(3.0-en)*y(1,1);
  dfdy(2,2,1) = el^2*s - el*(en-1.0)*y(1,2);
  dfdy(2,2,2) = -el*0.5*(3.0-en)*y(1,1);
  dfdy(2,1,1) = -el*0.5*(3.0-en)*y(2,2);
  dfdy(2,1,2) = -el*(en-1.0)*y(2,1);
```

d02tx_gafun.m

```
function [ga] = gafun(ya, neq, m, nlbc)
  global el;
  global en;
  global s;
  ga = zeros(nlbc, 1);

  ga(1) = ya(1,1);
  ga(2) = ya(1,2);
  ga(3) = ya(2,1);
```

d02tx_gajac.m

```
function [dgady] = gajac(ya, neq, m, nlbc)
  global el;
  global en;
  global s;
  dgady = zeros(nlbc, neq, 3);

  dgady(1,1,1) = 1;
  dgady(2,1,2) = 1;
  dgady(3,2,1) = 1;
```

d02tx_gbfun.m

```
function [gb] = gbfun(yb, neq, m, nrbc)
  global el;
  global en;
  global s;
  gb = zeros(nrbc, 1);

  gb(1) = yb(1,2);
  gb(2) = yb(2,1) - 1;
```

d02tx_gbjac.m

```
function [dgbdy] = gbjac(yb, neq, m, nrbc)
  global el;
  global en;
  global s;
  dgbdy = zeros(nrbc, neq, 3);

  dgbdy(1,1,2) = 1;
  dgbdy(2,2,1) = 1;
```

d02tx_guess.m

```
function [y, dym] = guess(x, neq, m)
  global el;
  global en;
  global s;
  y = zeros(neq, 3);
  dym = zeros(neq, 1);

  ex = x*el;
  expmx = exp(-ex);
  y(1,1) = -ex^2*expmx;
  y(1,2) = (-2*ex+ex^2)*expmx;
  y(1,3) = (-2+4*ex-ex^2)*expmx;
  y(2,1) = 1 - expmx;
  y(2,2) = expmx;
  dym(1) = (6-6*ex+ex^2)*expmx;
```

```
  dym(2) = -expmx;
```

```
m = [int32(3); int32(2)];
nlbc = int32(3);
nrbc = int32(2);
ncol = int32(6);
neq = int32(2);
tols = [0.00001; 0.00001];
nmesh = int32(21);
mxmesh = int32(250);
mesh = zeros(250,1);
ipmesh = zeros(250,1, 'int32');
ipmesh(1) = int32(1);
for i=2:nmesh
  mesh(i) = double(i-1)/double(nmesh-1);
  ipmesh(i) = int32(2);
end
ipmesh(nmesh) = int32(1);

global el;
el = 60;
global s;
s = 0.24;
global en;
en = 0.2;

ncont = int32(3);
mmax  = int32(3);

% Initialise
[work, iwork, ifail] = d02tv(m, nlbc, nrbc, ncol, tols, nmesh, mesh,
ipmesh);
% Solve
for j = 1:ncont
  fprintf('\n Tolerance = %8.1e, L = %8.3f, S = %6.4f\n\n', tols(1), el,
s);
  [work, iwork, ifail] = ...
    d02tk('d02tx_ffun', 'd02tx_fjac', 'd02tx_gafun', 'd02tx_gbfun', ...
    'd02tx_gajac', 'd02tx_gbjac', 'd02tx_guess', work, iwork);
  % Extract Mesh
  [nmesh, mesh, ipmesh, ermx, iermx, ijermx, ifail] = ...
    d02tz(mxmesh, work, iwork);
  fprintf(' Used a mesh of %d points\n', nmesh);
  fprintf(' Maximum error = %10.2e in interval %d for component %d\n\n',
...
    ermx, iermx, ijermx);
  % Print solution components on mesh
  fprintf(' Solution on original interval:\n');
  fprintf('      x        f          g\n');
  for i=1:16
    xx = double(i-1)*2/el;
    [y, work, ifail] = d02ty(xx, neq, mmax, work, iwork);
    fprintf(' %2.8f%11.4f%11.4f\n', xx*el, y(1,1), y(2,1));
  end
  for i=1:10
    xx = (30+(el-30)*double(i)/10)/el;
    [y, work, ifail] = d02ty(xx, neq, mmax, work, iwork);
    fprintf(' %2.8f%11.4f%11.4f\n', xx*el, y(1,1), y(2,1));
  end
  % Select Mesh for continuation
  if j < ncont
    el = 2*el;
    s = 0.6*s;
    nmesh = (nmesh+1)/2;
    [work, iwork, ifail] = d02tx(nmesh, mesh, ipmesh, work, iwork);
  end
end
```

```
Tolerance =  1.0e-05, L =   60.000, S = 0.2400

 Used a mesh of 21 points
 Maximum error =   2.66e-08 in interval 7 for component 1

 Solution on original interval:
      x        f          g
 0.00000000     0.0000     0.0000
 2.00000000    -0.9769     0.8011
 4.00000000    -2.0900     1.1459
 6.00000000    -2.6093     1.2389
 8.00000000    -2.5498     1.1794
 10.00000000    -2.1397     1.0478
 12.00000000    -1.7176     0.9395
 14.00000000    -1.5465     0.9206
 16.00000000    -1.6127     0.9630
 18.00000000    -1.7466     1.0068
 20.00000000    -1.8286     1.0244
 22.00000000    -1.8338     1.0185
 24.00000000    -1.7956     1.0041
 26.00000000    -1.7582     0.9940
 28.00000000    -1.7445     0.9926
 30.00000000    -1.7515     0.9965
 33.00000000    -1.7695     1.0019
 36.00000000    -1.7730     1.0018
 39.00000000    -1.7673     0.9998
 42.00000000    -1.7645     0.9993
 45.00000000    -1.7659     0.9999
 48.00000000    -1.7672     1.0002
 51.00000000    -1.7671     1.0001
 54.00000000    -1.7666     0.9999
 57.00000000    -1.7665     0.9999
 60.00000000    -1.7666     1.0000

 Tolerance =  1.0e-05, L =  120.000, S = 0.1440

 Used a mesh of 21 points
 Maximum error =   6.88e-06 in interval 7 for component 2

 Solution on original interval:
      x        f          g
 0.00000000     0.0000     0.0000
 2.00000000    -1.1406     0.7317
 4.00000000    -2.6531     1.1315
 6.00000000    -3.6721     1.3250
 8.00000000    -4.0539     1.3707
 10.00000000    -3.8285     1.3003
 12.00000000    -3.1339     1.1407
 14.00000000    -2.2469     0.9424
 16.00000000    -1.6146     0.8201
 18.00000000    -1.5472     0.8549
 20.00000000    -1.8483     0.9623
 22.00000000    -2.1761     1.0471
 24.00000000    -2.3451     1.0778
 26.00000000    -2.3236     1.0600
 28.00000000    -2.1784     1.0165
 30.00000000    -2.0214     0.9775
 39.00000000    -2.1109     1.0155
 48.00000000    -2.0362     0.9931
 57.00000000    -2.0709     1.0023
 66.00000000    -2.0588     0.9995
 75.00000000    -2.0616     1.0000
 84.00000000    -2.0615     1.0001
 93.00000000    -2.0611     0.9999
 102.00000000    -2.0614     1.0000
 111.00000000    -2.0613     1.0000
 120.00000000    -2.0613     1.0000

 Tolerance =  1.0e-05, L =  240.000, S = 0.0864
```

```
Used a mesh of 81 points
Maximum error =   3.30e-07 in interval 19 for component 2

Solution on original interval:
     x        f          g
0.00000000     0.0000     0.0000
2.00000000    -1.2756     0.6404
4.00000000    -3.1604     1.0463
6.00000000    -4.7459     1.3011
8.00000000    -5.8265     1.4467
10.00000000    -6.3412     1.5036
12.00000000    -6.2862     1.4824
14.00000000    -5.6976     1.3886
16.00000000    -4.6568     1.2263
18.00000000    -3.3226     1.0042
20.00000000    -2.0328     0.7718
22.00000000    -1.4035     0.6943
24.00000000    -1.6603     0.8218
26.00000000    -2.2975     0.9928
28.00000000    -2.8661     1.1139
30.00000000    -3.1641     1.1641
51.00000000    -2.5307     1.0279
72.00000000    -2.3520     0.9919
93.00000000    -2.3674     0.9975
114.00000000    -2.3799     1.0003
135.00000000    -2.3800     1.0002
156.00000000    -2.3792     1.0000
177.00000000    -2.3791     1.0000
198.00000000    -2.3792     1.0000
219.00000000    -2.3792     1.0000
240.00000000    -2.3792     1.0000
```