NAG Toolbox for MATLAB

d02tv

1 Purpose

d02tv is a setup function which must be called prior to the first call of the nonlinear two-point boundary-value solver d02tk.

2 Syntax

```
[rwork, iwork, ifail] = d02tv(m, nlbc, nrbc, ncol, tols, nmesh, mesh,
ipmesh, 'neq', neq, 'mxmesh', mxmesh, 'lrwork', lrwork, 'liwork',
liwork)
```

3 Description

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

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

over an interval $[a, b]$ subject to p (> 0) nonlinear boundary conditions at a and q (> 0) nonlinear boundary conditions at b, where $p + q = \sum_{i=1}^{n} m_i$. 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

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

See Section 8 for information on how boundary-value problems of a more general nature can be treated.

d02tv is used to specify an initial mesh, error requirements and other details. d02tk is then used to solve the boundary-value problem.

The solution function d02tk proceeds as follows. A modified Newton method is applied to the equations

$$y_i^{(m_i)}(x) - f_i(x, z(y(x))) = 0, \qquad i = 1, \ldots, n$$

and the boundary conditions. To solve these equations numerically the components y_i are approximated by piecewise polynomials v_{ij} using a monomial basis on the jth mesh sub-interval. The coefficients of the polynomials v_{ij} form the unknowns to be computed. Collocation is applied at Gaussian points

$$v_{ij}^{(m_i)}\left(x_{jk}\right) - f_i\left(x_{jk}, z\left(v\left(x_{jk}\right)\right)\right) = 0, \qquad i = 1, \ldots, n,$$

where x_{jk} is the kth collocation point in the jth mesh sub-interval. Continuity at the mesh points is imposed, that is

$$v_{ij}\left(x_{j+1}\right) - v_{i,j+1}\left(x_{j+1}\right) = 0, \qquad i = 1, 2, \ldots, n,$$

where x_{j+1} is the right-hand end of the jth mesh sub-interval. The linearized collocation equations and boundary conditions, together with the continuity conditions, form a system of linear algebraic equations which are solved using f01lh and f04lh. For use in the modified Newton method, an approximation to the solution on the initial mesh must be supplied via the procedure argument user-supplied (sub)program **guess** of d02tk.

The solver attempts to satisfy the conditions

$$\frac{\|y_i - v_i\|}{(1.0 + \|v_i\|)} \leq \mathbf{tols}(i), \qquad i = 1, 2, \ldots, n, \tag{1}$$

where v_i is the approximate solution for the ith solution component and **tols** is supplied by you. The mesh is refined by trying to equidistribute the estimated error in the computed solution over all mesh sub-intervals, and an extrapolation-like test (doubling the number of mesh sub-intervals) is used to check for (1).

The 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

Gill P E, Murray W and Wright M H 1981 *Practical Optimization* Academic Press

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

Schwartz I B 1983 Estimating regions of existence of unstable periodic orbits using computer-based techniques *SIAM J. Sci. Statist. Comput.* **20(1)** 106–120

5 Parameters

5.1 Compulsory Input Parameters

1: **m(neq) – int32 array**

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

Constraint: $1 \leq \mathbf{m}(i) \leq 4$, for $i = 1, 2, \ldots, n$.

2: **nlbc – int32 scalar**

p, the number of left boundary conditions defined at the left-hand end, a ($= \mathbf{mesh}(1)$).

Constraint: $\mathbf{nlbc} \geq 1$.

3: **nrbc – int32 scalar**

q, the number of right boundary conditions defined at the right-hand end, b ($= \mathbf{mesh}(\mathbf{nmesh})$).

Constraints:

$\mathbf{nrbc} \geq 1$;

$\mathbf{nlbc} + \mathbf{nrbc} = \sum_{i=1}^{n} \mathbf{m}(i)$.

4: **ncol – int32 scalar**

The number of collocation points to be used in each mesh sub-interval.

Constraint: $m_{\max} \leq$ **ncol** ≤ 7, where $m_{\max} = \max(\mathbf{m}(i))$.

5: **tols(neq) – double array**

The error requirement for the ith solution component.

Constraint: $100 \times$ ***machine precision*** $<$ **tols**$(i) < 1.0$, for $i = 1, 2, \dots, n$.

6: **nmesh – int32 scalar**

The number of points to be used in the initial mesh of the solution process.

Constraint: **nmesh** ≥ 6.

7: **mesh(mxmesh) – double array**

The positions of the initial **nmesh** mesh points. The remaining elements of **mesh** need not be set. You should try to place the mesh points in areas where you expect the solution to vary most rapidly. In the absence of any other information the points should be equally distributed on $[a, b]$.

mesh(1) must contain the left boundary point, a, and **mesh**(**nmesh**) must contain the right boundary point, b.

Constraint: **mesh**$(i) <$ **mesh**$(i+1)$, for $i = 1, 2, \dots,$ **nmesh** $- 1$.

8: **ipmesh(mxmesh) – int32 array**

ipmesh(i) specifies whether or not the initial mesh point defined in **mesh**(i), for $i = 1, \dots,$ **nmesh**, should be a fixed point in all meshes computed during the solution process. The remaining elements of **ipmesh** need not be set.

ipmesh$(i) = 1$

Indicates that **mesh**(i) should be a fixed point in all meshes.

ipmesh$(i) = 2$

Indicates that **mesh**(i) is not a fixed point.

Constraints:

ipmesh$(1) = 1$ and **ipmesh**(**nmesh**) $= 1$, (i.e., the left and right boundary points, a and b, must be fixed points, in all meshes);
ipmesh$(i) = 1$ or 2, for $i = 2, 3, \dots,$ **nmesh** $- 1$.

5.2 Optional Input Parameters

1: **neq – int32 scalar**

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

n, the number of ordinary differential equations to be solved.

Constraint: **neq** ≥ 1.

2: **mxmesh – int32 scalar**

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

the maximum number of mesh points to be used during the solution process.

Constraint: **mxmesh** $\geq 2 \times$ **nmesh** $- 1$.

3: **lrwork – int32 scalar**

Default: The dimension of the array **rwork**.

Suggested value: $\mathbf{lrwork} = \mathbf{mxmesh} \times \left(109 \times \mathbf{neq}^2 + 78 \times \mathbf{neq} + 7\right)$, which will permit **mxmesh** mesh points for a system of **neq** differential equations regardless of their order or the number of collocation points used.

Default: $\mathbf{mxmesh} \times \left(109 \times \mathbf{neq}^2 + 78 \times \mathbf{neq} + 7\right)$

Constraint: $\mathbf{lrwork} \geq 50 + \mathbf{neq} \times (m_{\max} \times (1 + \mathbf{neq} + \max(\mathbf{nlbc}, \mathbf{nrbc})) + 6) - k_n \times (k_n + 6) - m^* \times (k_n + m^* - 2) + \mathbf{mxmesh} \times ((m^* + 3)(2m^* + 3) - 3 + k_n(k_n + m^* + 6)) + \mathbf{mxmesh}/2$, where $m^* = \sum_{i=1}^{n} \mathbf{m}(i)$ and $k_n = \mathbf{ncol} \times \mathbf{neq}$.

4: **liwork – int32 scalar**

Default: The dimension of the array **iwork**.

Suggested value: $\mathbf{liwork} = \mathbf{mxmesh} \times (11 \times \mathbf{neq} + 6)$, which will permit **mxmesh** mesh points for a system of **neq** differential equations regardless of their order or the number of collocation points used.

Default: $\mathbf{mxmesh} \times (11 \times \mathbf{neq} + 6)$

Constraint: $\mathbf{liwork} \geq 23 + 3 \times \mathbf{neq} - k_n + \mathbf{mxmesh} \times (m^{\times} + k_n + 4)$.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **rwork(lrwork) – double array**

Contains information for use by d02tk. This **must** be the same array as will be supplied to d02tk. The contents of this array **must** remain unchanged between calls.

2: **iwork(liwork) – int32 array**

Contains information for use by d02tk. This **must** be the same array as will be supplied to d02tk. The contents of this array **must** remain unchanged between calls.

3: **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:

$\mathbf{ifail} = 1$

On entry, $\mathbf{neq} < 1$,
or $\mathbf{m}(i) < 1$, for some i,
or $\mathbf{m}(i) > 4$, for some i,
or $\mathbf{nmesh} < 6$,
or the values of **mesh** are not strictly increasing,
or $\mathbf{ipmesh}(i)$ is invalid for some i,
or $\mathbf{mxmesh} < 2 \times \mathbf{nmesh} - 1$,
or $\mathbf{ncol} < m_{\max}$, where $m_{\max} = \max(\mathbf{m}(i))$,
or $\mathbf{ncol} > 7$,
or $\mathbf{nlbc} < 1$,

or **nrbc** < 1,
or a value of **tols** is invalid,
or $\textbf{nlbc} + \textbf{nrbc} \neq \sum_{i=1}^{n} \textbf{m}(i)$,
or **lrwork** or **liwork** is too small.

7 Accuracy

Not applicable.

8 Further Comments

For problems where sharp changes of behaviour are expected over short intervals it may be advisable to:

– use a large value for **ncol**;

– cluster the initial mesh points where sharp changes in behaviour are expected;

– maintain fixed points in the mesh using the parameter **ipmesh** to ensure that the remeshing process does not inadvertently remove mesh points from areas of known interest before they are detected automatically by the algorithm.

8.1 Nonseparated Boundary Conditions

A boundary-value problem with nonseparated boundary conditions can be treated by transformation to an equivalent problem with separated conditions. As a simple example consider the system

$$y_1' = f_1(x, y_1, y_2)$$

$$y_2' = f_2(x, y_1, y_2)$$

on $[a, b]$ subject to the boundary conditions

$$\begin{aligned} g_1(y_1(a)) &= 0 \\ g_2(y_2(a), y_2(b)) &= 0. \end{aligned}$$

By adjoining the trivial ordinary differential equation

$$r' = 0,$$

which implies $r(a) = r(b)$, and letting $r(b) = y_2(b)$, say, we have a new system

$$\begin{aligned} y_1' &= f_1(x, y_1, y_2) \\ y_2' &= f_2(x, y_1, y_2) \\ r' &= 0, \end{aligned}$$

subject to the separated boundary conditions

$$\begin{aligned} g_1(y_1(a)) &= 0 \\ g_2(y_2(a), r(a)) &= 0 \\ y_2(b) - r(b) &= 0. \end{aligned}$$

There is an obvious overhead in adjoining an extra differential equation: the system to be solved is increased in size.

8.2 Multipoint Boundary Value Problems

Multipoint boundary-value problems, that is problems where conditions are specified at more than two points, can also be transformed to an equivalent problem with two boundary points. Each sub-interval defined by the multipoint conditions can be transformed onto the interval $[0, 1]$, say, leading to a larger set of differential equations. The boundary conditions of the transformed system consist of the original boundary conditions and the conditions imposed by the requirement that the solution components be continuous at the interior break points. For example, consider the equation

$$y^{(3)} = f\left(t, y, y^{(1)}, y^{(2)}\right) \qquad \text{on} \qquad [a, c]$$

subject to the conditions

$$\begin{aligned} y(a) &= A \\ y(b) &= B \\ y^{(1)}(c) &= C \end{aligned}$$

where $a < b < c$. This can be transformed to the system

$$\left.\begin{aligned} y_1^{(3)} &= f\left(t, y_1, y_1^{(1)}, y_1^{(2)}\right) \\ y_2^{(3)} &= f\left(t, y_2, y_2^{(1)}, y_2^{(2)}\right) \end{aligned}\right\} \quad \text{on } [0, 1]$$

where

$$\begin{aligned} y_1 &\equiv y \quad \text{on} \quad [a, b] \\ y_2 &\equiv y \quad \text{on} \quad [b, c], \end{aligned}$$

subject to the boundary conditions

$$\begin{aligned} y_1(0) &= A \\ y_1(1) &= B \\ y_2^{(1)}(1) &= C \\ y_2(0) &= B \quad (\text{from } y_1(1) = y_2(0)) \\ y_1^{(1)}(1) &= y_2^{(1)}(0) \\ y_1^{(2)}(1) &= y_2^{(2)}(0). \end{aligned}$$

In this instance two of the resulting boundary conditions are nonseparated but they may next be treated as described above.

8.3 High Order Systems

Systems of ordinary differential equations containing derivatives of order greater than four can always be reduced to systems of order suitable for treatment by d02tv and its related functions. For example suppose we have the sixth-order equation

$$y^{(6)} = -y.$$

Writing the variables $y_1 = y$ and $y_2 = y^{(4)}$ we obtain the system

$$\begin{aligned} y_1^{(4)} &= y_2 \\ y_2^{(2)} &= -y_1 \end{aligned}$$

which has maximal order four, or writing the variables $y_1 = y$ and $y_2 = y^{(3)}$ we obtain the system

$$\begin{aligned} y_1^{(3)} &= y_2 \\ y_2^{(3)} &= -y_1 \end{aligned}$$

which has maximal order three. The best choice of reduction by choosing new variables will depend on the structure and physical meaning of the system. Note that you will control the error in each of the variables y_1 and y_2. Indeed, if you wish to control the error in certain derivatives of the solution of an equation of order greater than one, then you should make those derivatives new variables.

8.4 Fixed Points and Singularities

The solver function d02tk employs collocation at Gaussian points in each sub-interval of the mesh. Hence the coefficients of the differential equations are not evaluated at the mesh points. Thus, fixed points should be specified in the mesh where either the coefficients are singular, or the solution has less smoothness, or where the differential equations should not be evaluated. Singular coefficients at boundary points often arise when physical symmetry is used to reduce partial differential equations to ordinary differential equations. These do not pose a direct numerical problem for using this code but they can severely impact its convergence.

8.5 Numerical Jacobians

The solver function d02tk requires an external procedure user-supplied (sub)program **fjac** to evaluate the partial derivatives of f_i with respect to the elements of $z(y)$ $\left(= \left(y_1, y_1^1, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right)\right)$. In cases where the partial derivatives are difficult to evaluate, numerical approximations can be used. However, this approach might have a negative impact on the convergence of the modified Newton method. You could consider the use of symbolic mathematic packages and/or automatic differentiation packages if available to you.

See Section 9 of the document for d02tz for an example using numerical approximations to the Jacobian. There central differences are used and each f_i is assumed to depend on all the components of z. This requires two evaluations of the system of differential equations for each component of z. The perturbation used depends on the size of each component of z and a minimum quantity dependent on the ***machine precision***. The cost of this approach could be reduced by employing an alternative difference scheme and/or by only perturbing the components of z which appear in the definitions of the f_i. A discussion on the choice of perturbation factors for use in finite difference approximations to partial derivatives can be found in Gill *et al.* 1981.

9 Example

```
m = ones(6, 1, 'int32');
nlbc = int32(3);
nrbc = int32(3);
ncol = int32(5);
tols = [1e-05; 1e-05; 1e-05; 1e-05; 1e-05; 1e-05];
nmesh = int32(11);
mesh = zeros(100, 1);
ipmesh = zeros(100, 1, 'int32');
for i=1:nmesh
  mesh(i) = double(i-1)/10;
  ipmesh(i) = int32(2);
end
ipmesh(1) = int32(1);
ipmesh(nmesh) = int32(1);
[rwork, iwork, ifail] = d02tv(m, nlbc, nrbc, ncol, tols, nmesh, mesh,
ipmesh)
```

```
rwork =
     array elided
iwork =
     array elided
ifail =
           0
```