NAG Toolbox for MATLAB

d02ej

1 Purpose

d02ej integrates a stiff system of first-order ordinary differential equations over an interval with suitable initial conditions, using a variable-order, variable-step method implementing the Backward Differentiation Formulae (BDF), until a user-specified function, if supplied, of the solution is zero, and returns the solution at points specified by you, if desired.

2 Syntax

```
[x, y, tol, ifail] = d02ej(x, xend, y, fcn, pederv, tol, relabs, output, g, 'n', n)
```

3 Description

d02ej advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x, y_1, y_2, \ldots, y_n), \qquad i = 1, 2, \ldots, n,$$

from $x =$ **x** to $x =$ **xend** using a variable-order, variable-step method implementing the BDF. The system is defined by user-supplied (sub)program **fcn**, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_n$ (see Section 5). The initial values of $y_1, y_2, \ldots, y_n$ must be given at $x =$ **x**.

The solution is returned via the (sub)program **output** at points specified by you, if desired: this solution is obtained by C^1 interpolation on solution values produced by the method. As the integration proceeds a check can be made on the user-specified function $g(x, y)$ to determine an interval where it changes sign. The position of this sign change is then determined accurately by C^1 interpolation to the solution. It is assumed that $g(x, y)$ is a continuous function of the variables, so that a solution of $g(x, y) = 0.0$ can be determined by searching for a change in sign in $g(x, y)$. The accuracy of the integration, the interpolation and, indirectly, of the determination of the position where $g(x, y) = 0.0$, is controlled by the parameters **tol** and **relabs**. The Jacobian of the system $y' = f(x, y)$ may be supplied in (sub)program **pederv**, if it is available.

For a description of BDF and their practical implementation see Hall and Watt 1976.

4 References

Hall G and Watt J M (ed.) 1976 *Modern Numerical Methods for Ordinary Differential Equations* Clarendon Press, Oxford

5 Parameters

5.1 Compulsory Input Parameters

1: **x – double scalar**

The initial value of the independent variable x.

Constraint: **x** $\neq$ **xend**.

2: **xend – double scalar**

The final value of the independent variable. If **xend** $<$ **x**, integration will proceed in the negative direction.

Constraint: **xend** $\neq$ **x**.

3: **y(n) – double array**

The initial values of the solution $y_1, y_2, \ldots, y_n$ at $x = \mathbf{x}$.

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

fcn must evaluate the functions f_i (i.e., the derivatives y_i') for given values of its arguments $x, y_1, \ldots, y_n$.

Its specification is:

```
[f] = fcn(x, y)
```

Input Parameters

1: **x – double scalar**

The value of the independent variable x.

2: **y(n) – double array**

The value of the variable y_i, for $i = 1, 2, \ldots, n$.

Output Parameters

1: **f(n) – double array**

The value of f_i, for $i = 1, 2, \ldots, n$.

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

pederv must evaluate the Jacobian of the system (that is, the partial derivatives $\frac{\partial f_i}{\partial y_j}$) for given values of the variables $x, y_1, y_2, \ldots, y_n$.

Its specification is:

```
[pw] = pederv(x, y)
```

Input Parameters

1: **x – double scalar**

The value of the independent variable x.

2: **y(n) – double array**

The value of the variable y_i, for $i = 1, 2, \ldots, n$.

Output Parameters

1: **pw(n,n) – double array**

The value of $\frac{\partial f_i}{\partial y_j}$, for $i, j = 1, 2, \ldots, n$.

If you do not wish to supply the Jacobian, the actual parameter **pederv must** be the string `'d02ejy'`. **d02ejy** is included in the NAG Fortran Library.

6: **tol – double scalar**

Must be set to a **positive** tolerance for controlling the error in the integration. Hence **tol** affects the determination of the position where $g(x, y) = 0.0$, if user-supplied real function **g** is supplied.

d02ej has been designed so that, for most problems, a reduction in **tol** leads to an approximately proportional reduction in the error in the solution. However, the actual relation between **tol** and the accuracy achieved cannot be guaranteed. You are strongly recommended to call d02ej with more than one value for **tol** and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge, you might compare the results obtained by calling d02ej with **tol** $= 10^{-p}$ and **tol** $= 10^{-p-1}$ if p correct decimal digits are required in the solution.

Constraint: **tol** > 0.0.

7: **relabs – string**

The type of error control. At each step in the numerical solution an estimate of the local error, est, is made. For the current step to be accepted the following condition must be satisfied:

$$est = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(e_i/(\tau_r \times |y_i| + \tau_a)\right)^2} \leq 1.0$$

where τ_r and τ_a are defined by

relabs	τ_r	τ_a
'M'	**tol**	**tol**
'A'	0.0	**tol**
'R'	**tol**	ϵ
'D'	**tol**	ϵ

where ϵ is a small machine-dependent number and e_i is an estimate of the local error at y_i, computed internally. If the appropriate condition is not satisfied, the step size is reduced and the solution is recomputed on the current step. If you wish to measure the error in the computed solution in terms of the number of correct decimal places, then **relabs** should be set to 'A' on entry, whereas if the error requirement is in terms of the number of correct significant digits, then **relabs** should be set to 'R'. If you prefer a mixed error test, then **relabs** should be set to 'M', otherwise if you have no preference, **relabs** should be set to the default 'D'. Note that in this case 'D' is taken to be 'R'.

Constraint: **relabs** = 'A', 'M', 'R' or 'D'.

8: **output – string containing name of m-file**

output permits access to intermediate values of the computed solution (for example to print or plot them), at successive user-specified points. It is initially called by d02ej with **xsol** $=$ **x** (the initial value of x). You must reset **xsol** to the next point (between the current **xsol** and **xend**) where **output** is to be called, and so on at each call to **output**. If, after a call to **output**, the reset point **xsol** is beyond **xend**, d02ej will integrate to **xend** with no further calls to **output**; if a call to **output** is required at the point **xsol** $=$ **xend**, then **xsol** must be given precisely the value **xend**.

Its specification is:

```
[xsol] = output(xsol, y)
```

Input Parameters

1: **xsol – double scalar**

The value of the independent variable x.

You must set **xsol** to the next value of x at which **output** is to be called.

2: **y**(n) **– double array**

The computed solution at the point **xsol**.

Output Parameters

1: **xsol – double scalar**

The value of the independent variable x.

You must set **xsol** to the next value of x at which **output** is to be called.

If you do not wish to access intermediate output, the actual parameter **output must** be the string `'d02ejx'`. **d02ejx** is included in the NAG Fortran Library.

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

g must evaluate the function $g(x,y)$ for specified values x,y. It specifies the function g for which the first position x where $g(x,y) = 0$ is to be found.

Its specification is:

```
[result] = g(x, y)
```

Input Parameters

1: **x – double scalar**

The value of the independent variable x.

2: **y**(n) **– double array**

The value of the variable y_i, for $i = 1,2,\ldots,n$.

Output Parameters

1: **result – double scalar**

The result of the function.

If you do not require the root-finding option, the actual parameter **g must** be the string `'d02ejw'`. **d02ejw** is included in the NAG Fortran Library.

5.2 Optional Input Parameters

1: **n – int32 scalar**

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

n, the number of differential equations.

Constraint: **n** ≥ 1.

5.3 Input Parameters Omitted from the MATLAB Interface

w, iw

5.4 Output Parameters

1: **x – double scalar**

If user-supplied real function **g** is supplied by you, **x** contains the point where $g(x,y) = 0.0$, unless $g(x,y) \neq 0.0$ anywhere on the range **x** to **xend**, in which case, **x** will contain **xend**. If **g** is not supplied **x** contains **xend**, unless an error has occurred, when it contains the value of x at the error.

2: **y**$(\mathbf{n})$ **– double array**

The computed values of the solution at the final point $x =$ **x**.

3: **tol – double scalar**

Normally unchanged. However if the range **x** to **xend** is so short that a small change in **tol** is unlikely to make any change in the computed solution, then, on return, **tol** has its sign changed.

4: **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 $= 1$

On entry, **tol** ≤ 0.0,
or **x** $=$ **xend**,
or **n** ≤ 0,
or **relabs** $\neq$ 'M','A','R','D',
or **iw** $< (12 + \mathbf{n}) \times \mathbf{n} + 50$.

ifail $= 2$

With the given value of **tol**, no further progress can be made across the integration range from the current point $x = \mathbf{x}$. (See Section 5 for a discussion of this error test.) The components $\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(n)$ contain the computed values of the solution at the current point $x = \mathbf{x}$. If you have supplied user-supplied real function **g**, then no point at which $g(x, y)$ changes sign has been located up to the point $x = \mathbf{x}$.

ifail $= 3$

tol is too small for d02ej to take an initial step. **x** and $\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(n)$ retain their initial values.

ifail $= 4$

xsol lies behind **x** in the direction of integration, after the initial call to (sub)program **output**, if the **output** option was selected.

ifail $= 5$

A value of **xsol** returned by the (sub)program **output** lies behind the last value of **xsol** in the direction of integration, if the **output** option was selected.

ifail $= 6$

At no point in the range **x** to **xend** did the function $g(x, y)$ change sign, if user-supplied real function **g** was supplied. It is assumed that $g(x, y) = 0$ has no solution.

ifail $= 7$ (c05az)

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

ifail $= 8$ (d02xk)

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

ifail $= 9$

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

7 Accuracy

The accuracy of the computation of the solution vector **y** may be controlled by varying the local error tolerance **tol**. In general, a decrease in local error tolerance should lead to an increase in accuracy. You are advised to choose **relabs** = 'R' unless you have a good reason for a different choice. It is particularly appropriate if the solution decays.

If the problem is a root-finding one, then the accuracy of the root determined will depend strongly on $\frac{\partial g}{\partial x}$ and $\frac{\partial g}{\partial y_i}$, for $i = 1, 2, \ldots, n$. Large values for these quantities may imply large errors in the root.

8 Further Comments

If more than one root is required, then to determine the second and later roots d02ej may be called again starting a short distance past the previously determined roots. Alternatively you may construct your own root-finding code using d02nb (and other functions in sub-chapter D02M/N), d02xk and c05az.

If it is easy to code, you should supply the (sub)program **pederv**. However, it is important to be aware that if **pederv** is coded incorrectly, a very inefficient integration may result and possibly even a failure to complete the integration (see **ifail** = 2).

9 Example

d02ej_fcn.m

```
function f = fcn(x, y)
  f = zeros(3,1);
  f(1) = -0.04*y(1) + 1.0d4*y(2)*y(3);
  f(2) = 0.04*y(1) - 1.0d4*y(2)*y(3) - 3.0d7*y(2)*y(2);
  f(3) = 3.0d7*y(2)*y(2);
```

d02ej_g.m

```
function result = g(x, y)
  result = y(1) - 0.9;
```

d02ej_output.m

```
function xsolOut = output(xsol, y)
  fprintf('%3.5f %+3.5f %+3.5f %+3.5f\n', xsol, y(1), y(2), y(3));
  xsolOut = xsol + 2;
```

d02ej_pederv.m

```
function pw = pederv(x, y)
  pw = zeros(3,3);
  pw(1,1) = -0.04d0;
  pw(1,2) = 1.0d4*y(3);
  pw(1,3) = 1.0d4*y(2);
  pw(2,1) = 0.04d0;
  pw(2,2) = -1.0d4*y(3) - 6.0d7*y(2);
  pw(2,3) = -1.0d4*y(2);
  pw(3,1) = 0.0d0;
  pw(3,2) = 6.0d7*y(2);
  pw(3,3) = 0.0d0;
```

```
x = 0;
```

```
xend = 10;
y = [1;
     0;
     0];
tol = 0.001;
relabs = 'Default';
[xOut, yOut, tolOut, ifail] = ...
         d02ej(x, xend, y, 'd02ej_fcn', 'd02ej_pederv', tol, relabs,
'd02ej_output', 'd02ej_g')
```

```
0.00000 +1.00000 +0.00000 +0.00000
2.00000 +0.94163 +0.00003 +0.05834
4.00000 +0.90551 +0.00002 +0.09447
xOut =
    4.3767
yOut =
    0.9000
    0.0000
    0.1000
tolOut =
   1.0000e-03
ifail =
           0
```