NAG Toolbox for MATLAB

d02nv

1 Purpose

d02nv is a setup function which must be must be called prior to an integrator in sub-chapter D02M/N, if Backward Differentiation Formulae (BDF) are to be used.

2 Syntax

```
[con, rwork, ifail] = d02nv(ldysav, sdysav, maxord, method, petzld, con,
tcrit, hmin, hmax, h0, maxstp, mxhnil, norm_p, rwork)
```

3 Description

An integrator setup function must be called before the call to any integrator in this sub-chapter. This setup function, d02nv, makes the choice of the BDF integrator and permits you to define options appropriate to this choice.

4 References

See the D02M/N Sub-chapter Introduction.

5 Parameters

5.1 Compulsory Input Parameters

1: **ldysav – int32 scalar**

A bound on the maximum number of differential equations to be solved.

Constraint: **ldysav** ≥ 1.

2: **sdysav – int32 scalar**

The second dimension of the array **ysav** that will be supplied to the integrator, as declared in the (sub)program from which the integrator is called.

Constraint: **sdysav** $\geq$ **maxord** $+ 1$.

3: **maxord – int32 scalar**

The maximum order to be used for the BDF method.

Constraint: $0 <$ **maxord** ≤ 5.

4: **method – string**

Specifies the method to be used to solve the system of nonlinear equations arising on each step of the BDF code.

method = 'N'

A modified Newton iteration is used.

method = 'F'

Functional iteration is used.

method = 'D'

The modified Newton iteration is used.

Note: a linear algebra setup function must be called even when using functional iteration, since if difficulty is encountered a switch is made to a modified Newton method.

Only the first character of the actual parameter **method** is passed to d02nv; hence it is permissible for the actual argument to be more descriptive e.g., 'Newton', 'Functional iteration' or 'Default' in a call to d02nv.

Constraint: **method** = 'N', 'F' or 'D'.

5: **petzld – logical scalar**

Specifies whether the Petzold local error test is to be used. If **petzld** is set to **true** on entry, then the Petzold local error test is used, otherwise a conventional test is used. The Petzold test results in extra overhead cost but is more stable and reliable for differential/algebraic equations.

6: **con**(**6**) **– double array**

Values to be used to control step size choice during integration. If any **con**$(i) = 0.0$ on entry, it is replaced by its default value described below. In most cases this is the recommended setting.

con(1), **con**(2) and **con**(3) are factors used to bound step size changes. If the current step size h fails, then the modulus of the next step size is bounded by **con**$(1) \times |h|$. The default value of **con**(1) is 2.0. Note that the new step size may be used with a method of different order to the failed step. If the initial step size is h, then the modulus of the step size on the second step is bounded by **con**$(3) \times |h|$. At any other stage in the integration, if the current step size is h, then the modulus of the next step size is bounded by **con**$(2) \times |h|$. The default values are 10.0 for **con**(2) and 1000.0 for **con**(3).

con(4), **con**(5) and **con**(6) are 'tuning' constants used in determining the next order and step size. They are used to scale the error estimates used in determining whether to keep the same order of the BDF method, decrease the order or increase the order respectively. The larger the value of **con**(i), for $i = 4, 5, 6$, the less likely the choice of the corresponding order. The default values are: **con**$(4) = 1.2$, **con**$(5) = 1.3$, **con**$(6) = 1.4$.

Constraints:

$0.0 <$ **con**$(1) <$ **con**$(2) <$ **con**(3);
con$(i) > 1.0$, for $i = 2, 3, \ldots, 6$.

These constraints must be satisfied after any zero values have been replaced by their default values.

7: **tcrit – double scalar**

A point beyond which integration must not be attempted. The use of **tcrit** is described under the parameter **itask** in the specification for the integrator (e.g., see d02nb). A value, 0.0 say, must be specified even if **itask** subsequently specifies that **tcrit** will not be used.

8: **hmin – double scalar**

The minimum absolute step size to be allowed. Set **hmin** = 0.0 if this option is not required.

9: **hmax – double scalar**

The maximum absolute step size to be allowed. Set **hmax** = 0.0 if this option is not required.

10: **h0 – double scalar**

The step size to be attempted on the first step. Set **h0** $= 0.0$ if the initial step size is to be calculated internally.

11: **maxstp – int32 scalar**

The maximum number of steps to be attempted during one call to the integrator after which it will return with **ifail** $= 2$. Set **maxstp** $= 0$ if no limit is to be imposed.

12: **mxhnil – int32 scalar**

The maximum number of warnings printed (if **itrace** ≥ 0) per problem when $t + h = t$ on a step ($h =$ current step size). If **mxhnil** ≤ 0, a default value of 10 is assumed.

13: **norm_p – string**

Indicates the type of norm to be used.

norm_p $=$ 'M'

Maximum norm.

norm_p $=$ 'A'

Averaged L2 norm.

norm_p $=$ 'D'

Is the same as **norm_p** $=$ 'A'.

If $vnorm$ denotes the norm of the vector v of length n, then for the averaged L2 norm

$$vnorm = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(v_i/w_i)^2},$$

while for the maximum norm

$$vnorm = \max_i |v_i/w_i|.$$

If you wish to weight the maximum norm or the L2 norm, then **rtol** and **atol** should be scaled appropriately on input to the integrator (see under **itol** in the specification of the integrator for the formulation of the weight vector w_i from **rtol** and **atol**, e.g., see d02nb).

Only the first character to the actual parameter **norm_p** is passed to d02nv; hence it is permissible for the actual argument to be more descriptive e.g., 'Maximum', 'Average L2' or 'Default' in a call to d02nv.

Constraint: **norm_p** $=$ 'M', 'A' or 'D'.

14: **rwork**$(\mathbf{50} + \mathbf{4} \times \mathbf{ldysav})$ **– double array**

This must be the same workspace array as the array **rwork** supplied to the integrator. It is used to pass information from the setup function to the integrator and therefore the contents of this array must not be changed before calling the integrator.

5.2 Optional Input Parameters

None.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **con**(6) **– double array**

The values actually used by d02nv.

2: **rwork**$(\mathbf{50} + \mathbf{4} \times \mathbf{ldysav})$ **– double array**

This must be the same workspace array as the array **rwork** supplied to the integrator. It is used to pass information from the setup function to the integrator and therefore the contents of this array must not be changed before calling the integrator.

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:

ifail $= 1$

On entry, an illegal input was detected.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

d02nb_fcn.m

```
function [f, ires] = fcn(neq, t, y, ires)
      f = zeros(3,1);
      f(1) = -0.04d0*y(1) + 1.0d4*y(2)*y(3);
      f(2) = 0.04d0*y(1) - 1.0d4*y(2)*y(3) - 3.0d7*y(2)*y(2);
      f(3) = 3.0d7*y(2)*y(2);
```

d02nb_jac.m

```
function p = jac(neq, t, y, h, d)
  p = zeros(neq, neq);
  hxd = h*d;
  p(1,1) = 1.0d0 - hxd*(-0.04d0);
  p(1,2) = -hxd*(1.0d4*y(3));
  p(1,3) = -hxd*(1.0d4*y(2));
  p(2,1) = -hxd*(0.04d0);
  p(2,2) = 1.0d0 - hxd*(-1.0d4*y(3)-6.0d7*y(2));
  p(2,3) = -hxd*(-1.0d4*y(2));
  p(3,2) = -hxd*(6.0d7*y(2));
  p(3,3) = 1.0d0 - hxd*(0.0d0);
```

d02nb_monitr.m

```
function [hnext, y, imon, inln, hmin, hmax] = ...
    monitr(neq, neqmax, t, hlast, hnext, y, ydot, ysave, r, acor, imon,
```

```
hmin, hmax, nqu)
  inln=int32(0);
```

```
neq = int32(3);
t = 0;
tout = 10;
y = [1;
     0;
     0];
rwork = zeros(62, 1);
rtol = [0.0001];
atol = [1e-07];
itol = int32(1);
inform = zeros(23, 1, 'int32');
ysave = zeros(3, 6);
wkjac = zeros(12, 1);
itask = int32(4);
itrace = int32(0);
[const, rwork, ifail] = ...
    d02nv(int32(3), int32(6), int32(5), 'Newton', false, zeros(6), ...
    10, 1e-10, 10, 0, int32(200), int32(5), 'Average-L2', rwork);
[rwork, ifail] = ...
    d02ns(int32(3), int32(3), 'Numerical', int32(12), rwork);
[tOut, yOut, ydot, rworkOut, informOut, ysaveOut, wkjacOut, ifail] = ...
     d02nb(neq, t, tout, y, rwork, rtol, atol, itol, inform, 'd02nb_fcn',
...
    ysave, 'd02nb_jac', wkjac, 'd02nb_monitr', itask, itrace)
```

```
tOut =
    10
yOut =
    0.8414
    0.0000
    0.1586
ydot =
   -0.0079
   -0.0000
    0.0079
rworkOut =
     array elided
informOut =
          55
         132
          17
           3
           3
          79
           0
           3
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
ysaveOut =
    0.8414   -0.0041    0.0001   -0.0000    0.0000   -0.0000
    0.0000   -0.0000    0.0000   -0.0000    0.0000   -0.0000
    0.1586    0.0041   -0.0001    0.0000   -0.0000    0.0000
```

```
wkjacOut =
    1.0100
         0
   -0.0100
 -380.6996
 -247.4940
  629.1936
   -0.0408
   -0.0040
    2.5831
    1.0000
    3.0000
    3.0000
ifail =
           0
```