NAG Toolbox

nag_ode_ivp_stiff_bdf (d02nv)

1 Purpose

nag_ode_ivp_stiff_bdf (d02nv) is a setup function which must be called prior to linear algebra setup functions and integrators from the SPRINT suite of functions, if Backward Differentiation Formulae (BDF) are to be used.

2 Syntax

```
[con, rwork, ifail] = nag_ode_ivp_stiff_bdf(neqmax, sdysav, maxord, method,
petzld, con, tcrit, hmin, hmax, h0, maxstp, mxhnil, norm_p, rwork)
[con, rwork, ifail] = d02nv(neqmax, 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 linear algebra setup function or integrator from the SPRINT suite of functions in this sub-chapter. This setup function, nag_ode_ivp_stiff_bdf (d02nv), makes the choice of the BDF integrator and permits you to define options appropriate to this choice. Alternative choices of integrator from this suite are the BLEND method and the DASSL implementation of the BDF method which can be chosen by initial calls to nag_ode_ivp_stiff_blend (d02nw) or nag_ode_ivp_stiff_dassl (d02mv) respectively.

4 References

See the D02M–N Sub-chapter Introduction.

5 Parameters

5.1 Compulsory Input Parameters

1: **neqmax** – INTEGER

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

Constraint: **neqmax** ≥ 1.

2: **sdysav** – INTEGER

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** – INTEGER

The maximum order to be used for the BDF method.

Constraint: $0 <$ **maxord** ≤ 5.

4: **method** – CHARACTER(1)

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'
A 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 argument **method** is passed to nag_ode_ivp_stiff_bdf (d02nv); hence it is permissible for the actual argument to be more descriptive e.g., 'Newton', 'Functional iteration' or 'Default' in a call to nag_ode_ivp_stiff_bdf (d02nv).

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

5: **petzld** – LOGICAL

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**) – REAL (KIND=nag_wp) array

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

$\mathbf{con}(1)$, $\mathbf{con}(2)$, and $\mathbf{con}(3)$ 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 $\mathbf{con}(1) \times |h|$. The default value of $\mathbf{con}(1)$ 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 $\mathbf{con}(3) \times |h|$. 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 $\mathbf{con}(2) \times |h|$. The default values are 10.0 for $\mathbf{con}(2)$ and 1000.0 for $\mathbf{con}(3)$.

$\mathbf{con}(4)$, $\mathbf{con}(5)$ and $\mathbf{con}(6)$ 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 $\mathbf{con}(i)$, for $i = 4, 5, 6$, the less likely the choice of the corresponding order. The default values are: $\mathbf{con}(4) = 1.2$, $\mathbf{con}(5) = 1.3$, $\mathbf{con}(6) = 1.4$.

Constraints:

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

$0.0 < \mathbf{con}(1) \leq \mathbf{con}(2) \leq \mathbf{con}(3)$;
$\mathbf{con}(i) \geq 1.0$, for $i = 2, 3, \ldots, 6$.

7: **tcrit** – REAL (KIND=nag_wp)

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

8: **hmin** – REAL (KIND=nag_wp)

The minimum absolute step size to be allowed. Set $\mathbf{hmin} = 0.0$ if this option is not required.

9: **hmax** – REAL (KIND=nag_wp)

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

10: **h0** – REAL (KIND=nag_wp)

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

11: **maxstp** – INTEGER

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** – INTEGER

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** – CHARACTER(1)

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 nag_ode_ivp_stiff_exp_fulljac (d02nb)).

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

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

14: **rwork**$(\mathbf{50} + \mathbf{4} \times \mathbf{neqmax})$ – REAL (KIND=nag_wp) 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 Output Parameters

1: **con**(**6**) – REAL (KIND=nag_wp) array

The values actually used by nag_ode_ivp_stiff_bdf (d02nv).

2: **rwork**(**50** + **4** × **neqmax**) – REAL (KIND=nag_wp) array

3: **ifail** – INTEGER

ifail = 0 unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail = 1

On entry, an illegal input was detected.

ifail = −99

An unexpected error has been triggered by this routine. Please contact NAG.

ifail = −399

Your licence key may have expired or may not have been installed correctly.

ifail = −999

Dynamic memory allocation failed.

7 Accuracy

Not applicable.

8 Further Comments

None.

9 Example

See Section 10 in nag_ode_ivp_stiff_exp_fulljac (d02nb).

9.1 Program Text

```
function d02nv_example

fprintf('d02nv example results\n\n');

% Initialize variables and arrays for setup routine

n      = nag_int(3);
ord    = nag_int(5);
sdy    = nag_int(ord + 1);
petzld = false;
con    = zeros(6);
tcrit  = 10;
hmin   = 1.0e-10;
hmax   = 10;
h0     = 0;
maxstp = nag_int(200);
mxhnil = nag_int(5);
lrwork = 50 + 4*n;
```

```
rwork  = zeros(lrwork);

% Setup integration method using d02nv.
[const, rwork, ifail] = d02nv(n, sdy, ord, 'Newton', petzld, con, tcrit, ...
                              hmin, hmax, h0, maxstp, mxhnil, 'Average-L2', ...
                              rwork);
% Setup for analytical Jacbian using d02ns
nwkjac = nag_int(n*(n+1));
[rwork, ifail] = d02ns(n, n, 'Analytical', nwkjac, rwork);

% Initialize variables and arrays for integration
t      = 0;
tout   = 10;
y      = [1; 0; 0];
rtol   = [0.0001];
atol   = [1e-07];
itol   = nag_int(1);
inform = zeros(23, 1, nag_int_name);
ysave  = zeros(n, sdy);
wkjac  = zeros(nwkjac, 1);
itask  = nag_int(4);
itrace = nag_int(0);

% Integrate ODE from t=0 to t=tout, no monitoring, using d02nb
[t, y, ydot, rwork, inform, ysave, wkjac, ifail] = d02nb(t, tout, y, rwork, ...
          rtol, atol, itol, inform, @fcn, ysave, @jac, wkjac, 'd02nby', ...
          itask, itrace);

fprintf('Solution y and derivative y'' at t = %7.4f is:\n',t);
fprintf('\n %10s %10s\n','y','y''');
for i=1:n
  fprintf(' %10.4f %10.4f\n',y(i),ydot(i));
end

function [f, ires] = fcn(neq, t, y, ires)
% Evaluate derivative vector.
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);

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

9.2 Program Results

```
     d02nv example results

Solution y and derivative y' at t = 10.0000 is:

          y          y'
     0.8414    -0.0079
     0.0000    -0.0000
     0.1586     0.0079
```