NAG Toolbox

nag_ode_ivp_stiff_dassl (d02mv)

1 Purpose

nag_ode_ivp_stiff_dassl (d02mv) is an integration method specific setup function which must be called prior to linear algebra setup functions and integrators from the SPRINT suite of functions, if the DASSL implementation of Backward Differentiation Formulae (BDF) is to be used. Note that this method is also available, independent from the SPRINT suite, using nag_ode_dae_dassl_gen (d02ne)

2 Syntax

```
[con, rwork, ifail] = nag_ode_ivp_stiff_dassl(neqmax, sdysav, maxord, con, tcrit, hmin, hmax, h0, maxstp, mxhnil, norm_p, rwork)
[con, rwork, ifail] = d02mv(neqmax, sdysav, maxord, 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_dassl (d02mv), makes the choice of the DASSL integrator and permits you to define options appropriate to this choice. Alternative choices of integrator from this suite are the BDF method and the BLEND method which can be chosen by initial calls to nag_ode_ivp_stiff_bdf (d02nv) or nag_ode_ivp_stiff_blend (d02nw) 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 (e.g., see nag_ode_ivp_stiff_exp_fulljac (d02nb)).

Constraint: **sdysav** $\geq$ **maxord** $+ 3$.

3: **maxord** – INTEGER

The maximum order to be used for the BDF method. If **maxord** $= 0$ or **maxord** > 5 then **maxord** $= 5$ is assumed.

Constraint: **maxord** ≥ 0.

4: **con**(**3**) – REAL (KIND=nag_wp) 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).

Constraints:

These constraints must be satisfied after any zero values have been replaced by default values.

$0.0 <$ **con**$(1) <$ **con**$(2) <$ **con**(3);
con$(2) > 1.0$;
con$(3) > 1.0$.

5: **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.

6: **hmin** – REAL (KIND=nag_wp)

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

7: **hmax** – REAL (KIND=nag_wp)

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

8: **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.

9: **maxstp** – INTEGER

The maximum number of steps to be attempted during one call to the integrator after which it will return with **ifail** $= 2$ (e.g., see nag_ode_ivp_stiff_exp_fulljac (d02nb)). Set **maxstp** $= 0$ if no limit is to be imposed.

10: **mxhnil** – INTEGER

The maximum number of warnings printed (if **itrace** ≥ 0, e.g., see nag_ode_ivp_stiff_exp_fulljac (d02nb)) per problem when $t + h = t$ on a step ($h =$ current step size). If **mxhnil** ≤ 0, a default value of 10 is assumed.

11: **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 `A'.

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

$$vnormB = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{v_i}{w_i}\right)^2},$$

while for the maximum norm

$$vnorm = \max_{1\leq i\leq n}\left|\frac{v_i}{w_i}\right|.$$

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_dassl (d02mv); 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_dassl (d02mv).

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

12: **rwork**(**50** + **4** × **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**(**3**) – REAL (KIND=nag_wp) array

The values actually to be used by the integration function.

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, **neqmax** < 1,
or **sdysav** < **maxord** + 3,
or **maxord** < 0,
or **maxord** > 5,
or invalid value for element of the array **con**,
or **norm_p** ≠ 'M', 'A' or 'D'.

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

This example solves the plane pendulum problem defined by the equations:

$$\begin{aligned} x' &= u \\ y' &= v \\ u' &= -\lambda x \\ v' &= -\lambda y - 1 \\ x^2 + y^2 &= 1 \end{aligned}$$

The additional algebraic constraint $xu + yv = 0$ can be derived, and after appropriate substitution and manipulation to avoid a singular Jacobian solves the equations:

$$\begin{aligned} y_1' &= y_3 - y_6 y_1 \\ y_2' &= y_4 - y_6 y_2 \\ y_3' &= -y_5 y_1 \\ y_4' &= -y_5 y_2 - 1 \\ 0 &= y_1 y_3 + y_2 y_4 \\ 0 &= y_1^2 + y_2^2 - 1 \end{aligned}$$

with given initial conditions and derivatives.

9.1 Program Text

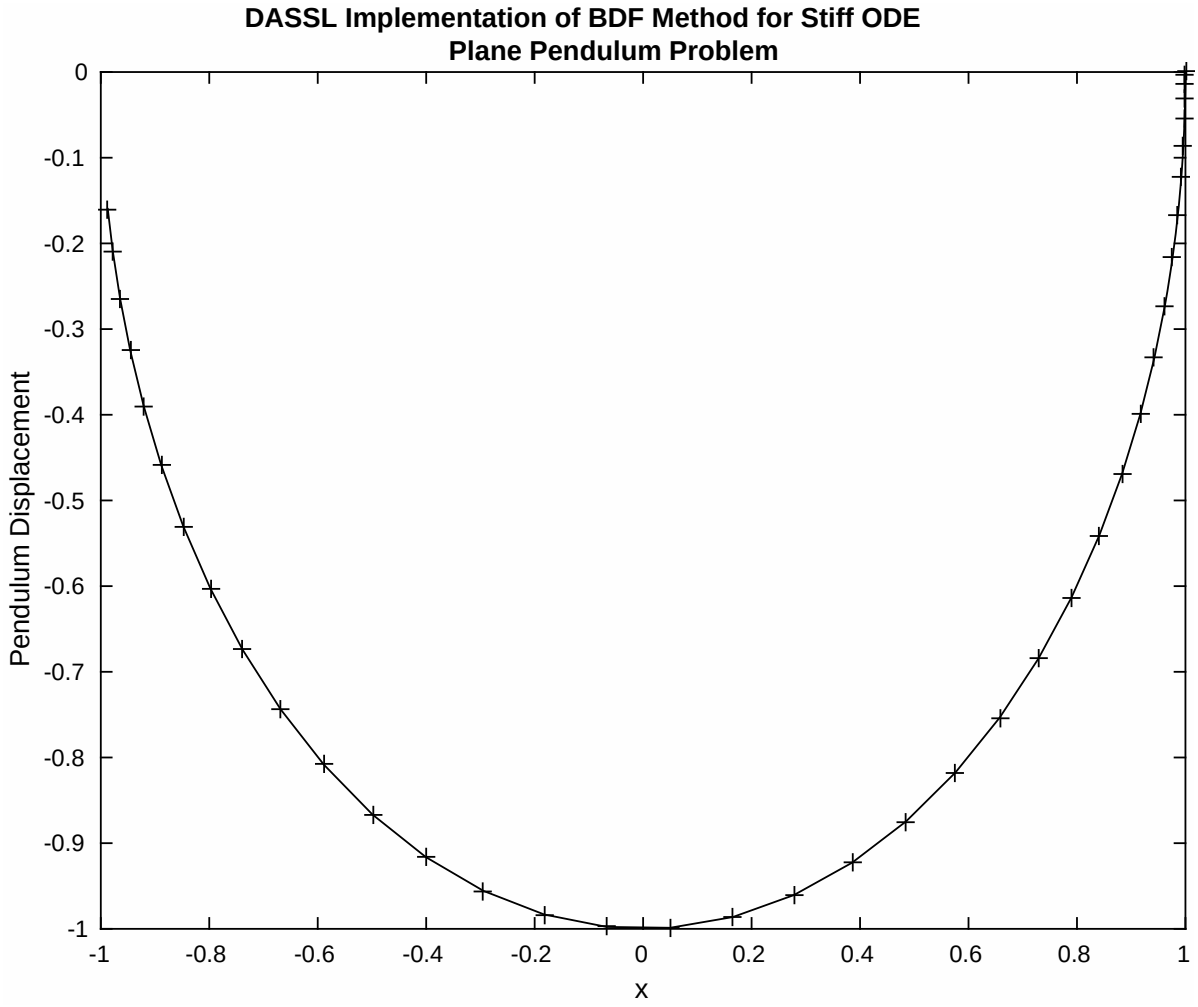
```
function d02mv_example

fprintf('d02mv example results\n\n');

% Initialize setup variables and arrays.
neq    = nag_int(6);
neqmax = nag_int(neq);
nwkjac = nag_int(neqmax*(neqmax + 1));
maxord = nag_int(5);
sdysav = nag_int(maxord+3);
maxstp = nag_int(5000);
mxhnil = nag_int(5);

h0    = 0.0;
hmax  = 0.0;
hmin  = 1.0e-10;
tcrit = pi;

const  = zeros(3, 1);
rwork  = zeros(50+4*neqmax, 1);

% d02mv is an integration method-specific setup routine to be called prior
% to linear algebra setup and integrator routines if the DASSL
% implementation of BDF is to be used.

[const, rwork, ifail] = d02mv(neqmax, sdysav, maxord, const, tcrit, hmin, ...
                              hmax, h0, maxstp, mxhnil, 'Average-L2', rwork);

% d02ns is a setup routine (specifically for full matrix linear algebra)
```

```
% to be called prior to a routine from sub-chapter d02n-m (e.g. d02ng,
% as here).

[rwork, ifail] = d02ns(neq, neqmax, 'Analytic', nwkjac, rwork);

% Initialize integration variables and arrays
wkjac  = zeros(nwkjac, 1);
ysave  = zeros(neq, sdysav);
inform(1:23) = nag_int(0);

t    = 0;
tout = pi;
nstep = 38;
itol = nag_int(1);
rtol = [1e-03];
atol = [1e-06];
itask = nag_int(4);
itrace = nag_int(0);
lderiv(1:2) = true;
y(1:neq) = 0;
y(1)     = 1;
ydot(1:neq) = 0;
ydot(1) = y(3) - y(6)*y(1);
ydot(2) = y(4) - y(6)*y(2);
ydot(3) = -y(5)*y(1);
ydot(4) = -y(5)*y(2) - 1.0;
ydot(5) = -3*y(4);

% Output header and initial results.
fprintf(['\nPendulum problem with relative tolerance = %7.1e\n', ...
    '                  and absolute tolerance = %7.1e\n\n'], ...
    rtol(1), atol(1));
fprintf('   t        y1      y2      y3      y4      y5      y6\n');
fprintf(' %6.4f  %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f\n', t, y);

% Prepare to store results for plotting, then loop over values for the
% independent variable.
tkeep = zeros(nstep+1,1);
ykeep = zeros(neq,nstep+1);
tkeep(1) = t;
ykeep(:,1) = y;
for istep = 1:nstep
    tout = istep/nstep*pi;
    [t, tout, y, ydot, rwork, inform, ysave, wkjac, lderiv, ifail] = ...
        d02ng(t, tout, y, ydot, rwork, rtol, atol, itol, inform, @resid, ...
              ysave, @jac, wkjac, @monitr, lderiv, itask, itrace);

    % Store current results for plotting.
    tkeep(istep+1) = t;
    ykeep(:,istep+1) = y;
end

% Output final results.
fprintf(' %6.4f  %7.4f %7.4f %7.4f %7.4f %7.4f %7.4f\n', t, y);
% Plot results.
fig1 = figure;
display_plot(ykeep(1,:), ykeep(2,:));

function pdj = jac(neq, t, y, ydot, h, d, pdj)
% Evaluate the Jacobian.

pdj = zeros(neq, neq);
hxd = h*d;
pdj(1,1) =  (1.0 + hxd*y(6)); pdj(1,3) = -hxd;      pdj(1,6) =  hxd*y(1);
pdj(2,2) =  (1.0 + hxd*y(6)); pdj(2,4) = -hxd;      pdj(2,6) =  hxd*y(2);
pdj(3,1) =         hxd*y(5);  pdj(3,3) =  1.0;      pdj(3,5) =  hxd*y(1);
pdj(4,2) =         hxd*y(5);  pdj(4,4) =  1.0;      pdj(4,5) =  hxd*y(2);
pdj(5,1) =        -hxd*y(3);  pdj(5,3) = -hxd*y(1);
pdj(5,2) =        -hxd*y(4);  pdj(5,4) = -hxd*y(2);
pdj(6,1) =      -2*hxd*y(1);
pdj(6,2) =      -2*hxd*y(2);
```

```
function [hnext, y, imon, inln, hmin, hmax] = monitr(neq, neqmax, ...
    t, hlast, hnext, y, ydot, ysave, r, acor, imon, hmin, hmax, nqu)
  inln = nag_int(0);

function [r, ires] = resid(neq, t, y, ydot, ires)
% Evaluate the residue.
r = zeros(neq,1);
if ires == -1
    r(1:4) = -ydot(1:4);
    r(5:6) = 0.0;
else
    r(1) =  y(3) - y(6)*y(1) - ydot(1);
    r(2) =  y(4) - y(6)*y(2) - ydot(2);
    r(3) =       -y(5)*y(1) - ydot(3);
    r(4) =       -y(5)*y(2) - ydot(4) - 1;
    r(5) =  y(1)*y(3) + y(2)*y(4);
    r(6) =  y(1)*y(1) + y(2)*y(2) - 1;
end

function display_plot(x, y)
% Formatting for title and axis labels.
% Plot results.
plot(x, y, '-+');
% Add title.
title({'DASSL Implementation of BDF Method for Stiff ODE',...
       'Plane Pendulum Problem'});
% Label the axes.
xlabel('x'); ylabel('Pendulum Displacement');
```

9.2 Program Results

```
     d02mv example results

Pendulum problem with relative tolerance = 1.0e-03
                  and absolute tolerance = 1.0e-06

   t        y1      y2      y3      y4      y5      y6
 0.0000   1.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 3.1416  -0.9872 -0.1597 -0.0902  0.5579  0.4790 -0.0000
```


DASSL Implementation of BDF Method for Stiff ODE
Plane Pendulum Problem
Pendulum Displacement
x
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1