NAG Toolbox

nag_ode_ivp_stiff_errest (d02za)

1 Purpose

nag_ode_ivp_stiff_errest (d02za) calculates the weighted norm of the local error estimate from inside a **monitr** called from an integrator in Sub-chapter D02M–N (e.g., see nag_ode_ivp_stiff_exp_fulljac (d02nb)).

2 Syntax

```
[result, ifail] = nag_ode_ivp_stiff_errest(v, w, 'neq', neq)
[result, ifail] = d02za(v, w, 'neq', neq)
```

3 Description

nag_ode_ivp_stiff_errest (d02za) is for use with the forward communication integrators nag_ode_ivp_stiff_exp_fulljac (d02nb), nag_ode_ivp_stiff_exp_bandjac (d02nc), nag_ode_ivp_stiff_exp_sparjac (d02nd), nag_ode_ivp_stiff_imp_fulljac (d02ng), nag_ode_ivp_stiff_imp_bandjac (d02nh) and nag_ode_ivp_stiff_imp_sparjac (d02nj) and the reverse communication integrators nag_ode_ivp_stiff_exp_revcom (d02nm) and nag_ode_ivp_stiff_imp_revcom (d02nn). It must be used only inside **monitr** (if this option is selected) for the forward communication functions or on the equivalent return for the reverse communication functions. It may be used to evaluate the norm of the scaled local error estimate, $\|v\|$, where the weights used are contained in w and the norm used is as defined by an earlier call to the integrator setup function (nag_ode_ivp_stiff_dassl (d02mv), nag_ode_ivp_stiff_bdf (d02nv) or nag_ode_ivp_stiff_blend (d02nw)). Its use is described under the description of **monitr** in the specifications for the forward communication integrators mentioned above.

4 References

None.

5 Parameters

5.1 Compulsory Input Parameters

1: **v**(**neq**) – REAL (KIND=nag_wp) array

The vector, the weighted norm of which is to be evaluated by nag_ode_ivp_stiff_errest (d02za). **v** is calculated internally by the integrator being used.

2: **w**(**neq**) – REAL (KIND=nag_wp) array

The weights, calculated internally by the integrator, to be used in the norm evaluation.

5.2 Optional Input Parameters

1: **neq** – INTEGER

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

The number of differential equations, as defined for the integrator being used.

5.3 Output Parameters

1: **result**

The result of the function.

2: **ifail** – INTEGER

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

6 Error Indicators and Warnings

Note: nag_ode_ivp_stiff_errest (d02za) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail $= 1$ (*warning*)

The value of the norm would either overflow or is close to overflowing. A value close to the square root of the largest number on the computer is returned.

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

The result is calculated close to ***machine precision*** except in the case when the function exits with **ifail** $= 1$.

8 Further Comments

nag_ode_ivp_stiff_errest (d02za) should only be used within **monitr** associated with the integrators in Sub-chapter D02M–N (e.g., see nag_ode_ivp_stiff_exp_fulljac (d02nb)). Its use and only valid calling sequence are fully documented in the description of **monitr** in the function documents for the integrators.

9 Example

This example solves the well-known stiff Robertson problem

$$\begin{array}{rcrcrcl} a' & = & -0.04a & + & 1.0\text{E}4bc & & \\ b' & = & 0.04a & - & 1.0\text{E}4bc & - & 3.0\text{E}7b^2 \\ c' & = & & & & & 3.0\text{E}7b^2 \end{array}$$

over the range $[0, 10]$ with initial conditions $a = 1.0$ and $b = c = 0.0$ using scalar error control (**itol** $= 1$) and computation of the solution at **tout** $= 10.0$ with **tcrit** (e.g., see nag_ode_ivp_stiff_dassl (d02mv)) set to 10.0 (**itask** $= 4$). A BDF integrator (setup function nag_ode_ivp_stiff_bdf (d02nv)) is used and a modified Newton method is selected. This example illustrates the use of nag_ode_ivp_stiff_errest (d02za) within a monitor function **monitr** to output intermediate results during the integration. The same problem is solved in the example program for nag_ode_ivp_stiff_exp_fulljac (d02nb) where no monitoring was performed and so no intermediate solution information is output.

9.1 Program Text

```
     function d02za_example

fprintf('d02za example results\n\n');

global isol tkeep ykeep

% Initialize setup variables and arrays.
neq    = nag_int(3);
neqmax = nag_int(neq);
nwkjac = nag_int(neqmax*(neqmax + 1));
maxord = nag_int(5);
sdysav = nag_int(maxord+1);
maxstp = nag_int(200);
mxhnil = nag_int(5);

h0    = 0;
hmax  = 10;
hmin  = 1.0e-10;
tcrit = 10;
petzld = false;

const  = zeros(6, 1);
rwork  = zeros(50+4*neqmax, 1);

% d02nv is a setup routine to be called prior to d02nb.
[const, rwork, ifail] = d02nv( ...
                              neqmax, sdysav, maxord, 'Newton', petzld, ...
                              const, tcrit, hmin, hmax, h0, maxstp, ...
                              mxhnil, 'Average-L2', rwork);
% d02ns determines how d02nb evaluates the Jacobian.
[rwork, ifail] = d02ns( ...
                       neq, neqmax, 'Analytical', nwkjac, rwork);

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

% First case. Integrate to tout without passing tout (tcrit=tout and itask=4)
%             use B.D.F formulae with a Newton method.
%             Evaluate Jacobian numerically (d02nbz), no monitoring (d02nby).
t      = 0.0;
tout   = 10.0;
itask  = nag_int(4);
itrace = nag_int(0);
y      = [1; 0; 0];
itol   = nag_int(1);
rtol   = [0.0001];
atol   = [1e-07];

isol          = 1;
tkeep(isol)   = t;
ykeep(1:3,isol) = y;

% Output initial and final solutions.
fprintf('  Using Analytical Jacobian\n\n');
fprintf('     x           y_1          y_2           y_3\n');
fprintf(' %8.3f      %5.1f        %5.1f         %5.1f\n', t, y);

[t, y, ydot, rwork, inform, ysave, wkjac, ifail] = ...
d02nb( ...
       t, tout, y, rwork, rtol, atol, itol, inform, @fcn, ysave, ...
       @jac, wkjac, @monitr, itask, itrace);
fprintf(' %8.3f      %5.1f        %5.1f         %5.1f\n', t, y);

% Plot results.
fig1 = figure;
display_plot(tkeep, ykeep)
```

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

function p = jac(neq, t, y, h, d, p)
  % Evaluate the Jacobian.
  p = zeros(neq, neq);
  hxd = h*d;
  p(1,1) = 1 - hxd*(-0.04);
  p(1,2) =   - hxd*(1.0d4*y(3));
  p(1,3) =   - hxd*(1.0d4*y(2));
  p(2,1) =   - hxd*(0.04);
  p(2,2) = 1 - 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 - hxd*(0);

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

  global isol tkeep ykeep

  inln=nag_int(0);
  if (imon == nag_int(1))
    isol = isol + 1;
    tkeep(isol) = t;
    ykeep(1:3,isol) = y;
  end

function display_plot(x,y)
  % Formatting for title and axis labels.
  % Plot one of the curves and then add the other two.
  hline3 = plot(x, y(3,:));
  hold on;
  [haxes, hline1, hline2] = plotyy(x, 100*y(2,:), x, y(1,:));
  % Set the axis limits and the tick specifications to beautify the plot.
  set(haxes(1), 'YLim', [0 0.0045]);
  set(haxes(1), 'XMinorTick', 'on', 'YMinorTick', 'on');
  set(haxes(1), 'YTick', [0:0.001:0.004]);
  set(haxes(2), 'YLim', [0.995 1.005]);
  set(haxes(2), 'YMinorTick', 'on');
  set(haxes(2), 'YTick', [0.995:0.002:1.005]);
  set(haxes(1), 'XLim', [-0.005 0.1]);
  set(haxes(2), 'XLim', [-0.005 0.1]);
  set(gca, 'box', 'off');
  % Add title.
  th = title({'Stiff Robertson Problem','Using BDF with Modified Newton'});
  set(th,'position',[0.05,0.004]);
  % Label the x axis, and both y axes.
  xlabel('x');
  ylabel(haxes(1), 'Solution (100*b,c)');
  ylabel(haxes(2), 'Solution (a)');
  % Add a legend.
  legend('c', '100*b', 'a', 'Location', 'East');
  % Set some features of the three lines
  set(hline1, 'Linewidth', 0.5, 'Marker', '+', 'LineStyle', '-', ...
     'Color', 'Magenta');
  set(hline2, 'Linewidth', 0.5, 'Marker', '*', 'LineStyle', ':');
  set(hline3, 'Linewidth', 0.5, 'Marker', 'x', 'LineStyle', '--');
  hold off;
```

9.2 Program Results

```
d02za example results

Using Analytical Jacobian

   x          y_1          y_2          y_3
 0.000        1.0          0.0          0.0
10.000        0.8          0.0          0.2
```