NAG Toolbox

nag_ode_dae_dassl_gen (d02ne)

1 Purpose

nag_ode_dae_dassl_gen (d02ne) is a function for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations.

2 Syntax

```
[t, y, ydot, rtol, atol, itask, icom, com, user, ifail] = nag_ode_dae_dassl_gen
(t, tout, y, ydot, rtol, atol, itask, res, jac, icom, com, 'neq', neq, 'user',
user)
[t, y, ydot, rtol, atol, itask, icom, com, user, ifail] = d02ne(t, tout, y,
ydot, rtol, atol, itask, res, jac, icom, com, 'neq', neq, 'user', user)
```

3 Description

nag_ode_dae_dassl_gen (d02ne) is a general purpose function for integrating the initial value problem for a stiff system of implicit ordinary differential equations with coupled algebraic equations written in the form

$$F(t, y, y') = 0.$$

nag_ode_dae_dassl_gen (d02ne) uses the DASSL implementation of the Backward Differentiation Formulae (BDF) of orders one to five to solve a system of the above form for y (**y**) and y' (**ydot**). Values for **y** and **ydot** at the initial time must be given as input. These values must be consistent, (i.e., if **t**, **y**, **ydot** are the given initial values, they must satisfy $F(\mathbf{t}, \mathbf{y}, \mathbf{ydot}) = 0$). The function solves the system from $t =$ **t** to $t =$ **tout**.

An outline of a typical calling program for nag_ode_dae_dassl_gen (d02ne) is given below. It calls the DASSL implementation of the BDF integrator setup function nag_ode_dae_dassl_setup (d02mw) and the banded matrix setup function nag_ode_dae_dassl_linalg (d02np) (if required), and, if the integration needs to proceed, calls nag_ode_dae_dassl_cont (d02mc) before continuing the integration.

```
.
.
.
% Initialize the integrator
[...] = d02mw(...);
% Is the Jacobian matrix banded?
if (banded)
  [...] = d02np(...);
end
% Set DT to the required temporal resolution
% Set TEND to the final time
% Call the integrator for each temporal value
while 1
  [tout,...] = d02ne(...);
  if (tout>=tend || itask<0)
    break
  end
  if (itask ~= 1)
    tout = min(tout + dt, tend);
  end
  % Print the solution
  [...] = d02mc(...);
end
.
```

.
.

4 References

None.

5 Parameters

5.1 Compulsory Input Parameters

1: **t** – REAL (KIND=nag_wp)

On initial entry: the initial value of the independent variable, t.

2: **tout** – REAL (KIND=nag_wp)

The next value of t at which a computed solution is desired.

On initial entry: **tout** is used to determine the direction of integration. Integration is permitted in either direction (see also **itask**).

Constraint: **tout** $\neq$ **t**.

3: **y**(**neq**) – REAL (KIND=nag_wp) array

On initial entry: the vector of initial values of the dependent variables y.

4: **ydot**(**neq**) – REAL (KIND=nag_wp) array

On initial entry: **ydot** must contain approximations to the time derivatives y' of the vector y evaluated at the initial value of the independent variable.

5: **rtol**(:) – REAL (KIND=nag_wp) array

The dimension of the array **rtol** depends on the value of **itol** as set in nag_ode_dae_dassl_setup (d02mw); it must be at least **neq** if **itol** $= true$ and at least 1 if **itol** $= false$

The relative local error tolerance.

Constraint: **rtol**$(i) \geq 0.0$, for $i = 1, 2, \ldots, n$

where $n =$ **neq** when **itol** $= true$ and $n = 1$ otherwise.

6: **atol**(:) – REAL (KIND=nag_wp) array

The dimension of the array **atol** depends on the value of **itol** as set in nag_ode_dae_dassl_setup (d02mw); it must be at least **neq** if **itol** $= true$ and at least 1 if **itol** $= false$

The absolute local error tolerance.

Constraint: **atol**$(i) \geq 0.0$, for $i = 1, 2, \ldots, n$

where $n =$ **neq** when **itol** $= true$ and $n = 1$ otherwise.

7: **itask** – INTEGER

On initial entry: need not be set.

8: **res** – SUBROUTINE, supplied by the user.

res must evaluate the residual

$$R = F(t, y, y').$$

```
[r, ires, user] = res(neq, t, y, ydot, ires, user)
```

Input Parameters

1: **neq** – INTEGER

The number of differential-algebraic equations being solved.

2: **t** – REAL (KIND=nag_wp)

t, the current value of the independent variable.

3: **y**(**neq**) – REAL (KIND=nag_wp) array

y_i, for $i = 1, 2, \ldots,$ **neq**, the current solution component.

4: **ydot**(**neq**) – REAL (KIND=nag_wp) array

The derivative of the solution at the current point t.

5: **ires** – INTEGER

Is always equal to zero.

6: **user** – INTEGER array

res is called from nag_ode_dae_dassl_gen (d02ne) with the object supplied to nag_ode_dae_dassl_gen (d02ne).

Output Parameters

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

r(i) must contain the ith component of R, for $i = 1, 2, \ldots,$ **neq** where

$$R = F(\mathbf{t}, \mathbf{y}, \mathbf{ydot}).$$

2: **ires** – INTEGER

ires should normally be left unchanged. However, if an illegal value of **y** is encountered, **ires** should be set to -1; nag_ode_dae_dassl_gen (d02ne) will then attempt to resolve the problem so that illegal values of **y** are not encountered. **ires** should be set to -2 if you wish to return control to the calling (sub)routine; this will cause nag_ode_dae_dassl_gen (d02ne) to exit with **ifail** $= 23$.

3: **user** – INTEGER array

9: **jac** – SUBROUTINE, supplied by the NAG Library or the user.

Evaluates the matrix of partial derivatives, J, where

$$J_{ij} = \frac{\partial F_i}{\partial y_j} + \mathbf{cj} \times \frac{\partial F_i}{\partial y'_j}, \quad i, j = 1, 2, \ldots, \mathbf{neq}.$$

If this option is not required, the actual argument for **jac** must be the string `nag_ode_dae_dassl_gen_dummy_jac (d02nez)`. (nag_ode_dae_dassl_gen_dummy_jac (d02nez) is included in the NAG Toolbox.) You must indicate to the integrator whether this option is to be used by setting the argument **jceval** appropriately in a call to the setup function nag_ode_dae_dassl_setup (d02mw).

```
[pd, user] = jac(neq, t, y, ydot, pd, cj, user)
```

Input Parameters

1: **neq** – INTEGER

The number of differential-algebraic equations being solved.

2: **t** – REAL (KIND=nag_wp)

t, the current value of the independent variable.

3: **y**(**neq**) – REAL (KIND=nag_wp) array

y_i, for $i = 1, 2, \ldots,$ **neq**, the current solution component.

4: **ydot**(**neq**) – REAL (KIND=nag_wp) array

The derivative of the solution at the current point t.

5: **pd**(:) – REAL (KIND=nag_wp) array

pd is preset to zero before the call to **jac**.

6: **cj** – REAL (KIND=nag_wp)

cj is a scalar constant which will be defined in nag_ode_dae_dassl_gen (d02ne).

7: **user** – INTEGER array

jac is called from nag_ode_dae_dassl_gen (d02ne) with the object supplied to nag_ode_dae_dassl_gen (d02ne).

Output Parameters

1: **pd**(:) – REAL (KIND=nag_wp) array

If the Jacobian is full then **pd**$((j-1) \times$ **neq** $+ i) = J_{ij}$, for $i = 1, 2, \ldots,$ **neq** and $j = 1, 2, \ldots,$ **neq**; if the Jacobian is banded then **pd**$((j-1) \times (2$**ml** $+$ **mu** $+ 1) +$ **ml** $+$ **mu** $+ i - j + 1) = J_{ij}$, for $\max(1, j -$ **mu**$) \leq i \leq \min(n, j +$ **ml**$)$; (see also in nag_lapack_dgbtrf (f07bd)).

2: **user** – INTEGER array

10: **icom**(**50** + **neq**) – INTEGER array

icom contains information which is usually of no interest, but is necessary for subsequent calls. However you may find the following useful:

icom(22)
The order of the method to be attempted on the next step.

icom(23)
The order of the method used on the last step.

icom(26)
The number of steps taken so far.

icom(27)
The number of calls to **res** so far.

icom(28)
The number of evaluations of the matrix of partial derivatives needed so far.

icom(29)
The total number of error test failures so far.

icom(30)
The total number of convergence test failures so far.

11: **com**($lcom$) – REAL (KIND=nag_wp) array

$lcom$, the dimension of the array, must satisfy the constraint $lcom \geq 40 + (maxorder + 4) \times \mathbf{neq} + \mathbf{neq} \times p + q$ where $maxorder$ is the maximum order that can be used by the integration method (see **maxord** in nag_ode_dae_dassl_setup (d02mw)); $p = \mathbf{neq}$ when the Jacobian is full and $p = (2 \times \mathbf{ml} + \mathbf{mu} + 1)$ when the Jacobian is banded; and, $q = (\mathbf{neq}/(\mathbf{ml} + \mathbf{mu} + 1)) + 1$ when the Jacobian is to be evaluated numerically and $q = 0$ otherwise.

com contains information which is usually of no interest, but is necessary for subsequent calls. However you may find the following useful:

com(3)
The step size to be attempted on the next step.

com(4)
The current value of the independent variable, i.e., the farthest point integration has reached. This will be different from **t** only when interpolation has been performed (**itask** $= 3$).

5.2 Optional Input Parameters

1: **neq** – INTEGER

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

The number of differential-algebraic equations to be solved.

Constraint: **neq** ≥ 1.

2: **user** – INTEGER array

user is not used by nag_ode_dae_dassl_gen (d02ne), but is passed to **res** and **jac**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Output Parameters

1: **t** – REAL (KIND=nag_wp)

On intermediate exit: t, the current value of the independent variable.

On final exit: the value of the independent variable at which the computed solution y is returned (usually at **tout**).

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

On intermediate exit: the computed solution vector, y, evaluated at $t = T$.

On final exit: the computed solution vector, evaluated at t (usually $t =$ **tout**).

3: **ydot**(**neq**) – REAL (KIND=nag_wp) array

The time derivatives y' of the vector y at the last integration point.

4: **rtol**(:) – REAL (KIND=nag_wp) array

The dimension of the array **rtol** depends on the value of **itol** as set in nag_ode_dae_dassl_setup (d02mw); it will be **neq** if **itol** $= true$ and at least 1 if **itol** $= false$

rtol remains unchanged unless nag_ode_dae_dassl_gen (d02ne) exits with **ifail** $= 16$ in which case the values may have been increased to values estimated to be appropriate for continuing the integration.

5: **atol**(:) – REAL (KIND=nag_wp) array

The dimension of the array **atol** depends on the value of **itol** as set in nag_ode_dae_dassl_setup (d02mw); it will be **neq** if **itol** $= true$ and at least 1 if **itol** $= false$

atol remains unchanged unless nag_ode_dae_dassl_gen (d02ne) exits with **ifail** $= 16$ in which case the values may have been increased to values estimated to be appropriate for continuing the integration.

6: **itask** – INTEGER

The task performed by the integrator on successful completion or an indicator that a problem occurred during integration.

itask $= 2$
The integration to **tout** was successfully completed (**t** $=$ **tout**) by stepping exactly to **tout**.

itask $= 3$
The integration to **tout** was successfully completed (**t** $=$ **tout**) by stepping past **tout**. **y** and **ydot** are obtained by interpolation.

itask < 0
Different negative values of **itask** returned correspond to different failure exits. **ifail** should always be checked in such cases and the corrective action taken where appropriate.

itask must remain **unchanged** between calls to nag_ode_dae_dassl_gen (d02ne).

7: **icom**(**50** $+$ **neq**) – INTEGER array

8: **com**($lcom$) – REAL (KIND=nag_wp) array

9: **user** – INTEGER array

10: **ifail** – INTEGER

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

6 Error Indicators and Warnings

Note: nag_ode_dae_dassl_gen (d02ne) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail $= 1$

Constraint: **neq** ≥ 1.

ifail $= 3$

Constraint: **tout** $\neq$ **t**.

tout is behind **t** in the direction of h.

tout is too close to **t** to start integration.

ifail $= 6$

Some element of **rtol** is less than zero.

ifail $= 7$

Some element of **atol** is less than zero.

ifail $= 8$

A previous call to this function returned with **itask** $= \langle value \rangle$ and no appropriate action was taken.

ifail $= 11$

Either the initialization function has not been called prior to the first call of this function or a communication array has become corrupted.

ifail $= 12$

Either the initialization function has not been called prior to the first call of this function or a communication array has become corrupted.

ifail $= 13$

com array is of insufficient length; length required $= \langle value \rangle$; actual length $lcom = \langle value \rangle$.

ifail $= 14$

All elements of **rtol** and **atol** are zero.

ifail $= 15$

Maximum number of steps taken on this call before reaching **tout**.

ifail $= 16$

Too much accuracy requested for precision of machine.

ifail $= 17$

A solution component has become zero when a purely relative tolerance (zero absolute tolerance) was selected for that component.

ifail $= 18$

The error test failed repeatedly with $|h| = hmin$.

ifail $= 19$

The corrector repeatedly failed to converge with $|h| = hmin$.

ifail $= 20$

The iteration matrix is singular.

ifail $= 21$

The corrector could not converge and the error test failed repeatedly.

ifail $= 22$

ires was set to -1 during a call to **res** and could not be resolved.

ifail $= 23$

ires was set to -2 during a call to **res**.

ifail $= 24$

The initial **ydot** could not be computed.

ifail $= 25$

Repeated occurrences of input constraint violations have been 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

The accuracy of the numerical solution may be controlled by a careful choice of the arguments **rtol** and **atol**. You are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is, you are advised to choose **itol** $= 0$ with **atol**(1) small but positive).

8 Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem. For banded systems the cost is proportional to **neq** $\times$ $(\mathbf{ml} + \mathbf{mu} + 1)^2$, while for full systems the cost is proportional to $\mathbf{neq}^3$. Note however that for moderately sized problems which are only mildly nonlinear the cost may be dominated by factors proportional to **neq** $\times$ $(\mathbf{ml} + \mathbf{mu} + 1)$ and $\mathbf{neq}^2$ respectively.

9 Example

For this function two examples are presented. There is a single example program for nag_ode_dae_dassl_gen (d02ne), with a main program and the code to solve the two example problems given in Example 1 (EX1) and Example 2 (EX2).

Example 1 (EX1)

This example solves the well-known stiff Robertson problem written in implicit form

$$\begin{array}{rcrcrcrcl} r_1 &=& -0.04a &+& 1.0\text{E}4bc & & &-& a' \\ r_2 &=& 0.04a &-& 1.0\text{E}4bc &-& 3.0\text{E}7b^2 &-& b' \\ r_3 &=& & & & & 3.0\text{E}7b^2 &-& c' \end{array}$$

with initial conditions $a = 1.0$ and $b = c = 0.0$ over the range $[0, 0.1]$ the BDF method (setup function nag_ode_dae_dassl_setup (d02mw) and nag_ode_dae_dassl_linalg (d02np)).

Example 2 (EX2)

This example illustrates the use of nag_ode_dae_dassl_gen (d02ne) to solve a simple algebraic problem by continuation. The equation $4 - 2y + 0.1e^y t = 0$ from $t = 0$ (where $y = 2$) to $t = 1$.

9.1 Program Text

```
     function d02ne_example

fprintf('d02ne example results\n\n');

fprintf('Example 1\n');
ex1;
fprintf('\nExample 2\n');
ex2;

function ex1
  % Stiff Robertson problem

  % Initialize the problem, specifying that the Jacobian is to be
  % evaluated analytically using the provided routine jac.
  neq    = nag_int(3);
  maxord = nag_int(5);
  jceval = 'Analytic';
  hmax   = 0;
  h0     = 0;
  itol   = nag_int(1);
  lcom   = nag_int(200);
  [icom, com, ifail] = d02mw(...
                             neq, maxord, jceval, hmax, h0, itol, lcom);

  % Specify that the Jacobian is banded
  mu = nag_int(2);
  ml = nag_int(1);
  [icom, ifail] = d02np(neq, ml, mu, icom);

  % Set initial values
  rtol = [1e-3; 1e-3; 1e-3];
  atol = [1e-6; 1e-6; 1e-6];
  y    = [1; 0; 0];
  ydot = zeros(neq,1);
  t    = 0;
  tout = 0.02;

  % Use the user parameter to pass the band dimensions through to jac.
  % An alternative would be to hard code values for ml and mu in jac.
  user = {ml, mu};

  fprintf('\n    t              y\n');
  fprintf('%8.4f   %12.6f %12.6f %12.6f\n', t, y);

  itask = nag_int(0);
  % Obtain the solution at 5 equally spaced values of T.
  for j = 1:5
    if ifail == 0
      [t, y, ydot, rtol, atol, itask, icom, com, user, ifail] = ...
      d02ne(...
            t, tout, y, ydot, rtol, atol, itask, @res1, @jac1, ...
            icom, com, 'user', user);
      fprintf('%8.4f   %12.6f %12.6f %12.6f\n', t, y);
      tout = tout + 0.02;
      icom = d02mc(icom);
    end
  end

  fprintf('\nThe integrator completed task, ITASK = %d\n', itask);

function ex2

  % Setup
  neq = nag_int(1);
  maxord = nag_int(5);
  jceval = 'Analytic';
  hmax   = 0;
  h0     = 0;
  itol   = nag_int(1);
```

```
  lcom   = nag_int(200);
  [icom, com, ifail] = d02mw(...
                             neq, maxord, jceval, hmax, h0, itol, lcom);

  % Set initial values
  rtol = [0];
  atol = [1e-8];
  t      = 0;
  tout = 0.2;
  y = [2];
  ydot = [0];
  fprintf('\n    t            y\n');
  fprintf('%8.4f   %12.6f\n', t, y);

  itask = nag_int(0);
  % Obtain the solution at 5 equally spaced values on continuation path.
  for j = 1:5
    if ifail == 0
      [t, y, ydot, rtol, atol, itask, icom, com, user, ifail] = ...
      d02ne(...
            t, tout, y, ydot, rtol, atol, itask, @res2, @jac2, ...
            icom, com);
      fprintf('%8.4f   %12.6f\n', t, y);
      tout = tout + 0.2;
      icom = d02mc(icom);
    end
  end
  fprintf('\nThe integrator completed task, ITASK = %d\n', itask);

function [pd, user] = jac1(neq, t, y, ydot, pd, cj, user)
  ml = user{1};
  mu = user{2};

  stride = 2*ml+mu+1;
  % Main diagonal pdfull(i,i), i=1,neq
  md = mu + ml + 1;
  pd(md) = -0.04 - cj;
  pd(md+stride) = -1.0e4*y(3) - 6.0e7*y(2) - cj;
  pd(md+2*stride) = -cj;
  % 1 sub-diagonal pdfull(i+1:i), i=1,neq-1
  ms = md + 1;
  pd(ms) = 0.04;
  pd(ms+stride) = 6.0e7*y(2);
  % First super-diagonal pdfull(i-1,i), i=2, neq
  ms = md - 1;
  pd(ms+stride) = 1.0e4*y(3);
  pd(ms+2*stride) = -1.0e4*y(2);
  % Second super-diagonal pdfull(i-2,i), i=3, neq
  ms = md - 2;
  pd(ms+2*stride) = 1.0e4*y(2);

function [r, ires, user] = res1(neq, t, y, ydot, ires, user)
  r = zeros(neq, 1);
  r(1) = -0.04*y(1) + 1.0e4*y(2)*y(3)                    - ydot(1);
  r(2) =  0.04*y(1) - 1.0e4*y(2)*y(3) - 3.0e7*y(2)*y(2) - ydot(2);
  r(3) =                              3.0e7*y(2)*y(2) - ydot(3);

function [pd, user] = jac2(neq, t, y, ydot, pd, cj, user)
  pd(1) = -2*y(1) + 0.1*t*y(1)*exp(y(1));

function [r, ires, user] = res2(neq, t, y, ydot, ires, user)
  r(1) = 4 - y(1)^2 + t*0.1*exp(y(1));
```

9.2 Program Results

```
     d02ne example results

Example 1

    t             y
  0.0000       1.000000     0.000000     0.000000
  0.0200       0.999204     0.000036     0.000760
  0.0400       0.998415     0.000036     0.001549
  0.0600       0.997631     0.000036     0.002333
  0.0800       0.996852     0.000036     0.003112
  0.1000       0.996080     0.000036     0.003884

The integrator completed task, ITASK = 3

Example 2

    t           y
  0.0000       2.000000
  0.2000       2.038016
  0.4000       2.078379
  0.6000       2.121462
  0.8000       2.167736
  1.0000       2.217821

The integrator completed task, ITASK = 3
```