NAG Library Function Document

nag_dae_ivp_dassl_gen (d02nec)

1 Purpose

nag_dae_ivp_dassl_gen (d02nec) is a function for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations.

2 Specification

```
#include <nag.h>
#include <nagd02.h>

void nag_dae_ivp_dassl_gen (Integer neq, double *t, double tout, double y[],
     double ydot[], double rtol[], double atol[], Integer *itask,
     void (*res)(Integer neq, double t, const double y[],
          const double ydot[], double r[], Integer *ires, Nag_Comm *comm),
     void (*jac)(Integer neq, double t, const double y[],
          const double ydot[], double pd[], double cj, Nag_Comm *comm),
     Integer icom[], double com[], Integer lcom, Nag_Comm *comm,
     NagError *fail)
```

3 Description

nag_dae_ivp_dassl_gen (d02nec) 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_dae_ivp_dassl_gen (d02nec) 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 $t = \mathbf{t}$ to $t = \mathbf{tout}$.

An outline of a typical calling program for nag_dae_ivp_dassl_gen (d02nec) is given below. It calls the DASSL implementation of the BDF integrator setup function nag_dae_ivp_dassl_setup (d02mwc) and the banded matrix setup function nag_dae_ivp_dassl_linalg (d02npc) (if required), and, if the integration needs to proceed, calls nag_dae_ivp_dassl_cont (d02mcc) before continuing the integration.

```
/* declarations */
   EXTERN res, jac
        .
        .
        .
 /* Initialize the integrator */
   nag_dae_ivp_dassl_setup(...);
 /* Is the Jacobian matrix banded? */
   if (banded) {nag_dae_ivp_dassl_linalg(...);}

 /* Set dt to the required temporal resolution */
 /* Set tend to the final time */
 /* Call the integrator for each temporal value */
   itask = 0;
   while (tout<tend && itask>-1) {
     nag_dae_ivp_dassl_gen (...);
     if (itask != 1)
        tout = MIN(tout+dt,tend);
        /* Print the solution */
```

```
    }
        .
        .
        .
```

4 References

None.

5 Arguments

1: **neq** – Integer *Input*

On entry: the number of differential-algebraic equations to be solved.

Constraint: **neq** ≥ 1.

2: **t** – double * *Input/Output*

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

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**).

3: **tout** – double *Input*

On entry: 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**.

4: **y**[**neq**] – double *Input/Output*

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

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

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

5: **ydot**[**neq**] – double *Input/Output*

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.

On exit: the time derivatives y' of the vector y at the last integration point.

6: **rtol**[dim] – double *Input/Output*

Note: the dimension, dim, of the array **rtol** depends on the value of **vector_tol** as set in nag_dae_ivp_dassl_setup (d02mwc); it must be at least

neq when **vector_tol** = Nag_TRUE;
1 when **vector_tol** = Nag_FALSE.

On entry: the relative local error tolerance.

Constraint: **rtol**$[i-1] \geq 0.0$, for $i = 1, 2, \ldots, n$

where $n =$ **neq** when **vector_tol** = Nag_TRUE and $n = 1$ otherwise.

On exit: **rtol** remains unchanged unless nag_dae_ivp_dassl_gen (d02nec) exits with **fail.code** = NE_ODE_TOL in which case the values may have been increased to values estimated to be appropriate for continuing the integration.

7: **atol**[dim] – double *Input/Output*

Note: the dimension, *dim*, of the array **atol** depends on the value of **vector_tol** as set in nag_dae_ivp_dassl_setup (d02mwc); it must be at least

neq when **vector_tol** = Nag_TRUE;
1 when **vector_tol** = Nag_FALSE.

On entry: the absolute local error tolerance.

Constraint: **atol**[$i-1$] ≥ 0.0, for $i = 1, 2, \ldots, n$

where $n =$ **neq** when **vector_tol** = Nag_TRUE and $n = 1$ otherwise.

On exit: **atol** remains unchanged unless nag_dae_ivp_dassl_gen (d02nec) exits with **fail.code** = NE_ODE_TOL in which case the values may have been increased to values estimated to be appropriate for continuing the integration.

8: **itask** – Integer * *Input/Output*

On initial entry: need not be set.

On exit: 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. **fail** should always be checked in such cases and the corrective action taken where appropriate.

itask must remain **unchanged** between calls to nag_dae_ivp_dassl_gen (d02nec).

9: **res** – function, supplied by the user *External Function*

res must evaluate the residual

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

The specification of **res** is:

```
void res (Integer neq, double t, const double y[],
     const double ydot[], double r[], Integer *ires, Nag_Comm *comm)
```

1: **neq** – Integer *Input*

On entry: the number of differential-algebraic equations being solved.

2: **t** – double *Input*

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

3: **y**[**neq**] – const double *Input*

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

4: **ydot**[**neq**] – const double *Input*

On entry: the derivative of the solution at the current point t.

5: **r**[**neq**] – double *Output*

On exit: **r**$[i-1]$ must contain the ith component of R, for $i = 1, 2, \ldots,$ **neq** where

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

6: **ires** – Integer * *Input/Output*

On entry: is always equal to zero.

On exit: **ires** should normally be left unchanged. However, if an illegal value of **y** is encountered, **ires** should be set to -1; nag_dae_ivp_dassl_gen (d02nec) 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 function; this will cause nag_dae_ivp_dassl_gen (d02nec) to exit with **fail.code** = NE_RES_FLAG.

7: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **res**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_dae_ivp_dassl_gen (d02nec) you may allocate memory and initialize these pointers with various quantities for use by **res** when called from nag_dae_ivp_dassl_gen (d02nec) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

10: **jac** – function, supplied by the user *External Function*

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** may be specified as NULLFN. 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_dae_ivp_dassl_setup (d02mwc).

The specification of **jac** is:

```
void jac (Integer neq, double t, const double y[],
     const double ydot[], double pd[], double cj, Nag_Comm *comm)
```

1: **neq** – Integer *Input*

On entry: the number of differential-algebraic equations being solved.

2: **t** – double *Input*

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

3: **y**[**neq**] – const double *Input*

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

4: **ydot**[**neq**] – const double *Input*

On entry: the derivative of the solution at the current point t.

5: **pd**[dim] – double *Input/Output*

Note: the dimension of the array **pd** will be **neq** $\times$ **neq** when the Jacobian is full and will be $(2 \times \mathbf{ml} + \mathbf{mu} + 1) \times \mathbf{neq}$ when the Jacobian is banded (that is, a prior call to nag_dae_ivp_dassl_linalg (d02npc) has been made).

On entry: **pd** is preset to zero before the call to **jac**.

On exit: if the Jacobian is full then $\mathbf{pd}[(j-1) \times \mathbf{neq} + i - 1] = J_{ij}$, for $i = 1, 2, \ldots, \mathbf{neq}$ and $j = 1, 2, \ldots, \mathbf{neq}$; if the Jacobian is banded then $\mathbf{pd}[(j-1) \times (2\mathbf{ml} + \mathbf{mu} + 1) + \mathbf{ml} + \mathbf{mu} + i - j] = J_{ij}$, for $\max(1, j - \mathbf{mu}) \leq i \leq \min(n, j + \mathbf{ml})$; (see also in nag_dgbtrf (f07bdc)).

6: **cj** – double *Input*

On entry: **cj** is a scalar constant which will be defined in nag_dae_ivp_dassl_gen (d02nec).

7: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **jac**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_dae_ivp_dassl_gen (d02nec) you may allocate memory and initialize these pointers with various quantities for use by **jac** when called from nag_dae_ivp_dassl_gen (d02nec) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

11: **icom**[$\mathbf{50} + \mathbf{neq}$] – Integer *Communication Array*

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

icom[21]
The order of the method to be attempted on the next step.

icom[22]
The order of the method used on the last step.

icom[25]
The number of steps taken so far.

icom[26]
The number of calls to **res** so far.

icom[27]
The number of evaluations of the matrix of partial derivatives needed so far.

icom[28]
The total number of error test failures so far.

icom[29]
The total number of convergence test failures so far.

12: **com**[**lcom**] – double *Communication Array*

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

com[2]
The step size to be attempted on the next step.

com[3]
: 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** $= 3$).

13: **lcom** – Integer *Input*

On entry: the dimension of the array **com**.

Constraint: **lcom** $\geq 40 + (maxorder + 4) \times$ **neq** + **neq** $\times p + q$ where $maxorder$ is the maximum order that can be used by the integration method (see **maxord** in nag_dae_ivp_dassl_setup (d02mwc)); $p =$ **neq** when the Jacobian is full and $p = (2 \times$ **ml** + **mu** $+ 1)$ when the Jacobian is banded; and, $q = ($**neq**$/($**ml** + **mu** $+ 1)) + 1$ when the Jacobian is to be evaluated numerically and $q = 0$ otherwise.

14: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

15: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_ARRAY_INPUT

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

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

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

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_CONV_CONT

The corrector could not converge and the error test failed repeatedly. **t** = ⟨*value*⟩. Stepsize h = ⟨*value*⟩.

The corrector repeatedly failed to converge with $|h| = hmin$. **t** = ⟨*value*⟩. Stepsize h = ⟨*value*⟩.

NE_CONV_JACOBG

The iteration matrix is singular. **t** = ⟨*value*⟩. Stepsize h = ⟨*value*⟩.

NE_CONV_ROUNDOFF

The error test failed repeatedly with $|h| = hmin$. **t** = ⟨*value*⟩. Stepsize h = ⟨*value*⟩.

NE_INITIALIZATION

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

NE_INT

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

NE_INT_2

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

NE_INT_ARG_LT

On entry, **neq** $= \langle value \rangle$.
Constraint: **neq** ≥ 1.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_MAX_STEP

Maximum number of steps taken on this call before reaching **tout**: **t** $= \langle value \rangle$, maximum number of steps $= \langle value \rangle$.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_ODE_TOL

A solution component has become zero when a purely relative tolerance (zero absolute tolerance) was selected for that component. **t** $= \langle value \rangle$, **y**$[I - 1] = \langle value \rangle$ for component $I = \langle value \rangle$.

Too much accuracy requested for precision of machine. **rtol** and **atol** were increased by scale factor R. Try running again with these scaled tolerances. **t** $= \langle value \rangle$, $R = \langle value \rangle$.

NE_REAL_2

tout is behind **t** in the direction of h: **tout** $-$ **t** $= \langle value \rangle$, $h = \langle value \rangle$.

tout is too close to **t** to start integration: **tout** $-$ **t** $= \langle value \rangle$: $hmin = \langle value \rangle$.

NE_REAL_ARG_EQ

On entry, **t** $= \langle value \rangle$.
Constraint: **tout** $\neq$ **t**.

NE_RES_FLAG

ires was set to -1 during a call to **res** and could not be resolved. **t** $= \langle value \rangle$. Stepsize $h = \langle value \rangle$.

ires was set to -2 during a call to **res**. **t** $= \langle value \rangle$. Stepsize $= \langle value \rangle$.

Repeated occurrences of input constraint violations have been detected. This could result in a potential infinite loop: **itask** $= \langle value \rangle$. Current violation corresponds to exit with **fail**.**code** $= \langle value \rangle$.

NE_SINGULAR_POINT

The initial **ydot** could not be computed. **t** $= \langle value \rangle$. Stepsize $h = \langle value \rangle$.

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 **vector_tol** = Nag_FALSE with **atol**[0] small but positive).

8 Parallelism and Performance

nag_dae_ivp_dassl_gen (d02nec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_dae_ivp_dassl_gen (d02nec) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 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 $\mathbf{neq} \times (\mathbf{ml} + \mathbf{mu} + 1)^2$, 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 $\mathbf{neq} \times (\mathbf{ml} + \mathbf{mu} + 1)$ and $\mathbf{neq}^2$ respectively.

10 Example

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

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

with initial conditions $a = 1.0$ and $b = c = 0.0$ over the range $[0, 0.1]$ the BDF method (setup function nag_dae_ivp_dassl_setup (d02mwc) and nag_dae_ivp_dassl_linalg (d02npc)).

10.1 Program Text

```
/* nag_dae_ivp_dassl_gen (d02nec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 *
 */

/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd02.h>

#ifdef __cplusplus
extern "C"
{
```

```
#endif
  static void NAG_CALL res(Integer neq, double t, const double y[],
                           const double ydot[], double r[], Integer *ires,
                           Nag_Comm *comm);
  static void NAG_CALL jac(Integer neq, double t, const double y[],
                           const double ydot[], double *pd, double cj,
                           Nag_Comm *comm);
  static void NAG_CALL myjac(Integer neq, Integer ml, Integer mu, double t,
                             const double y[], const double ydot[],
                             double *pd, double cj);
#ifdef __cplusplus
}
#endif
int main(void)
{
  /*Integer scalar and array declarations */
  Integer exit_status = 0, maxord = 5;
  Nag_Comm comm;
  Integer neq, licom, mu, ml, lcom;
  Integer i, itask, j;
  Nag_Boolean vector_tol;
  Integer *icom = 0;
  NagError fail;
  /*Double scalar and array declarations */
  double dt, h0, hmax, t, tout;
  double *atol = 0, *com = 0, *rtol = 0, *y = 0, *ydot = 0;
  static double ruser[2] = { -1.0, -1.0 };

  INIT_FAIL(fail);

  printf("nag_dae_ivp_dassl_gen (d02nec) Example Program Results\n\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  /* Set problem parameters required to allocate arrays */
  neq = 3;
  ml = 1;
  mu = 2;
  licom = 50 + neq;
  lcom = 40 + (maxord + 4) * neq + (2 * ml + mu + 1) * neq +
         2 * (neq / (ml + mu + 1) + 1);
  if (!(atol = NAG_ALLOC(neq, double)) || !(com = NAG_ALLOC(lcom, double))
      || !(rtol = NAG_ALLOC(neq, double)) || !(y = NAG_ALLOC(neq, double))
      || !(ydot = NAG_ALLOC(neq, double))
      || !(comm.iuser = NAG_ALLOC(2, Integer))
      || !(icom = NAG_ALLOC(licom, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }
  /* Initialize the problem, specifying that the Jacobian is to be */
  /* evaluated analytically using the provided routine jac.        */
  h0 = 0.0;
  hmax = 0.0;
  vector_tol = Nag_TRUE;
  /*
   * nag_dae_ivp_dassl_setup (d02mwc)
   * Implicit DAE/ODEs, stiff ivp, setup for nag_dae_ivp_dassl_gen (d02nec)
   */
  nag_dae_ivp_dassl_setup(neq, maxord, Nag_AnalyticalJacobian, hmax, h0,
                          vector_tol, icom, licom, com, lcom, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dae_ivp_dassl_setup (d02mwc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
  /* Specify that the Jacobian is banded.
   *
```

```
   * nag_dae_ivp_dassl_linalg (d02npc)
   * ODE/DAEs, ivp, linear algebra setup routine for
   * nag_dae_ivp_dassl_gen (d02nec)
   */
  nag_dae_ivp_dassl_linalg(neq, ml, mu, icom, licom, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_dae_ivp_dassl_linalg (d02npc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Set initial values */
  t = 0.00e0;
  tout = 0.00e0;
  dt = 0.020e0;
  for (i = 0; i < neq; i++) {
    rtol[i] = 1.00e-3;
    atol[i] = 1.00e-6;
    y[i] = 0.00e0;
    ydot[i] = 0.00e0;
  }
  y[0] = 1.00e0;
  /* Use the comm.iuser array to pass the band dimensions through to jac. */
  /* An alternative would be to hard code values for ml and mu in jac.    */
  comm.iuser[0] = ml;
  comm.iuser[1] = mu;
  printf("    t          y(1)         y(2)         y(3)\n");
  printf("%8.4f", t);
  for (i = 0; i < neq; i++)
    printf("%12.6f%s", y[i], (i + 1) % 3 ? " " : "\n");
  itask = 0;
  /*     Obtain the solution at 5 equally spaced values of t. */
  for (j = 0; j < 5; j++) {
    tout = tout + dt;
    /*
     * nag_dae_ivp_dassl_gen (d02nec)
     * dassl integrator
     */
    nag_dae_ivp_dassl_gen(neq, &t, tout, y, ydot, rtol, atol, &itask, res,
                          jac, icom, com, lcom, &comm, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_dae_ivp_dassl_gen (d02nec).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
    printf("%8.4f", t);
    for (i = 0; i < neq; i++)
      printf("%12.6f%s", y[i], (i + 1) % 3 ? " " : "\n");
    /*
     * nag_dae_ivp_dassl_cont (d02mcc)
     * dassl method continuation resetting function
     */
    nag_dae_ivp_dassl_cont(icom);
  }
  printf("\n");
  printf("  The integrator completed task, ITASK = %4" NAG_IFMT "\n", itask);

END:
  NAG_FREE(atol);
  NAG_FREE(com);
  NAG_FREE(rtol);
  NAG_FREE(y);
  NAG_FREE(ydot);
  NAG_FREE(comm.iuser);
  NAG_FREE(icom);

  return exit_status;
}
```

```
static void NAG_CALL res(Integer neq, double t, const double y[],
                         const double ydot[], double r[], Integer *ires,
                         Nag_Comm *comm)
{
  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback res, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  r[0] = (-(0.040e0 * y[0])) + 1.00e4 * y[1] * y[2] - ydot[0];
  r[1] = 0.040e0 * y[0] - 1.00e4 * y[1] * y[2] - 3.00e7 * y[1] * y[1] -
         ydot[1];
  r[2] = 3.00e7 * y[1] * y[1] - ydot[2];
  return;
}

static void NAG_CALL jac(Integer neq, double t, const double y[],
                         const double ydot[], double *pd, double cj,
                         Nag_Comm *comm)
{
  Integer ml, mu;
  if (comm->user[1] == -1.0) {
    printf("(User-supplied callback jac, first invocation.)\n");
    comm->user[1] = 0.0;
  }
  ml = comm->iuser[0];
  mu = comm->iuser[1];
  myjac(neq, ml, mu, t, y, ydot, pd, cj);
  return;
}

static void NAG_CALL myjac(Integer neq, Integer ml, Integer mu, double t,
                           const double y[], const double ydot[], double *pd,
                           double cj)
{
  Integer md, ms;
  Integer pdpd;
  pdpd = 2 * ml + mu + 1;
#define PD(I, J) pd[(J-1)*pdpd + I - 1]
  /*     Main diagonal PDFULL(i,i), i=1,neq */
  md = mu + ml + 1;
  PD(md, 1) = -0.040e0 - cj;
  PD(md, 2) = -1.00e4 * y[2] - 6.00e7 * y[1] - cj;
  PD(md, 3) = -cj;
  /*     1 Subdiagonal PDFULL(i+1:i), i=1,neq-1 */
  ms = md + ml;
  PD(ms, 1) = 0.040e0;
  PD(ms, 2) = 6.00e7 * y[1];
  /*     First superdiagonal PDFULL(i-1,i), i=2, neq */
  ms = md - 1;
  PD(ms, 2) = 1.00e4 * y[2];
  PD(ms, 3) = -1.00e4 * y[1];
  /*     Second superdiagonal PDFULL(i-2,i), i=3, neq */
  ms = md - 2;
  PD(ms, 3) = 1.00e4 * y[1];
  return;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_dae_ivp_dassl_gen (d02nec) Example Program Results

    t          y(1)         y(2)         y(3)
  0.0000    1.000000     0.000000     0.000000
(User-supplied callback res, first invocation.)
(User-supplied callback jac, first invocation.)
  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
```