# NAG Library Function Document

# nag_dae_ivp_dassl_setup (d02mwc)

## 1 Purpose

nag_dae_ivp_dassl_setup (d02mwc) is a setup function which must be called prior to the integrator nag_dae_ivp_dassl_gen (d02nec), if the DASSL implementation of Backward Differentiation Formulae (BDF) is to be used.

## 2 Specification

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

```
void nag_dae_ivp_dassl_setup (Integer neq, Integer maxord,
     Nag_EvaluateJacobian jceval, double hmax, double h0,
     Nag_Boolean vector_tol, Integer icom[], Integer licom, double com[],
     Integer lcom, NagError *fail)
```

## 3 Description

This integrator setup function must be called before the first call to the integrator nag_dae_ivp_dassl_gen (d02nec). nag_dae_ivp_dassl_setup (d02mwc) permits you to define options for the DASSL integrator, such as: whether the Jacobian is to be provided or is to be approximated numerically by the integrator; the initial and maximum step-sizes for the integration; whether relative and absolute tolerances are system wide or per system equation; and the maximum order of BDF method permitted.

## 4 References

None.

## 5 Arguments

1: **neq** – Integer *Input*

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

*Constraint*: **neq** $\geq 1$.

2: **maxord** – Integer *Input*

*On entry*: the maximum order to be used for the BDF method. Orders up to 5th order are available; setting **maxord** $> 5$ means that the maximum order used will be 5.

*Constraint*: $1 \leq$ **maxord**.

3: **jceval** – Nag_EvaluateJacobian *Input*

*On entry*: specifies the technique to be used to compute the Jacobian.

**jceval** = Nag_NumericalJacobian
The Jacobian is to be evaluated numerically by the integrator.

**jceval** = Nag_AnalyticalJacobian
You must supply a function to evaluate the Jacobian on a call to the integrator.

*Constraint*: **jceval** = Nag_NumericalJacobian or Nag_AnalyticalJacobian.

4: **hmax** – double *Input*

*On entry*: the maximum absolute step size to be allowed. Set $\mathbf{hmax} = 0.0$ if this option is not required.

*Constraint*: $\mathbf{hmax} \geq 0.0$.

5: **h0** – double *Input*

*On entry*: the step size to be attempted on the first step. Set $\mathbf{h0} = 0.0$ if the initial step size is calculated internally.

6: **vector_tol** – Nag_Boolean *Input*

*On entry*: a value to indicate the form of the local error test.

**vector_tol** = Nag_FALSE
: **rtol** and **atol** are single element vectors.

**vector_tol** = Nag_TRUE
: **rtol** and **atol** are vectors. This should be chosen if you want to apply different tolerances to each equation in the system.

See nag_dae_ivp_dassl_gen (d02nec).

**Note**: The tolerances must either both be single element vectors or both be vectors of length **neq**.

7: **icom**[**licom**] – Integer *Communication Array*

*On exit*: used to communicate details of the task to be carried out to the integration function nag_dae_ivp_dassl_gen (d02nec).

8: **licom** – Integer *Input*

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

*Constraint*: $\mathbf{licom} \geq \mathbf{neq} + 50$.

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

*On exit*: used to communicate problem parameters to the integration function nag_dae_ivp_dassl_gen (d02nec). This must be the same communication array as the array **com** supplied to nag_dae_ivp_dassl_gen (d02nec). In particular, the values of **hmax** and **h0** are contained in **com**.

10: **lcom** – Integer *Input*

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

*Constraints*:

the array **com** must be large enough for the requirements of nag_dae_ivp_dassl_gen (d02nec). That is:

if the system Jacobian is dense, $\mathbf{lcom} \geq 40 + (\mathbf{maxord} + 4) \times \mathbf{neq} + \mathbf{neq}^2$;
if the system Jacobian is banded,
$\mathbf{lcom} \geq 40 + (\mathbf{maxord} + 4) \times \mathbf{neq} + (2 \times \mathbf{ml} + \mathbf{mu} + 1) \times \mathbf{neq} + 2 \times (\mathbf{neq}/(\mathbf{ml} + \mathbf{mu} + 1) + 1)$.

Here **ml** and **mu** are the lower and upper bandwidths respectively that are to be specified in a subsequent call to nag_dae_ivp_dassl_linalg (d02npc).

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

The NAG error argument (see Section 3.6 in the Essential Introduction).

## 6 Error Indicators and Warnings

**NE_BAD_PARAM**

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

**NE_INT_ARG_GT**

On entry, **licom** = ⟨*value*⟩ and **neq** = ⟨*value*⟩.
Constraint: **licom** $\geq 50 +$ **neq**.

**NE_INT_ARG_LT**

On entry, **maxord** = ⟨*value*⟩.
Constraint: **maxord** $\geq 1$.

On entry, **neq** = ⟨*value*⟩.
Constraint: **neq** $\geq 1$.

**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.

**NE_REAL_ARG_LT**

On entry, **hmax** = ⟨*value*⟩.
Constraint: **hmax** $\geq 0.0$.

## 7 Accuracy

Not applicable.

## 8 Parallelism and Performance

Not applicable.

## 9 Further Comments

None.

## 10 Example

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

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

Differentiating the algebraic constraint once, a new algebraic constraint is obtained

$$xu + yv = 0.$$

Differentiating the algebraic constraint one more time, substituting for $x'$, $y'$, $u'$, $v'$ and using $x^2 + y^2 - 1 = 0$, the corresponding DAE system includes the differential equations and the algebraic equation in $\lambda$:

$$u^2 + v^2 - \lambda - y = 0.$$

We solve the reformulated DAE system

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

For our experiments, we take consistent initial values

$$y_1(0) = 1,\ y_2(0) = 0,\ y_3(0) = 0,\ y_4(0) = 1 \text{ and } y_5(0) = 1$$

at $t = 0$.

## 10.1 Program Text

```
/* nag_dae_ivp_dassl_setup (d02mwc) Example Program.
 *
 * Copyright 2009, Numerical Algorithms Group.
 *
 * Mark 9, 2009.
 *
 */

/* Pre-processor includes */
#include <stdio.h>
#include <math.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);
#ifdef __cplusplus
}
#endif
int main(void)
{
  /* Scalars */
  Integer     exit_status = 0;
  Integer     i, itask, neq, maxord, licom, lcom;
  double      h0, hmax, g1, g2, t, tout;
  /* Arrays */
  static double ruser[2] = {-1.0, -1.0};
  Integer     *icom = 0;
  double      *atol = 0, *com = 0, *rtol = 0, *y = 0, *ydot = 0;
  /* NAG types */
  Nag_Boolean vector_tol;
  Nag_Comm    comm;
  NagError    fail;

  INIT_FAIL(fail);

  printf("nag_dae_ivp_dassl_setup (d02mwc) Example Program Results\n\n");

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

  /* Set problem size and allocate accordingly */
  neq = 5;
  maxord = 5;
  licom = 50+neq;
  lcom = 40+(maxord+4+neq)*neq;
```

```
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)) ||
    !(icom = NAG_ALLOC(licom, Integer)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

/* Use vector of tolerances */
vector_tol = Nag_TRUE;
for (i = 0; i < neq; i++)
  {
    rtol[i] = 1.00e-8;
    atol[i] = 1.00e-8;
  }
/* Set up integrator to use supplied Jacobian, default step-sizes and
 * vector tolerances using nag_dae_ivp_dassl_setup (d02mwc).
 */
h0   = 0.0;
hmax = 0.0;
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;
  }

/* Set initial values*/
y[0] = 1.0;
y[1] = 0.0;
y[2] = 0.0;
y[3] = 1.0;
y[4] = 1.0;
for (i = 0; i < neq; i++) ydot[i] = 0.0;
t    = 0.0;
tout = 1.0;

/* Print header and initial values */
printf("%7s%12s%12s%12s%12s%12s\n", "t", "y_1", "y_2", "y_3", "y_4", "y_5");
printf("   %6.4f", t);
for (i = 0; i < neq; i++)
  printf("%11.6f%s", y[i], (i+1)%5?" ":"\n");

itask = 0;
while ((itask >= 0) && (itask <= 3) && (t < tout)) {
  /* Integrate using  nag_dae_ivp_dassl_gen (d02nec). */
  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;
    }
  else
    {
      printf("   %6.4f", t);
      for (i = 0; i < neq; i++) printf("%11.6f%s", y[i], (i+1)%5?" ":"\n");
      printf("\n  d02nec  returned with ITASK = %4ld\n\n", itask);
    }
}
if ((itask >= 0) && (itask <= 3))
  {
```

```
      g1 = y[0]*y[0] + y[1]*y[1] - 1.0;
      g2 = y[0]*y[2] + y[1]*y[3];
      printf("  The position-level constraint G1 =  %13.4e\n", g1);
      printf("  The velocity-level constraint G2 =  %13.4e\n", g2);
    }

 END:
  NAG_FREE(atol);
  NAG_FREE(com);
  NAG_FREE(rtol);
  NAG_FREE(y);
  NAG_FREE(ydot);
  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] = y[2] - ydot[0];
  r[1] = y[3] - ydot[1];
  r[2] = -y[4]*y[0] - ydot[2];
  r[3] = -y[4]*y[1] - ydot[3] - 1.0;
  r[4] = y[2]*y[2] + y[3]*y[3] - y[4] - y[1];
  return;
}
static void NAG_CALL jac(Integer neq, double t, const double y[],
                         const double ydot[], double *pd, double cj,
                         Nag_Comm *comm)
{
  Integer pdpd;
  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback jac, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  pdpd = neq;
#define PD(I, J) pd[(J-1)*pdpd + I-1]
  PD(1, 1) = -cj;
  PD(1, 3) = 1.0;
  PD(2, 2) = -cj;
  PD(2, 4) = 1.0;
  PD(3, 1) = -y[4];
  PD(3, 3) = -cj;
  PD(3, 5) = -y[0];
  PD(4, 2) = -y[4];
  PD(4, 4) = -cj;
  PD(4, 5) = -y[1];
  PD(5, 2) = -1.0;
  PD(5, 3) = 2.0*y[2];
  PD(5, 4) = 2.0*y[3];
  PD(5, 5) = -1.0;
  return;
}
```

## 10.2 Program Data

None.

### 10.3 Program Results

```
nag_dae_ivp_dassl_setup (d02mwc) Example Program Results

      t         y_1        y_2        y_3        y_4        y_5
   0.0000   1.000000   0.000000   0.000000   1.000000   1.000000
(User-supplied callback res, first invocation.)
(User-supplied callback jac, first invocation.)
   1.0000   0.867349   0.497701  -0.033748   0.058813  -0.493103

  d02nec  returned with ITASK =    3

  The position-level constraint G1 =    -8.5802e-09
  The velocity-level constraint G2 =    -3.0051e-08
```