NAG Library Function Document

nag_ode_ivp_adams_gen (d02cjc)

1 Purpose

nag_ode_ivp_adams_gen (d02cjc) integrates a system of first order ordinary differential equations over a range with suitable initial conditions, using a variable-order, variable-step Adams method until a user-specified function, if supplied, of the solution is zero, and returns the solution at specified points, if desired.

2 Specification

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

```
void nag_ode_ivp_adams_gen (Integer neq,
     void (*fcn)(Integer neq, double x, const double y[], double f[],
          Nag_User *comm),
     double *x, double y[], double xend, double tol, Nag_ErrorControl err_c,
     void (*output)(Integer neq, double *xsol, const double y[], Nag_User *comm),
     double (*g)(Integer neq, double x, const double y[], Nag_User *comm),
     Nag_User *comm, NagError *fail)
```

3 Description

nag_ode_ivp_adams_gen (d02cjc) advances the solution of a system of ordinary differential equations

$$y_i' = f_i(x, y_1, y_2, \ldots, y_{\mathbf{neq}}), \qquad i = 1, 2, \ldots, \mathbf{neq},$$

from $x = \mathbf{x}$ to $x = \mathbf{xend}$ using a variable-order, variable-step Adams method. The system is defined by **fcn**, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_{\mathbf{neq}}$. The initial values of $y_1, y_2, \ldots, y_{\mathbf{neq}}$ must be given at $x = \mathbf{x}$.

The solution is returned via **output** at specified points, if desired: this solution is obtained by C^1 interpolation on solution values produced by the method. As the integration proceeds a check can be made on the user-specified function $g(x, y)$ to determine an interval where it changes sign. The position of this sign change is then determined accurately. It is assumed that $g(x, y)$ is a continuous function of the variables, so that a solution of $g(x, y) = 0.0$ can be determined by searching for a change in sign in $g(x, y)$. The accuracy of the integration, the interpolation and, indirectly, of the determination of the position where $g(x, y) = 0.0$, is controlled by the arguments **tol** and **err_c**.

For a description of Adams methods and their practical implementation see Hall and Watt (1976).

4 References

Hall G and Watt J M (ed.) (1976) *Modern Numerical Methods for Ordinary Differential Equations* Clarendon Press, Oxford

5 Arguments

1: **neq** – Integer *Input*

On entry: the number of differential equations.

Constraint: $\mathbf{neq} \geq 1$.

2: **fcn** – function, supplied by the user *External Function*

fcn must evaluate the first derivatives y_i' (i.e., the functions f_i) for given values of their arguments $x, y_1, y_2, \ldots, y_{\mathbf{neq}}$.

The specification of **fcn** is:

```
void fcn (Integer neq, double x, const double y[], double f[], Nag_User *comm)
```

1: **neq** – Integer *Input*

On entry: the number of differential equations.

2: **x** – double *Input*

On entry: the value of the independent variable x.

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

On entry: **y**$[i-1]$ holds the value of the variable y_i, for $i = 1, 2, \ldots, \mathbf{neq}$.

4: **f**[**neq**] – double *Output*

On exit: **f**$[i-1]$ must contain the value of f_i, for $i = 1, 2, \ldots, \mathbf{neq}$.

5: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm** → **p** should be cast to the required type, e.g., `struct user *s = (struct user *)comm → p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

3: **x** – double * *Input/Output*

On entry: the initial value of the independent variable x.

Constraint: $\mathbf{x} \neq \mathbf{xend}$.

On exit: if g is supplied, **x** contains the point where $g(x, y) = 0.0$, unless $g(x, y) \neq 0.0$ anywhere on the range **x** to **xend**, in which case, **x** will contain **xend**. If g is not supplied **x** contains **xend**, unless an error has occurred, when it contains the value of x at the error.

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

On entry: the initial values of the solution $y_1, y_2, \ldots, y_{\mathbf{neq}}$ at $x = \mathbf{x}$.

On exit: the computed values of the solution at the final point $x = \mathbf{x}$.

5: **xend** – double *Input*

On entry: the final value of the independent variable.

xend < **x**
Integration proceeds in the negative direction.

Constraint: $\mathbf{xend} \neq \mathbf{x}$.

6: **tol** – double *Input*

On entry: a positive tolerance for controlling the error in the integration. Hence **tol** affects the determination of the position where $g(x, y) = 0.0$, if g is supplied.

nag_ode_ivp_adams_gen (d02cjc) has been designed so that, for most problems, a reduction in **tol** leads to an approximately proportional reduction in the error in the solution. However, the actual relation between **tol** and the accuracy achieved cannot be guaranteed. You are strongly recommended to call nag_ode_ivp_adams_gen (d02cjc) with more than one value for **tol** and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge, you might compare the results obtained by calling nag_ode_ivp_adams_gen (d02cjc) with **tol** $= 10.0^{-p}$ and **tol** $= 10.0^{-p-1}$ where p correct decimal digits are required in the solution.

Constraint: **tol** > 0.0.

7: **err_c** – Nag_ErrorControl *Input*

On entry: the type of error control. At each step in the numerical solution an estimate of the local error, *est*, is made. For the current step to be accepted the following condition must be satisfied:

$$est = \sqrt{\sum_{i=1}^{\mathbf{neq}} (e_i/(\tau_r \times |y_i| + \tau_a))^2} \leq 1.0$$

where τ_r and τ_a are defined by

err_c	τ_r	τ_a
Nag_Relative	**tol**	ϵ
Nag_Absolute	0.0	**tol**
Nag_Mixed	**tol**	**tol**

where ϵ is a small machine-dependent number and e_i is an estimate of the local error at y_i, computed internally. If the appropriate condition is not satisfied, the step size is reduced and the solution is recomputed on the current step. If you wish to measure the error in the computed solution in terms of the number of correct decimal places, then **err_c** should be set to Nag_Absolute on entry, whereas if the error requirement is in terms of the number of correct significant digits, then **err_c** should be set to Nag_Relative. If you prefer a mixed error test, then **err_c** should be set to Nag_Mixed. The recommended value for **err_c** is Nag_Mixed and this should be chosen unless there are good reasons for a different choice.

Constraint: **err_c** = Nag_Relative, Nag_Absolute or Nag_Mixed.

8: **output** – function, supplied by the user *External Function*

output permits access to intermediate values of the computed solution (for example to print or plot them), at successive user-specified points. It is initially called by nag_ode_ivp_adams_gen (d02cjc) with **xsol** = **x** (the initial value of x). You must reset **xsol** to the next point (between the current **xsol** and **xend**) where **output** is to be called, and so on at each call to **output**. If, after a call to **output**, the reset point **xsol** is beyond **xend**, nag_ode_ivp_adams_gen (d02cjc) will integrate to **xend** with no further calls to **output**; if a call to **output** is required at the point **xsol** = **xend**, then **xsol** must be given precisely the value **xend**.

The specification of **output** is:

```
void output (Integer neq, double *xsol, const double y[], Nag_User *comm)
```

1: **neq** – Integer *Input*

On entry: the number of differential equations.

2: **xsol** – double * *Input/Output*

On entry: the value of the independent variable x.

On exit: you must set **xsol** to the next value of x at which **output** is to be called.

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

On entry: the computed solution at the point **xsol**.

4: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm** → **p** should be cast to the required type, e.g., `struct user *s = (struct user *)comm → p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

If you do not wish to access intermediate output, the actual argument **output** must be the NAG defined null function pointer `NULLFN`.

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

g must evaluate $g(x, y)$ for specified values x, y. It specifies the function g for which the first position x where $g(x, y) = 0$ is to be found.

The specification of **g** is:

```
double g (Integer neq, double x, const double y[], Nag_User *comm)
```

1: **neq** – Integer *Input*

On entry: the number of differential equations.

2: **x** – double *Input*

On entry: the value of the independent variable x.

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

On entry: **y**$[i - 1]$ holds the value of the variable y_i, for $i = 1, 2, \ldots,$ **neq**.

4: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm** → **p** should be cast to the required type, e.g., `struct user *s = (struct user *)comm → p`, to obtain the original object's address with appropriate type. (See the argument **comm** below.)

If you do not require the root finding option, the actual argument **g** must be the NAG defined null double function pointer `NULLDFN`.

10: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **p**, of type Pointer, allows you to communicate information to and from **fcn**, **output** and **g**. An object of the required type should be declared, e.g., a structure, and its address assigned to the pointer **p** by means of a cast to Pointer in the calling program. E.g. `comm.p = (Pointer)&s`. The type pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

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

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

6 Error Indicators and Warnings

NE_2_REAL_ARG_EQ

On entry, **x** = ⟨*value*⟩ while **xend** = ⟨*value*⟩. These arguments must satisfy **x** ≠ **xend**.

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument **err_c** had an illegal value.

NE_INT_ARG_LT

On entry, **neq** = ⟨*value*⟩.
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.

NE_NO_SIGN_CHANGE

No change in sign of the function $g(x, y)$ was detected in the integration range.

NE_REAL_ARG_LE

On entry, **tol** must not be less than or equal to 0.0: **tol** = ⟨*value*⟩.

NE_TOL_PROGRESS

The value of **tol**, ⟨*value*⟩, is too small for the function to make any further progress across the integration range. Current value of **x** = ⟨*value*⟩.

NE_TOL_TOO_SMALL

The value of **tol**, ⟨*value*⟩, is too small for the function to take an initial step.

NE_XSOL_INCONSIST

On call ⟨*value*⟩ to the supplied print function **xsol** was set to a value behind the previous value of **xsol** in the direction of integration.
Previous **xsol** = ⟨*value*⟩, **xend** = ⟨*value*⟩, new **xsol** = ⟨*value*⟩.

NE_XSOL_NOT_RESET

On call ⟨*value*⟩ to the supplied print function **xsol** was not reset.

NE_XSOL_SET_WRONG

xsol was set to a value behind **x** in the direction of integration by the first call to the supplied print function.
The integration range is [⟨*value1*⟩, ⟨*value2*⟩], **xsol** = ⟨*value*⟩.

7 Accuracy

The accuracy of the computation of the solution vector **y** may be controlled by varying the local error tolerance **tol**. In general, a decrease in local error tolerance should lead to an increase in accuracy. You are advised to choose **err_c** = Nag_Mixed unless you have a good reason for a different choice.

If the problem is a root-finding one, then the accuracy of the root determined will depend on the properties of $g(x, y)$. You should try to code **g** without introducing any unnecessary cancellation errors.

8 Further Comments

If more than one root is required then nag_ode_ivp_adams_roots (d02qfc) should be used.

If the function fails with error exit of **fail.code** = NE_TOL_TOO_SMALL, then it can be called again with a larger value of **tol** if this has not already been tried. If the accuracy requested is really needed and cannot be obtained with this function, the system may be very stiff (see below) or so badly scaled that it cannot be solved to the required accuracy.

If the function fails with error exit of **fail.code** = NE_TOL_PROGRESS, it is probable that it has been called with a value of **tol** which is so small that a solution cannot be obtained on the range **x** to **xend**. This can happen for well-behaved systems and very small values of **tol**. You should, however, consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity (infinite value) of the solution, the function will usually stop with error exit of **fail.code** = NE_TOL_PROGRESS, unless overflow occurs first. Numerical integration cannot be continued through a singularity, and analytic treatment should be considered;

(b) for 'stiff' equations where the solution contains rapidly decaying components, the function will use very small steps in x (internally to nag_ode_ivp_adams_gen (d02cjc)) to preserve stability. This will exhibit itself by making the computing time excessively long, or occasionally by an exit with **fail.code** = NE_TOL_PROGRESS. Adams methods are not efficient in such cases.

9 Example

We illustrate the solution of four different problems. In each case the differential system (for a projectile) is

$$\begin{aligned} y' &= \tan\phi \\ v' &= \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi} \\ \phi' &= \frac{-0.032}{v^2} \end{aligned}$$

over an interval **x** = 0.0 to **xend** = 10.0 starting with values $y = 0.5$, $v = 0.5$ and $\phi = \pi/5$. We solve each of the following problems with local error tolerances 1.0e−4 and 1.0e−5.

(i) To integrate to $x = 10.0$ producing output at intervals of 2.0 until a point is encountered where $y = 0.0$.

(ii) As (i) but with no intermediate output.

(iii) As (i) but with no termination on a root-finding condition.

(iv) As (i) but with no intermediate output and no root-finding termination condition.

9.1 Program Text

```
/* nag_ode_ivp_adams_gen (d02cjc) Example Program.
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 * Mark 3 revised, 1994.
 * Mark 7 revised, 2001.
 *
 */

#include <nag.h>
#include <nagx04.h>
#include <math.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <nagd02.h>
#include <nagx01.h>

#ifdef __cplusplus
```

```
extern "C" {
#endif
static void NAG_CALL out(Integer neq, double *xsol, double y[], Nag_User *comm);
static void NAG_CALL fcn(Integer neq, double x, double y[], double f[],
                         Nag_User *comm);
static double NAG_CALL g(Integer neq, double x, double y[], Nag_User *comm);
#ifdef __cplusplus
}
#endif

struct user
{
  double  xend, h;
  Integer k;
  FILE    *fpout;
};

int main(int argc, char *argv[])
{
  FILE        *fpout;
  Integer     exit_status = 0, i, j, neq;
  Nag_User    comm;
  double      pi, tol, x, y[3];
  struct user s;
  NagError    fail;

  INIT_FAIL(fail);

  /* Check for command-line IO options */
  fpout = nag_example_file_io(argc, argv, "-results", NULL);
  fprintf(fpout, "nag_ode_ivp_adams_gen (d02cjc) Example Program Results\n");

  /* For communication with function out()
   * assign address of user defined structure
   * to Nag pointer.
   */
  comm.p = (Pointer)

  neq = 3;
  s.xend = 10.0;
  s.fpout = fpout;
  /* nag_pi (x01aac).
   * pi
   */
  pi = nag_pi;
  fprintf(fpout, "\nCase 1: intermediate output, root-finding\n");
  for (j = 4; j <= 5; ++j)
    {
      tol = pow(10.0, (double)(-j));
      fprintf(fpout, "\n  Calculation with tol = %10.1e\n", tol);
      x = 0.0;
      y[0] = 0.5;
      y[1] = 0.5;
      y[2] = pi / 5.0;
      s.k = 4;
      s.h = (s.xend - x) / (double)(s.k + 1);
      fprintf(fpout, "\n     X         Y(1)         Y(2)         Y(3)\n");

      /* nag_ode_ivp_adams_gen (d02cjc).
       * Ordinary differential equation solver using a
       * variable-order variable-step Adams method (Black Box)
       */
      nag_ode_ivp_adams_gen(neq, fcn, &x, y, s.xend, tol, Nag_Mixed, out, g,
                            &comm, &fail);
      if (fail.code != NE_NOERROR)
        {
          fprintf(fpout, "Error from nag_ode_ivp_adams_gen (d02cjc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
```

```
      fprintf(fpout, "\n  Root of Y(1) = 0.0 at %7.3f\n", x);
      fprintf(fpout, "\n  Solution is");
      for (i = 0; i < 3; ++i)
        fprintf(fpout, "%10.5f", y[i]);
      fprintf(fpout, "\n");
    }
  fprintf(fpout, "\n\nCase 2: no intermediate output, root-finding\n");
  for (j = 4; j <= 5; ++j)
    {
      tol = pow(10.0, (double)(-j));
      fprintf(fpout, "\n  Calculation with tol = %10.1e\n", tol);
      x = 0.0;
      y[0] = 0.5;
      y[1] = 0.5;
      y[2] = pi / 5.0;

      /* nag_ode_ivp_adams_gen (d02cjc), see above. */
      nag_ode_ivp_adams_gen(neq, fcn, &x, y, s.xend, tol, Nag_Mixed, NULLFN, g,
                            &comm, &fail);
      if (fail.code != NE_NOERROR)
        {
          fprintf(fpout, "Error from nag_ode_ivp_adams_gen (d02cjc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
      fprintf(fpout, "\n  Root of Y(1) = 0.0 at %7.3f\n", x);
      fprintf(fpout, "\n  Solution is");
      for (i = 0; i < 3; ++i)
        fprintf(fpout, "%10.5f", y[i]);
      fprintf(fpout, "\n");
    }
  fprintf(fpout, "\n\nCase 3: intermediate output, no root-finding\n");
  for (j = 4; j <= 5; ++j)
    {
      tol = pow(10.0, (double)(-j));
      fprintf(fpout, "\n  Calculation with tol = %10.1e\n", tol);
      x = 0.0;
      y[0] = 0.5;
      y[1] = 0.5;
      y[2] = pi / 5.0;
      s.k = 4;
      s.h = (s.xend - x) / (double)(s.k + 1);
      fprintf(fpout, "\n     X         Y(1)         Y(2)         Y(3)\n");

      /* nag_ode_ivp_adams_gen (d02cjc), see above. */
      nag_ode_ivp_adams_gen(neq, fcn, &x, y, s.xend, tol, Nag_Mixed, out,
                            NULLDFN, &comm, &fail);
      if (fail.code != NE_NOERROR)
        {
          fprintf(fpout, "Error from nag_ode_ivp_adams_gen (d02cjc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
    }

  fprintf(fpout, "\n\nCase 4: no intermediate output, no root-finding");
  fprintf(fpout, " ( integrate to xend)\n");
  for (j = 4; j <= 5; ++j)
    {
      tol = pow(10.0, (double)(-j));
      fprintf(fpout, "\n  Calculation with tol = %10.1e\n", tol);
      x = 0.0;
      y[0] = 0.5;
      y[1] = 0.5;
      y[2] = pi / 5.0;
      fprintf(fpout, "\n     X         Y(1)         Y(2)         Y(3)\n");
      fprintf(fpout, "%8.2f", x);
      for (i = 0; i < 3; ++i)
```

```
        fprintf(fpout, "%13.5f", y[i]);
      fprintf(fpout, "\n");

      /* nag_ode_ivp_adams_gen (d02cjc), see above. */
      nag_ode_ivp_adams_gen(neq, fcn, &x, y, s.xend, tol, Nag_Mixed, NULLFN,
                            NULLDFN, &comm, &fail);
      if (fail.code != NE_NOERROR)
        {
          fprintf(fpout, "Error from nag_ode_ivp_adams_gen (d02cjc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      fprintf(fpout, "%8.2f", x);
      for (i = 0; i < 3; ++i)
        fprintf(fpout, "%13.5f", y[i]);
      fprintf(fpout, "\n");
    }
 END:
  if (fpout != stdout) fclose(fpout);
  return exit_status;
}


static void NAG_CALL out(Integer neq, double *xsol, double y[],
                         Nag_User *comm)
{
  Integer     i;
  struct user *s = (struct user *) comm->p;

  fprintf(s->fpout, "%8.2f", *xsol);
  for (i = 0; i < 3; ++i)
    {
      fprintf(s->fpout, "%13.5f", y[i]);
    }
  fprintf(s->fpout, "\n");
  *xsol = s->xend - (double) s->k * s->h;
  s->k--;
}                                /* out */


static void NAG_CALL fcn(Integer neq, double x, double y[], double f[],
                         Nag_User *comm)
{
  double pwr;

  f[0] = tan(y[2]);
  f[1] = -0.032*tan(y[2])/y[1] - 0.02*y[1]/cos(y[2]);

  pwr = y[1];
  f[2] = -0.032/(pwr*pwr);
}                                /* fcn */


static double NAG_CALL g(Integer neq, double x, double y[], Nag_User *comm)
{
  return y[0];
}                                /* g */
```

9.2 Program Data

None.

9.3 Program Results

```
nag_ode_ivp_adams_gen (d02cjc) Example Program Results

Case 1: intermediate output, root-finding

  Calculation with tol =    1.0e-04

     X         Y(1)         Y(2)         Y(3)
   0.00      0.50000      0.50000      0.62832
   2.00      1.54931      0.40548      0.30662
   4.00      1.74229      0.37433     -0.12890
   6.00      1.00554      0.41731     -0.55068

  Root of Y(1) = 0.0 at   7.288

  Solution is  -0.00000   0.47486  -0.76011

  Calculation with tol =    1.0e-05

     X         Y(1)         Y(2)         Y(3)
   0.00      0.50000      0.50000      0.62832
   2.00      1.54933      0.40548      0.30662
   4.00      1.74232      0.37433     -0.12891
   6.00      1.00552      0.41731     -0.55069

  Root of Y(1) = 0.0 at   7.288

  Solution is  -0.00000   0.47486  -0.76010


Case 2: no intermediate output, root-finding

  Calculation with tol =    1.0e-04

  Root of Y(1) = 0.0 at   7.288

  Solution is  -0.00000   0.47486  -0.76011

  Calculation with tol =    1.0e-05

  Root of Y(1) = 0.0 at   7.288

  Solution is  -0.00000   0.47486  -0.76010


Case 3: intermediate output, no root-finding

  Calculation with tol =    1.0e-04

     X         Y(1)         Y(2)         Y(3)
   0.00      0.50000      0.50000      0.62832
   2.00      1.54931      0.40548      0.30662
   4.00      1.74229      0.37433     -0.12890
   6.00      1.00554      0.41731     -0.55068
   8.00     -0.74589      0.51299     -0.85371
  10.00     -3.62813      0.63325     -1.05152

  Calculation with tol =    1.0e-05

     X         Y(1)         Y(2)         Y(3)
   0.00      0.50000      0.50000      0.62832
   2.00      1.54933      0.40548      0.30662
   4.00      1.74232      0.37433     -0.12891
   6.00      1.00552      0.41731     -0.55069
   8.00     -0.74601      0.51299     -0.85372
  10.00     -3.62829      0.63326     -1.05153


Case 4: no intermediate output, no root-finding ( integrate to xend)
```

```
Calculation with tol =    1.0e-04

   X         Y(1)         Y(2)         Y(3)
  0.00      0.50000      0.50000      0.62832
 10.00     -3.62813      0.63325     -1.05152

Calculation with tol =    1.0e-05

   X         Y(1)         Y(2)         Y(3)
  0.00      0.50000      0.50000      0.62832
 10.00     -3.62829      0.63326     -1.05153
```