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 points specified by the user, if desired.

2. Specification

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

void nag_ode_ivp_adams_gen(Integer neq,
            void (*fcn)(Integer neq, double x, 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, double y[],
                           Nag_User *comm),
            double (*g)(Integer neq, double x, double y[], Nag_User *comm),
            Nag_User *comm, NagError *fail)
```

3. Description

The function 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 a function **fcn** supplied by the user, 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 the user-supplied function **output** at points specified by the user, 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 parameters **tol** and **err_c**.

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

4. Parameters

neq

Input: the number of differential equations.
Constraint: $\mathbf{neq} \geq 1$.

fcn

The function **fcn**, supplied by the user, 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, double y[], double f[], Nag_User *comm)
```

neq
: Input: the number of differential equations.

x
: Input: the value of the independent variable x.

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

f[neq]
: Output: **f**$[i-1]$ must contain the value of f_i, for $i = 1, 2, \ldots,$**neq**.

comm
: Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
: Input/Output: 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.)

x

Input: the initial value of the independent variable x.
Constraint: **x** $\neq$ **xend**.
Output: if g is supplied by the user, **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 by the user **x** contains **xend**, unless an error has occurred, when it contains the value of x at the error.

y[neq]

Input: the initial values of the solution $y_1, y_2, \ldots, y_{\mathbf{neq}}$ at $x =$ **x**.
Output: the computed values of the solution at the final point $x =$ **x**.

xend

Input: the final value of the independent variable. If **xend** $<$ **x**, integration proceeds in the negative direction.
Constraint: **xend** $\neq$ **x**.

tol

Input: 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 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. The user is strongly recommended to call nag_ode_ivp_adams_gen with more than one value for **tol** and to compare the results obtained to estimate their accuracy. In the absence of any prior knowledge, the user might compare the results obtained by calling nag_ode_ivp_adams_gen 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.

err_c

Input: 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:

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

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 the user wishes 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 the user prefers 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**.

output

The 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 with **xsol** = **x** (the initial value of x). The user 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 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, double y[], Nag_User *comm)
```

neq
Input: the number of differential equations.

xsol
Input: the value of the independent variable x.
Output: the user must set **xsol** to the next value of x at which **output** is to be called.

y[neq]
Input: the computed solution at the point **xsol**.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: 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 the user does not wish to access intermediate output, the actual argument **output** must be the NAG defined null function pointer `NULLFN`.

g

The 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, double y[], Nag_User *comm)
```

neq
Input: the number of differential equations.

x
Input: the value of the independent variable x.

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

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: 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 the user does not require the root finding option, the actual argument **g** must be the Nag defined null double function pointer `NULLDFN`.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:

p - Pointer
Input/Output: The pointer **p**, of type Pointer, allows the user to communicate information to and from the user-defined functions **fcn()**, **output()** and **g()**. An object of the required type should be declared by the user, 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.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

NE_INT_ARG_LT
On entry, **neq** must not be less than 1: **neq** = ⟨*value*⟩.

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

NE_2_REAL_ARG_EQ
On entry, **x** = ⟨*value*⟩ while **xend** = ⟨*value*⟩. These parameters must satisfy **x** ≠ **xend**.

NE_BAD_PARAM
On entry parameter **err_c** had an illegal value.

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

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*⟩.

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

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_ALLOC_FAIL
Memory allocation failed.

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 consult NAG for assistance.

6. 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**. The user 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) 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.

6.1. 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. Users are advised to choose **err_c** = **Nag_Mixed** unless they 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 $g(x, y)$. The user should try to code **g** without introducing any unnecessary cancellation errors.

6.2. References

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

7. See Also

nag_ode_ivp_bdf_gen (d02ejc)
nag_ode_ivp_rk_range (d02pcc)
nag_ode_ivp_adams_roots (d02qfc)

8. Example

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

$$y' = \tan\phi$$

$$v' = \frac{-0.032\tan\phi}{v} - \frac{0.02v}{\cos\phi}$$

$$\phi' = \frac{-0.032}{v^2}$$

over an interval $\mathbf{x} = 0.0$ to $\mathbf{xend} = 10.0$ 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.

8.1. Program Text

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

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

#ifdef NAG_PROTO
static void out(Integer neq, double *xsol, double y[], Nag_User *comm);
#else
static void out();
#endif

#ifdef NAG_PROTO
static void fcn(Integer neq, double x, double y[], double f[], Nag_User *comm);
#else
static void fcn();
#endif

#ifdef NAG_PROTO
static double g(Integer neq, double x, double y[], Nag_User *comm);
#else
static double g();
#endif

struct user
{
  double xend, h;
  Integer k;
};

main()
{
```

```
Integer i, j, neq;
double x, pi, tol;
double y[3];
Nag_User comm;
struct user s;

Vprintf("d02cjc Example Program Results\n");

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

neq = 3;
s.xend = 10.0;
pi = X01AAC;
Vprintf("\nCase 1: intermediate output, root-finding\n");
for (j = 4; j <= 5; ++j)
  {
    tol = pow(10.0, (double)(-j));
    Vprintf("\n  Calculation with tol = %8.1e\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);
    Vprintf("\n     X         Y(1)         Y(2)         Y(3)\n");

    d02cjc(neq, fcn, &x, y, s.xend, tol, Nag_Mixed, out, g, &comm,
           NAGERR_DEFAULT);

    Vprintf("\n  Root of Y(1) = 0.0 at %7.3f\n", x);
    Vprintf("\n  Solution is");
    for (i = 0; i < 3; ++i)
      Vprintf("%10.5f", y[i]);
    Vprintf("\n");
  }
Vprintf("\n\nCase 2: no intermediate output, root-finding\n");
for (j = 4; j <= 5; ++j)
  {
    tol = pow(10.0, (double)(-j));
    Vprintf("\n  Calculation with tol = %8.1e\n", tol);
    x = 0.0;
    y[0] = 0.5;
    y[1] = 0.5;
    y[2] = pi / 5.0;

    d02cjc(neq, fcn, &x, y, s.xend, tol, Nag_Mixed, NULLFN, g, &comm,
           NAGERR_DEFAULT);

    Vprintf("\n  Root of Y(1) = 0.0 at %7.3f\n", x);
    Vprintf("\n  Solution is");
    for (i = 0; i < 3; ++i)
      Vprintf("%10.5f", y[i]);
    Vprintf("\n");
  }
Vprintf("\n\nCase 3: intermediate output, no root-finding\n");
for (j = 4; j <= 5; ++j)
  {
    tol = pow(10.0, (double)(-j));
    Vprintf("\n  Calculation with tol = %8.1e\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);
    Vprintf("\n     X         Y(1)         Y(2)         Y(3)\n");
```

```
      d02cjc(neq, fcn, &x, y, s.xend, tol, Nag_Mixed, out, NULLDFN, &comm,
             NAGERR_DEFAULT);
    }

  Vprintf("\n\nCase 4: no intermediate output, no root-finding");
  Vprintf(" ( integrate to xend)\n");
  for (j = 4; j <= 5; ++j)
    {
      tol = pow(10.0, (double)(-j));
      Vprintf("\n  Calculation with tol = %8.1e\n", tol);
      x = 0.0;
      y[0] = 0.5;
      y[1] = 0.5;
      y[2] = pi / 5.0;
      Vprintf("\n     X         Y(1)         Y(2)         Y(3)\n");
      Vprintf("%8.2f", x);
      for (i = 0; i < 3; ++i)
        Vprintf("%13.5f", y[i]);
      Vprintf("\n");

      d02cjc(neq, fcn, &x, y, s.xend, tol, Nag_Mixed, NULLFN,
             NULLDFN, &comm, NAGERR_DEFAULT);

      Vprintf("%8.2f", x);
      for (i = 0; i < 3; ++i)
        Vprintf("%13.5f", y[i]);
      Vprintf("\n");
    }
  exit(EXIT_SUCCESS);
}                               /* main */


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

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

}                               /* out */


#ifdef NAG_PROTO
static void fcn(Integer neq, double x, double y[], double f[], Nag_User *comm)
#else
     static void fcn(neq, x, y, f, comm)
     Integer neq;
     double x;
     double y[], f[];
     Nag_User *comm;
#endif
{
  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 */


#ifdef NAG_PROTO
static double g(Integer neq, double x, double y[], Nag_User *comm)
#else
     static double g(neq, x, y, comm)
     Integer neq;
     double x;
     double y[];
     Nag_User *comm;
#endif
{
  return y[0];
}                               /* g */
```

8.2. Program Data

None.

8.3. Program Results

```
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
```