nag_ode_ivp_adams_roots (d02qfc)

1. Purpose

nag_ode_ivp_adams_roots (d02qfc) is a function for integrating a non-stiff system of first order ordinary differential equations using a variable-order variable-step Adams method. A root-finding facility is provided.

2. Specification

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

void nag_ode_ivp_adams_roots(Integer neqf,
            void (*fcn)(Integer neqf, double x, double y[],
                        double f[], Nag_User *comm),
            double *t, double y[], double tout,
            double (*g) (Integer neqf, double x, double y[],
                         double yp[], Integer k, Nag_User *comm),
            Nag_User *comm, Nag_ODE_Adams *opt, NagError *fail)
```

3. Description

Given the initial values $x, y_1, y_2, \ldots, y_{\mathbf{neqf}}$ the function integrates a non-stiff system of first order ordinary differential equations of the type, $y_i' = f_i(x, y_1, y_2, \ldots, y_{\mathbf{neqf}})$, for $i = 1, 2, \ldots,$**neqf**, from $x =$ **t** to $x =$ **tout** 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 $y_1, y_2, \ldots, y_{\mathbf{neqf}}$, and $y_1, y_2, \ldots, y_{\mathbf{neqf}}$ are supplied at $x =$ **t**. The function is capable of finding roots (values of x) of prescribed event functions of the form

$$g_j(x, y, y') = 0, \qquad j = 1, 2, \ldots, \mathbf{neqg}.$$

(See nag_ode_ivp_adams_setup (d02qwc) for the specification of **neqg**).

Each g_j is considered to be independent of the others so that roots are sought of each g_j individually. The root reported by the function will be the first root encountered by any g_j. Two techniques for determining the presence of a root in an integration step are available: the sophisticated method described in Watts (1985) and a simplified method whereby sign changes in each g_j are looked for at the ends of each integration step. The event functions are defined by a function **g** supplied by the user which evaluates g_j in terms of $x, y_1, \ldots, y_{\mathbf{neqf}}$ and $y_1', \ldots, y_{\mathbf{neqf}}'$. In one-step mode the function returns an approximation to the solution at each integration point. In interval mode this value is returned at the end of the integration range. If a root is detected this approximation is given at the root. The user selects the mode of operation, the error control, the root-finding technique and various integration inputs by a prior call of the setup routine nag_ode_ivp_adams_setup (d02qwc).

For a description of the practical implementation of an Adams formula see Shampine and Gordon (1975) and Shampine and Watts (1979).

4. Parameters

neqf

Input: the number of differential equations.
Constraint: **neqf** ≥ 1.

fcn

The function **fcn** must evaluate the functions f_i (that is the first derivatives y_i') for given values of its arguments $x, y_1, y_2, \ldots, y_{\mathbf{neqf}}$.
The specification of **fcn** is:

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

neqf
Input: the number of differential equations.

x
Input: the current value of the argument x.

y[neqf]
Input: **y**$[i-1]$ contains the current value of the argument y_i, for $i = 1, 2, \ldots,$**neqf**.

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

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.

t

Input: after a call to nag_ode_ivp_adams_setup (d02qwc) with **state** = **Nag_NewStart** (i.e., an initial entry), **t** must be set to the initial value of the independent variable x.
Output: the value of x at which y has been computed. This may be an intermediate output point, a root, **tout**, or a point at which an error has occurred. If the integration is to be continued, possibly with a new value for **tout**, **t** must not be changed.

y[neqf]
Input: the initial values of the solution $y_1, y_2, \ldots, y_{\mathbf{neqf}}$.
Output: the computed values of the solution at the exit value of **t**. If the integration is to be continued, possibly with a new value for **tout**, these values must not be changed.

tout
Input: the next value of x at which a computed solution is required. For the initial **t**, the input value of **tout** is used to determine the direction of integration. Integration is permitted in either direction. If **tout** = **t** on exit, **tout** must be reset beyond **t in the direction of integration**, before any continuation call.

g
The function **g** must evaluate a given component of $g(x, y, y')$ at a specifed point.

If root-finding is not required the actual argument for **g must** be the NAG defined null double function pointer `NULLDFN`.
The specification of **g** is:

```
double g(Integer neqf, double x, double y[], double yp[], Integer k,
         Nag_User *comm)
```

neqf
Input: the number of differential equations.

x
Input: the current value of the independent variable.

y[neqf]
Input: the current values of the dependent variables.

yp[neqf]
Input: the current values of the derivatives of the dependent variables.

k
Input: the component of g which must be evaluated.

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.

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()** 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`.

opt
Input: pointer to a structure of type Nag_ODE_Adams as initialised by the setup function nag_ode_ivp_adams_setup (d02qwc).
Output: the following structure members hold information as follows (see also Section 6):

root - Boolean
Output: if root-finding was required (**neqg** > 0 in a call to the setup function nag_ode_ivp_adams_setup (d02qwc)), then **root** specifies whether or not the output value of the parameter **t** is a root of one of the event functions. If **root** = **FALSE**, then no root was detected, whereas **root** = **TRUE** indicates a root.
If root-finding was not required (**neqg** $= 0$) then on exit **root** = **FALSE**.
If **root** = **FALSE**, then **opt.index, opt.type, opt.events** and **opt.resids** are indeterminate.

index - Integer
Output: the index k of the event equation $g_k(x, y, y') = 0$ for which the root has been detected.

type - Integer
Output: information about the root detected for the event equation defined by **opt.index**. The possible values of **type** with their interpretations are as follows:

type $= 1$
a simple root, or lack of distinguishing information available;
type $= 2$
a root of even multiplicity is believed to have been detected, that is no change in sign of the event function was found;
type $= 3$

a high order root of odd multiplicity;
type = 4
a possible root, but due to high multiplicity or a clustering of roots accurate evaluation of the event function was prohibited by round-off error and/or cancellation.

In general, the accuracy of the root is less reliable for values of **type** > 1.

events - Integer *
Output: array pointer containing information about the kth event function on a very small interval containing the root, **t**. All roots lying in this interval are considered indistinguishable numerically and therefore should be regarded as defining a root at **t**. The possible values of **events**$[j-1]$, $j = 1, 2, \ldots,$**neqg**, with their interpretations are as follows:

events$[j-1] = 0$
the jth event function did not have a root;
events$[j-1] = -1$
the jth event function changed sign from positive to negative about a root, in the direction of integration;
events$[j-1] = 1$
the jth event function changed sign from negative to positive about a root, in the direction of integration;
events$[j-1] = 2$
a root was identified, but no change in sign was observed.

resids - double *
Output: array pointer, **opt.resids**$[j-1]$, $j = 1, 2, \ldots,$**neqg**, contains value of the jth event function computed at the root, **t**

yp - double *
Output: array pointer to the approximate derivative of the solution component y_i at the output value of **t**. These values are obtained by the evaluation of $y' = f(x, y)$ except when the output value of the parameter **t** is **tout** and **opt.tcurr** $\neq$ **tout**, in which case they are obtained by interpolation.

tcurr - double
Output: the value of the independent variable which the integrator has actually reached. **tcurr** will always be at least as far as the output value of the argument **t** in the direction of integration, but may be further.

hlast - double
Output: the last successful step size used in the integration.

hnext - double
Output: the next step size which the integration would attempt.

ord_last - Integer
Output: the order of the method last used (successfully) in the integration.

ord_next - Integer
Output: the order of the method which the integration would attempt on the next step.

nsuccess - Integer
Output: the number of integration steps attempted that have been successful since the start of the current problem.

nfail - Integer
Output: the number of integration steps attempted that have failed since the start of the current problem.

tolfac - double
Output: a tolerance scale factor, **tolfac** ≥ 1.0, returned when nag_ode_ivp_adams_roots exits with **fail.code** = **NE_ODE_TOL**. If **rtol** and **atol** are uniformly scaled up by a factor of **tolfac** and nag_ode_ivp_adams_setup (d02qwc) is called, the next call to nag_ode_ivp_adams_roots is deemed likely to succeed.

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

5. Error Indications and Warnings

NE_NO_SETUP
The setup function nag_ode_ivp_adams_setup (d02qwc) has not been called.

NE_SETUP_ERROR
The call to setup function nag_ode_ivp_adams_setup (d02qwc) produced an error.

NE_NEQF
The value of **neqf** supplied is not the same as that given to the setup function nag_ode_ivp_adams_setup (d02qwc). **neqf** = ⟨*value*⟩ but the value given to nag_ode_ivp_adams_setup (d02qwc) was ⟨*value*⟩.

NE_T_SAME_TOUT
On entry **tout** = **t**, **t** is ⟨*value*⟩.

NE_T_CHANGED
The value of **t** has been changed from ⟨*value*⟩ to ⟨*value*⟩. This is not permitted on a continuation call.

NE_DIRECTION_CHANGE
The value of **tout**, ⟨*value*⟩, indicates a change in the integration direction. This is not permitted on a continuation call.

NE_TOUT_TCRIT
tout = ⟨*value*⟩ but **crit** was set **TRUE** in setup call and integration cannot be attempted beyond **tcrit** = ⟨*value*⟩.

NE_MAX_STEP
The maximum number of steps have been attempted.

If integration is to be continued then the routine may be called again and a further **max_step** steps will be attempted (see nag_ode_ivp_adams_setup (d02qwc) for details of **max_step**).

NE_ODE_TOL
The error tolerances are too stringent. **rtol** and **atol** should be scaled up by the factor **opt.tolfac** and the integration function re-entered. **opt.tolfac** = ⟨*value*⟩ (see Section 6).

NE_WEIGHT_ZERO
An error weight has become zero during the integration, see d02qwc document; **atol**[⟨*value*⟩] was set to 0.0 but **y**[⟨*value*⟩] is now 0.0. Integration successful as far as **t** = ⟨*value*⟩.

The value of the array index is returned in **fail.errnum**.

NE_STIFF_PROBLEM
The problem appears to be stiff.

(See Chapter Introduction for a discussion of the term 'stiff'). Although it is inefficient to use this integrator to solve stiff problems, integration may be continued by resetting **fail.code** and calling the routine again.

NE_SINGULAR_POINT
A change in sign of an event function has been detected but the root-finding process appears to have converged to a singular point of **t** rather than a root.

Integration may be continued by calling the routine again.

NE_NO_G_FUN
Root finding has been requested by setting **neqg** > 0, **neqg** = ⟨*value*⟩, but argument **g** is a null function.

6. Further Comments

If the function fails with **fail.code** = **NE_ODE_TOL**, then the combination of **atol** and **rtol** may be so small that a solution cannot be obtained, in which case the function should be called again

using larger values for **rtol** and/or **atol** when calling the setup function nag_ode_ivp_adams_setup (d02qwc). If the accuracy requested is really needed then the user should consider whether there is a more fundamental difficulty. For example:

(a) in the region of a singularity the solution components will usually be of a large magnitude. The function could be used in one-step mode to monitor the size of the solution with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary;

(b) for 'stiff' equations, where the solution contains rapidly decaying components, the function will require a very small step size to preserve stability. This will usually be exhibited by excessive computing time and sometimes an error exit with **fail.code** = **NE_ODE_TOL**, but usually an error exit with **fail.code** = **NE_MAX_STEP** or **NE_STIFF_PROBLEM**. The Adams methods are not efficient in such cases. A high proportion of failed steps (see parameter **opt.nfail**) may indicate stiffness but there may be other reasons for this phenomenon.

nag_ode_ivp_adams_roots can be used for producing results at short intervals (for example, for graph plotting); the user should set **crit** = **TRUE** and **tcrit** to the last output point required in a prior call to nag_ode_ivp_adams_setup (d02qwc) and then set **tout** appropriately for each output point in turn in the call to nag_ode_ivp_adams_roots.

The structure **opt** will contain pointers which have been allocated memory by calls to nag_ode_ivp_adams_setup (d02qwc). This allocated memory is then accessed by nag_ode_ivp_adams_roots and, if required, nag_ode_ivp_adams_interp (d02qzc). When all calls to these functions have been completed the function nag_ode_ivp_adams_free (d02qyc) may be called to free memory allocated to the structure.

6.1. Accuracy

The accuracy of integration is determined by the parameters **vectol**, **rtol** and **atol** in a prior call to nag_ode_ivp_adams_setup (d02qwc). Note that only the local error at each step is controlled by these parameters. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the properties of the differential equation system. The code is designed so that a reduction in the tolerances should lead to an approximately proportional reduction in the error. The user is strongly recommended to call nag_ode_ivp_adams_roots with more than one set of tolerances and to compare the results obtained to estimate their accuracy.

The accuracy obtained depends on the type of error test used. If the solution oscillates around zero a relative error test should be avoided, whereas if the solution is exponentially increasing an absolute error test should not be used. If different accuracies are required for different components of the solution then a component-wise error test should be used. For a description of the error test see the specifications of the parameters **vectol**, **atol** and **rtol** in the routine document for nag_ode_ivp_adams_setup (d02qwc).

The accuracy of any roots located will depend on the accuracy of integration and may also be restricted by the numerical properties of $g(x, y, y')$. When evaluating g the user should try to write the code so that unnecessary cancellation errors will be avoided.

6.2. References

Shampine L F and Gordon M K (1975) *Computer Solution of Ordinary Differential Equations - The Initial Value Problem* W H Freeman & Co., San Fransisco.

Shampine L F and Watts H A (1979) *DEPAC - Design of a user oriented package of ODE solvers* Sandia National Laboratory Report SAND79-2374.

Watts H A (1985) *RDEAM - An Adams ODE Code with Root Solving Capability* Sandia National Laboratory Report SAND85-1595.

7. See Also

nag_ode_ivp_adams_gen (d02cjc)
nag_ode_ivp_adams_setup (d02qwc)
nag_ode_ivp_adams_free (d02qyc)
nag_ode_ivp_adams_interp (d02qzc)

8. Example

We solve the equation

$$y'' = -y, \qquad y(0) = 0,\ y'(0) = 1$$

reposed as

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= -y_1 \end{aligned}$$

over the range $[0, 10.0]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$ using vector error control (**vectol** = **TRUE**) and computation of the solution at **tout** = 10.0 with **tcrit** = 10.0 (**crit** = **TRUE**). Also, we use nag_ode_ivp_adams_roots to locate the positions where $y_1 = 0.0$ or where the first component has a turning point, that is $y_1' = 0.0$.

8.1. Program Text

```
/* nag_ode_ivp_adams_roots(d02qfc) Example Program
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 */

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

#ifdef NAG_PROTO
static void ftry02(Integer neqf, double x, double y[], double yp[],
                   Nag_User *comm);
#else
static void ftry02();
#endif

#ifdef NAG_PROTO
static double gtry02(Integer neqf, double x, double y[], double yp[],
                     Integer k, Nag_User *comm);
#else
static double gtry02();
#endif

#define NEQF 2
#define NEQG 2

main()
{
  double y[NEQF], atol[NEQF], rtol[NEQF];
  Boolean crit, alter_g, vectol, one_step, sophist;
  double t, tout, tcrit;
  Integer i, max_step, neqf, neqg;
  Nag_Start state;
  Nag_ODE_Adams opt;

  Vprintf("d02qfc Example Program Results\n");

  neqf = NEQF;
  neqg = NEQG;
  tcrit = 10.0;
  state = Nag_NewStart;
  vectol = TRUE;
  one_step = FALSE;
  crit = TRUE;
  max_step = 0;
  sophist = TRUE;
  for (i = 0; i <= 1; ++i)
    {
```

```
      rtol[i] = 0.0001;
      atol[i] = 1e-06;
    }

  d02qwc(&state, neqf, vectol, atol, rtol, one_step, crit,
         tcrit, 0.0, max_step, neqg, &alter_g, sophist, &opt,
         NAGERR_DEFAULT);

  t = 0.0;
  tout = tcrit;
  y[0] = 0.0;
  y[1] = 1.0;

  do
    {
      d02qfc(neqf, ftry02, &t, y, tout, gtry02, NAGUSER_DEFAULT, &opt,
             NAGERR_DEFAULT);

      if (opt.root)
        {
          Vprintf("\nRoot at %11.5e\n", t);
          Vprintf("for event equation %1ld", opt.index);
          Vprintf(" with type %1ld", opt.type);
          Vprintf(" and residual %11.5e\n", opt.resids[opt.index-1]);

          Vprintf(" Y(1) = %11.5e   Y'(1) = %11.5e\n", y[0], opt.yp[0]);

          for (i = 1; i <= neqg; ++i)
            {
              if (i != opt.index && opt.events[i-1] != 0)
                {
                  Vprintf("and also for event equation %1ld", i);
                  Vprintf(" with type %1ld", opt.events[i-1]);
                  Vprintf(" and residual %11.5e\n", opt.resids[i-1]);
                }
            }
        }

    } while (opt.tcurr < tout && opt.root);

  /* Free the memory which was allocated by
   * d02qwc to the pointers inside opt.
   */
  d02qyc(&opt);

  exit(EXIT_SUCCESS);
}                               /* main */


#ifdef NAG_PROTO
static void ftry02(Integer neqf, double x, double y[], double yp[],
                   Nag_User *comm)
#else
     static void ftry02(neqf, x, y, yp, comm)
     Integer neqf;
     double x;
     double y[], yp[];
     Nag_User *comm;
#endif
{
  yp[0] = y[1];
  yp[1] = -y[0];
}                               /* ftry02 */


#ifdef NAG_PROTO
static double gtry02(Integer neqf, double x, double y[], double yp[],
                     Integer k, Nag_User *comm)
#else
     static double gtry02(neqf, x, y, yp, k, comm)
```

```
        Integer neqf;
        double x;
        double y[], yp[];
        Integer k;
        Nag_User *comm;
#endif
{
  if (k == 1) return yp[0];
  else return y[0];
}                                /* gtry02 */
```

8.2. Program Data

None.

8.3. Program Results

```
d02qfc Example Program Results

Root at 0.00000e+00
for event equation 2 with type 1 and residual 0.00000e+00
 Y(1) = 0.00000e+00   Y'(1) = 1.00000e+00

Root at 1.57076e+00
for event equation 1 with type 1 and residual -5.92381e-16
 Y(1) = 1.00003e+00   Y'(1) = -5.92381e-16

Root at 3.14151e+00
for event equation 2 with type 1 and residual -1.28576e-16
 Y(1) = -1.28576e-16   Y'(1) = -1.00012e+00

Root at 4.71228e+00
for event equation 1 with type 1 and residual 3.54189e-16
 Y(1) = -1.00010e+00   Y'(1) = 3.54189e-16

Root at 6.28306e+00
for event equation 2 with type 1 and residual 2.47328e-15
 Y(1) = 2.47328e-15   Y'(1) = 9.99979e-01

Root at 7.85379e+00
for event equation 1 with type 1 and residual -3.20697e-15
 Y(1) = 9.99970e-01   Y'(1) = -3.20697e-15

Root at 9.42469e+00
for event equation 2 with type 1 and residual -2.90637e-15
 Y(1) = -2.90637e-15   Y'(1) = -9.99854e-01
```