# NAG Library Function Document

# 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, const double y[], double f[],
           Nag_User *comm),
     double *t, double y[], double tout,
     double  (*g)(Integer neqf, double x, const double y[],
             const 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, $y_i' = f_i\left(x, y_1, y_2, \ldots, y_{\mathbf{neqf}}\right)$, for $i = 1, 2, \ldots, \mathbf{neqf}$, from $x = \mathbf{t}$ to $x = \mathbf{tout}$ using a variable-order variable-step Adams' method. The system is defined by **fcn**, which evaluates $f_i$ in terms of $x$ and $y_1, y_2, \ldots, y_{\mathbf{neqf}}$, and $y_1, y_2, \ldots, y_{\mathbf{neqf}}$ are supplied at $x = \mathbf{t}$. The function is capable of finding roots (values of $x$) of prescribed event functions of the form

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

(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 **g**, 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. You need to select the mode of operation, the error control, the root-finding technique and various integration inputs with a prior call of the setup function 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 References

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

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

Watts H A (1985) RDEAM – An Adams ODE code with root solving capability *Report SAND85-1595* Sandia National Laboratory

## 5 Arguments

1: **neqf** – Integer *Input*

*On entry*: the number of differential equations.

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

2: **fcn** – function, supplied by the user *External 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, const double y[], double f[],
      Nag_User *comm)
```

1: **neqf** – Integer *Input*

*On entry*: the number of differential equations.

2: **x** – double *Input*

*On entry*: the current value of the argument $x$.

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

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

4: **f**[**neqf**] – double *Output*

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

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.

3: **t** – double * *Input/Output*

*On entry*: 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 $x$.

*On exit*: 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.

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

*On entry*: the initial values of the solution $y_1, y_2, \ldots, y_{\mathbf{neqf}}$.

*On exit*: 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.

5: **tout** – double *Input*

*On entry*: 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.

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

**g** must evaluate a given component of $g(x, y, y')$ at a specified 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, const double y[],
        const double yp[], Integer k, Nag_User *comm)
```

1: **neqf** – Integer *Input*

*On entry*: the number of differential equations.

2: **x** – double *Input*

*On entry*: the current value of the independent variable.

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

*On entry*: the current values of the dependent variables.

4: **yp**[**neqf**] – const double *Input*

*On entry*: the current values of the derivatives of the dependent variables.

5: **k** – Integer *Input*

*On entry*: the component of $g$ which must be evaluated.

6: **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.

7: **comm** – Nag_User *

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

**p** – Pointer

*On entry/exit*: the pointer **comm**→**p**, of type Pointer, allows you to communicate information to and from **fcn** and **g**. An object of the required type should be declared,

e.g., a structure , and its address assigned to the pointer **comm→p** by means of a cast to Pointer in the calling program. E.g. `comm.p = (Pointer)&s`.

8: **opt** – Nag_ODE_Adams *

Pointer to a structure of type Nag_ODE_Adams as initialized by the setup function nag_ode_ivp_adams_setup (d02qwc) with the following members:

**root** – Nag_Boolean *Output*

*On exit*: 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 argument **t** is a root of one of the event functions. If **root** = Nag_FALSE, then no root was detected, whereas **root** = Nag_TRUE indicates a root.

If root-finding was not required (**neqg** $= 0$) then on exit **root** = Nag_FALSE.

If **root** = Nag_FALSE, then **opt→index**, **opt→type**, **opt→events** and **opt→resids** are indeterminate.

**index** – Integer *Output*

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

**type** – Integer *Output*

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

If **type** $= 1$, a simple root, or lack of distinguishing information available.

If **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.

If **type** $= 3$, a high order root of odd multiplicity.

If **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*

*On exit*: array pointer containing information about the $k$th 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 $j$th event function did not have a root;

**events**$[j-1] = -1$, the $j$th event function changed sign from positive to negative about a root, in the direction of integration;

**events**$[j-1] = 1$, the $j$th 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*

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

**yp** – double *Output*

*On exit*: 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 argument **t** is **tout** and **opt**→**tcurr** $\neq$ **tout**, in which case they are obtained by interpolation.

**tcurr** – double *Output*

*On exit*: 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*

*On exit*: the last successful step size used in the integration.

**hnext** – double *Output*

*On exit*: the next step size which the integration would attempt.

**ord_last** – Integer *Output*

*On exit*: the order of the method last used (successfully) in the integration.

**ord_next** – Integer *Output*

*On exit*: the order of the method which the integration would attempt on the next step.

**nsuccess** – Integer *Output*

*On exit*: the number of integration steps attempted that have been successful since the start of the current problem.

**nfail** – Integer *Output*

*On exit*: the number of integration steps attempted that have failed since the start of the current problem.

**tolfac** – double *Output*

*On exit*: a tolerance scale factor, **tolfac** $\geq 1.0$, returned when nag_ode_ivp_adams_roots (d02qfc) 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 (d02qfc) is deemed likely to succeed.

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

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

## 6 Error Indicators and Warnings

**NE_DIRECTION_CHANGE**

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

**NE_MAX_STEP**

The maximum number of steps have been attempted. If integration is to be continued then the function 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_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_NO_G_FUN**

Root finding has been requested by setting **neqg** $> 0$, **neqg** $= \langle value \rangle$, but argument **g** is a null function.

**NE_NO_SETUP**

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

**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** $= \langle value \rangle$ (see Section 9).

**NE_SETUP_ERROR**

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

**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 function again.

**NE_STIFF_PROBLEM**

The problem appears to be stiff. (See the d02 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** and calling the function again.

**NE_T_CHANGED**

The value of **t** has been changed from $\langle value \rangle$ to $\langle value \rangle$. This is not permitted on a continuation call.

**NE_T_SAME_TOUT**

On entry, **tout** $=$ **t**, **t** is $\langle value \rangle$.

**NE_TOUT_TCRIT**

**tout** $= \langle value \rangle$ but **crit** was set Nag_TRUE in setup call and integration cannot be attempted beyond **tcrit** $= \langle value \rangle$.

**NE_WEIGHT_ZERO**

An error weight has become zero during the integration, see d02qwc document; **atol**$[\langle value \rangle]$ was set to 0.0 but **y**$[\langle value \rangle]$ is now 0.0. Integration successful as far as **t** $= \langle value \rangle$. The value of the array index is returned in **fail.errnum**.

## 7 Accuracy

The accuracy of integration is determined by the arguments **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 arguments. 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. You are strongly recommended to call nag_ode_ivp_adams_roots (d02qfc) 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 arguments **vectol**, **atol** and **rtol** in the function 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 $g$ you should try to write the code so that unnecessary cancellation errors will be avoided.

## 8 Parallelism and Performance

Not applicable.

## 9 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 you 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 argument **opt**→**nfail**) may indicate stiffness but there may be other reasons for this phenomenon.

nag_ode_ivp_adams_roots (d02qfc) can be used for producing results at short intervals (for example, for graph plotting); you should set **crit** = Nag_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 (d02qfc).

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 (d02qfc) 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.

## 10 Example

We solve the equation

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

reposed as

$$y_1' = y_2$$
$$y_2' = -y_1$$

over the range $[0, 10.0]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$ using vector error control (**vectol** = Nag_TRUE) and computation of the solution at **tout** = 10.0 with **tcrit** = 10.0 (**crit** = Nag_TRUE). Also, we use nag_ode_ivp_adams_roots (d02qfc) to locate the positions where $y_1 = 0.0$ or where the first component has a turning point, that is $y_1' = 0.0$.

## 10.1 Program Text

```
/* nag_ode_ivp_adams_roots (d02qfc) Example Program.
 *
 * Copyright 1991 Numerical Algorithms Group.
 *
 * Mark 2, 1991.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

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

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL ftry02(Integer neqf, double x, const double y[],
                            double yp[], Nag_User *comm);
static double NAG_CALL gtry02(Integer neqf, double x, const double y[],
                              const double yp[], Integer k, Nag_User *comm);
#ifdef __cplusplus
}
#endif

#define NEQF 2
#define NEQG 2
int main(void)
{
  static Integer use_comm[2] = {1, 1};
  Nag_Boolean   alter_g, crit, one_step, sophist, vectol;
  Integer       exit_status = 0, i, max_step, neqf, neqg;
  NagError      fail;
  Nag_ODE_Adams opt;
  Nag_Start     state;
  Nag_User      comm;
  double        *atol = 0, *rtol = 0, t, tcrit, tout, *y = 0;

  INIT_FAIL(fail);

  printf("nag_ode_ivp_adams_roots (d02qfc) Example Program Results\n");

  /* For communication with user-supplied functions: */
  comm.p = (Pointer)

  neqf = NEQF;
  neqg = NEQG;
  if (neqf < 1)
    {
      exit_status = 1;
      return exit_status;
    }
  else
    {
      if (!(y = NAG_ALLOC(neqf, double)) ||
          !(atol = NAG_ALLOC(neqf, double)) ||
          !(rtol = NAG_ALLOC(neqf, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  tcrit = 10.0;
  state = Nag_NewStart;
  vectol = Nag_TRUE;
  one_step = Nag_FALSE;
  crit = Nag_TRUE;
```

```
  max_step = 0;
  sophist = Nag_TRUE;
  for (i = 0; i <= 1; ++i)
    {
      rtol[i] = 0.0001;
      atol[i] = 1e-06;
    }

  /* nag_ode_ivp_adams_setup (d02qwc).
   * Setup function for nag_ode_ivp_adams_roots (d02qfc)
   */
  nag_ode_ivp_adams_setup(&state, neqf, vectol, atol, rtol, one_step, crit,
                          tcrit, 0.0, max_step, neqg, &alter_g, sophist, &opt,
                          &fail);
  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ode_ivp_adams_setup (d02qwc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }

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

  do
    {
      /* nag_ode_ivp_adams_roots (d02qfc).
       * Ordinary differential equation solver using Adams method
       * (sophisticated use)
       */
      nag_ode_ivp_adams_roots(neqf, ftry02, &t, y, tout, gtry02,
                              &comm, &opt, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_ode_ivp_adams_roots (d02qfc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }


      if (opt.root)
        {
          printf("\nRoot at %14.5e\n", t);
          printf("for event equation %1ld", opt.index);
          printf(" with type %1ld", opt.type);
          printf(" and residual %14.5e\n", opt.resids[opt.index-1]);

          printf(" Y(1) = %14.5e   Y'(1) = %14.5e\n", y[0], opt.yp[0]);

          for (i = 1; i <= neqg; ++i)
            {
              if (i != opt.index && opt.events[i-1] != 0)
                {
                  printf("and also for event equation %1ld", i);
                  printf(" with type %1ld", opt.events[i-1]);
                  printf(" and residual %14.5e\n", opt.resids[i-1]);
                }
            }
        }
    } while (opt.tcurr < tout && opt.root);

  /* Free the memory which was allocated by
   * nag_ode_ivp_adams_setup (d02qwc) to the pointers inside opt.
   */
  /* nag_ode_ivp_adams_free (d02qyc).
   * Freeing function for use with nag_ode_ivp_adams_roots (d02qfc)
   */
```

```
  nag_ode_ivp_adams_free(&opt);
 END:
  NAG_FREE(y);
  NAG_FREE(atol);
  NAG_FREE(rtol);
  return exit_status;
}


static void NAG_CALL fcn(Integer neqf, double x, const double y[], double
                         yp[], Nag_User *comm)
{
  Integer *use_comm = (Integer *)comm->p;

  if (use_comm[0])
    {
      printf("(User-supplied callback fcn, first invocation.)\n");
      use_comm[0] = 0;
    }

  yp[0] = y[1];
  yp[1] = -y[0];
}                               /* fcn */


static double NAG_CALL gcn(Integer neqf, double x, const double y[], double
                           const yp[], Integer k, Nag_User *comm)
{
  Integer *use_comm = (Integer *)comm->p;

  if (use_comm[1])
    {
      printf("(User-supplied callback gcn, first invocation.)\n");
      use_comm[1] = 0;
    }

  if (k == 1) return yp[0];
  else return y[0];
}  /* gcn */
```

## 10.2 Program Data

None.

## 10.3 Program Results

```
nag_ode_ivp_adams_roots (d02qfc) Example Program Results
(User-supplied callback fcn, first invocation.)
(User-supplied callback gcn, first invocation.)

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 -5.90726e-16
 Y(1) = 1.00003e+00   Y'(1) = -5.90726e-16

Root at 3.14151e+00
for event equation 2 with type 1 and residual -1.28281e-16
 Y(1) = -1.28281e-16   Y'(1) = -1.00012e+00

Root at 4.71228e+00
for event equation 1 with type 1 and residual 3.59623e-16
 Y(1) = -1.00010e+00   Y'(1) = 3.59623e-16

Root at 6.28306e+00
for event equation 2 with type 1 and residual 2.47333e-15
 Y(1) = 2.47333e-15   Y'(1) = 9.99979e-01
```

```
Root at 7.85379e+00
for event equation 1 with type 1 and residual -3.20716e-15
 Y(1) = 9.99970e-01   Y'(1) = -3.20716e-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
```