NAG Toolbox

nag_ode_ivp_adams_roots (d02qf)

1 Purpose

nag_ode_ivp_adams_roots (d02qf) 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 Syntax

```
[t, y, root, rwork, iwork, ifail] = nag_ode_ivp_adams_roots(fcn, t, y, tout, g, neqg, rwork, iwork, 'neqf', neqf)
[t, y, root, rwork, iwork, ifail] = d02qf(fcn, t, y, tout, g, neqg, rwork, iwork, 'neqf', neqf)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 22: *lrwork* and *liwork* were removed from the interface

At Mark 22: *lrwork* and *liwork* were removed from the interface.

3 Description

Given the initial values $x, y_1, y_2, \ldots, y_{\mathbf{neqf}}$ nag_ode_ivp_adams_roots (d02qf) integrates a non-stiff system of first-order differential equations of the type

$$y_i' = f_i\left(x, y_1, y_2, \ldots, y_{\mathbf{neqf}}\right), \quad i = 1, 2, \ldots, \mathbf{neqf},$$

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 (d02qw) 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** supplied by you 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 select the mode of operation, the error control, the root-finding technique and various optional inputs by a prior call to the setup function nag_ode_ivp_adams_setup (d02qw).

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 Parameters

5.1 Compulsory Input Parameters

1: **fcn** – SUBROUTINE, supplied by the user.

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

```
[f] = fcn(neqf, x, y)
```

Input Parameters

1: **neqf** – INTEGER

The number of differential equations.

2: **x** – REAL (KIND=nag_wp)

The current value of the argument x.

3: **y**(**neqf**) – REAL (KIND=nag_wp) array

y_i, for $i = 1, 2, \ldots,$ **neqf**, the current value of the argument.

Output Parameters

1: **f**(**neqf**) – REAL (KIND=nag_wp) array

The value of f_i, for $i = 1, 2, \ldots,$ **neqf**.

2: **t** – REAL (KIND=nag_wp)

After a call to nag_ode_ivp_adams_setup (d02qw) with **statef** = 'S' (i.e., an initial entry), **t** must be set to the initial value of the independent variable x.

3: **y**(**neqf**) – REAL (KIND=nag_wp) array

The initial values of the solution $y_1, y_2, \ldots, y_{\mathbf{neqf}}$.

4: **tout** – REAL (KIND=nag_wp)

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.

5: **g** – REAL (KIND=nag_wp) FUNCTION, supplied by the user.

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 string `nag_ode_ivp_adams_roots_dummy_g (d02qfz)`. (nag_ode_ivp_adams_roots_dummy_g (d02qfz) is included in the NAG Toolbox.)

```
[result] = g(neqf, x, y, yp, k)
```

Input Parameters

1: **neqf** – INTEGER

The number of differential equations being solved.

2: **x** – REAL (KIND=nag_wp)

The current value of the independent variable.

3: **y**(**neqf**) – REAL (KIND=nag_wp) array

The current values of the dependent variables.

4: **yp**(**neqf**) – REAL (KIND=nag_wp) array

The current values of the derivatives of the dependent variables.

5: **k** – INTEGER

The component of g which must be evaluated.

Output Parameters

1: **result**

The value of the component of $g(x, y, y')$ at the specified point.

6: **neqg** – INTEGER

The number of event functions which you are defining for root-finding. If root-finding is not required the value for **neqg** must be ≤ 0. Otherwise it must be the same argument **neqg** used in the prior call to nag_ode_ivp_adams_setup (d02qw).

7: **rwork**($lrwork$) – REAL (KIND=nag_wp) array

This **must** be the same argument **rwork** as supplied to nag_ode_ivp_adams_setup (d02qw). It is used to pass information from nag_ode_ivp_adams_setup (d02qw) to nag_ode_ivp_adams_roots (d02qf), and from nag_ode_ivp_adams_roots (d02qf) to nag_ode_ivp_adams_diag (d02qx), nag_ode_ivp_adams_rootdiag (d02qy) and nag_ode_ivp_adams_interp (d02qz). Therefore the contents of this array **must not** be changed before the call to nag_ode_ivp_adams_roots (d02qf) or calling any of the functions nag_ode_ivp_adams_diag (d02qx), nag_ode_ivp_adams_rootdiag (d02qy) and nag_ode_ivp_adams_interp (d02qz).

8: **iwork**($liwork$) – INTEGER array

This **must** be the same argument **iwork** as supplied to nag_ode_ivp_adams_setup (d02qw). It is used to pass information from nag_ode_ivp_adams_setup (d02qw) to nag_ode_ivp_adams_roots (d02qf), and from nag_ode_ivp_adams_roots (d02qf) to nag_ode_ivp_adams_diag (d02qx), nag_ode_ivp_adams_rootdiag (d02qy) and nag_ode_ivp_adams_interp (d02qz). Therefore the contents of this array **must not** be changed before the call to nag_ode_ivp_adams_roots (d02qf) or calling any of the functions nag_ode_ivp_adams_diag (d02qx), nag_ode_ivp_adams_rootdiag (d02qy) and nag_ode_ivp_adams_interp (d02qz).

5.2 Optional Input Parameters

1: **neqf** – INTEGER

Default: the dimension of the array **y**.

The number of first-order ordinary differential equations to be solved by nag_ode_ivp_adams_roots (d02qf). It must contain the same value as the argument **neqf** used in a prior call to nag_ode_ivp_adams_setup (d02qw).

Constraint: **neqf** ≥ 1.

5.3 Output Parameters

1: **t** – REAL (KIND=nag_wp)

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.

2: **y**(**neqf**) – REAL (KIND=nag_wp) array

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.

3: **root** – LOGICAL

If root-finding was required (**neqg** > 0 on entry), then **root** specifies whether or not the output value of the argument **t** is a root of one of the event functions. If **root** $= false$, then no root was detected, whereas **root** $= true$ indicates a root and you should make a call to nag_ode_ivp_adams_rootdiag (d02qy) for further information.

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

4: **rwork**($lrwork$) – REAL (KIND=nag_wp) array

5: **iwork**($liwork$) – INTEGER array

6: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, the integrator detected an illegal input, or nag_ode_ivp_adams_setup (d02qw) has not been called before the call to the integrator.

This error may be caused by overwriting elements of **rwork** and **iwork**.

ifail $= 2$ (*warning*)

The maximum number of steps has been attempted (at a cost of about 2 calls to **fcn** per step). (See argument **maxstp** in nag_ode_ivp_adams_setup (d02qw).) If integration is to be continued then you need only reset **ifail** and call the function again and a further **maxstp** steps will be attempted.

ifail $= 3$

The step size needed to satisfy the error requirements is too small for the ***machine precision*** being used. (See argument **tolfac** in nag_ode_ivp_adams_diag (d02qx).)

ifail $= 4$ (*warning*)

Some error weight w_i became zero during the integration (see arguments **vectol**, **rtol** and **atol** in nag_ode_ivp_adams_setup (d02qw).) Pure relative error control (**atol** $= 0.0$) was requested on a variable (the ith) which has now become zero. (See argument **badcmp** in nag_ode_ivp_adams_diag (d02qx).) The integration was successful as far as **t**.

ifail $= 5$ (*warning*)

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 **ifail** and calling the function again.

ifail $= 6$ (*warning*)

A change in sign of an event function has been detected but the root-finding process appears to have converged to a singular point **t** rather than a root. Integration may be continued by resetting **ifail** and calling the function again.

ifail $= 7$

The code has detected two successive error exits at the current value of **t** and cannot proceed. Check all input variables.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

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 (d02qw). 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 (d02qf) 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 (d02qw).

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 Further Comments

If nag_ode_ivp_adams_roots (d02qf) fails with **ifail** $= 3$ 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 with larger values for **rtol** and/or **atol** (see nag_ode_ivp_adams_setup (d02qw)). 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. nag_ode_ivp_adams_roots (d02qf) 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 **ifail** $= 3$, but usually an error exit with **ifail** $= 2$ or 5. The Adams' methods are not efficient in such cases and you should consider using a function from the Sub-chapter D02M–N. A high proportion of failed steps (see argument **nfail**) may indicate stiffness but there may be other reasons for this phenomenon.

nag_ode_ivp_adams_roots (d02qf) can be used for producing results at short intervals (for example, for graph plotting); you should set **crit** $= true$ and **tcrit** to the last output point required in a prior call to nag_ode_ivp_adams_setup (d02qw) and then set **tout** appropriately for each output point in turn in the call to nag_ode_ivp_adams_roots (d02qf).

9 Example

This example solves the equation

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

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** $= true$) and computation of the solution at **tout** $= 10.0$ with **tcrit** $= 10.0$ (**crit** $= true$). Also, we use nag_ode_ivp_adams_roots (d02qf) to locate the positions where $y_1 = 0.0$ or where the first component has a turning-point, that is $y_1' = 0.0$.

9.1 Program Text

```
function d02qf_example

fprintf('d02qf example results\n\n');

% Initialize parameters for Adams method setup
neqf   = nag_int(2);
neqg   = nag_int(2);
vectol = true;
atol   = [1e-06;  1e-06];
rtol   = [0.0001; 0.0001];
onestp = false;
crit   = true;
tcrit  = 10;
hmax   = 0;
maxstp = nag_int(0);
alterg = false;
sophst = true;

lrwork = 23*(1+neqf) + 14*neqg;
rwork  = zeros(lrwork,1);
liwork = 21 + 4*neqg;
iwork  = zeros(liwork, 1, nag_int_name);
```

```
% Adams method setup
[statef, alterg, rwork, iwork, ifail] = ...
  d02qw(...
        'Start', neqf, vectol, atol, rtol, onestp, crit, tcrit, hmax, ...
        maxstp, neqg, alterg, sophst, rwork, iwork);

% Adams method integration
t      = 0;
y      = [0; 1];
tout   = 10;
root   = true;

while (t<tout && root)
  [t, y, root, rwork, iwork, ifail] = ...
    d02qf(...
          @fcn, t, y, tout, @g, neqg, rwork, iwork);

  if (root)
    [yp, tcurr, hlast, hnext, odlast, odnext, nsucc, nfail, tolfac, ...
     badcmp, ifail] = d02qx(...
                            neqf, rwork, iwork);

    % Display solution and root statistics.
    [index, itype, events, resids, ifail] = ...
      d02qy(...
            neqg, rwork, iwork);

    fprintf('\nRoot at %13.5e\n',t);
    fprintf('for event equation %2d with type %3d and residual %13.5e\n',...
           index, itype, resids(index))
    fprintf(' y(1) = %13.5e  y''(1) = %13.5e\n',y(1), yp(1));
    ind = find(events);
    for i = ind
      if i~=index
        fprintf('and also for event equation %2d with type %3d',i, events(i));
        fprintf(' and residual %13.5e\n',resids(i));
      end
    end
  end
end

function f = fcn(neqf, x, y)
  f=zeros(neqf,1);
  f(1)=y(2);
  f(2)=-y(1);

function result = g(neqf, x, y, yp, k)
  if (k == 1)
    result = yp(1);
  else
    result = y(1);
  end
```

9.2 Program Results

```
     d02qf example 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  -5.20417e-17
 y(1) =   1.00003e+00  y'(1) =  -5.20417e-17

Root at   3.14151e+00
for event equation  2 with type   1 and residual  -1.27676e-15
 y(1) =  -1.27676e-15  y'(1) =  -1.00012e+00

Root at   4.71228e+00
```

```
for event equation  1 with type   1 and residual   1.67921e-15
 y(1) =  -1.00010e+00  y'(1) =   1.67921e-15

Root at   6.28306e+00
for event equation  2 with type   1 and residual   2.65066e-15
 y(1) =   2.65066e-15  y'(1) =   9.99979e-01

Root at   7.85379e+00
for event equation  1 with type   4 and residual  -7.63278e-17
 y(1) =   9.99970e-01  y'(1) =  -7.63278e-17

Root at   9.42469e+00
for event equation  2 with type   4 and residual  -6.86950e-16
 y(1) =  -6.86950e-16  y'(1) =  -9.99854e-01
```