NAG Toolbox for MATLAB

d02qf

1 Purpose

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] = d02qf(fcn, t, y, tout, g, neqg,
rwork, iwork, 'neqf', neqf, 'lrwork', lrwork, 'liwork', liwork)
```

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 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 user-supplied (sub)program **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

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

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 user-supplied real function **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 of the setup function 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 – string containing name of m-file**

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

Its specification is:

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

Input Parameters

1: **neqf – int32 scalar**

the number of differential equations.

2: **x – double scalar**

The current value of the argument x.

3: **y(neqf) – double array**

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

Output Parameters

1: **f(neqf) – double array**

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

2: **t – double scalar**

After a call to 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) – double array**

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

4: **tout – double scalar**

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 – string containing name of m-file**

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 `'d02qfz'`. **d02qfz** is included in the NAG Fortran Library.

Its specification is:

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

Input Parameters

1: **neqf – int32 scalar**

The number of differential equations being solved.

2: **x – double scalar**

The current value of the independent variable.

3: **y(neqf) – double array**

The current values of the dependent variables.

4: **yp(neqf) – double array**

The current values of the derivatives of the dependent variables.

5: **k – int32 scalar**

The component of g which must be evaluated.

Output Parameters

1: **result – double scalar**

The result of the function.

6: **neqg – int32 scalar**

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 parameter **neqg** used in the prior call to d02qw.

7: **rwork(lrwork) – double array**

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

8: **iwork(liwork) – int32 array**

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

5.2 Optional Input Parameters

1: **neqf – int32 scalar**

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

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

Constraint: **neqf** ≥ 1.

2: **lrwork – int32 scalar**

Default: The dimension of the array **rwork**.

This must be the same parameter **lrwork** as supplied to d02qw.

3: **liwork – int32 scalar**

Default: The dimension of the array **iwork**.

This must be the same parameter **liwork** as supplied to d02qw.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **t – double scalar**

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) – double 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 scalar**

If root-finding was required (**neqg** > 0 on entry), 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 and you should make a call to d02qy for further information.

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

4: **rwork(lrwork) – double array**

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

5: **iwork(liwork) – int32 array**

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

6: **ifail – int32 scalar**

0 unless the function detects an error (see Section 6).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, the integrator detected an illegal input, or d02qw has not been called prior to the call to the integrator.

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

ifail $= 2$

The maximum number of steps has been attempted (at a cost of about 2 calls to user-supplied (sub)program **fcn** per step). (See parameter **maxstp** in 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 parameter **tolfac** in d02qx.)

ifail $= 4$

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

ifail $= 5$

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 $= 6$

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.

7 Accuracy

The accuracy of integration is determined by the parameters **vectol**, **rtol** and **atol** in a prior call to d02qw. 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. You are strongly recommended to call 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 parameters **vectol**, **atol** and **rtol** in the function document for 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 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 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. 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 **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 parameter **NFAIL**) may indicate stiffness but there may be other reasons for this phenomenon.

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 d02qw and then set **tout** appropriately for each output point in turn in the call to d02qf.

9 Example

```
d02qf_fcn.m

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

```
d02qf_g.m

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

```
t = 0;
y = [0; 1];
tout = 10;
neqg = int32(2);
rwork = zeros(97,1);
iwork = zeros(29, 1, 'int32');
[statefOut, altergOut, rwork, iwork, ifail] = ...
     d02qw('S', int32(2), true, [1e-06;1e-06], [0.0001; 0.0001], false,
true, ...
    10, 0, int32(0), int32(2), false, true, rwork, iwork);
[tOut, yOut, root, rworkOut, iworkOut, ifail] = ...
    d02qf('d02qf_fcn', t, y, tout, 'd02qf_g', neqg, rwork, iwork)
```

```
tOut =
     0
yOut =
     0
     1
root =
     1
rworkOut =
     array elided
iworkOut =
            2
            2
            0
            0
            1
           97
           29
            1
            1
            1
            2
            1
            0
            0
            1
            0
         1000
            0
           0
           0
           0
           2
           0
           0
           0
           0
           0
           0
           0
ifail =
           0
```