

NAG Toolbox for MATLAB

d02qg

1 Purpose

d02qg is a reverse communication 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, irevcm, trvcm, yrvcm, yprvcm, kgrvcm, rwork, iwork, ifail]
= d02qg(t, y, tout, neqg, irevcm, grvcm, kgrvcm, rwork, iwork, 'neqf',
neqf, 'lrwork', lrwork, 'liwork', liwork)
```

3 Description

Given the initial values $x, y_1, y_2, \dots, y_{\text{neqf}}$ d02qg integrates a non-stiff system of first-order differential equations of the type, $y'_i = f_i(x, y_1, y_2, \dots, y_{\text{neqf}})$, for $i = 1, 2, \dots, \text{neqf}$, from $x = \mathbf{t}$ to $x = \mathbf{tout}$ using a variable-order variable-step Adams method. You define the system by reverse communication, evaluating f_i in terms of x and $y_1, y_2, \dots, y_{\text{neqf}}$, and $y_1, y_2, \dots, y_{\text{neqf}}$ are supplied at $x = \mathbf{t}$ by d02qg. The function is capable of finding roots (values of x) of prescribed event functions of the form

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

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. You also define each g_j by reverse communication. 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.

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

Note: this function uses **reverse communication**. Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the **parameter IREVCM**. Between intermediate exits and re-entries, **all parameters other than all parameters other than grvcm and rwork must remain unchanged**.

5.1 Compulsory Input Parameters

1: **t** – double scalar

On initial entry: that is after a call to d02qw with **statef** = 'S', **t** must be set to the initial value of the independent variable x .

2: **y(neqf)** – double array

On initial entry: the initial values of the solution $y_1, y_2, \dots, y_{\text{neqf}}$.

3: **tout** – double scalar

On initial 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.

4: **neqg** – int32 scalar

On initial entry: 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 value as the parameter **neqg** used in the prior call to d02qw.

5: **irevcm** – int32 scalar

On initial entry: must have the value 0.

6: **grvcm** – double scalar

On initial entry: need not be set.

On intermediate re-entry: with **irevcm** = 9, 10, 11 or 12, **grvcm** must contain the value of $g_k(x, y, y')$, where k is given by **kgrvcm**.

7: **kgrvcm** – int32 scalar

On intermediate re-entry: with **irevcm** = 9, 10, 11 or 12, **kgrvcm** must remain unchanged from a previous call to d02qg.

8: **rwork(lrwork)** – double array

This **must** be the same parameter **rwork** as supplied to d02qw. It is used to pass information from d02qw to d02qg, and from d02qg to d02qx, d02qy and d02qz. Therefore the contents of this array **must not** be changed before the call to d02qg or calling any of the functions d02qx, d02qy and d02qz.

9: **iwork(liwork)** – int32 array

This **must** be the same parameter **iwork** as supplied to d02qw. It is used to pass information from d02qw to d02qg, and from d02qg to d02qx, d02qy and d02qz. Therefore the contents of this array **must not** be changed before the call to d02qg 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**.

On initial entry: the number of first-order ordinary differential equations to be solved by d02qg. It must contain the same value as the parameter **neqf** used in the prior call to d02qw.

Constraint: **neqf** ≥ 1 .

2: **lrwork – int32 scalar**

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

On initial entry: This must be the same parameter **lrwork** as supplied to d02qw.

3: **liwork – int32 scalar**

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

On initial entry: This must be the same parameter **liwork** as supplied to d02qw.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters1: **t – double scalar**

On final 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.

2: **y(neqf) – double array**

On final 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.

3: **root – logical scalar**

On final exit: 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 **root** = **false**.

4: **irevcn – int32 scalar**

On intermediate exit: specifies what action you must take before re-entering d02qg with **irevcn** unchanged.

irevcn = 1, 2, 3, 4, 5, 6 or 7

Indicates that you must supply $y' = f(x, y)$, where x is given by **trvcn** and y_i is returned in **y(i)**, for $i = 1, 2, \dots, \text{neqf}$ when **yrvcn** = 0 and **rwork(yrvcn + i - 1)**, for $i = 1, 2, \dots, \text{neqf}$ when **yrvcn** ≠ 0. y'_i should be placed in location **rwork(yprvcn + i - 1)**, for $i = 1, 2, \dots, \text{neqf}$.

irevcn = 8

Indicates that the current step was not successful due to error test failure. The only information supplied to you on this return is the current value of the independent variable **t**, as given by **trvcn**. No values must be changed before re-entering d02qg. This facility enables you to determine the number of unsuccessful steps.

irevcn = 9, 10, 11 or 12

Indicates that you must supply $g_k(x, y, y')$, where k is given by **kgrvcn**, x is given by **trvcn**, y_i is given by **y(i)** and y'_i is given by **rwork(yprvcn - 1 + i)**. The result g_k should be placed in the variable **grvcn**.

On final exit: has the value 0, which indicates that an output point or root has been reached or an error has occurred (see **ifail**).

5: **trvcm – double scalar**

On intermediate exit: the current value of the independent variable.

6: **yrvcm – int32 scalar**

On intermediate exit: with **irevcm** = 1, 2, 3, 4, 5, 6, 7, 9, 10, 11 or 12, **yrvcm** specifies the locations of the dependent variables y for use in evaluating the differential system or the event functions.

yrvcm = 0

y_i is given by $y(i)$, for $i = 1, 2, \dots, \mathbf{neqf}$.

yrvcm \neq 0

y_i is given by **rwork**(**yrvcm** + $i - 1$), for $i = 1, 2, \dots, \mathbf{neqf}$.

7: **yprvcm – int32 scalar**

On intermediate exit: with **irevcm** = 1, 2, 3, 4, 5, 6 or 7, **yprvcm** specifies the positions in **rwork** at which you should place the derivatives y' . y'_i should be placed in location **rwork**(**yprvcm** + $i - 1$), for $i = 1, 2, \dots, \mathbf{neqf}$.

With **irevcm** = 9, 10, 11 or 12, **yprvcm** specifies the locations of the derivatives y' for use in evaluating the event functions. y'_i is given by **rwork**(**yprvcm** + $i - 1$), for $i = 1, 2, \dots, \mathbf{neqf}$.

8: **kgrvcm – int32 scalar**

On intermediate exit: with **irevcm** = 9, 10, 11 or 12, **kgrvcm** specifies which event function $g_k(x, y, y')$ you must evaluate.

9: **rwork(lrwork) – double array**

This **must** be the same parameter **rwork** as supplied to d02qw. It is used to pass information from d02qw to d02qg, and from d02qg to d02qx, d02qy and d02qz. Therefore the contents of this array **must not** be changed before the call to d02qg or calling any of the functions d02qx, d02qy and d02qz.

10: **iwork(liwork) – int32 array**

This **must** be the same parameter **iwork** as supplied to d02qw. It is used to pass information from d02qw to d02qg, and from d02qg to d02qx, d02qy and d02qz. Therefore the contents of this array **must not** be changed before the call to d02qg or calling any of the functions d02qx, d02qy and d02qz.

11: **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 derivative evaluations 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 i th) 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 property 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 d02qg 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**, **rtol** and **atol** 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 d02qg 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**. 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 sub-chapter D02M/N. A high proportion of failed steps (see parameter **nfail** in d02qx) may indicate stiffness but there may be other reasons for this phenomenon.

d02qg 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 d02qg.

9 Example

```

t = 0;
y = [0.5;
     0.5;
     0.2*pi];
neqg = int32(1);
grvcm = 0;
kgrvcm = int32(0);
rwork = zeros(106, 1);
iwork = zeros(25, 1, 'int32');

[statef, alterg, rwork, iwork, ifail] = ...
    d02qw('S', int32(3), false, [1e-4], [1e-7], false, true, ...
    10, 2, int32(500), int32(1), false, true, rwork, iwork);

j = 1;
tinc = 2;
ifail = int32(0);

fprintf('\n   T           Y(1)      Y(2)      Y(3)\n');
fprintf('%7.4f %7.4f %7.4f %7.4f\n', t, y(1), y(2), y(3));
soln = [t, transpose(y)];

while (j < 5 && ifail == int32(0))
    tout = double(j)*tinc;
    irevcm = int32(0);

    [t, y, root, irevcm, trvcm, yrvcm, yprvcm, kgrvcm, rwork, iwork, ifail]
    = ...
        d02qg(t, y, tout, neqg, irevcm, grvcm, kgrvcm, rwork, iwork);

    while (irevcm ~= 0)
        if (irevcm < 8)
            if (yrvcm == 0)
                rwork(yprvcm) = tan(y(3));
                rwork(yprvcm+1) = -0.032*tan(y(3))/y(2) - 0.02*y(2)/cos(y(3));
                rwork(yprvcm+2) = -0.032/y(2)^2;
            else
                rwork(yprvcm) = tan(rwork(yrvcm+2));
                rwork(yprvcm+1) = ...
                    -0.032*tan(rwork(yrvcm+2))/rwork(yrvcm+1) - ...
                    0.02*rwork(yrvcm+1)/cos(rwork(yrvcm+2));
                rwork(yprvcm+2) = -0.032/rwork(yrvcm+1)^2;
            end
        elseif (irevcm > 8)
            grvcm = y(1);
        end
        [t, y, root, irevcm, trvcm, yrvcm, yprvcm, kgrvcm, rwork, iwork,

```

```
ifail] = ...  
    d02qg(t, y, tout, neqg, irevcm, grvcm, kgrvcm, rwork, iwork);  
end  
  
if (ifail == 0)  
    fprintf('%7.4f %7.4f %7.4f %7.4f\n', t, y(1), y(2), y(3));  
    soln = [soln; t, transpose(y)];  
end  
j = j + 1;  
end
```

T	Y(1)	Y(2)	Y(3)
0.0000	0.5000	0.5000	0.6283
2.0000	1.5493	0.4055	0.3066
4.0000	1.7424	0.3743	-0.1289
6.0000	1.0058	0.4173	-0.5507
7.2879	-0.0000	0.4749	-0.7601