NAG Toolbox for MATLAB

d02pc

1 Purpose

d02pc solves the initial value problem for a first-order system of ordinary differential equations using Runge–Kutta methods.

2 Syntax

```
[tgot, ygot, ypgot, ymax, work, ifail] = d02pc(f, neq, twant, ygot, ymax, work)
```

3 Description

d02pc and its associated functions (d02pv, d02py and d02pz) solve the initial value problem for a first-order system of ordinary differential equations. The functions, based on Runge–Kutta methods and derived from RKSUITE (see Brankin *et al.* 1991), integrate

$$y' = f(t,y) \qquad \text{given} \qquad y(t_0) = y_0$$

where y is the vector of n solution components and t is the independent variable.

d02pc is designed for the usual task, namely to compute an approximate solution at a sequence of points. You must first call d02pv to specify the problem and how it is to be solved. Thereafter you call d02pc repeatedly with successive values of **twant**, the points at which you require the solution, in the range from **tstart** to **tend** (as specified in d02pv). In this manner d02pc returns the point at which it has computed a solution **tgot** (usually **twant**), the solution there (**ygot**) and its derivative (**ypgot**). If d02pc encounters some difficulty in taking a step toward **twant**, then it returns the point of difficulty (**tgot**) and the solution and derivative computed there (**ygot** and **ypgot**, respectively).

In the call to d02pv you can specify either the first step size for d02pc to attempt or that it compute automatically an appropriate value. Thereafter d02pc estimates an appropriate step size for its next step. This value and other details of the integration can be obtained after any call to d02pc by a call to d02py. The local error is controlled at every step as specified in d02pv. If you wish to assess the true error, you must set **errass** = **true** in the call to d02pv. This assessment can be obtained after any call to d02pc by a call to d02pz.

For more complicated tasks, you are referred to functions d02pd, d02px and d02pw, all of which are used by d02pc.

4 References

Brankin R W, Gladwell I and Shampine L F 1991 RKSUITE: A suite of Runge–Kutta codes for the initial value problems for ODEs *SoftReport 91-S1* Southern Methodist University

5 Parameters

5.1 Compulsory Input Parameters

1: **f – string containing name of m-file**

f must evaluate the functions f_i (that is the first derivatives y_i') for given values of the arguments t, y_i.

Its specification is:

```
[yp] = f(t, y)
```

Input Parameters

1: **t – double scalar**

t, the current value of the independent variable.

2: **y**(n) **– double array**

The current values of the dependent variables, y_i, for $i = 1, 2, \ldots, n$.

Output Parameters

1: **yp**(n) **– double array**

The values of f_i, for $i = 1, 2, \ldots, n$.

2: **neq – int32 scalar**

n, the number of ordinary differential equations in the system to be solved by the integration function.

Constraint: **neq** ≥ 1.

3: **twant – double scalar**

t, the next value of the independent variable where a solution is desired.

Constraint: **twant** must be closer to **tend** than the previous value of **tgot** (or **tstart** on the first call to d02pc); see d02pv for a description of **tstart** and **tend**. **twant** must not lie beyond **tend** in the direction of integration.

4: **ygot**$(*)$ **– double array**

Note: the dimension of the array **ygot** must be at least n.

On the first call to d02pc, **ygot** need not be set. On all subsequent calls **ygot** must remain unchanged.

5: **ymax**$(*)$ **– double array**

Note: the dimension of the array **ymax** must be at least n.

On the first call to d02pc, **ymax** need not be set. On all subsequent calls **ymax** must remain unchanged.

6: **work**$(*)$ **– double array**

Note: the dimension of the array **work** must be at least **lenwrk** (see d02pv).

This **must** be the same array as supplied to d02pv. It **must** remain unchanged between calls.

5.2 Optional Input Parameters

None.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **tgot – double scalar**

The value of the independent variable t at which a solution has been computed. On successful exit with **ifail** $= 0$, **tgot** will equal **twant**. On exit with **ifail** > 1, a solution has still been computed at the value of **tgot** but in general **tgot** will not equal **twant**.

2: **ygot**$(*)$ **– double array**

Note: the dimension of the array **ygot** must be at least n.

An approximation to the true solution at the value of **tgot**. At each step of the integration to **tgot**, the local error has been controlled as specified in d02pv. The local error has still been controlled even when **tgot** $\neq$ **twant**, that is after a return with **ifail** > 1.

3: **ypgot**$(*)$ **– double array**

Note: the dimension of the array **ypgot** must be at least n.

An approximation to the first derivative of the true solution at **tgot**.

4: **ymax**$(*)$ **– double array**

Note: the dimension of the array **ymax** must be at least n.

ymax(i) contains the largest value of $|y_i|$ computed at any step in the integration so far.

5: **work**$(*)$ **– double array**

Note: the dimension of the array **work** must be at least **lenwrk** (see d02pv).

Information about the integration for use on subsequent calls to d02pc or other associated functions.

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, an invalid input value for **twant** was detected or an invalid call to d02pc was made, for example without a previous call to the setup function d02pv. You cannot continue integrating the problem.

ifail $= 2$

This return is possible only when **method** $= 3$ has been selected in the preceding call of d02pv. d02pc is being used inefficiently because the step size has been reduced drastically many times to get answers at many values of **twant**. If you really need the solution at this many points, you should change to **method** $= 2$ because it is (much) more efficient in this situation. To change **method**, restart the integration from **tgot**, **ygot** by a call to d02pv. If you wish to continue with **method** $= 3$, just call d02pc again without altering any of the arguments other than **ifail**. The monitor of this kind of inefficiency will be reset automatically so that the integration can proceed.

ifail $= 3$

A considerable amount of work has been expended in the (primary) integration. This is measured by counting the number of calls to the user-supplied (sub)program **f**. At least 5000 calls have been made since the last time this counter was reset. Calls to **f** in a secondary integration for global error assessment (when **errass** $=$ **true** in the call to d02pv) are not counted in this total. The integration was interrupted, so **tgot** is not equal to **twant**. If you wish to continue on towards **twant**, just call

d02pc again without altering any of the arguments other than **ifail**. The counter measuring work will be reset to zero automatically.

ifail $= 4$

It appears that this problem is stiff. The methods implemented in d02pc can solve such problems, but they are inefficient. You should change to another code based on methods appropriate for stiff problems. The integration was interrupted so **tgot** is not equal to **twant**. If you want to continue on towards **twant**, just call d02pc again without altering any of the arguments other than **ifail**. The stiffness monitor will be reset automatically.

ifail $= 5$

It does not appear possible to achieve the accuracy specified by **tol** and **thres** in the call to d02pv with the precision available on the computer being used and with this value of **method**. You cannot continue integrating this problem. A larger value for **method**, if possible, will permit greater accuracy with this precision. To increase **method** and/or continue with larger values of **tol** and/or **thres**, restart the integration from **tgot**, **ygot** by a call to d02pv.

ifail $= 6$

(This error exit can only occur if **errass** $=$ **true** in the call to d02pv.) The global error assessment may not be reliable beyond the current integration point **tgot**. This may occur because either too little or too much accuracy has been requested or because $f(t,y)$ is not smooth enough for values of t just past **tgot** and current values of the solution y. The integration cannot be continued. This return does not mean that you cannot integrate past **tgot**, rather that you cannot do it with **errass** $=$ **true**. However, it may also indicate problems with the primary integration.

7 Accuracy

The accuracy of integration is determined by the parameters **tol** and **thres** in a prior call to d02pv (see the function document for d02pv for further details and advice). 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 system.

8 Further Comments

If d02pc returns with **ifail** $= 5$ and the accuracy specified by **tol** and **thres** is really required then you should consider whether there is a more fundamental difficulty. For example, the solution may contain a singularity. In such a region the solution components will usually be large in magnitude. Successive output values of **ygot** and **ymax** should be monitored (or d02pd should be used since this takes one integration step at a time) 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.

Performance statistics are available after any return from d02pc by a call to d02py. If **errass** $=$ **true** in the call to d02pv, global error assessment is available after any return from d02pc (except when **ifail** $= 1$) by a call to d02pz.

After a failure with **ifail** $= 5$ or 6 the diagnostic functions d02py and d02pz may be called only once.

If d02pc returns with **ifail** $= 4$ then it is advisable to change to another code more suited to the solution of stiff problems. d02pc will not return with **ifail** $= 4$ if the problem is actually stiff but it is estimated that integration can be completed using less function evaluations than already computed.

9 Example

d02pc_f.m

```
function [yp] = f(t, y)
```

```
   yp = zeros(2, 1);
   yp(1) = y(2);
   yp(2) = -y(1);
```

```
tstart = 0;
ystart = [0; 1];
tend = 6.283185307179586;
tol = 0.001;
thres = [1e-08; 1e-08];
method = int32(1);
task = 'Usual Task';
errass = false;
lenwrk = int32(64);
neq = int32(2);
twant = 0.7853981633974483;
ygot = [0; 0];
ymax = [0; 0];
[work, ifail] = ...
       d02pv(tstart, ystart, tend, tol, thres, method, task, errass,
lenwrk);
[tgot, ygotOut, ypgot, ymaxOut, workOut, ifail] = ...
    d02pc('d02pc_f', neq, twant, ygot, ymax, work)
```

```
tgot =
    0.7854
ygotOut =
    0.7069
    0.7068
ypgot =
    0.7058
   -0.7084
ymaxOut =
    0.7069
    1.0000
workOut =
     array elided
ifail =
           0
```