NAG Toolbox for MATLAB

d02la

1 Purpose

d02la is a function for integrating a non-stiff system of second-order ordinary differential equations using Runge–Kutta–Nystrom techniques.

2 Syntax

```
[t, y, yp, ydp, rwork, ifail] = d02la(fcn, t, tend, y, yp, ydp, rwork,
'neq', neq, 'lrwork', lrwork)
```

3 Description

Given the initial values $x, y_1, y_2, \ldots, y_{\mathbf{neq}}, y'_1, y'_2, \ldots, y'_{\mathbf{neq}}$ d02la integrates a non-stiff system of second-order differential equations of the type

$$y''_i = f_i\left(x, y_1, y_2, \ldots, y_{\mathbf{neq}}\right), \qquad i = 1, 2, \ldots, \mathbf{neq},$$

from $x = \mathbf{t}$ to $x = \mathbf{tend}$ using a Runge–Kutta–Nystrom formula pair. The system is defined by user-supplied (sub)program **fcn**, which evaluates f_i in terms of x and $y_1, y_2, \ldots, y_{\mathbf{neq}}$, where $y_1, y_2, \ldots, y_{\mathbf{neq}}$ are supplied at x.

There are two Runge–Kutta–Nystrom formula pairs implemented in this function. The lower order method is intended if you have moderate accuracy requirements and may be used in conjunction with the interpolation function d02lz to produce solutions and derivatives at user-specified points. The higher order method is intended if you have high accuracy requirements.

In one-step mode the function returns approximations to the solution, derivative and f_i at each integration point. In interval mode these values are returned at the end of the integration range. You select the order of the method, the mode of operation, the error control and various optional inputs by a prior call of d02lx.

For a description of the Runge–Kutta–Nystrom formula pairs see Dormand *et al.* 1986a and Dormand *et al.* 1986b and for a description of their practical implementation see Brankin *et al.* 1989.

4 References

Brankin R W, Dormand J R, Gladwell I, Prince P J and Seward W L 1989 Algorithm 670: A Runge–Kutta–Nystrom Code *ACM Trans. Math. Software* **15** 31–40

Dormand J R, El–Mikkawy M E A and Prince P J 1986a Families of Runge–Kutta–Nystrom formulae *Mathematical Report TPMR 86-1* Teesside Polytechnic

Dormand J R, El–Mikkawy M E A and Prince P J 1986b High order embedded Runge–Kutta–Nystrom formulae *Mathematical Report TPMR 86-2* Teesside Polytechnic

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 second derivatives y''_i) for given values of its arguments x, $y_1, y_2, \ldots, y_{\mathbf{neq}}$.

Its specification is:

```
[f] = fcn(neq, t, y)
```

Input Parameters

1: **neq – int32 scalar**

the number of differential equations.

2: **t – double scalar**

The value of the argument x.

3: **y(neq) – double array**

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

Output Parameters

1: **f(neq) – double array**

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

2: **t – double scalar**

The initial value of the independent variable x.

Constraint: **t** $\neq$ **tend**.

3: **tend – double scalar**

The end point of the range of integration. If **tend** $<$ **t** on initial entry, integration will proceed in the negative direction. **tend** may be reset, in the direction of integration, before any continuation call.

4: **y(neq) – double array**

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

5: **yp(neq) – double array**

The initial values of the derivatives $y'_1, y'_2, \ldots, y'_{\mathbf{neq}}$.

6: **ydp(neq) – double array**

Must be unchanged from a previous call to d02la.

7: **rwork(lrwork) – double array**

This **must** be the same parameter **rwork** as supplied to d02lx. It is used to pass information from d02lx to d02la, and from d02la to both d02ly and d02lz. Therefore the contents of this array **must not** be changed before the call to d02la or calling either of the functions d02ly and d02lz.

5.2 Optional Input Parameters

1: **neq – int32 scalar**

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

the number of second-order ordinary differential equations to be solved by d02la. It must contain the same value as the parameter **neq** used in a prior call of d02lx.

Constraint: **neq** ≥ 1.

2: **lrwork – int32 scalar**

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

This must be the same parameter **lrwork** as supplied to d02lx.

5.3 Input Parameters Omitted from the MATLAB Interface

None.

5.4 Output Parameters

1: **t – double scalar**

The value of the independent variable, which is usually **tend**, unless an error has occurred or the code is operating in one-step mode. If the integration is to be continued, possibly with a new value for **tend**, **t** must not be changed.

2: **y**(**neq**) **– 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 **tend**, these values must not be changed.

3: **yp**(**neq**) **– double array**

The computed values of the derivatives at the exit value of **t**. If the integration is to be continued, possibly with a new value for **tend**, these values must not be changed.

4: **ydp**(**neq**) **– double array**

The computed values of the second derivative at the exit value of **t**, unless illegal input is detected, in which case the elements of **ydp** may not have been initialized. If the integration is to be continued, possibly with a new value for **tend**, these values must not be changed.

5: **rwork**(**lrwork**) **– double array**

This **must** be the same parameter **rwork** as supplied to d02lx. It is used to pass information from d02lx to d02la, and from d02la to both d02ly and d02lz. Therefore the contents of this array **must not** be changed before the call to d02la or calling either of the functions d02ly and d02lz.

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 $= 1$

Illegal input detected, i.e., one of the following conditions:

on any call, **t** $=$ **tend**, or the value of **neq** or **lrwork** has been altered;

on a continuation call, the direction of integration has been changed;

d02lx had not been called previously, or the previous call of d02lx resulted in an error exit.

This error exit can be caused if elements of **rwork** have been overwritten.

ifail $= 2$

The maximum number of steps has been attempted. (See parameter **maxstp** in d02lx.) 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 $= 3$

In order to satisfy the error requirements, the step size needed is too small for the ***machine precision*** being used.

ifail $= 4$

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

ifail $= 5$

The code has detected inefficient use of the integration method. The step size has been reduced by a significant amount too often in order to hit the output points specified by **tend**. (Of the last 100 or more successful steps more than 10% are steps with sizes that have had to be reduced by a factor of greater than a half.)

7 Accuracy

The accuracy of integration is determined by the parameters **tol**, **thres** and **thresp** in a prior call of d02lx. Note that only the local error at each step is controlled by these parameters. The error estimates obtained are not strict bounds, but they are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the system. The code is designed so that a reduction in **tol** should lead to an approximately proportional reduction in the error. You are strongly recommended to call d02la with more than one value for **tol** and 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. For a description of the error test see the specifications of the parameters **tol**, **thres** and **thresp** in function document d02lx.

8 Further Comments

If d02la fails with **ifail** $= 3$ then the value of **tol** may be so small that a solution cannot be obtained, in which case the function should be called again with a larger value for **tol**. 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. d02la 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) if the solution contains fast oscillatory 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** $= 3$. The Runge–Kutta–Nystrom methods are not efficient in such cases and you should consider reposing your problem as a system of first-order ordinary differential equations and then using a function from sub-chapter D02M/N with the Blend formulae (see d02nw).

d02la can be used for producing results at short intervals (for example, for tabulation), in two ways. By far the less efficient is to call d02la successively over short intervals, $t + (i-1) \times h$ to $t + i \times h$, although this is the only way if the higher order method has been selected and precisely **not** what it is intended for. A more efficient way, **only** for use when the lower order method has been selected, is to use d02la in one-step mode. The output values of parameters **y**, **yp**, **ydp**, **t** and **rwork** are set correctly for a call of d02lz to compute the solution and derivative at the required points.

9 Example

d02la_fcn.m

```
function f = fcn(neq, t, y)
  r = sqrt(y(1)^2+y(2)^2)^3;
```

```
   f = zeros(2,1);
   f(1) = -y(1)/r;
   f(2) = -y(2)/r;
```

```
t = 0;
tend = 20;
y = [0.5;
     0];
yp = [0;
     1.732050807568877];
ydp = zeros(2, 1);
rwork = zeros(56,1);
[startOut, rwork, ifail] = ...
     d02lx(0, 1e-4, zeros(2,1), zeros(2,1), int32(0), true, true, false,
rwork);
[tOut, yOut, ypOut, ydpOut, rworkOut, ifail] = ...
    d02la('d02la_fcn', t, tend, y, yp, ydp, rwork)
```

```
tOut =
    0.0624
yOut =
    0.4923
    0.1075
ypOut =
   -0.2464
    1.7054
ydpOut =
   -3.8480
   -0.8406
rworkOut =
     array elided
ifail =
           0
```