NAG Toolbox for MATLAB

d02nh

1 Purpose

d02nh is a forward communication function for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations when the Jacobian is a banded matrix.

2 Syntax

```
[t, tout, y, ydot, rwork, inform, ysav, wkjac, jacpvt, lderiv, ifail] =
d02nh(neq, t, tout, y, ydot, rwork, rtol, atol, itol, inform, resid,
ysav, jac, wkjac, jacpvt, monitr, lderiv, itask, itrace, 'sdysav',
sdysav, 'nwkjac', nwkjac, 'njcpvt', njcpvt)
```

3 Description

d02nh is a general purpose function for integrating the initial value problem for a stiff system of implicit ordinary differential equations coupled with algebraic equations, written in the form

$$A(t,y)y' = g(t,y).$$

It is designed specifically for the case where the resulting Jacobian is a banded matrix (see the description of (sub)program **jac**).

Both interval and step oriented modes of operation are available and also modes designed to permit intermediate output within an interval oriented mode.

An outline of a typical calling program for d02nh is given below. It calls the banded matrix linear algebra setup function d02nt, and the Backward Differentiation Formula (BDF) integrator setup function d02nv, and its diagnostic counterpart d02ny.

```
.
.
.
[...] = d02nv(...);
[...] = d02nt(...);
[..., ifail] = d02nh(...);
if (ifail ~= 1 and ifail < 14)
  [...] = d02ny(...);
end
.
.
.
```

The linear algebra setup function d02nt and one of the integrator setup functions, d02mv, d02nv or d02nw, must be called prior to the call of d02nh. The integrator diagnostic function d02ny may be called after the call to d02nh. There is also a function, d02nz, designed to permit you to change step size on a continuation call to d02nh without restarting the integration process.

4 References

See the D02M/N Sub-chapter Introduction.

5 Parameters

5.1 Compulsory Input Parameters

1: **neq – int32 scalar**

The number of differential equations to be solved.

Constraint: **neq** ≥ 1.

2: **t – double scalar**

t, the value of the independent variable. The input value of **t** is used only on the first call as the initial point of the integration.

3: **tout – double scalar**

The next value of t at which a computed solution is desired. For the initial t, the input value of **tout** is used to determine the direction of integration. Integration is permitted in either direction (see also **itask**).

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

4: **y**(**ldysav**) **– double array**

ldysav, the first dimension of the array, must be at least **neq**.

The values of the dependent variables (solution). On the first call the first **neq** elements of y must contain the vector of initial values.

5: **ydot**(**ldysav**) **– double array**

ldysav, the first dimension of the array, must be at least **neq**.

If **lderiv**(1) = **true**, **ydot** must contain approximations to the time derivatives y' of the vector y.

If **lderiv**(1) = **false**, **ydot** need not be set on entry.

6: **rwork**$(\mathbf{50} + \mathbf{4} \times \mathbf{ldysav})$ **– double array**

7: **rtol**$(*)$ **– double array**

Note: the dimension of the array **rtol** must be at least 1 if **itol** $= 1$ or **itol** $= 2$, and at least **neq** otherwise.

The relative local error tolerance.

Constraint: **rtol**$(i) \geq 0.0$ for all relevant i (see **itol**).

8: **atol**$(*)$ **– double array**

Note: the dimension of the array **atol** must be at least 1 if **itol** $= 1$ or **itol** $= 3$, and at least **neq** otherwise.

The absolute local error tolerance.

Constraint: **atol**$(i) \geq 0.0$ for all relevant i (see **itol**).

9: **itol – int32 scalar**

A value to indicate the form of the local error test. **itol** indicates to d02nh whether to interpret either or both of **rtol** or **atol** as a vector or a scalar. The error test to be satisfied is $\|e_i/w_i\| < 1.0$, where w_i is defined as follows:

itol	**rtol**	**atol**	w_i
1	scalar	scalar	**rtol**$(1) \times \lvert y_i \rvert +$ **atol**(1)
2	scalar	vector	**rtol**$(1) \times \lvert y_i \rvert +$ **atol**(i)

3	vector	scalar	$\mathbf{rtol}(i) \times \lvert y_i \rvert + \mathbf{atol}(1)$
4	vector	vector	$\mathbf{rtol}(i) \times \lvert y_i \rvert + \mathbf{atol}(i)$

e_i is an estimate of the local error in y_i, computed internally, and the choice of norm to be used is defined by a previous call to an integrator setup function.

Constraint: $1 \leq \mathbf{itol} \leq 4$.

10: **inform(23) – int32 array**

11: **resid – string containing name of m-file**

resid must evaluate the residual

$$r = g(t,y) - A(t,y)y'$$

in one case and

$$r = -A(t,y)y'$$

in another.

Its specification is:

```
[r, ires] = resid(neq, t, y, ydot, ires)
```

Input Parameters

1: **neq – int32 scalar**

the number of equations being solved.

2: **t – double scalar**

t, the current value of the independent variable.

3: **y(neq) – double array**

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

4: **ydot(neq) – double array**

The value of y_i' at t, for $i = 1, 2, \ldots,$ **neq**.

5: **ires – int32 scalar**

The form of the residual that must be returned in the array **r**.

ires $= -1$

The residual defined by equation (2) must be returned.

ires $= 1$

The residual defined by equation (1) must be returned.

Should be unchanged unless one of the following actions is required of the integrator, in which case **ires** should be set accordingly.

ires $= 2$

Indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to **ifail** $= 11$.

ires $= 3$

Indicates to the integrator that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to

try to avoid this condition. If this is not possible the integrator returns to the calling (sub)program with the error indicator set to **ifail** $= 7$.

ires $= 4$

Indicates to the integrator to stop its current operation and to enter the (sub)program **monitr** immediately with parameter **imon** $= -2$.

Output Parameters

1: **r(neq) – double array**

r(i) must contain the ith component of r_i, for $i = 1, 2, \ldots,$ **neq**, where

$$r = g(t,y) - A(t,y)y' \tag{1}$$

or

$$r = -A(t,y)y' \tag{2}$$

and where the definition of r is determined by the input value of **ires**.

2: **ires – int32 scalar**

The form of the residual that must be returned in the array **r**.

ires $= -1$

The residual defined by equation (2) must be returned.

ires $= 1$

The residual defined by equation (1) must be returned.

Should be unchanged unless one of the following actions is required of the integrator, in which case **ires** should be set accordingly.

ires $= 2$

Indicates to the integrator that control should be passed back immediately to the calling (sub)program with the error indicator set to **ifail** $= 11$.

ires $= 3$

Indicates to the integrator that an error condition has occurred in the solution vector, its time derivative or in the value of t. The integrator will use a smaller time step to try to avoid this condition. If this is not possible the integrator returns to the calling (sub)program with the error indicator set to **ifail** $= 7$.

ires $= 4$

Indicates to the integrator to stop its current operation and to enter the (sub)program **monitr** immediately with parameter **imon** $= -2$.

12: **ysav(ldysav,sdysav) – double array**

ldysav, the first dimension of the array, must be at least **neq**.

An appropriate value for **sdysav** is described in the specifications of the integrator setup functions d02mv, d02nv and d02nw. This value must be the same as that supplied to the integrator setup function.

13: **jac – string containing name of m-file**

jac must evaluate the Jacobian of the system. If this option is not required, the actual argument for **jac** must be the string `'d02nhz'`. **d02nhz** is included in the NAG Fortran Library. You must indicate whether this option is to be used by setting the parameter **jceval** appropriately in a call to the linear algebra setup function d02nt.

First we must define the system of nonlinear equations which is solved internally by the integrator. The time derivative, y', generated internally, has the form

$$y' = (y - z)/(hd),$$

where h is the current step size and d is a parameter that depends on the integration method in use. The vector y is the current solution and the vector z depends on information from the previous time steps. This means that $\frac{d}{dy'}(\ ) = \frac{1}{(hd)}\frac{d}{dy}(\ )$. The system of nonlinear equations that is solved has the form

$$A(t,y)y' - g(t,y) = 0$$

but is solved in the form

$$r(t,y) = 0,$$

where r is the function defined by

$$r(t,y) = (hd)(A(t,y)(y - z)/(hd) - g(t,y)).$$

It is the Jacobian matrix $\dfrac{\partial r}{\partial y}$ that you must supply in **jac** as follows:

$$\frac{\partial r_i}{\partial y_j} = a_{ij}(t,y) + (hd)\frac{\partial}{\partial y_j}\left(\sum_{k=1}^{\mathbf{neq}} a_{ik}(t,y)y'_k - g_i(t,y)\right).$$

Its specification is:

```
[p] = jac(neq, t, y, ydot, h, d, ml, mu, p)
```

Input Parameters

1: **neq – int32 scalar**

the number of equations being solved.

2: **t – double scalar**

t, the current value of the independent variable.

3: **y(neq) – double array**

The current solution component y_i, for $i = 1, 2, \ldots,$ **neq**.

4: **ydot(neq) – double array**

The derivative of the solution at the current point t.

5: **h – double scalar**

The current step size.

6: **d – double scalar**

The parameter d which depends on the integration method.

7: **ml – int32 scalar**

8: **mu – int32 scalar**

The number of subdiagonals and superdiagonals respectively in the band.

9: **p(ml + mu + 1,neq) – double array**

Is set to zero.

Elements of the Jacobian matrix $\frac{\partial r}{\partial y}$ stored as specified by the following pseudo-code

```
for i = 1:neq
  j1 = max(i-ml,1);
  j2 = min(i+mu,neq);
  for j = j1:j2
    k = min(ml+1-i,0)+j;
    p(k,i) = δr/δy(i,j);
  end
end
```

See also the function document for f07bd.

Only nonzero elements of this array need be set, since it is preset to zero before the call to **jac**.

Output Parameters

1: **p(ml + mu + 1,neq) – double array**

Is set to zero.

Elements of the Jacobian matrix $\frac{\partial r}{\partial y}$ stored as specified by the following pseudo-code

```
for i = 1:neq
  j1 = max(i-ml,1);
  j2 = min(i+mu,neq);
  for j = j1:j2
    k = min(ml+1-i,0)+j;
    p(k,i) = δr/δy(i,j);
  end
end
```

See also the function document for f07bd.

Only nonzero elements of this array need be set, since it is preset to zero before the call to **jac**.

14: **wkjac(nwkjac) – double array**

This value must be the same as that supplied to the linear algebra setup function d02nt.

Constraint: **nwkjac** $\geq (2m_L + m_U + 1) \times$ **ldysav**, where m_L and m_U are the number of subdiagonals and superdiagonals respectively in the band defined by a call to d02nt.

15: **jacpvt(njcpvt) – int32 array**

the size of the array **jacpvt**. This value must be the same as that supplied to the linear algebra setup function d02nt.

Constraint: **njcpvt** $\geq$ **ldysav**.

16: **monitr – string containing name of m-file**

monitr performs tasks requested by you. If this option is not required, then the actual argument for **monitr** must be the string `'d02nby'`. **d02nby** is included in the NAG Fortran Library.

Its specification is:

```
[hnext, y, imon, inln, hmin, hmax] = monitr(neq, ldysav, t, hlast,
hnext, y, ydot, ysav, r, acor, imon, hmin, hmax, nqu)
```

Input Parameters

1: **neq – int32 scalar**

the number of equations being solved.

2: **ldysav – int32 scalar**

An upper bound on the number of equations to be solved.

3: **t – double scalar**

The current value of the independent variable.

4: **hlast – double scalar**

The last step size successfully used by the integrator.

5: **hnext – double scalar**

The step size that the integrator proposes to take on the next step.

The next step size to be used. If this is different from the input value, then **imon** must be set to 4.

6: **y**(**ldysav**) **– double array**

ldysav, the first dimension of the array, must be at least **neq**.

y, the values of the dependent variables evaluated at t.

These values must not be changed unless **imon** is set to 2.

7: **ydot**(**ldysav**) **– double array**

ldysav, the first dimension of the array, must be at least **neq**.

The time derivatives y' of the vector y.

8: **ysav**(**ldysav,***sdysav*) **– double array**

ldysav, the first dimension of the array, must be at least **neq**.

Workspace to enable you to carry out interpolation using either of the functions d02xj or d02xk.

9: **r**(**ldysav**) **– double array**

ldysav, the first dimension of the array, must be at least **neq**.

If **imon** $= 0$ and **inln** $= 3$, the first **neq** elements contain the residual vector $A(t,y)y' - g(t,y)$.

10: **acor**(**ldysav,2**) **– double array**

ldysav, the first dimension of the array, must be at least **neq**.

With **imon** $= 1$, **acor**$(i,1)$ contains the weight used for the ith equation when the norm is evaluated, and **acor**$(i,2)$ contains the estimated local error for the ith equation. The scaled local error at the end of a timestep may be obtained by calling the double function d02za as follows:

```
[errloc, ifail] = d02za(acor(1:neq,2), acor(1:neq,1));
% Check ifail before proceeding
```

11: **imon – int32 scalar**

A flag indicating under what circumstances **monitr** was called:

imon $= -2$

Entry from the integrator after **ires** $= 4$ (set in user-supplied (sub)program **resid**) caused an early termination (this facility could be used to locate discontinuities).

imon $= -1$

The current step failed repeatedly.

imon $= 0$

Entry after a call to the internal nonlinear equation solver (see **inln**).

imon $= 1$

The current step was successful.

May be reset to determine subsequent action in d02nh.

imon $= -2$

Integration is to be halted. A return will be made from the integrator to the calling (sub)program with **ifail** $= 12$.

imon $= -1$

Allow the integrator to continue with its own internal strategy. The integrator will try up to three restarts unless **imon** is set $\neq -1$ on exit.

imon $= 0$

Return to the internal nonlinear equation solver, where the action taken is determined by the value of **inln**.

imon $= 1$

Normal exit to the integrator to continue integration.

imon $= 2$

Restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The **monitr** provided solution **y** will be used for the initial conditions.

imon $= 3$

Try to continue with the same step size and order as was to be used before the call to **monitr**. **hmin** and **hmax** may be altered if desired.

imon $= 4$

Continue the integration but using a new value **hnext** and possibly new values of **hmin** and **hmax**.

12: **hmin – double scalar**

The minimum step size to be taken on the next step.

The minimum step size to be used. If this is different from the input value, then **imon** must be set to 3 or 4.

13: **hmax – double scalar**

The maximum step size to be taken on the next step.

The maximum step size to be used. If this is different from the input value, then **imon** must be set to 3 or 4. If **hmax** is set to zero, no limit is assumed.

14: **nqu – int32 scalar**

The order of the integrator used on the last step. This is supplied to enable you to carry out interpolation using either of the functions d02xj or d02xk.

Output Parameters

1: **hnext – double scalar**

The step size that the integrator proposes to take on the next step.

The next step size to be used. If this is different from the input value, then **imon** must be set to 4.

2: **y**(**ldysav**) **– double array**

y, the values of the dependent variables evaluated at t.

These values must not be changed unless **imon** is set to 2.

3: **imon – int32 scalar**

A flag indicating under what circumstances **monitr** was called:

imon $= -2$

Entry from the integrator after **ires** $= 4$ (set in user-supplied (sub)program **resid**) caused an early termination (this facility could be used to locate discontinuities).

imon $= -1$

The current step failed repeatedly.

imon $= 0$

Entry after a call to the internal nonlinear equation solver (see **inln**).

imon $= 1$

The current step was successful.

May be reset to determine subsequent action in d02nh.

imon $= -2$

Integration is to be halted. A return will be made from the integrator to the calling (sub)program with **ifail** $= 12$.

imon $= -1$

Allow the integrator to continue with its own internal strategy. The integrator will try up to three restarts unless **imon** is set $\neq -1$ on exit.

imon $= 0$

Return to the internal nonlinear equation solver, where the action taken is determined by the value of **inln**.

imon $= 1$

Normal exit to the integrator to continue integration.

imon $= 2$

Restart the integration at the current time point. The integrator will restart from order 1 when this option is used. The **monitr** provided solution **y** will be used for the initial conditions.

imon $= 3$

Try to continue with the same step size and order as was to be used before the call to **monitr**. **hmin** and **hmax** may be altered if desired.

imon $= 4$

Continue the integration but using a new value **hnext** and possibly new values of **hmin** and **hmax**.

4: **inln – int32 scalar**

The action to be taken by the internal nonlinear equation solver when **monitr** is exited with **imon** $= 0$. By setting **inln** $= 3$ and returning to the integrator, the residual vector is evaluated and placed in the array **r**, and then **monitr** is called again. At present this is the only option available: **inln** must not be set to any other value.

5: **hmin – double scalar**

The minimum step size to be taken on the next step.

The minimum step size to be used. If this is different from the input value, then **imon** must be set to 3 or 4.

6: **hmax – double scalar**

The maximum step size to be taken on the next step.

The maximum step size to be used. If this is different from the input value, then **imon** must be set to 3 or 4. If **hmax** is set to zero, no limit is assumed.

17: **lderiv**(2) **– logical array**

lderiv(1) must be set to **true**, if you have supplied both an initial y and an initial y'. **lderiv**(1) must be set to **false**, if only the initial y has been supplied.

lderiv(2) must be set to **true** if the integrator is to use a modified Newton method to evaluate the initial y and y'. Note that y and y', if supplied, are used as initial estimates. This method involves taking a small step at the start of the integration, and if **itask** $= 6$ on entry, **t** and **tout** will be set to the result of taking this small step. **lderiv**(2) must be set to **false** if the integrator is to use functional iteration to evaluate the initial y and y', and if this fails a modified Newton method will then be attempted. **lderiv**(2) $=$ **true** is recommended if there are implicit equations or the initial y and y' are zero.

18: **itask – int32 scalar**

The task to be performed by the integrator.

itask $= 1$

Normal computation of output values of $y(t)$ at $t =$ **tout** (by overshooting and interpolating).

itask $= 2$

Take one step only and return.

itask $= 3$

Stop at the first internal integration point at or beyond $t =$ **tout** and return.

itask $= 4$

Normal computation of output values of $y(t)$ at $t =$ **tout** but without overshooting $t =$ **tcrit**. **tcrit** must be specified as an option in one of the integrator setup functions prior to the first call to the integrator, or specified in the optional input function prior to a continuation call. **tcrit** may be equal to or beyond **tout**, but not before it, in the direction of integration.

itask $= 5$

Take one step only and return, without passing **tcrit**. **tcrit** must be specified as under **itask** $= 4$.

itask $= 6$

The integrator will solve for the initial values of y and y' only and then return to the calling (sub)program without doing the integration. This option can be used to check the initial values of y and y'. Functional iteration or a 'small' backward Euler method used in conjunction with a damped Newton iteration is used to calculate these values (see **lderiv**). Note that if a backward Euler step is used then the value of t will have been advanced a short distance from the initial point.

Note: if d02nh is recalled with a different value of **itask** (and **tout** altered), then the initialization procedure is repeated, possibly leading to different initial conditions.

Constraint: $1 \leq$ **itask** ≤ 6.

19: **itrace – int32 scalar**

The level of output that is printed by the integrator. **itrace** may take the value -1, 0, 1, 2 or 3.

itrace < -1

-1 is assumed and similarly if **itrace** > 3, then 3 is assumed.

itrace $= -1$

No output is generated.

itrace $= 0$

Only warning messages are printed on the current error message unit (see x04aa).

itrace > 0

Warning messages are printed as above, and on the current advisory message unit (see x04ab) output is generated which details Jacobian entries, the nonlinear iteration and the time integration. The advisory messages are given in greater detail the larger the value of **itrace**.

5.2 Optional Input Parameters

1: **sdysav – int32 scalar**

Default: The second dimension of the array **ysav**.

An appropriate value for **sdysav** is described in the specifications of the integrator setup functions d02mv, d02nv and d02nw. This value must be the same as that supplied to the integrator setup function.

2: **nwkjac – int32 scalar**

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

This value must be the same as that supplied to the linear algebra setup function d02nt.

Constraint: **nwkjac** $\geq (2m_L + m_U + 1) \times$ **ldysav**, where m_L and m_U are the number of subdiagonals and superdiagonals respectively in the band defined by a call to d02nt.

3: **njcpvt – int32 scalar**

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

the size of the array **jacpvt**. This value must be the same as that supplied to the linear algebra setup function d02nt.

Constraint: **njcpvt** $\geq$ **ldysav**.

5.3 Input Parameters Omitted from the MATLAB Interface

ldysav

5.4 Output Parameters

1: **t – double scalar**

The value at which the computed solution y is returned (usually at **tout**).

2: **tout – double scalar**

Normally unchanged. However when **itask** $= 6$, then **tout** contains the value of **t** at which initial values have been computed without performing any integration. See descriptions of **itask** and **lderiv**.

3: **y**(**ldysav**) **– double array**

The computed solution vector, evaluated at t (usually $t =$ **tout**).

4: **ydot**(**ldysav**) **– double array**

The time derivatives y' of the vector y at the last integration point.

5: **rwork**($\mathbf{50} + \mathbf{4} \times$ **ldysav**) **– double array**

6: **inform**(**23**) **– int32 array**

7: **ysav**(**ldysav,sdysav**) **– double array**

8: **wkjac**(**nwkjac**) **– double array**

9: **jacpvt**(**njcpvt**) **– int32 array**

10: **lderiv**(**2**) **– logical array**

lderiv(1) is normally unchanged. However if **itask** $= 6$ and internal initialization was successful then **lderiv**(1) $=$ **true**.

lderiv(2) $=$ **true**, if implicit equations were detected. Otherwise **lderiv**(2) $=$ **false**.

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 $= 1$

An illegal input was detected on entry, or after an internal call to (sub)program **monitr**. If **itrace** > -1, then the form of the error will be detailed on the current error message unit (see x04aa).

ifail $= 2$

The maximum number of steps specified has been taken (see the description of optional inputs in the integrator setup functions and the optional input continuation d02nz).

ifail $= 3$

With the given values of **rtol** and **atol** no further progress can be made across the integration range from the current point **t**. The components $\mathbf{y}(1), \mathbf{y}(2), \ldots, \mathbf{y}(\mathbf{neq})$ contain the computed values of the solution at the current point **t**.

ifail $= 4$

There were repeated error test failures on an attempted step, before completing the requested task, but the integration was successful as far as **t**. The problem may have a singularity, or the local error requirements may be inappropriate.

ifail $= 5$

There were repeated convergence test failures on an attempted step, before completing the requested task, but the integration was successful as far as **t**. This may be caused by an inaccurate Jacobian matrix or one which is incorrectly computed.

ifail $= 6$

Some error weight w_i became zero during the integration (see the description of **itol**). Pure relative error control (**atol**$(i) = 0.0$) was requested on a variable (the ith) which has now vanished. The integration was successful as far as **t**.

ifail $= 7$

The user-supplied (sub)program **resid** set its error flag (**ires** $= 3$) continually despite repeated attempts by the integrator to avoid this.

ifail $= 8$

lderiv$(1) =$ **false** on entry but the internal initialization function was unable to initialize y' (more detailed information may be directed to the current error message unit, see x04aa).

ifail $= 9$

A singular Jacobian $\dfrac{\partial r}{\partial y}$ has been encountered. You should check the problem formulation and Jacobian calculation.

ifail $= 10$

An error occurred during Jacobian formulation or back-substitution (a more detailed error description may be directed to the current error message unit, see x04aa).

ifail $= 11$

The user-supplied (sub)program **resid** signalled the integrator to halt the integration and return (**ires** $= 2$). Integration was successful as far as **t**.

ifail $= 12$

The (sub)program **monitr** set **imon** $= -2$ and so forced a return but the integration was successful as far as **t**.

ifail $= 13$

The requested task has been completed, but it is estimated that a small change in **rtol** and **atol** is unlikely to produce any change in the computed solution. (Only applies when you are not operating in one step mode, that is when **itask** $\neq 2$ or 5.)

ifail $= 14$

The values of **rtol** and **atol** are so small that the function is unable to start the integration.

ifail $= 15$

The linear algebra setup function d02nt was not called before the call to d02nh.

7 Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the parameters **rtol** and **atol**, and to a much lesser extent by the choice of norm. You are advised to use scalar error control unless the components of the solution are expected to be poorly scaled. For the type of decaying solution typical of many stiff problems, relative error control with a small absolute error threshold will be most appropriate (that is, you are advised to choose **itol** $= 1$ with **atol**(1) small but positive).

8 Further Comments

The cost of computing a solution depends critically on the size of the differential system and to a lesser extent on the degree of stiffness of the problem. For d02nh the cost is proportional to **neq** $\times$ $(\mathbf{ml} + \mathbf{mu} + 1)^2$ though for problems which are only mildly nonlinear the cost may be dominated by factors proportional to **neq** $\times$ $(\mathbf{ml} + \mathbf{mu} + 1)$ except for very large problems.

In general, you are advised to choose the BDF option (setup function d02nv) but if efficiency is of great importance and especially if it is suspected that $\dfrac{\partial}{\partial y}\left(A^{-1}g\right)$ has complex eigenvalues near the imaginary axis for some part of the integration, you should try the BLEND option (setup function d02nw).

9 Example

d02nh_jac.m

```
function p = jac(neq, t, y, ydot, h, d, ml, mu, pIn)
  p = zeros(ml+mu+1, neq);
```

d02nh_monitr.m

```
function [hnext, y, imon, inln, hmin, hmax] = ...
    monitr(neq, neqmax, t, hlast, hnext, y, ydot, ysave, r, acor, imon,
hmin, hmax, nqu)

  inln = int32(0);
```

d02nh_resid.m

```
function [r, ires] = resid(neq, t, y, ydot, ires)

  r = zeros(neq,1);
```

```
   r(1) = -ydot(1) - ydot(2) - ydot(3);
   r(2) = -ydot(2);
   r(3) = -ydot(3);
   if (ires == 1) ;
      r(1) = 0.0d0 + r(1);
      r(2) = 0.04d0*y(1) - 1.0d4*y(2)*y(3) - 3.0d7*y(2)*y(2) + r(2);
      r(3) = 3.0d7*y(2)*y(2) + r(3);
   end ;
```

```
neq = int32(3);
t = 0;
tout = 10;
y = [1;
     0;
     0];
ydot = zeros(3, 1);
rwork = zeros(62, 1);
rtol = [0.0001];
atol = [1e-06;
     1e-07;
     1e-06];
itol = int32(2);
inform = zeros(23, 1, 'int32');
ysave = zeros(3, 6);
wkjac = zeros(15, 1);
jacpvt = zeros(3, 1, 'int32');
lderiv = [false;
     false];
itask = int32(6);
itrace = int32(0);
[const, rwork, ifail] = ...
    d02nv(int32(3), int32(6), int32(5), 'Newton', false, zeros(6), ...
    0, 1e-10, 10, 1e-4, int32(200), int32(5), 'Average-L2', rwork);
[rwork, ifail] = ...
    d02nt(int32(3), int32(3), 'Numerical', int32(1), int32(2), int32(15),
...
    int32(3), rwork);
[tOut, toutOut, yOut, ydotOut, rworkOut, informOut, ysaveOut, ...
 wkjacOut, jacpvtOut, lderivOut, ifail] = ...
    d02nh(neq, t, tout, y, ydot, rwork, rtol, atol, itol, inform, ...
    'd02nh_resid', ysave, 'd02nh_jac', wkjac, jacpvt, 'd02nh_monitr', ...
    lderiv, itask, itrace)
```

```
WARNING... EQUATION(=I1) AND POSSIBLY OTHER EQUATIONS ARE
 IMPLICIT AND IN CALCULATING THE INITIAL VALUES THE EQNS
 WILL BE TREATED AS IMPLICIT.
 IN ABOVE MESSAGE I1 =         1
tOut =
   3.1500e-06
toutOut =
   3.1500e-06
yOut =
    1.0000
    0.0000
    0.0000
ydotOut =
   -0.0400
    0.0400
    0.0000
rworkOut =
     array elided
informOut =
           0
          10
           1
           0
           1
           2
```

```
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
           0
ysaveOut =
     0     0     1     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
wkjacOut =
    1.0000
    1.0000
    1.0000
         0
    1.0000
    0.0000
         0
         0
    1.0000
         0
         0
         0
   -0.0000
   -0.0000
    1.0000
jacpvtOut =
           1
           2
           3
lderivOut =
     1
     0
ifail =
           0
```