NAG Toolbox

nag_ode_ivp_stiff_imp_revcom (d02nn)

1 Purpose

nag_ode_ivp_stiff_imp_revcom (d02nn) is a reverse communication function for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations.

2 Syntax

```
[t, tout, y, ydot, rwork, inform, ysav, wkjac, jacpvt, imon, inln, ires,
irevcm, lderiv, ifail] = nag_ode_ivp_stiff_imp_revcom(t, tout, y, ydot, rwork,
rtol, atol, itol, inform, ysav, wkjac, jacpvt, imon, inln, ires, irevcm,
lderiv, itask, itrace, 'neq', neq, 'sdysav', sdysav, 'nwkjac', nwkjac)
[t, tout, y, ydot, rwork, inform, ysav, wkjac, jacpvt, imon, inln, ires,
irevcm, lderiv, ifail] = d02nn(t, tout, y, ydot, rwork, rtol, atol, itol,
inform, ysav, wkjac, jacpvt, imon, inln, ires, irevcm, lderiv, itask, itrace,
'neq', neq, 'sdysav', sdysav, 'nwkjac', nwkjac)
```

Note: the interface to this routine has changed since earlier releases of the toolbox:

At Mark 22: *njcpvt* was removed from the interface; **neq** was made optional.

3 Description

nag_ode_ivp_stiff_imp_revcom (d02nn) 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

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

An outline of a typical calling program is given below:

```
%
 % data initialization
 %
 call linear algebra setup routine
 call integrator setup routine
 irevcm = int32(0);
 [..., irevcm, ...] = d02nn();
 while (irevcm > 0)
   if (irevcm > 7 and irevcm < 11)
     if (irevcm == 8)
       supply the jacobian matrix                              (i)
     elseif (irevcm == 9)
       perform monitoring tasks requested by the user         (ii)
     elseif (irevcm == 10)
       indicates an unsuccessful step
     end  else    evaluate the residual                                (iii)
   end
   [..., irevcm, ...] = d02nn();
 end
 %
 % post processing (optional linear algebra diagnostic call
 % (sparse case only), optional integrator diagnostic call)
 %
```

There are three major operations that may be required of the calling function on an intermediate return (**irevcm** $\neq 0$) from nag_ode_ivp_stiff_imp_revcom (d02nn); these are denoted (i), (ii) and (iii).

The following sections describe in greater detail exactly what is required of each of these operations.

(i) **Supply the Jacobian matrix**

You need only provide this facility if the argument **jceval** = 'A' (or **jceval** = 'F' if using sparse matrix linear algebra) in a call to the linear algebra setup function (see **jceval** in nag_ode_ivp_stiff_sparjac_setup (d02nu)). If the Jacobian matrix is to be evaluated numerically by the integrator, then the remainder of section (i) can be ignored.

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

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

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

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

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

where f is the function defined by

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

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

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

where t, h and d are located in **rwork**(19), **rwork**(16) and **rwork**(20) respectively and the arrays **y** and **ydot** contain the current solution and time derivatives respectively. Only the nonzero elements of the Jacobian need be set, since the locations where it is to be stored are preset to zero.

Hereafter in this document this operation will be referred to as jac.

(ii) **Perform tasks requested by you**

This operation is essentially a monitoring function and additionally provides the opportunity of changing the current values of **y**, **ydot**, **hnext** (the step size that the integrator proposes to take on the next step), **hmin** (the minimum step size to be taken on the next step), and **hmax** (the maximum step size to be taken on the next step). The scaled local error at the end of a time step may be obtained by calling the double function nag_ode_ivp_stiff_errest (d02za) as follows:

```
[errloc, ifail] = d02za(rwork(51+neqmax:51+neqmax+neq-1), rwork(51:51
+neq-1));
 % Check ifail before proceeding
```

The following gives details of the location within the array **rwork** of variables that may be of interest to you:

Variable	Specification	Location
tcurr	the current value of the independent variable	**rwork**(19)
hlast	last step size successfully used by the integrator	**rwork**(15)
hnext	step size that the integrator proposes to take on the next step	**rwork**(16)
hmin	minimum step size to be taken on the next step	**rwork**(17)
hmax	maximum step size to be taken on the next step	**rwork**(18)
nqu	the order of the integrator used on the last step	**rwork**(10)

You are advised to consult the description of **monitr** in nag_ode_ivp_stiff_imp_fulljac (d02ng) for details on what optional input can be made.

If either **y** or **ydot** are changed, then **imon** must be set to 2 before return to nag_ode_ivp_stiff_imp_revcom (d02nn). If either of the values **hmin** or **hmax** are changed, then **imon** must be set ≥ 3 before return to nag_ode_ivp_stiff_imp_revcom (d02nn). If **hnext** is changed, then **imon** must be set to 4 before return to nag_ode_ivp_stiff_imp_revcom (d02nn).

In addition you can force nag_ode_ivp_stiff_imp_revcom (d02nn) to evaluate the residual vector

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

by setting **imon** $= 0$ and **inln** $= 3$ and then returning to nag_ode_ivp_stiff_imp_revcom (d02nn); on return to this monitoring operation the residual vector will be stored in **rwork**$(50 + 2 \times$ **neq** $+ i)$, for $i = 1, 2, \ldots,$ **neq**.

Hereafter in this document this operation will be referred to as monitr.

(iii) **Evaluate the residual**

This operation must evaluate the residual

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

in one case and the reduced residual

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

in another, where t is located in **rwork**(19). The form of the residual that is returned is determined by the value of **ires** returned by nag_ode_ivp_stiff_imp_revcom (d02nn). If **ires** $= -1$, then the residual defined by equation (2) above must be returned; if **ires** $= 1$, then the residual returned by equation (1) above must be returned.

Hereafter in this document this operation will be referred to as resid.

4 References

See the D02M–N Sub-chapter Introduction.

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 argument **irevcm**. Between intermediate exits and re-entries, **all arguments other than ydot, rwork, wkjac, imon, inln and ires must remain unchanged**.

5.1 Compulsory Input Parameters

1: **t** – REAL (KIND=nag_wp)

On initial entry: 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.

2: **tout** – REAL (KIND=nag_wp)

On initial entry: 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**.

3: **y**(**neq**) – REAL (KIND=nag_wp) array

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

4: **ydot**(**neq**) – REAL (KIND=nag_wp) array

On initial entry: if **lderiv**$(1) = true$, **ydot** must contain approximations to the time derivatives y' of the vector y. If **lderiv**$(1) = false$, then **ydot** need not be set on entry.

5: **rwork**(**50** + **4** × **neq**) – REAL (KIND=nag_wp) array

On initial entry: must be the same array as used by one of the method setup functions nag_ode_ivp_stiff_dassl (d02mv), nag_ode_ivp_stiff_bdf (d02nv) or nag_ode_ivp_stiff_blend (d02nw), and by one of the storage setup functions nag_ode_ivp_stiff_fulljac_setup (d02ns), nag_ode_ivp_stiff_bandjac_setup (d02nt) or nag_ode_ivp_stiff_sparjac_setup (d02nu). The contents of **rwork** must not be changed between any call to a setup function and the first call to nag_ode_ivp_stiff_imp_revcom (d02nn).

On intermediate re-entry: must contain residual evaluations as described under the argument **irevcm**.

6: **rtol**(:) – REAL (KIND=nag_wp) array

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

On initial entry: the relative local error tolerance.

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

7: **atol**(:) – REAL (KIND=nag_wp) array

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

On initial entry: the absolute local error tolerance.

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

8: **itol** – INTEGER

On initial entry: a value to indicate the form of the local error test. **itol** indicates to nag_ode_ivp_stiff_imp_revcom (d02nn) 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**$(1) \times \|y_i\| +$ **atol**(1)
2	scalar	vector	**rtol**$(1) \times \|y_i\| +$ **atol**(i)
3	vector	scalar	**rtol**$(i) \times \|y_i\| +$ **atol**(1)
4	vector	vector	**rtol**$(i) \times \|y_i\| +$ **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: **itol** = 1, 2, 3 or 4.

9: **inform**(**23**) – INTEGER array

10: **ysav**($ldysav$, **sdysav**) – REAL (KIND=nag_wp) array

11: **wkjac**(**nwkjac**) – REAL (KIND=nag_wp) array

On intermediate re-entry: elements of the Jacobian as defined under the description of **irevcm**. If a numerical Jacobian was requested then **wkjac** is used for workspace.

12: **jacpvt**($njcpvt$) – INTEGER array

On initial entry: the dimension of the array **jacpvt**. The actual size depends on the linear algebra method used. An appropriate value for $njcpvt$ is described in the specifications of the linear algebra setup functions nag_ode_ivp_stiff_bandjac_setup (d02nt) and nag_ode_ivp_stiff_sparjac_

setup (d02nu) for banded and sparse matrix linear algebra respectively. This value must be the same as that supplied to the linear algebra setup function. When full matrix linear algebra is chosen, the array **jacpvt** is not used and hence $njcpvt$ should be set to 1.

13: **imon** – INTEGER

On intermediate re-entry: may be reset to determine subsequent action in nag_ode_ivp_stiff_imp_revcom (d02nn).

imon $= -2$
Integration is to be halted. A return will be made from nag_ode_ivp_stiff_imp_revcom (d02nn) to the calling (sub)program with **ifail** $= 12$.

imon $= -1$
Allow nag_ode_ivp_stiff_imp_revcom (d02nn) to continue with its own internal strategy. The integrator will try up to three restarts unless **imon** $\neq -1$.

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 nag_ode_ivp_stiff_imp_revcom (d02nn) 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 internal initialization module solves for new values of y and y' by using the values supplied in **y** and **ydot** by the **monitr** operation (see Section 3) as initial estimates.

imon $= 3$
Try to continue with the same step size and order as was to be used before entering the **monitr** operation (see Section 3). **hmin** and **hmax** may be altered if desired.

imon $= 4$
Continue the integration but using a new value of **hnext** and possibly new values of **hmin** and **hmax**.

14: **inln** – INTEGER

On intermediate re-entry: with **imon** $= 0$ and **irevcm** $= 9$, **inln** specifies the action to be taken by the internal nonlinear equation solver. By setting **inln** $= 3$ and returning to nag_ode_ivp_stiff_imp_revcom (d02nn), the residual vector is evaluated and placed in **rwork**$(50 + 2 \times$ **neq** $+ i)$, for $i = 1, 2, \ldots,$ **neq** and then the **monitr** operation (see Section 3) is invoked again. At present this is the only option available: **inln** must not be set to any other value.

15: **ires** – INTEGER

On intermediate re-entry: should be unchanged unless one of the following actions is required of nag_ode_ivp_stiff_imp_revcom (d02nn) in which case **ires** should be set accordingly.

ires $= 2$
Indicates to nag_ode_ivp_stiff_imp_revcom (d02nn) that control should be passed back immediately to the calling (sub)program with the error indicator set to **ifail** $= 11$.

ires $= 3$
Indicates to nag_ode_ivp_stiff_imp_revcom (d02nn) 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 nag_ode_ivp_stiff_imp_revcom (d02nn) returns to the calling (sub)program with the error indicator set to **ifail** $= 7$.

ires $= 4$
Indicates to nag_ode_ivp_stiff_imp_revcom (d02nn) to stop its current operation and to enter the **monitr** operation (see Section 3) immediately.

16: **irevcm** – INTEGER

On initial entry: must contain 0.

On intermediate re-entry: should remain unchanged.

Constraint: $0 \leq$ **irevcm** ≤ 11.

17: **lderiv**(**2**) – LOGICAL array

On initial entry: **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** – INTEGER

On initial entry: 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 before the first call to the integrator, or specified in the optional input function before a continuation call. **tcrit** (e.g., see nag_ode_ivp_stiff_bdf (d02nv)) 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** (e.g., see nag_ode_ivp_stiff_bdf (d02nv)). **tcrit** must be specified 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 nag_ode_ivp_stiff_imp_revcom (d02nn) 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** – INTEGER

On initial entry: 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 nag_file_set_unit_error (x04aa)).

itrace > 0
Warning messages are printed as above, and on the current advisory message unit (see nag_file_set_unit_advisory (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: **neq** – INTEGER

Default: the dimension of the arrays **y**, **ydot** and the first dimension of the array **ysav**. (An error is raised if these dimensions are not equal.)

On initial entry: the number of equations to be solved.

Constraint: **neq** ≥ 1.

2: **sdysav** – INTEGER

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

On initial entry: the second dimension of the array **ysav**. an appropriate value for **sdysav** is described in the specifications of the integrator setup functions nag_ode_ivp_stiff_dassl (d02mv), nag_ode_ivp_stiff_bdf (d02nv) and nag_ode_ivp_stiff_blend (d02nw). This value must be the same as that supplied to the integrator setup function.

3: **nwkjac** – INTEGER

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

On initial entry: the dimension of the array **wkjac**. the actual size depends on the linear algebra method used. An appropriate value for **nwkjac** is described in the specifications of the linear algebra setup functions nag_ode_ivp_stiff_fulljac_setup (d02ns), nag_ode_ivp_stiff_bandjac_setup (d02nt) and nag_ode_ivp_stiff_sparjac_setup (d02nu) for full, banded and sparse matrix linear algebra respectively. This value must be the same as that supplied to the linear algebra setup function.

5.3 Output Parameters

1: **t** – REAL (KIND=nag_wp)

On final exit: the value at which the computed solution y is returned (usually at **tout**).

2: **tout** – REAL (KIND=nag_wp)

Is unaltered unless **itask** $= 6$ and **lderiv**$(2) = \mathit{true}$ on entry (see also **itask** and **lderiv**) in which case **tout** will be set to the result of taking a small step at the start of the integration.

3: **y**(**neq**) – REAL (KIND=nag_wp) array

On final exit: the computed solution vector evaluated at **t** (usually $t =$ **tout**).

4: **ydot**(**neq**) – REAL (KIND=nag_wp) array

On final exit: contains the time derivatives y' of the vector y at the last integration point.

5: **rwork**(**50** + **4** × **neq**) – REAL (KIND=nag_wp) array

On intermediate exit: contains information for **jac**, **resid** and **monitr** operations as described under Section 3 and the argument **irevcm**.

6: **inform**(**23**) – INTEGER array

7: **ysav**($ldysav$, **sdysav**) – REAL (KIND=nag_wp) array

8: **wkjac**(**nwkjac**) – REAL (KIND=nag_wp) array

On intermediate exit: the Jacobian is overwritten.

9: **jacpvt**($njcpvt$) – INTEGER array

10: **imon** – INTEGER

On intermediate exit: used to pass information between nag_ode_ivp_stiff_imp_revcom (d02nn) and the **monitr** operation (see Section 3). With **irevcm** = 9, **imon** contains a flag indicating under what circumstances the return from nag_ode_ivp_stiff_imp_revcom (d02nn) occurred:

imon = −2
: Exit from nag_ode_ivp_stiff_imp_revcom (d02nn) after **ires** = 4 (set in the **resid** operation (see Section 3) caused an early termination (this facility could be used to locate discontinuities).

imon = −1
: The current step failed repeatedly.

imon = 0
: Exit from nag_ode_ivp_stiff_imp_revcom (d02nn) after a call to the internal nonlinear equation solver.

imon = 1
: The current step was successful.

11: **inln** – INTEGER

On intermediate exit: contains a flag indicating the action to be taken, if any, by the internal nonlinear equation solver.

12: **ires** – INTEGER

On intermediate exit: with **irevcm** = 1, 2, 3, 4, 5, 6, 7 or 11, **ires** specifies the form of the residual to be returned by the **resid** operation (see Section 3).

If **ires** = 1, then $-r = g(t,y) - A(t,y)y'$ must be returned.

If **ires** = −1, then $-\hat{r} = -A(t,y)y'$ must be returned.

13: **irevcm** – INTEGER

On intermediate exit: indicates what action you must take before re-entering nag_ode_ivp_stiff_imp_revcom (d02nn). The possible exit values of **irevcm** are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 or 11 which should be interpreted as follows:

irevcm = 1, 2, 3, 4, 5, 6, 7 or 11
: Indicates that a **resid** operation (see Section 3) is required: you must supply the residual of the system. For each of these values of **irevcm**, y_i is located in **y**(i), for $i = 1, 2, \ldots,$ **neq**.

For **irevcm** $= 1$, 3, 6 or 11, y'_i is located in **ydot**(i) and r_i should be stored in **rwork**$(50 + 2 \times$ **neq** $+ i)$, for $i = 1, 2, \ldots,$ **neq**.

For **irevcm** $= 2$, y'_i is located in **rwork**$(50 +$ **neq** $+ i)$ and r_i should be stored in **rwork**$(50 + 2 \times$ **neq** $+ i)$, for $i = 1, 2, \ldots,$ **neq**.

For **irevcm** $= 4$ or 7, y'_i is located in **ydot**(i) and r_i should be stored in **rwork**$(50 +$ **neq** $+ i)$, for $i = 1, 2, \ldots,$ **neq**.

For **irevcm** $= 5$, y'_i is located in **rwork**$(50 + 2 \times$ **neq** $+ i)$ and r_i should be stored in **ydot**(i), for $i = 1, 2, \ldots,$ **neq**.

irevcm $= 8$
Indicates that a **jac** operation (see Section 3) is required: you must supply the Jacobian matrix.

If full matrix linear algebra is being used, then the (i, j)th element of the Jacobian must be stored in **wkjac**$((j - 1) \times$ **neq** $+ i)$.

If banded matrix linear algebra is being used, then the (i, j)th element of the Jacobian must be stored in **wkjac**$((i - 1) \times m_B + k)$, where $m_B = m_L + m_U + 1$ and $k = \min(m_L - i + 1, 0) + j$; here m_L and m_U are the number of subdiagonals and superdiagonals, respectively, in the band.

If sparse matrix linear algebra is being used, then nag_ode_ivp_stiff_sparjac_enq (d02nr) must be called to determine which column of the Jacobian is required and where it should be stored.

```
[j, iplace] = d02nr(inform);
```

will return in **j** the number of the column of the Jacobian that is required and will set **iplace** $= 1$ or 2 (see nag_ode_ivp_stiff_sparjac_enq (d02nr)). If **iplace** $= 1$, you must store the nonzero element (i, j) of the Jacobian in **rwork**$(50 + 2 \times$ **neq** $+ i)$; otherwise it must be stored in **rwork**$(50 +$ **neq** $+ i)$.

irevcm $= 9$
Indicates that a **monitr** operation (see Section 3) can be performed.

irevcm $= 10$
Indicates that the current step was not successful, due to error test failure or convergence test failure. The only information supplied to you on this return is the current value of the variable t, located in **rwork**(19). No values must be changed before re-entering nag_ode_ivp_stiff_imp_revcom (d02nn); this facility enables you to determine the number of unsuccessful steps.

On final exit: **irevcm** $= 0$ indicating that the user-specified task has been completed or an error has been encountered (see the descriptions for **itask** and **ifail**).

14: **lderiv**(**2**) – LOGICAL array

On final exit: **lderiv**(1) is normally unchanged. However if **itask** $= 6$ and internal initialization was successful then **lderiv**$(1) = \mathit{true}$.

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

15: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

6 Error Indicators and Warnings

Errors or warnings detected by the function:

ifail $= 1$

On entry, the integrator detected an illegal input, or that a linear algebra and/or integrator setup function has not been called prior to the call to the integrator. If **itrace** ≥ 0, the form of the error will be detailed on the current error message unit (see nag_file_set_unit_error (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 function, nag_ode_ivp_stiff_contin (d02nz)).

ifail $= 3$ (*warning*)

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 $= 4$ (*warning*)

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 $= 5$ (*warning*)

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 $= 6$ (*warning*)

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 **resid** operation (see Section 3) set the 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 nag_file_set_unit_error (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 nag_file_set_unit_error (x04aa)).

ifail $= 11$ (*warning*)

The **resid** operation (see Section 3) signalled the integrator to halt the integration and return by setting **ires** $= 2$. Integration was successful as far as **t**.

ifail $= 12$ (*warning*)

The **monitr** operation (see Section 3) set **imon** $= -2$ and so forced a return but the integration was successful as far as **t**.

ifail $= 13$ (*warning*)

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 nag_ode_ivp_stiff_imp_revcom (d02nn) is unable to start the integration.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

The accuracy of the numerical solution may be controlled by a careful choice of the arguments **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; also on the type of linear algebra being used. For further details see Section 9 in nag_ode_ivp_stiff_imp_fulljac (d02ng), nag_ode_ivp_stiff_imp_bandjac (d02nh) and nag_ode_ivp_stiff_imp_sparjac (d02nj) of the documents for nag_ode_ivp_stiff_imp_fulljac (d02ng) (full matrix), nag_ode_ivp_stiff_imp_bandjac (d02nh) (banded matrix) or nag_ode_ivp_stiff_imp_sparjac (d02nj) (sparse matrix).

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

9 Example

We solve the well-known stiff Robertson problem written as a differential system in implicit form

$$\begin{aligned} r_1 &= (a' + b' + c') \\ r_2 &= 0.04a - 1.0\text{E}4bc - 3.0\text{E}7b^2 - b' \\ r_3 &= \qquad\qquad\qquad 3.0\text{E}7b^2 - c' \end{aligned}$$

over the range $[0, 10]$ with initial conditions $a = 1.0$ and $b = c = 0.0$ and with scalar error control (**itol** $= 1$). We integrate to the first internal integration point past **tout** $= 10.0$ (**itask** $= 3$), using a BDF method (setup function nag_ode_ivp_stiff_dassl (d02mv)) and a modified Newton method. We treat the Jacobian as sparse (setup function nag_ode_ivp_stiff_sparjac_setup (d02nu)) and we calculate it analytically. In this program we also illustrate the monitoring of step failures (**irevcm** $= 10$) and the forcing of a return when the component falls below 0.9 in the evaluation of the residual by setting **ires** $= 2$.

9.1 Program Text

```
function d02nn_example

fprintf('d02nn example results\n\n');

% Initialize integration method setup variables and arrays.
neq    = nag_int(3);
neqmax = nag_int(neq);
nwkjac = nag_int(neqmax*(neqmax + 1));
maxord = nag_int(5);
sdysav = nag_int(maxord+1);
maxstp = nag_int(200);
mxhnil = nag_int(5);

h0    = 0;
hmax  = 10;
hmin  = 1.0e-10;
tcrit = 0;
petzld = false;

const  = zeros(6, 1);
rwork  = zeros(50+4*neqmax, 1);

[const, rwork, ifail] = d02nv(neqmax, sdysav, maxord, 'Newton', petzld, ...
                              const, tcrit, hmin, hmax, h0, maxstp, ...
                              mxhnil, 'Average-L2', rwork);

% Sparse Jacobian supplied, setup
ia     = nag_int(0);
ja     = nag_int(0);
njcpvt = nag_int(150);
nwkjac = nag_int(100);
eta    = 1.0e-4;
u      = 0.1;
sens   = 0.0;
lblock = true;

[jacpvt, rwork, ifail] = d02nu(...
    neq, neqmax, 'Analytical', nwkjac, ia, ja, ...
    njcpvt, sens, u, eta, lblock, rwork);

% Integration input variable intialization
t      = 0;
tout   = 10;
y      = [1; 0; 0];
ydot   = [0; 0; 0];
rtol   = [0.0001];
atol   = [1e-07];
itol   = nag_int(1);
inform = zeros(23, 1, nag_int_name);
ysave  = zeros(neq, sdysav);
```

```
wkjac  = zeros(nwkjac,1);
imon   = nag_int(0);
inln   = nag_int(0);
ires   = nag_int(1);
irevcm = nag_int(0);
lderiv = [false; false];
itask  = nag_int(3);
itrace = nag_int(0);

nfails = 0;

% pointers into rwork locations
lacorb = neqmax + 50;
lsavrb = lacorb + neqmax;
l1 = lsavrb+1; l2 = l1+1; l3 = l2+1;
m1 = lacorb+1; m2 = m1+1; m3 = m2+1;

fprintf('  Analytic Jacobian\n\n');
fprintf('     x           y_1          y_2           y_3\n');
fprintf(' %8.3f      %5.1f        %5.1f         %5.1f\n', t, y);
first_time = true;

% Main reverse communication loop controlled by irevcm

while (irevcm ~= 0 || first_time)
  first_time = false;

  [t, tout, y, ydot, rwork, inform, ysave, wkjac, ...
   jacpvt, imon, inln, ires, irevcm, lderiv, ifail] = ...
     d02nn(t, tout, y, ydot, rwork, rtol, atol, itol, inform, ...
     ysave, wkjac, jacpvt, imon, inln, ires, irevcm, lderiv, itask, itrace);

  if irevcm == 1 || irevcm == 3 || irevcm == 6 || irevcm == 11
    % Equivalent to resid evaluation in forward communication routines
    % ydot stored in ydot, resid returned in rwork(l1)
    rwork(l1) = -ydot(1) - ydot(2) - ydot(3);
    rwork(l2) = -ydot(2);
    rwork(l3) = -ydot(3);
    if (ires == 1)
      rwork(l1) =                                                rwork(l1);
      rwork(l2) = 0.04*y(1) - 1e4*y(2)*y(3) - 3e7*y(2)*y(2) + rwork(l2);
      rwork(l3) =                             3e7*y(2)*y(2) + rwork(l3);
    end
  elseif (irevcm == 2)
    % Equivalent to resid evaluation in forward communication routines
    % ydot stored in rwork(l1:), resid returned in rwork(l1)
    rwork(l1) = -rwork(l1) - rwork(l2) - rwork(l3);
    rwork(l2) = -rwork(l2);
    rwork(l3) = -rwork(l3);
  elseif (irevcm == 4 || irevcm == 7)
    % Equivalent to resid evaluation in forward communication routines
    % ydot stored in ydot, resid returned in rwork(m1)
    rwork(m1) = -ydot(1) - ydot(2) - ydot(3);
    rwork(m2) = -ydot(2);
    rwork(m3) = -ydot(3);
    if (ires == 1)
      rwork(m1) =                                                rwork(m1);
      rwork(m2) = 0.04*y(1) - 1e4*y(2)*y(3) - 3e7*y(2)*y(2) + rwork(m2);
      rwork(m3) =                             3e7*y(2)*y(2) + rwork(m3);
    end
  elseif (irevcm == 5)
    % Equivalent to resid evaluation in forward communication routines
    % ydot stored in rwork(l1:), resid returned in ydot
    ydot(1) = -rwork(l1) - rwork(l2) - rwork(l);
    ydot(2) = 0.04*y(1) - 1e4*y(2)*y(3) - 3e7*y(2)*y(2) - rwork(l2);
    ydot(3) =                           3e7*y(2)*y(2) - rwork(l3);
  elseif (irevcm == 8)
    % Equivalent to jac evaluation in forward communication routines
    [j, iplace] = d02nr(inform);
    hxd = rwork(16)*rwork(20);
    if (iplace < 2)
```

```
      if (j < 2)
        rwork(l1) = 1 - hxd*(0);
        rwork(l2) = 0 - hxd*(0.04);
        rwork(l3) = 0 - hxd*(0);
      elseif (j == 2)
        rwork(l1) = 1 - hxd*(0);
        rwork(l2) = 1 - hxd*(-1e4*y(3)-6e7*y(2));
        rwork(l3) = 0 - hxd*(6e7*y(2));
      elseif (j > 2)
        rwork(l1) = 1 - hxd*(0);
        rwork(l2) = 0 - hxd*(-1e4*y(2));
        rwork(l3) = 1 - hxd*(0);
      end
    else
      if (j < 2)
        rwork(m1) = 1 - hxd*(0);
        rwork(m2) = 0 - hxd*(0.04);
        rwork(m3) = 0 - hxd*(0);
      elseif (j == 2)
        rwork(m1) = 1 - hxd*(0);
        rwork(m2) = 1 - hxd*(-1e4*y(3)-6e7*y(2));
        rwork(m3) = 0 - hxd*(6e7*y(2));
      elseif (j > 2)
        rwork(m1) = 1 - hxd*(0);
        rwork(m2) = 0 - hxd*(-1e4*y(2));
        rwork(m3) = 1 - hxd*(0);
      end
    end
  elseif (irevcm == 10)
    % Step failure
    nfails = nfails + 1;
  end
end

[hu, h, tcur, tolsf, nst, nre, nje, nqu, nq, nit, imxer, algequ, ifail] = ...
    d02ny(neq, neqmax, rwork, inform);

[y, ifail] = d02mz(tout, neq, neq, ysave, rwork);

% Print solution and diagnostics
fprintf(' %8.3f      %5.1f        %5.1f         %5.1f\n', t, y);
fprintf('\nDiagnostic information\n integration status:\n');
fprintf('   last and next step sizes = %8.5f, %8.5f\n',hu, h);
fprintf('   integration stopped at x = %8.5f\n',tcur);
fprintf(' algorithm statistics:\n');
fprintf('   number of time-steps and Newton iterations  = %5d %5d\n',nst,nit);
fprintf('   number of residual and jacobian evaluations = %5d %5d\n',nre,nje);
fprintf('   order of method last used and next to use   = %5d %5d\n',nqu,nq);
fprintf('   component with largest error                = %5d\n',imxer);
fprintf('   number of failed steps                      = %5d\n\n',nfails);

icall = nag_int(0);
[liwreq, liwusd, lrwreq, lrwusd, nlu, nz, ngp, isplit, igrow, nblock] = ...
    d02nx(icall, true, inform);

fprintf(' sparse jacobian statistics:\n');
fprintf('   njcpvt - required = %3d, used = %3d\n',liwreq,liwusd);
fprintf('   nwkjac - required = %3d, used = %3d\n',lrwreq,lrwusd);
fprintf('   No. of LU-decomps = %3d, No. of nonzeros = %3d\n',nlu,nz);
fprintf(['   No. of function calls to form Jacobian = %3d, try ', ...
    'isplit = %3d\n'], ngp, isplit);
fprintf([' Growth estimate = %d, No. of blocks on diagonal %3d\n\n'], ...
    igrow, nblock);
```

9.2 Program Results

```
     d02nn example results

  Analytic Jacobian

     x           y_1          y_2          y_3
    0.000        1.0          0.0          0.0
 Warning: Equation(=i1) and possibly other equations are
 implicit and in calculating the initial values the
 equations will be treated as implicit.
 In above message i1 =          1
   10.767        0.8          0.0          0.2

Diagnostic information
 integration status:
   last and next step sizes =  0.90181,  0.90181
   integration stopped at x = 10.76669
 algorithm statistics:
   number of time-steps and Newton iterations  =    55    80
   number of residual and jacobian evaluations =    89    17
   order of method last used and next to use   =     4     4
   component with largest error                =     3
   number of failed steps                      =     4

 sparse jacobian statistics:
   njcpvt - required =  99, used = 150
   nwkjac - required =  31, used =  75
   No. of LU-decomps =  17, No. of nonzeros =   8
   No. of function calls to form Jacobian =   0, try isplit =  73
 Growth estimate = 1531, No. of blocks on diagonal   1
```