NAG Toolbox

nag_ode_ivp_stiff_imp_fulljac (d02ng)

1 Purpose

nag_ode_ivp_stiff_imp_fulljac (d02ng) is a forward communication function for integrating stiff systems of implicit ordinary differential equations coupled with algebraic equations when the Jacobian is a full matrix.

2 Syntax

```
[t, tout, y, ydot, rwork, inform, ysav, wkjac, lderiv, ifail] =
nag_ode_ivp_stiff_imp_fulljac(t, tout, y, ydot, rwork, rtol, atol, itol, inform,
resid, ysav, jac, wkjac, monitr, lderiv, itask, itrace, 'neq', neq, 'sdysav',
sdysav)
[t, tout, y, ydot, rwork, inform, ysav, wkjac, lderiv, ifail] = d02ng(t, tout,
y, ydot, rwork, rtol, atol, itol, inform, resid, ysav, jac, wkjac, monitr,
lderiv, itask, itrace, 'neq', neq, 'sdysav', sdysav)
```

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

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

3 Description

nag_ode_ivp_stiff_imp_fulljac (d02ng) 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 full matrix (see the description of **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 nag_ode_ivp_stiff_imp_fulljac (d02ng) is given below. It calls the full matrix linear algebra setup function nag_ode_ivp_stiff_fulljac_setup (d02ns), the Backward Differentiation Formula (BDF) integrator setup function nag_ode_ivp_stiff_bdf (d02nv), and its diagnostic counterpart nag_ode_ivp_stiff_integ_diag (d02ny).

```
.
.
.
[...] = d02nv(...);
[...] = d02ns(...);
[..., ifail] = d02ng(...);
if (ifail ~= 1 and ifail < 14)
  [...] = d02ny(...);
end
.
.
.
```

The linear algebra setup function nag_ode_ivp_stiff_fulljac_setup (d02ns) and one of the integrator setup functions, nag_ode_ivp_stiff_dassl (d02mv), nag_ode_ivp_stiff_bdf (d02nv) or nag_ode_ivp_stiff_blend (d02nw), must be called prior to the call of nag_ode_ivp_stiff_imp_fulljac (d02ng). The integrator diagnostic function nag_ode_ivp_stiff_integ_diag (d02ny) may be called after the call to

nag_ode_ivp_stiff_imp_fulljac (d02ng). There is also a function, nag_ode_ivp_stiff_contin (d02nz), designed to permit you to change step size on a continuation call to nag_ode_ivp_stiff_imp_fulljac (d02ng) without restarting the integration process.

4 References

See the D02M–N Sub-chapter Introduction.

5 Parameters

5.1 Compulsory Input Parameters

1: **t** – REAL (KIND=nag_wp)

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)

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

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

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.

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

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

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

The absolute local error tolerance.

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

8: **itol** – INTEGER

A value to indicate the form of the local error test. **itol** indicates to nag_ode_ivp_stiff_imp_fulljac (d02ng) 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	$\mathbf{rtol}(i) \times \lvert y_i \rvert + \mathbf{atol}(1)$
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: **itol** $= 1$, 2, 3 or 4.

9: **inform**(**23**) – INTEGER array

10: **resid** – SUBROUTINE, supplied by the user.

resid must evaluate the residual

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

in one case and

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

in another.

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

Input Parameters

1: **neq** – INTEGER

The number of equations being solved.

2: **t** – REAL (KIND=nag_wp)

t, the current value of the independent variable.

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

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

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

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

5: **ires** – INTEGER

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

ires $= -1$
The residual defined in equation (2) must be returned.

ires $= 1$
The residual defined in equation (1) must be returned.

Output Parameters

1: **r**(**neq**) – REAL (KIND=nag_wp) array

r(i) must contain the ith component of r, 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** – INTEGER

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 **monitr** immediately with argument **imon** $= -2$.

11: **ysav**$(ldysav, \mathbf{sdysav})$ – REAL (KIND=nag_wp) array

$ldysav$, the first dimension of the array, must satisfy the constraint $ldysav \geq$ **neq**.

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.

12: **jac** – SUBROUTINE, supplied by the NAG Library or the user.

jac must evaluate the Jacobian of the system. If this option is not required, the actual argument for **jac** must be the string `nag_ode_ivp_stiff_imp_fulljac_dummy_jac (d02ngz)`. (nag_ode_ivp_stiff_imp_fulljac_dummy_jac (d02ngz) is included in the NAG Toolbox.) You must indicate to the integrator whether this option is to be used by setting the argument **jceval** appropriately in a call to the full linear algebra setup function nag_ode_ivp_stiff_fulljac_setup (d02ns).

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 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 it 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:

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

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

Input Parameters

1: **neq** – INTEGER

The number of equations being solved.

2: **t** – REAL (KIND=nag_wp)

t, the current value of the independent variable.

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

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

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

The derivative of the solution at the current point t.

5: **h** – REAL (KIND=nag_wp)

The current step size.

6: **d** – REAL (KIND=nag_wp)

The argument d which depends on the integration method.

7: **p**(**neq**, **neq**) – REAL (KIND=nag_wp) array

Is set to zero.

Output Parameters

1: **p**(**neq**, **neq**) – REAL (KIND=nag_wp) array

p(i, j) must contain $\dfrac{\partial r_i}{\partial y_j}$, for $i = 1, 2, \ldots,$ **neq** and $j = 1, 2, \ldots,$ **neq**.

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

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

$nwkjac$, the dimension of the array, must satisfy the constraint $nwkjac \geq ldysav \times (ldysav + 1)$.

This value must be the same as that supplied to the linear algebra setup function nag_ode_ivp_stiff_fulljac_setup (d02ns).

Constraint: $nwkjac \geq ldysav \times (ldysav + 1)$.

14: **monitr** – SUBROUTINE, supplied by the NAG Library or the user.

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

```
[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** – INTEGER

The number of equations being solved.

2: **ldysav** – INTEGER

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

3: **t** – REAL (KIND=nag_wp)

The current value of the independent variable.

4: **hlast** – REAL (KIND=nag_wp)

The last step size successfully used by the integrator.

5: **hnext** – REAL (KIND=nag_wp)

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

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

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

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

The time derivatives y' of the vector y.

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

Workspace to enable you to carry out interpolation using either of the functions nag_ode_ivp_stiff_nat_interp (d02xj) or nag_ode_ivp_stiff_c1_interp (d02xk).

9: **r**(**neq**) – REAL (KIND=nag_wp) array

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

10: **acor**(**neq**, **2**) – REAL (KIND=nag_wp) array

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 nag_ode_ivp_stiff_errest (d02za) as follows:

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

11: **imon** – INTEGER

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

imon $= -2$
Entry from the integrator after **ires** $= 4$ (set in **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.

12: **hmin** – REAL (KIND=nag_wp)

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

13: **hmax** – REAL (KIND=nag_wp)

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

14: **nqu** – INTEGER

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 nag_ode_ivp_stiff_nat_interp (d02xj) or nag_ode_ivp_stiff_c1_interp (d02xk).

Output Parameters

1: **hnext** – REAL (KIND=nag_wp)

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

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

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

3: **imon** – INTEGER

May be reset to determine subsequent action in nag_ode_ivp_stiff_imp_fulljac (d02ng).

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** $\neq -1$ on exit.

imon $= 0$
Return to the internal nonlinear equation solver, where the action taken is determined by the value of **inln** (see **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 solution **y**, provided by **monitr**, 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 of **hnext** and possibly new values of **hmin** and **hmax**.

4: **inln** – INTEGER

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** – REAL (KIND=nag_wp)

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** – REAL (KIND=nag_wp)

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.

15: **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.

16: **itask** – INTEGER

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** 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 nag_ode_ivp_stiff_imp_fulljac (d02ng) is recalled with a different value of **itask** (and **tout** altered), then the initialization procedure is repeated, possibly leading to different initial conditions.

Constraint: $1 \leq \mathbf{itask} \leq 6$.

17: **itrace** – INTEGER

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.)

The number of differential equations to be solved.

Constraint: **neq** ≥ 1.

2: **sdysav** – INTEGER

Default: 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.

5.3 Output Parameters

1: **t** – REAL (KIND=nag_wp)

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

2: **tout** – REAL (KIND=nag_wp)

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**(**neq**) – REAL (KIND=nag_wp) array

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

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

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

5: **rwork**$(\mathbf{50} + \mathbf{4} \times \mathbf{neq})$ – REAL (KIND=nag_wp) array

6: **inform**$(\mathbf{23})$ – INTEGER array

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

Communication array, used to store information between calls to nag_ode_ivp_stiff_imp_fulljac (d02ng).

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

$nwkjac = ldysav \times (ldysav + 1)$.

Communication array, used to store information between calls to nag_ode_ivp_stiff_imp_fulljac (d02ng).

9: **lderiv**$(\mathbf{2})$ – 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**$(2) = false$.

10: **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 $= 1$

An illegal input was detected on entry, or after an internal call to **monitr**. If **itrace** > -1, then 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 $= 3$

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 $= 7$

resid set its error flag (**ires** $= 3$) continually despite repeated attempts by the integrator to avoid this.

ifail $= 8$

lderiv$(1) = \mathit{false}$ 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*)

resid signalled the integrator to halt the integration and return (**ires** $= 2$). Integration was successful as far as **t**.

ifail $= 12$ (*warning*)

monitr 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_fulljac (d02ng) is unable to start the integration.

ifail $= 15$

The linear algebra setup function nag_ode_ivp_stiff_fulljac_setup (d02ns) was not called before the call to nag_ode_ivp_stiff_imp_fulljac (d02ng).

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. For nag_ode_ivp_stiff_imp_fulljac (d02ng) the cost is proportional to **neq**^{3}, though for problems which are only mildly nonlinear the cost may be dominated by factors proportional to **neq**^{2} except for very large problems.

In general, you are advised to choose the BDF 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

This example solves the well-known stiff Robertson problem written in implicit form

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

with initial conditions $a = 1.0$ and $b = c = 0.0$ over the range $[0, 0.1]$ with vector error control (**itol** $= 4$), the BDF method (setup function nag_ode_ivp_stiff_bdf (d02nv)) and functional iteration. The Jacobian is calculated numerically if the functional iteration encounters difficulty and the integration is in one-step mode (**itask** $= 2$), with C^0 interpolation to calculate the solution at intervals of 0.02 using nag_ode_ivp_stiff_nat_interp (d02xj) externally. nag_ode_ivp_stiff_exp_fulljac_dummy_monit (d02nby) is used for **monitr**.

9.1 Program Text

```
function d02ng_example

fprintf('d02ng example results\n\n');

% switch for diagnostics output
istat  = 1;

% Initialize setup variables and arrays.
neq    = nag_int(3);
neqmax = neq;
maxord = nag_int(5);
sdysav = maxord+1;
petzld = false;
tcrit  = 0;
hmin   = 1.0e-10;
hmax   = 10;
h0     = 0;
maxstp = nag_int(200);
mxhnil = nag_int(5);

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

% d02nv is a setup routine to be called prior to d02ng.
[const, rwork, ifail] = ...
d02nv( ...
```

```
       neqmax, sdysav, maxord, 'Functional', petzld, ...
       const, tcrit, hmin, hmax, h0, maxstp, mxhnil, 'Average-L2', rwork);

nwkjac = nag_int(neq*(neq+1));
% Numerical Jacobian.
[rwork, ifail] = d02ns( ...
                       neq, neqmax, 'Numerical', nwkjac, rwork);

% Initialize integration variables and arrays

sol   = zeros(neq, 1);
wkjac = zeros(nwkjac, 1);
ysave = zeros(neq, sdysav);
ydot  = zeros(neq, 1);
inform(1:23) = nag_int(0);
algequ = false(neq, 1);

% Integrate to tout by overshooting tout in one step mode (itask=2);
% use BDF with functional iteration; use vector tolerances (itol=4);
% dummy d02nbz and d02nby are used for Jacobian and monitor functions.
t    = 0;
tout = 0.1;
itask  = nag_int(2);
itrace = nag_int(0);
y      = [1; 0; 0];
lderiv = [false; false];
itol = nag_int(4);
rtol = [0.0001; 0.001; 0.0001];
atol = [1e-07; 1e-08; 1e-07];

% Output header and initial results.
fprintf('    x             y_1            y_2            y_3\n');
fprintf('%8.3f       %8.4f       %8.6f       %8.4f\n',t,y);

% Prepare to store results for plotting.
ncall = 1;
tkeep = t;
ykeep = y;
tstep = 0.02;

% xout is intermediate output points for printing solution.
iout = 1;
xout = iout*tstep;

while t < tout
  % Calculate solution at this point.
  [t, tout, y, ydot, rwork, inform, ysave, wkjac, lderiv, ifail] = ...
  d02ng( ...
         t, tout, y, ydot, rwork, rtol, atol, itol, inform, @resid, ...
         ysave, 'd02ngz', wkjac, 'd02nby', lderiv, itask, itrace);
  if (t<tstep)
    ncall = ncall+1;
    ykeep(:,ncall) = y;
    tkeep(ncall) = t;
  elseif (xout <= t)
    % Passed over intermediate output point (xout)
    % Interpolate the solution to get its value at xout.
    [hu, h, tcur, tolsf, nst, nre, nje, nqu, nq, niter, imxer, algequ, ...
     ifail] = d02ny( ...
                     neq, neqmax, rwork, inform);
    [sol, ifail] = d02xj( ...
                          xout, neq, ysave, neq, tcur, ...
                          nqu, hu, h, 'sdysav', sdysav);

    % Accumulate results for plotting.
    ncall = ncall + 1;
    ykeep(:,ncall) = sol;
    tkeep(ncall) = xout;
    % output intermediate results
    fprintf('%8.3f       %8.4f       %8.6f       %8.4f\n',xout,sol);
```

```
    % Bump the counter for the next output point.
    iout = iout + 1;
    xout = iout*tstep;
  end
end

% Output some statistics, if required.
if istat == 1
  [hu, h, tcur, tolsf, nst, nre, nje, nqu, nq, niter, imxer, algequ, ifail] ...
  = d02ny( ...
           neq, neqmax, rwork, inform);
  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,niter);
  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);
end
% Plot results.
fig1 = figure;
display_plot(tkeep, ykeep)

function [r, ires] = resid(neq, t, y, ydot, ires)
  % Evaluate the residual.
  r(1:3) = -ydot(1:3);
  if ires == 1
    r(1) = -0.04*y(1) + 1.0e4*y(2)*y(3) + r(1);
    r(2) =  0.04*y(1) - 1.0e4*y(2)*y(3) - 3.0e7*y(2)*y(2) + r(2);
    r(3) =  3.0e7*y(2)*y(2) + r(3);
  end

function display_plot(x,y)
  % Formatting for title and axis labels.
  % Plot one of the curves and then add the other two.
  hline3 = plot(x, y(3,:));
  hold on;
  [haxes, hline1, hline2] = plotyy(x, 100*y(2,:), x, y(1,:));
  % Set the axis limits and the tick specifications to beautify the plot.
  set(haxes(1), 'YLim', [0.0 0.0045]);
  set(haxes(1), 'XMinorTick', 'on', 'YMinorTick', 'on');
  set(haxes(1), 'YTick', [0:0.001:0.004]);
  set(haxes(2), 'YLim', [0.995 1.006]);
  set(haxes(2), 'YMinorTick', 'on');
  set(haxes(2), 'YTick', [0.995:0.002:1.005]);
  set(haxes(1), 'XLim', [-0.005 0.1]);
  set(haxes(2), 'XLim', [-0.005 0.1]);
  set(gca, 'box', 'off'); % no ticks on opposite axes.
  % Add title.
  ht = title({'Stiff Robertson Problem:','BDF and Functional Iteration'});
  set(ht,'Position',[0.05,0.004,0.0]);
  % Label the x axis, and both y axes.
  xlabel('x');
  ylabel(haxes(1), 'Solution (100*b,c)');
  ylabel(haxes(2), 'Solution (a)');
  % Add a legend.
  legend('c', '100*b', 'a', 'Location', 'Best');
  % Set some features of the three lines
  set(hline1,'Linewidth',0.5,'Marker','+','LineStyle','-','Color','red');
  set(hline2,'Linewidth',0.5,'Marker','*','LineStyle',':','Color','blue');
  set(hline3,'Linewidth',0.5,'Marker','x','LineStyle','--','Color','green');
  hold off;
```

9.2 Program Results

```
     d02ng example results

      x             y_1            y_2            y_3
    0.000          1.0000       0.000000         0.0000
    0.020          0.9992       0.000036         0.0008
    0.040          0.9984       0.000036         0.0016
    0.060          0.9976       0.000036         0.0023
    0.080          0.9969       0.000036         0.0031
    0.100          0.9961       0.000036         0.0039

Diagnostic information
 integration status:
   last and next step sizes =  0.00027,  0.00053
   integration stopped at x =  0.10016
 algorithm statistics:
   number of time-steps and Newton iterations  =   244     0
   number of residual and jacobian evaluations =   809     0
   order of method last used and next to use   =     1     1
   component with largest error                =     2
```