NAG Toolbox

nag_ode_ivp_stiff_exp_sparjac (d02nd)

1 Purpose

nag_ode_ivp_stiff_exp_sparjac (d02nd) is a forward communication function for integrating stiff systems of explicit ordinary differential equations when the Jacobian is a sparse matrix.

2 Syntax

```
[t, y, ydot, rwork, inform, ysav, wkjac, jacpvt, ifail] =
nag_ode_ivp_stiff_exp_sparjac(t, tout, y, rwork, rtol, atol, itol, inform, fcn,
ysav, jac, wkjac, jacpvt, monitr, itask, itrace, 'neq', neq, 'sdysav', sdysav)
[t, y, ydot, rwork, inform, ysav, wkjac, jacpvt, ifail] = d02nd(t, tout, y,
rwork, rtol, atol, itol, inform, fcn, ysav, jac, wkjac, jacpvt, monitr, itask,
itrace, 'neq', neq, 'sdysav', sdysav)
```

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

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

3 Description

nag_ode_ivp_stiff_exp_sparjac (d02nd) is a general purpose function for integrating the initial value problem for a stiff system of explicit ordinary differential equations,

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

It is designed specifically for the case where the Jacobian $\dfrac{\partial g}{\partial y}$ is a sparse matrix.

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_exp_sparjac (d02nd) is given below. It calls the sparse matrix linear algebra setup function nag_ode_ivp_stiff_sparjac_setup (d02nu), the Backward Differentiation Formula (BDF) integrator setup function nag_ode_ivp_stiff_bdf (d02nv), its diagnostic counterpart nag_ode_ivp_stiff_integ_diag (d02ny), and the sparse linear algebra diagnostic function nag_ode_ivp_stiff_sparjac_diag (d02nx).

```
.
.
.
[...] = d02nv(...);
[...] = d02nu(...);
[..., ifail] = d02nd(...);
if (ifail ~= 1 and ifail < 14)
  [...] = d02nx(...);
  [...] = d02ny(...);
end
.
.
.
```

The linear algebra setup function nag_ode_ivp_stiff_sparjac_setup (d02nu) and one of the integrator setup functions, 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_exp_sparjac (d02nd). Either or both of the integrator diagnostic function nag_ode_ivp_stiff_integ_diag (d02ny), or the sparse matrix linear algebra diagnostic function nag_ode_ivp_stiff_sparjac_diag (d02nx), may be called after the call to nag_ode_ivp_stiff_exp_sparjac

(d02nd). 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_exp_sparjac (d02nd) 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: **rwork**($\mathbf{50} + \mathbf{4} \times \mathbf{neq}$) – REAL (KIND=nag_wp) array

5: **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**).

6: **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**).

7: **itol** – INTEGER

A value to indicate the form of the local error test. **itol** indicates to nag_ode_ivp_stiff_exp_sparjac (d02nd) 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	**rtol**$(i) \times \lvert y_i \rvert +$ **atol**(1)
4	vector	vector	**rtol**$(i) \times \lvert y_i \rvert +$ **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.

8: **inform**(**23**) – INTEGER array

9: **fcn** – SUBROUTINE, supplied by the user.

fcn must evaluate the derivative vector for the explicit ordinary differential equation system, defined by $y' = g(t, y)$.

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

Input Parameters

1: **neq** – INTEGER

The number of differential 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: **ires** – INTEGER

ires $= 1$.

Output Parameters

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

The value y'_i, given by $y'_i = g_i(t, y)$, for $i = 1, 2, \ldots,$ **neq**.

2: **ires** – INTEGER

You may set **ires** as follows to indicate certain conditions in **fcn** to the integrator:

ires $= 1$
: Indicates a normal return from **fcn**, that is **ires** has not been altered by you and integration continues.

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** $= -2$.

10: **ysav**($ldysav$, **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 specification of the integrator setup functions 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.

11: **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_exp_sparjac_dummy_jac (d02ndz)`. (nag_ode_ivp_stiff_exp_sparjac_dummy_jac (d02ndz) 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 sparse linear algebra setup function nag_ode_ivp_stiff_sparjac_setup (d02nu).

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

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

but it is solved in the form

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

where r is the function defined by

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

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

$$\frac{\partial r_i}{\partial y_j} = 1 - (hd)\frac{\partial g_i}{\partial y_j}, \quad \text{if } i = j,$$

$$\frac{\partial r_i}{\partial y_j} = -(hd)\frac{\partial g_i}{\partial y_j}, \qquad \text{otherwise.}$$

```
[pdj] = jac(neq, t, y, h, d, j, pdj)
```

Input Parameters

1: **neq** – INTEGER

The number of differential 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: **h** – REAL (KIND=nag_wp)

The current step size.

5: **d** – REAL (KIND=nag_wp)

The argument d which depends on the integration method.

6: **j** – INTEGER

The column of the Jacobian that **jac** must return in the array **pdj**.

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

Is set to zero.

Output Parameters

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

pdj(i) should be set to the (i,j)th element of the Jacobian, where j is given by **j**. Only nonzero elements of this array need be set, since it is preset to zero before the call to **jac**.

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

The actual size depends on whether the sparsity structure is supplied or whether it is to be estimated. An appropriate value for $nwkjac$ is described in the specification of the linear algebra setup function nag_ode_ivp_stiff_sparjac_setup (d02nu). This value must be the same as that supplied to nag_ode_ivp_stiff_sparjac_setup (d02nu).

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

The actual size depends on whether the sparsity structure is supplied or whether it is to be estimated. An appropriate value for $njcpvt$ is described in the specification of the linear algebra setup function nag_ode_ivp_stiff_sparjac_setup (d02nu). This value must be the same as that supplied to nag_ode_ivp_stiff_sparjac_setup (d02nu).

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 differential equations being solved.

2: **ldysav** – INTEGER

An upper bound on the number of differential 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** $= 3$, the first **neq** elements contain the residual vector, $y' - g(t, y)$.

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 **fcn**) 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_exp_sparjac (d02nd).

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** (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: **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** (e.g., see nag_ode_ivp_stiff_dassl (d02mv)). **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** (e.g., see nag_ode_ivp_stiff_dassl (d02mv)). **tcrit** must be specified as under **itask** $= 4$.

Constraint: **itask** $= 1$, 2, 3, 4 or 5.

16: **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 array **y** 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 specification of the integrator setup functions 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: **y**(**neq**) – REAL (KIND=nag_wp) array

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

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

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

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

5: **inform**(**23**) – INTEGER array

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

Communication array, used to store information between calls to nag_ode_ivp_stiff_exp_sparjac (d02nd).

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

Communication array, used to store information between calls to nag_ode_ivp_stiff_exp_sparjac (d02nd).

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

Communication array, used to store information between calls to nag_ode_ivp_stiff_exp_sparjac (d02nd).

9: **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 **y**(1), **y**(2), . . . , **y**(**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 $= 7$

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

ifail $= 8$

Not used for the integrator.

ifail $= 9$

A singular Jacobian $\dfrac{\partial r}{\partial y}$ has been encountered. This error exit is unlikely to be taken when solving explicit ordinary differential equations. 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*)

fcn 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_exp_sparjac (d02nd) is unable to start the integration.

ifail $= 15$

The linear algebra setup function nag_ode_ivp_stiff_sparjac_setup (d02nu) was not called prior to calling nag_ode_ivp_stiff_exp_sparjac (d02nd).

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

Since numerical stability and memory are often conflicting requirements when solving ordinary differential systems where the Jacobian matrix is sparse, we provide a diagnostic function, nag_ode_ivp_stiff_sparjac_diag (d02nx), whose aim is to inform you how much memory is required to solve the problem and to give you some indication of numerical stability.

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 $\dfrac{\partial g}{\partial y}$ 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

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

over the range $[0, 10.0]$ with initial conditions $a = 1.0$ and $b = c = 0.0$ using scalar error control (**itol** $= 1$). The solution is computed up to 10.0 by overshooting and interpolating (**itask** $= 1$) and the intermediate solution computed on an equispaced mesh through **monitr**. The integration algorithm used is the BDF method (setup function nag_ode_ivp_stiff_bdf (d02nv)) and a modified Newton method is also used. The use of the `N' (Numerical) and `S' (Structural) options are illustrated in turn for calculating the Jacobian.

9.1 Program Text

```
function d02nd_example

fprintf('d02nd example results\n\n');

% For communication with monitr2.
global ncall ykeep tkeep tout

% For communication with interp.
global xout rwork

% Initialize setup variables and arrays.
neq    = nag_int(3);
neqmax = nag_int(neq);
```

```
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);

% d02nv is a setup routine to be called prior to d02nd.
[const, rwork, ifail] = ...
d02nv( ...
       neqmax, sdysav, maxord, 'Newton', petzld, ...
       const, tcrit, hmin, hmax, h0, maxstp, mxhnil, 'Average-L2', rwork);

nelts = 8;
nia = nag_int(neq+1);
nja = nag_int(nelts);
ia  = nag_int([1; 3; 6; 9]);
ja  = nag_int([1; 2; 1; 2; 3; 1; 2; 3]);
njcpvt = nag_int(150);
nwkjac = nag_int(50);
eta    = 1.0e-4;
u      = 0.1;
sens   = 0.0;
lblock = true;
% d02nu is a setup routine for sparse Jacobian to be called prior to d02nd.
[jacpvt, rwork, ifail] = ...
d02nu( ...
       neq, neqmax, 'Numerical', nwkjac, ia, ja, ...
       njcpvt, sens, u, eta, lblock, rwork, 'nia', nia, 'nja', nja);

% Initialize integration variables and arrays.
inform(1:23) = nag_int(0);
wkjac = zeros(nwkjac, 1);
ysave = zeros(neq, sdysav);

% First case.
% Integrate to tout by overshooting (itask=1) using using  BDF with Newton.
% Scalar relative and absolute tolerances.
% Numerical Jacobian.
t      = 0;
tout   = 10.0;
itask  = nag_int(1);
itrace = nag_int(0);
y      = [1; 0; 0];
itol   = nag_int(1);
rtol   = [1e-4];
atol   = [1e-7];
isplit = nag_int(0);

% Output initial condition.
fprintf('Numerical Jacobian, structure not supplied\n\n');
fprintf('    x           y_1            y_2            y_3\n');
fprintf('%8.3f      %8.5f      %8.5f      %8.5f\n',t,y);
xout = 2.0;

% Numerical Jacobian (d02ndz) with monitoring (monitr1).
[t, y, ydot, rwork, inform, ysave, wkjac, jacpvt,ifail] = ...
d02nd( ...
       t, tout, y, rwork, rtol, atol, itol, inform, @fcn, ysave, ...
       'd02ndz', wkjac, jacpvt, @monitr1, itask, itrace);

% d02ny is an integrator diagnostic routine which can be called after d02nd.
[hu, h, tcur, tolsf, nst, nre, nje, nqu, nq, niter, imxer, algequ, ifail] ...
= d02ny( ...
```

```
         neq, neqmax, rwork, inform);

% Output diagnostics.
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);

% d02nx is an optional output routine which can be called after d02nd.
icall = nag_int(0);
[liwreq, liwusd, lrwreq, lrwusd, nlu, nz, ngp, isplit, igrow, nblock] ...
= d02nx( ...
         icall, lblock, inform);

% Output diagnostics.
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);

% Second case.
% As first case but using known structure of Jacobian.
t      = 0;
y      = [1; 0; 0];
isplit = nag_int(0);

% Prepare to store results for plotting.  This gets done in monitr2.
ncall = nag_int(1);
tkeep = t;
ykeep = y;

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

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

fprintf('Numerical Jacobian, structure supplied\n\n');
fprintf('    x           y_1            y_2            y_3\n');
fprintf('%8.3f      %8.5f      %8.5f      %8.5f\n',t,y);
xout = 2.0;

[t, y, ydot, rwork, inform, ysave, wkjac, jacpvt,ifail] = ...
d02nd( ...
       t, tout, y, rwork, rtol, atol, itol, inform, @fcn, ysave, ...
       'd02ndz', wkjac, jacpvt, @monitr2, itask, itrace);

[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);
```

```
[liwreq, liwusd, lrwreq, lrwusd, nlu, nz, ngp, isplit, igrow, nblock] = ...
d02nx( ...
       icall, lblock, 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);
% Plot results.
fig1 = figure;
display_plot(tkeep,ykeep)

function [hnext, y, imon, inln, hmin, hmax] = ...
         monitr1(neq, neqmax, t, hlast, hnext, y, ydot, ysave, ...
                 r, acor, imon, hmin, hmax, nqu)
  % Print out solution at intermediate point if passed at this step.
  [imon] = interp(neq, t, hlast, hnext, ysave, imon, nqu);
  inln = nag_int(0);

function [hnext, y, imon, inln, hmin, hmax] = ...
         monitr2( neq, neqmax, t, hlast, hnext, y, ydot, ysave, ...
                  r, acor, imon, hmin, hmax, nqu)
  % Like monitr1 but also stores the current results for plotting later.
  % For communication with main routine.
  global ncall ykeep tkeep tout
  if (imon == 1 && t>0.00001 && t<tout)
    ncall = ncall + 1;
    ykeep(:,ncall) = y;
    tkeep (ncall) = t;
  end
  [imon] = interp(neq, t, hlast, hnext, ysave, imon, nqu);
  inln = nag_int(0);

function [imon] = interp(neq, t, hlast, hnext, ysave, imon, nqu)
  % This routine prints out intermediate values by interpolating.
  % For communication with main routine.
  global xout rwork
  dims = size(ysave);
  lbeg = dims(1) + 50 + 1;
  lend = lbeg + neq - 1;
  lw(1:neq) = rwork(lbeg:lend);
  % Print intermediate result if integration has passed it at this step.
  while (xout <= 10.0 && t-hlast < xout && xout <= t)
    % d02xk interpolates components of the solution of a system of ODEs,
    % as provided by (e.g.) d02nd.
    [sol, ifail] = d02xk(xout, neq, ysave, lw, t, nqu, hlast, hnext);
    fprintf('%8.3f %13.5f %13.5f %13.5f\n', xout, sol(1:3));
    xout = xout + 2;
  end

function [f, ires] = fcn(neq, t, y, ires)
  % Evaluate derivative vector.
  f = zeros(3,1);
  f(1) = -0.04d0*y(1) + 1.0d4*y(2)*y(3);
  f(2) = 0.04d0*y(1) - 1.0d4*y(2)*y(3) - 3.0d7*y(2)*y(2);
  f(3) = 3.0d7*y(2)*y(2);

function display_plot(tkeep,ykeep)
  % Plot one of the curves and then add the other two.
  hline3 = plot(tkeep,ykeep(3,:));
  hold on;
  [haxes, hline1, hline2] = plotyy(tkeep,ykeep(1,:), tkeep,ykeep(2,:));
  % Set the axis limits and the tick specifications to beautify the plot.
  set(haxes(1), 'YLim', [0 1.2]);
  set(haxes(1), 'XMinorTick', 'on', 'YMinorTick', 'on');
  set(haxes(1), 'YTick', [0:0.2:1]);
```

```
  set(haxes(2), 'YLim', [0 4e-5]);
  set(haxes(2), 'YMinorTick', 'on');
  set(haxes(2), 'YTick', [5e-6:5e-6:3.5e-5]);
  set(haxes(1), 'XLim', [-0.1 10]);
  set(haxes(2), 'XLim', [-0.1 10]);
  % Add title.
  ht = title({'Stiff Robertson Problem: BDF Method', ...
              'Jacobian: Sparse with Structure Supplied'});
  set(ht,'Position',[5,1.05,0.0]);
  % Label the x axis, and both y axes.
  xlabel('x');
  ylabel(haxes(1),'Solution (a,c)');
  ylabel(haxes(2),'Solution (b)');
  % Add a legend.
  legend('c','a','b','Location','West');
  % Set some features of the 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

```
     d02nd example results

Numerical Jacobian, structure not supplied

    x           y_1           y_2           y_3
   0.000       1.00000       0.00000       0.00000
   2.000       0.94161       0.00003       0.05836
   4.000       0.90552       0.00002       0.09446
   6.000       0.87927       0.00002       0.12072
   8.000       0.85855       0.00002       0.14144
  10.000       0.84137       0.00002       0.15863

Diagnostic information
 integration status:
   last and next step sizes =  0.90236,  0.90236
   integration stopped at x = 10.76936
 algorithm statistics:
   number of time-steps and Newton iterations  =    55    78
   number of residual and jacobian evaluations =   136    16
   order of method last used and next to use   =     4     4
   component with largest error                =     3
 sparse jacobian statistics:
   njcpvt - required = 100, used = 150
   nwkjac - required =  29, used =  50)
   No. of LU-decomps =  16, No. of nonzeros =   7
   No. of function calls to form Jacobian =   3, try isplit =  73
 Growth estimate = 1108, No. of blocks on diagonal   1

Numerical Jacobian, structure supplied

    x           y_1           y_2           y_3
   0.000       1.00000       0.00000       0.00000
   2.000       0.94161       0.00003       0.05837
   4.000       0.90551       0.00002       0.09446
   6.000       0.87926       0.00002       0.12072
   8.000       0.85854       0.00002       0.14144
  10.000       0.84135       0.00002       0.15863

Diagnostic information
 integration status:
   last and next step sizes =  0.90178,  0.90178
   integration stopped at x = 10.76621
 algorithm statistics:
   number of time-steps and Newton iterations  =    55    78
   number of residual and jacobian evaluations =   129    16
   order of method last used and next to use   =     4     4
   component with largest error                =     3
```

```
sparse jacobian statistics:
  njcpvt - required = 106, used = 150
  nwkjac - required =  31, used =  50
  No. of LU-decomps =  16, No. of nonzeros =   8
  No. of function calls to form Jacobian =   3, try isplit =  73
Growth estimate = 277504, No. of blocks on diagonal   1
```