# NAG Toolbox

# nag_ode_ivp_stiff_c1_interp (d02xk)

## 1 Purpose

nag_ode_ivp_stiff_c1_interp (d02xk) interpolates components of the solution of a system of first-order ordinary differential equations from information provided by the integrators in Sub-chapter D02M–N. It provides $C^1$ interpolation suitable for general use.

## 2 Syntax

```
[sol, ifail] = nag_ode_ivp_stiff_c1_interp(xsol, m, ysav, acor, x, nqu, hu, h, 'sdysav', sdysav, 'neq', neq)
[sol, ifail] = d02xk(xsol, m, ysav, acor, x, nqu, hu, h, 'sdysav', sdysav, 'neq', neq)
```

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

At Mark 22: **neq** was made optional.

## 3 Description

nag_ode_ivp_stiff_c1_interp (d02xk) evaluates the first $m$ components of the solution of a system of ordinary differential equations at any point using $C^1$ polynomial interpolation based on information generated by the integrator. This information must be passed unchanged to nag_ode_ivp_stiff_c1_interp (d02xk). nag_ode_ivp_stiff_c1_interp (d02xk) should not normally be used to extrapolate outside the range of values obtained from the above functions.

It may be used with the D02N functions only when the BDF integration method is being employed (setup function nag_ode_ivp_stiff_bdf (d02nv)), provided the Petzold error test was not selected.

## 4 References

None.

## 5 Parameters

### 5.1 Compulsory Input Parameters

1: **xsol** – REAL (KIND=nag_wp)

The point at which the first $m$ components of the solution are to be evaluated. **xsol** should not be an extrapolation point, that is **xsol** should satisfy $(\mathbf{xsol} - \mathbf{x}) \times \mathbf{hu} \leq 0.0$. Extrapolation is permitted but not recommended.

2: **m** – INTEGER

The number of components of the solution whose values at **xsol** are required. The first $m$ components are evaluated.

*Constraint*: $1 \leq \mathbf{m} \leq \mathbf{neq}$.

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

$ldysav$, the first dimension of the array, must satisfy the constraint $ldysav \geq 1$.

The values provided in the argument **ysav** on return from the integrator.

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

The value returned in position $(ldysav + 50 + i)$, for $i = 1, 2, \ldots,$ **neq**, of the argument **rwork** returned by the integrator. If one of the forward communication D02N functions is being employed and nag_ode_ivp_stiff_c1_interp (d02xk) is to be used in **monitr**, then **acor**$(i)$ must contain the value given in position $(i, 2)$ of the **monitr** argument **acor**, for $i = 1, 2, \ldots,$ **neq** (e.g., see nag_ode_ivp_stiff_exp_fulljac (d02nb)).

5: **x** – REAL (KIND=nag_wp)

The latest value at which the solution has been computed, as provided in the argument **tcur** on return from the optional output nag_ode_ivp_stiff_integ_diag (d02ny).

6: **nqu** – INTEGER

The order of the method used up to the latest value at which the solution has been computed, as provided in the argument **nqu** on return from the optional output nag_ode_ivp_stiff_integ_diag (d02ny).

*Constraint*: **nqu** $\geq 1$.

7: **hu** – REAL (KIND=nag_wp)

The last successful step used, that is the step used in the integration to get to **x**, as provided in the argument **hu** on return from the optional output nag_ode_ivp_stiff_integ_diag (d02ny).

8: **h** – REAL (KIND=nag_wp)

The next step size to be attempted in the integration, as provided in the argument **h** on return from the optional output nag_ode_ivp_stiff_integ_diag (d02ny).

## 5.2 Optional Input Parameters

1: **sdysav** – INTEGER

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

The value used for the argument **sdysav** when calling the integrator.

*Constraint*: **sdysav** $\geq$ **nqu** $+ 1$.

2: **neq** – INTEGER

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

The value used for the argument **neq** when calling the integrator.

*Constraint*: $1 \leq$ **neq** $\leq ldysav$.

## 5.3 Output Parameters

1: **sol**(**m**) – REAL (KIND=nag_wp) array

The calculated value of the $i$th component of the solution at **xsol**, for $i = 1, 2, \ldots, m$.

2: **ifail** – INTEGER

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

If nag_ode_ivp_stiff_c1_interp (d02xk) is to be used for extrapolation, **ifail** must be set to 1 before entry. It is then essential to test the value of **ifail** on exit for **ifail** $= 1$ or 2.

## 6 Error Indicators and Warnings

Errors or warnings detected by the function:

**ifail** $= 1$

On entry, **m** $< 1$,
or **neq** $< 1$,
or $ldysav < 1$,
or **neq** $> ldysav$,
or **m** $>$ **neq**,
or **nqu** $< 1$,
or **sdysav** $<$ **nqu** $+ 1$,
or the BDF integrator was not previously used,
or the Petzold error test, if applicable, was used.

**ifail** $= 2$

On entry, **hu** $= 0.0$ or **h** $= 0.0$. This error can only occur if **h** and **hu** have been changed by you or possibly if the integrator has failed before calling nag_ode_ivp_stiff_c1_interp (d02xk).

**ifail** $= 3$ (*warning*)

nag_ode_ivp_stiff_c1_interp (d02xk) has been called for extrapolation. Before returning with this error exit, the value of the solution at **xsol** is calculated and placed in **sol**.

**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 solution values returned will be of a similar accuracy to those computed by the integrator.

## 8 Further Comments

nag_ode_ivp_stiff_c1_interp (d02xk) provides a $C^1$ interpolant and as such is ideal for most applications, for example for tabulation and root-finding. In general nag_ode_ivp_stiff_c1_interp (d02xk) should be preferred to nag_ode_ivp_stiff_nat_interp (d02xj) for interpolation as the latter provides only a $C^0$ interpolant. nag_ode_ivp_stiff_nat_interp (d02xj) is the natural interpolant employed by the BDF method and it is supplied only to permit you to reproduce the internal values used by the integrator.

## 9 Example

See Section 10 in nag_ode_ivp_stiff_exp_sparjac (d02nd) and nag_ode_ivp_stiff_exp_revcom (d02nm).

## 9.1 Program Text

```
     function d02xk_example

fprintf('d02xk example results\n\n');

% Initialize setup 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);

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

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

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

% Integrate to tout by overshooting (itask = 1).
% At monitoring stages output intermediate solutions after interpolating.
t     = 0;
tout  = 10.0;
itask = nag_int(1);
iout  = 1;
xout  = 2;
y     = [1; 0; 0];
itol  = nag_int(1);
rtol  = [1e-04];
atol  = [1e-07];

% Output header and initial results.
fprintf('\n      x           y(1)       y(2)          y(3)\n');
fprintf(' %8.3f %12.4f %12.2e %11.4f\n', t, y);

% bogus initial entry values for unitialized input parameters.
irevcm = nag_int(-999); imon = irevcm; inln = irevcm; ires = irevcm;

% Ask for warning messages only.
itrace = nag_int(0);

% The reverse communication process is controlled by the value of irevcm
% (which is 0 for the final exit).
while (irevcm ~= 0)
  [t, y, ydot, rwork, inform, ysave, wkjac, jacpvt, imon, inln, ires, ...
   irevcm, ifail] = d02nm( ...
                           t, tout, y, ydot, rwork, rtol, atol, itol, ...
                           inform, ysave, wkjac, jacpvt, imon, inln, ...
                           ires, irevcm, itask, itrace, 'neq', neq, ...
```

```
                          'sdysav', sdysav, 'nwkjac', nwkjac);

  % Evaluate the derivative, then use irevcm to place it.
  f(1) = -0.04*y(1) + 1.0e4*y(2)*y(3);
  f(2) =  0.04*y(1) - 1.0e4*y(2)*y(3) - 3.0e7*y(2)*y(2);
  f(3) =  3.0e7*y(2)*y(2);
  if (irevcm == 1 || irevcm == 3 )
    lrw = 50+2*neq;
    rwork(lrw+1:lrw+neq) = f;
  elseif (irevcm == 4)
    lrw = 50+neq;
    rwork(lrw+1:lrw+neq) = f;
  elseif (irevcm == 5)
    ydot = f;
  elseif (irevcm == 9 && imon == 1)
    % Extract useful information about the current step.
    tc    = rwork(19);
    hlast = rwork(15);
    hnext = rwork(16);
    nqu   = nag_int(rwork(10));

    % xout is intermediate output point.
    while (xout <= tc)
      lrw = 50+neq;
      [sol, ifail] = d02xk( ...
                           xout, neq, ysave, rwork(lrw+1:lrw+neq), ...
                           tc, nqu, hlast, hnext);
      % Output interpolated result, and save it for plotting.
      fprintf(' %8.3f %12.4f %12.2e %11.4f\n', xout, sol);
      iout = iout + 1;
      xout = iout*2;
    end
  end
end
```

## 9.2 Program Results

```
     d02xk example results

      x           y(1)       y(2)          y(3)
    0.000       1.0000     0.00e+00      0.0000
    2.000       0.9416     2.70e-05      0.0584
    4.000       0.9055     2.24e-05      0.0945
    6.000       0.8793     1.96e-05      0.1207
    8.000       0.8585     1.77e-05      0.1414
   10.000       0.8414     1.62e-05      0.1586
```