**nag_ode_ivp_rk_interp (d02pxc)**

1. **Purpose**

   **nag_ode_ivp_rk_interp (d02pxc)** is a function to compute the solution of a system of ordinary differential equations using interpolation anywhere on an integration step taken by **nag_ode_ivp_rk_onestep (d02pdc)**.

2. **Specification**

   ```c
   #include <nag.h>
   #include <nagd02.h>

   void nag_ode_ivp_rk_interp(Integer neq, double twant, Nag_SolDeriv request, Integer nwant,
   double ywant[], double ypwant[],
   void (*f) (Integer neq, double t, double y[], double yp[], Nag_User *comm),
   Nag_ODE_RK *opt, Nag_User *comm, NagError *fail)
   ```

3. **Description**

   This function and its associated functions (**nag_ode_ivp_rk_setup (d02pvc)**, **nag_ode_ivp_rk_onestep (d02pdc)**, **nag_ode_ivp_rk_reset_tend (d02pwc)**, **nag_ode_ivp_rk_errass (d02pzc)**) solve the initial value problem for a first order system of ordinary differential equations. The functions, based on Runge-Kutta methods and derived from RKSUITE (Brankin et al, 1991) integrate

   \[ y' = f(t, y) \quad \text{given} \quad y(t_0) = y_0 \]

   where \( y \) is the vector of \( neq \) solution components and \( t \) is the independent variable.

   **nag_ode_ivp_rk_onestep (d02pdc)** computes the solution at the end of an integration step. Using the information computed on that step **nag_ode_ivp_rk_interp** computes the solution by interpolation at any point on that step. It cannot be used if method = Nag_RK_7_8 was specified in the call to set-up function **nag_ode_ivp_rk_setup (d02pvc)**.

4. **Parameters**

   - **neq**
     - Input: the number of ordinary differential equations in the system.
     - Constraint: \( neq \geq 1 \).
   - **twant**
     - Input: the value of the independent variable, \( t \), where a solution is desired.
   - **request**
     - Input: determines whether the solution and/or its first derivative are computed as follows:
       - request = Nag_Sol - compute approximate solution only
       - request = Nag_Der - compute approximate first derivative of the solution only
       - request = Nag_SolDer - compute both approximate solution and first derivative.
     - Constraint: request = Nag_Sol or Nag_Der or Nag_SolDer.
   - **nwant**
     - Input: the number of components of the solution to be computed. The first \( nwant \) components are evaluated.
     - Constraint: \( 1 \leq nwant \leq neq \).
   - **ywant[nwant]**
     - Output: an approximation to the first \( nwant \) components of the solution at \( twant \) when specified by request.
nag_ode_ivp_rk_interp

ypwant[nwant]
Output: an approximation to the first nwant components of the first derivative of the solution at twant when specified by request.

f
This function must evaluate the functions $f_i$ (that is the first derivatives $y_i'$) for given values of the arguments $t, y_i$. It must be the same procedure as supplied to nag_ode_ivp_rkonestep (d02pdc).

```c
void f (Integer neq, double t, double y[], double yp[], Nag_User *comm)
```

- **neq**
  Input: the number of differential equations.

- **t**
  Input: the current value of the independent variable, $t$.

- **y[neq]**
  Input: the current values of the dependent variables, $y_i$ for $i = 1, 2, \ldots, neq$.

- **yp[neq]**
  Output: the values of $f_i$ for $i = 1, 2, \ldots, neq$.

- **comm**
  Input/Output: pointer to a structure of type Nag_User with the following member:
  - **p** - Pointer
    Input/Output: The pointer `comm->p` should be cast to the required type, e.g. `struct user *s = (struct user *)comm->p`, to obtain the original object’s address with appropriate type. (See the argument `comm` below.)

opt
Input: the structure of type Nag_ODE_RK as output from nag_ode_ivp_rkonestep (d02pdc). This structure must not be changed by the user.
Output: some members of opt are changed internally.

comm
Input/Output: pointer to a structure of type Nag_User with the following member:
- **p** - Pointer
  Input/Output: the pointer $p$, of type Pointer, allows the user to communicate information to and from the user-defined function $f()$. An object of the required type should be declared by the user, e.g. a structure, and its address assigned to the pointer $p$ by means of a cast to Pointer in the calling program, e.g. `comm.p = (Pointer)&s`. The type pointer will be `void *` with a C compiler that defines `void *` and `char *` otherwise.

fail
The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. Error Indications and Warnings

**NE_PREV_CALL**
The previous call to a function had resulted in a severe error. You must call nag_ode_ivp_rksetup (d02pvc) to start another problem.

**NE_RK_INVALID_CALL**
The function to be called as specified in the setup function nag_ode_ivp_rksetup (d02pvc) was nag_ode_ivp_rkrange (d02pcc). However the actual call was made to nag_ode_ivp_rkinterp. This is not permitted.

**NE_MISSING_CALL**
Previous call to nag_ode_ivp_rkonestep (d02pdc) has not been made, hence nag_ode_ivp_rk_interp must not be called.
NE_PREV_CALL_INI
The previous call to the function nag_ode_ivp_rk_onestep (d02pdc) resulted in a severe error. You must call nag_ode_ivp_rk_setup (d02pvc) to start another problem.

NE_NEQ
The value of neq supplied is not the same as that given to the setup function nag_ode_ivp_rk_setup (d02pvc). neq = ⟨value⟩ but the value given to nag_ode_ivp_rk_setup (d02pvc) was ⟨value⟩.

NE_BAD_PARAM
On entry parameter request had an illegal value.

NE_2_INT_ARG_GT
On entry nwant = ⟨value⟩ while neq = ⟨value⟩. These parameters must satisfy neq ≤ nwant.

NE_2_INT_ARG_LT
On entry, nwant must not be less than 1: nwant = ⟨value⟩.

NE_ALLOC_FAIL
Memory allocation failed.

NE_RK_PX_METHOD
Interpolation is not available with method = Nag_RK_7_8. Either use method = Nag_RK_2_3 or Nag_RK_4_5 for which interpolation is available. Alternatively use nag_ode_ivp_rk_reset_tend (d02pwc) to make nag_ode_ivp_rk_onestep (d02pdc) step exactly to the points where you want output.

NE_MEMORY_FREED
Internally allocated memory has been freed by a call to nag_ode_ivp_rk_free (d02pvc) without a subsequent call to the set up function nag_ode_ivp_rk_setup (d02pvc).

6. Further Comments
None.

6.1. Accuracy
The computed values will be of a similar accuracy to that computed by nag_ode_ivp_rk_onestep (d02pdc).

6.2. References
Brankin R W, Gladwell I and Shampine L F (1991) RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs SoftReport 91–S1, Department of Mathematics, Southern Methodist University, Dallas, TX 75275, U.S.A.

7. See Also
nag_ode_ivp_rk_onestep (d02pdc)
nag_ode_ivp_rk_setup (d02pvc)
nag_ode_ivp_rk_reset_tend (d02pwc)
nag_ode_ivp_rk_free (d02pvc)

8. Example
We solve the equation
\[ y'' = -y, \quad y(0) = 0, \quad y'(0) = 1 \]
reposed as
\[ y_1 = y_2, \quad y_2' = -y_1 \]
over the range \([0, 2\pi] \) with initial conditions \( y_1 = 0.0 \) and \( y_2 = 1.0 \). We use relative error control with threshold values of \( 1.0e-8 \) for each solution component. nag_ode_ivp_rk_onestep (d02pdc) is used to integrate the problem one step at a time and nag_ode_ivp_rk_interpol is used to compute the first component of the solution and its derivative at intervals of length \( \pi/8 \) across the range whenever these points lie in one of those integration steps. We use a moderate order Runge-Kutta method (method = Nag_RK_4_5) with tolerances tol = 1.0e–3 and tol = 1.0e–4 in turn so that we may compare the solutions. The value of \( \pi \) is obtained by using X01AAC.
8.1. Program Text

/* nag_ode_ivp_rk_interp(d02pxc) Example Program
 */

#include <nag.h>
#include <stdio.h>
#include <nag_stdlib.h>
#include <math.h>
#include <nagd02.h>
#include <nagx01.h>

#ifdef NAG_PROTO
static void f(Integer neq, double t1, double y[], double yp[], Nag_User *comm);
#else
static void f();
#endif
#define NEQ 2
#define NWANT 1
#define ZERO 0.0
#define ONE 1.0
#define TWO 2.0
#define FOUR 4.0

main()
{
    Integer neq, nwant;
    double hstart, pi, tnow, tend, tol, tstart, tinc, twant;
    Integer i, j, nout;
    double thres[NEQ], ynow[NEQ], ypnow[NEQ], ystart[NEQ], ywant[NWANT];
    double ypwant[NWANT];
    Nag_RK_method method;
    Nag_ErrorAssess errass;
    Nag_ODE_RK opt;
    Nag_User comm;

    Vprintf("d02pxc Example Program Results\n");
    /* Set initial conditions and input for d02pvc */
    neq = NEQ;
    method = Nag_RK_4_5;
    pi = X01AAC;
    tstart = ZERO;
    ystart[0] = ZERO;
    ystart[1] = ONE;
    tend = TWO*pi;
    for (i=0; i<neq; i++)
        thres[i] = 1.0e-8;
    errass = Nag_ErrorAssess_off;
    hstart = ZERO;

    /* Set control for output */
    nwant = NWANT;
    nout = 16;
    tinc = tend/nout;
    for (i=1; i<=2; i++)
    {
        if (i==1) tol = 1.0e-3;
        if (i==2) tol = 1.0e-4;
        d02pvc(neq, tstart, ystart, tend, tol, thres, method,
             Nag_RK_onestep, errass, hstart, &opt, NAGERR_DEFAULT);
    }
Vprintf("\nCalculation with tol = %8.1e\n\n",tol);
Vprintf (" t y1 y2\n\n");
Vprintf("%8.3f %8.4f %8.4f\n", tstart, ystart[0], ystart[1]);
j = nout - 1;
twant = tend - j*tinc;

do {
    d02pdc(neq, f, &tnow, ynow, ypnow, &opt, &comm, NAGERR_DEFAULT);
    while (twant<=tnow) {
        d02pxc(neq, twant, Nag_SolDer, nwant, ywant, ypwant, f,
            &opt, &comm, NAGERR_DEFAULT);
        Vprintf("%8.3f %8.4f %8.4f\n", twant, ywant[0],
            ypwant[0]);
        j = j - 1;
        twant = tend - j*tinc;
    }
} while (tnow<tend);

Vprintf("\nCost of the integration in evaluations of f is %ld\n\n", opt.totfcn);
d02ppc(&opt);
}
exit(EXIT_SUCCESS);
#endif NAG_PROTO

static void f(Integer neq, double t, double y[], double yp[], Nag_User *comm)
#else
static void f(neq, t, y, yp, comm)
    Integer neq;
    double t;
    double y[], yp[];
    Nag_User *comm;
#endif
{
    yp[0] = y[1];
    yp[1] = -y[0];
}

8.2. Program Data
None.

8.3. Program Results
d02pxc Example Program Results
Calculation with tol = 1.0e-03

<table>
<thead>
<tr>
<th>t</th>
<th>y1</th>
<th>y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>0.393</td>
<td>0.3827</td>
<td>0.9239</td>
</tr>
<tr>
<td>0.785</td>
<td>0.7071</td>
<td>0.7071</td>
</tr>
<tr>
<td>1.178</td>
<td>0.9239</td>
<td>0.3826</td>
</tr>
<tr>
<td>1.571</td>
<td>1.0000</td>
<td>-0.0001</td>
</tr>
<tr>
<td>1.963</td>
<td>0.9238</td>
<td>-0.3828</td>
</tr>
<tr>
<td>2.356</td>
<td>0.7070</td>
<td>-0.7073</td>
</tr>
<tr>
<td>2.749</td>
<td>0.3825</td>
<td>-0.9240</td>
</tr>
<tr>
<td>3.142</td>
<td>-0.0002</td>
<td>-0.9999</td>
</tr>
<tr>
<td>3.534</td>
<td>-0.3829</td>
<td>-0.9238</td>
</tr>
<tr>
<td>3.927</td>
<td>-0.7072</td>
<td>-0.7069</td>
</tr>
<tr>
<td>4.320</td>
<td>-0.9239</td>
<td>-0.3823</td>
</tr>
<tr>
<td>4.712</td>
<td>-0.9999</td>
<td>0.0004</td>
</tr>
<tr>
<td>5.105</td>
<td>-0.9236</td>
<td>0.3830</td>
</tr>
<tr>
<td>5.498</td>
<td>-0.7068</td>
<td>0.7073</td>
</tr>
<tr>
<td>5.890</td>
<td>-0.3823</td>
<td>0.9239</td>
</tr>
<tr>
<td>6.283</td>
<td>0.0004</td>
<td>0.9998</td>
</tr>
</tbody>
</table>
Cost of the integration in evaluations of f is 68

Calculation with tol = 1.0e-04

<table>
<thead>
<tr>
<th>t</th>
<th>y1</th>
<th>y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
<tr>
<td>0.393</td>
<td>0.3827</td>
<td>0.9239</td>
</tr>
<tr>
<td>0.785</td>
<td>0.7071</td>
<td>0.7071</td>
</tr>
<tr>
<td>1.178</td>
<td>0.9239</td>
<td>0.3827</td>
</tr>
<tr>
<td>1.571</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>1.963</td>
<td>0.9239</td>
<td>-0.3827</td>
</tr>
<tr>
<td>2.356</td>
<td>0.7071</td>
<td>-0.7071</td>
</tr>
<tr>
<td>2.749</td>
<td>0.3827</td>
<td>-0.9239</td>
</tr>
<tr>
<td>3.142</td>
<td>0.0000</td>
<td>-1.0000</td>
</tr>
<tr>
<td>3.534</td>
<td>-0.3827</td>
<td>-0.9239</td>
</tr>
<tr>
<td>3.927</td>
<td>-0.7071</td>
<td>-0.7071</td>
</tr>
<tr>
<td>4.320</td>
<td>-0.9239</td>
<td>-0.3827</td>
</tr>
<tr>
<td>4.712</td>
<td>-1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>5.105</td>
<td>-0.9239</td>
<td>0.3827</td>
</tr>
<tr>
<td>5.498</td>
<td>-0.7071</td>
<td>0.7071</td>
</tr>
<tr>
<td>5.890</td>
<td>-0.3826</td>
<td>0.9239</td>
</tr>
<tr>
<td>6.283</td>
<td>0.0000</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Cost of the integration in evaluations of f is 105