

# NAG Library Function Document

## nag\_ode\_ivp\_adams\_interp (d02qzc)

### 1 Purpose

nag\_ode\_ivp\_adams\_interp (d02qzc) interpolates components of the solution of a non-stiff system of first order ordinary differential equations from information provided by nag\_ode\_ivp\_adams\_roots (d02qfc). Normally this function will be used in conjunction with the integration function, nag\_ode\_ivp\_adams\_roots (d02qfc), operating in one-step mode.

### 2 Specification

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

void nag_ode_ivp_adams_interp (Integer neqf, double twant, Integer nwant,
    double ywant[], double ypwant[], Nag_ODE_Adams *opt, NagError *fail)
```

### 3 Description

nag\_ode\_ivp\_adams\_interp (d02qzc) evaluates the first **nwant** components of the solution of a non-stiff system of first order ordinary differential equations at any point using the method of Watts and Shampine (1986) and information generated by nag\_ode\_ivp\_adams\_roots (d02qfc). nag\_ode\_ivp\_adams\_interp (d02qzc) should not normally be used to extrapolate outside the current range of the values produced by the integration function.

### 4 References

Watts H A and Shampine L F (1986) Smoother interpolants for Adams codes *SIAM J. Sci. Statist. Comput.* **7** 334–345

### 5 Arguments

- 1: **neqf** – Integer *Input*  
*On entry:* the number of differential equations.  
*Constraint:* **neqf** ≥ 1.
- 2: **twant** – double *Input*  
*On entry:* the point at which components of the solution and derivative are to be evaluated. **twant** should not normally be an extrapolation point, that is **twant** should satisfy  
**opt**→**tcurr** – **opt**→**hlast** ≤ **twant** ≤ **opt**→**tcurr**.  
or if integration is proceeding in the negative direction  
**opt**→**tcurr** – **opt**→**hlast** ≥ **twant** ≥ **opt**→**tcurr**.  
Extrapolation is permitted but not recommended and a **fail** value of NW\_EXTRAPOLATION is returned whenever extrapolation is attempted.
- 3: **nwant** – Integer *Input*  
*On entry:* the number of components of the solution and derivative whose values, at **twant**, are required. The first **nwant** components are evaluated.  
*Constraint:* 1 ≤ **nwant** ≤ **neqf**.

- 4: **ywant[nwant]** – double *Output*  
*On exit:* **ywant**[ $i - 1$ ] contains the calculated value of the  $i$ th component of the solution at **twant**, for  $i = 1, 2, \dots, \mathbf{nwant}$ .
- 5: **ypwant[nwant]** – double *Output*  
*On exit:* **ypwant**[ $i - 1$ ] contains the calculated value of the  $i$ th component of the derivative at **twant**, for  $i = 1, 2, \dots, \mathbf{nwant}$ .
- 6: **opt** – Nag\_ODE\_Adams \* *Input*  
*On entry:* the structure of type Nag\_ODE\_Adams as output from the integration function nag\_ode\_ivp\_adams\_roots (d02qfc). The structure **must** be passed unchanged. (See Section 9 for comments about deallocation of memory from **opt**.)
- 7: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_INT\_ARG\_LT

On entry, **nwant** =  $\langle value \rangle$ .  
Constraint: **nwant**  $\geq 1$ .

### NE\_NEQF

The value of **neqf** supplied is not the same as that given to the setup function nag\_ode\_ivp\_adams\_setup (d02qwc). **neqf** =  $\langle value \rangle$  but the value given to nag\_ode\_ivp\_adams\_setup (d02qwc) was  $\langle value \rangle$ .

### NE\_NO\_INTEGRATE

The integrator function nag\_ode\_ivp\_adams\_roots (d02qfc) has not been called.

### NE\_NO\_STEPS

No successful integration steps were taken in the call(s) to the integration function nag\_ode\_ivp\_adams\_roots (d02qfc).

### NE\_NWANT\_GT

**nwant** is greater than the value of **neqf** given to the setup function nag\_ode\_ivp\_adams\_setup (d02qwc). **nwant** =  $\langle value \rangle$ , **neqf** =  $\langle value \rangle$ .

### NW\_EXTRAPOLATION

Extrapolation requested, **twant** =  $\langle value \rangle$ .

## 7 Accuracy

The error in interpolation is of a similar order to the error arising from the integration. The same order of accuracy can be expected when extrapolating using nag\_ode\_ivp\_adams\_interp (d02qzc). However, the actual error in extrapolation will, in general, be much larger than for interpolation.

## 8 Parallelism and Performance

nag\_ode\_ivp\_adams\_interp (d02qzc) is not threaded in any implementation.

## 9 Further Comments

When interpolation for only a few components is required then it is more efficient to order the components of interest so that they are numbered first.

The structure **opt** will contain pointers which have been allocated memory during a call to `nag_ode_ivp_adams_setup` (d02qwc). This allocated memory is used by `nag_ode_ivp_adams_roots` (d02qfc) and `nag_ode_ivp_adams_interp` (d02qzc). When all calls to these functions have been completed the function `nag_ode_ivp_adams_free` (d02qyc) may be called to free the allocated memory from the structure.

## 10 Example

This example solves the equation

$$y'' = -y, \quad y(0) = 0, y'(0) = 1$$

reposed as

$$\begin{aligned} y'_1 &= y_2 \\ y'_2 &= -y_1 \end{aligned}$$

over the range  $[0, \pi/2]$  with initial conditions  $y_1 = 0$  and  $y_2 = 1$  using vector error control (**vectol** = Nag.TRUE) and `nag_ode_ivp_adams_roots` (d02qfc) in one-step mode (**one\_step** = Nag.TRUE). `nag_ode_ivp_adams_interp` (d02qzc) is used to provide solution values at intervals of  $\pi/16$ .

### 10.1 Program Text

```
/* nag_ode_ivp_adams_interp (d02qzc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

#ifdef __cplusplus
extern "C"
{
#endif
    static void NAG_CALL ftry03(Integer neqf, double x, const double y[],
                                double yp[], Nag_User *comm);
#ifdef __cplusplus
}
#endif

#define NEQF 2
#define TSTART 0.0

int main(void)
{
    static Integer use_comm[1] = { 1 };
    Nag_Boolean alter_g, crit, one_step, sophist, vectol;
    Integer exit_status = 0, i, j, max_step, neqf, neqg, nwant;
    NagError fail;
    Nag_ODE_Adams opt;
    Nag_Start state;
    Nag_User comm;
```

```

double *atol = 0, hmax, pi, *rtol = 0, t, tcrit, tinc, tout, twant, *y = 0;
double *ypwant = 0, *ywant = 0;

INIT_FAIL(fail);

printf("nag_ode_ivp_adams_interp (d02qzc) Example Program Results\n");

/* For communication with user-supplied functions: */
comm.p = (Pointer) &use_comm;

/* nag_pi (x01aac).
 * pi
 */
pi = nag_pi;
state = Nag_NewStart;
neqf = NEQF;
if (neqf >= 1) {
    if (!(atol = NAG_ALLOC(neqf, double)) ||
        !(rtol = NAG_ALLOC(neqf, double)) ||
        !(y = NAG_ALLOC(neqf, double)) ||
        !(ywant = NAG_ALLOC(neqf, double)) ||
        !(ypwant = NAG_ALLOC(neqf, double)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
        goto END;
    }
}
else {
    exit_status = 1;
    return exit_status;
}

neqg = 0;
sophist = Nag_FALSE;
vectol = Nag_TRUE;
for (i = 0; i < 2; ++i) {
    atol[i] = 1e-08;
    rtol[i] = 0.0001;
}
one_step = Nag_TRUE;
crit = Nag_TRUE;
tinc = pi * 0.0625;
tcrit = tinc * 8.0;
tout = tcrit;
max_step = 500;
hmax = 2.0;
t = TSTART;
twant = TSTART + tinc;
nwant = 2;
y[0] = 0.0;
y[1] = 1.0;
printf("\n      T          Y(1)    Y(2)\n");
printf("    %6.4f    %7.4f %7.4f  \n", t, y[0], y[1]);

/* nag_ode_ivp_adams_setup (d02qwc).
 * Setup function for nag_ode_ivp_adams_roots (d02qfc)
 */
nag_ode_ivp_adams_setup(&state, neqf, vectol, atol, rtol, one_step, crit,
                        tcrit, hmax, max_step, neqg, &alter_g, sophist,
                        &opt, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_ivp_adams_setup (d02qwc).\n%s\n",
          fail.message);
    exit_status = 1;
    goto END;
}

j = 1;

while (t < tout && fail.code == NE_NOERROR) {

```

```

/* nag_ode_ivp_adams_roots (d02qfc).
 * Ordinary differential equation solver using Adams method
 * (sophisticated use)
 */
nag_ode_ivp_adams_roots(neqf, ftry03, &t, y, tout, NULLDFN,
                        &comm, &opt, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_ivp_adams_roots (d02qfc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
}

while (twant <= t && fail.code == NE_NOERROR) {
    /* nag_ode_ivp_adams_interp (d02qzc).
     * Interpolation function for use with
     * nag_ode_ivp_adams_roots (d02qfc)
     */
    nag_ode_ivp_adams_interp(neqf, twant, nwant, ywant, ypwant, &opt,
                             &fail);
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_ode_ivp_adams_interp (d02qzc).\n%s\n",
               fail.message);
        exit_status = 1;
        goto END;
    }

    printf(" %6.4f    %7.4f %7.4f  \n", twant, ywant[0], ywant[1]);
    ++j;
    twant = (double) j * tinc + 0.0;
}
}
/* Free the memory which was allocated by
 * nag_ode_ivp_adams_setup (d02qwc) to the pointers inside opt.
 */
/* nag_ode_ivp_adams_free (d02qyc).
 * Freeing function for use with nag_ode_ivp_adams_roots
 * (d02qfc)
 */
nag_ode_ivp_adams_free(&opt);

END:
    NAG_FREE(atol);
    NAG_FREE(rtol);
    NAG_FREE(y);
    NAG_FREE(ywant);
    NAG_FREE(ypwant);
    return exit_status;
}

static void NAG_CALL ftry03(Integer neqf, double x, const double y[], double
                             yp[], Nag_User *comm)
{
    Integer *use_comm = (Integer *) comm->p;

    if (use_comm[0]) {
        printf("(User-supplied callback ftry03, first invocation.)\n");
        use_comm[0] = 0;
    }

    yp[0] = y[1];
    yp[1] = -y[0];
} /* ftry03 */

```

## 10.2 Program Data

None.

### 10.3 Program Results

nag\_ode\_ivp\_adams\_interp (d02qzc) Example Program Results

| T  | Y(1)   | Y(2)    |
|--|--------|---------|
| 0.0000   | 0.0000 | 1.0000  |
| (User-supplied callback ftry03, first invocation.) |        |         |
| 0.1963   | 0.1951 | 0.9808  |
| 0.3927   | 0.3827 | 0.9239  |
| 0.5890   | 0.5556 | 0.8315  |
| 0.7854   | 0.7071 | 0.7071  |
| 0.9817   | 0.8315 | 0.5556  |
| 1.1781   | 0.9239 | 0.3827  |
| 1.3744   | 0.9808 | 0.1951  |
| 1.5708   | 1.0000 | -0.0000 |

---