NAG Library Function Document

nag_ode_ivp_rkts_interp (d02psc)

1 Purpose

nag_ode_ivp_rkts_interp (d02psc) computes the solution of a system of ordinary differential equations using interpolation anywhere on an integration step taken by nag_ode_ivp_rkts_onestep (d02pfc).

2 Specification

```
#include <nag.h>
#include <nagd02.h>
void nag_ode_ivp_rkts_interp (Integer n, double twant, Nag_SolDeriv reqest,
     Integer nwant, double ywant[], double ypwant[],
     void (*f)(double t, Integer n, const double y[], double yp[],
          Nag_Comm *comm),
     double wcomm[], Integer lwcomm, Nag_Comm *comm, Integer iwsav[],
     double rwsav[], NagError *fail)
```

3 Description

nag_ode_ivp_rkts_interp (d02psc) and its associated functions (nag_ode_ivp_rkts_onestep (d02pfc), nag_ode_ivp_rkts_setup (d02pqc), nag_ode_ivp_rkts_reset_tend (d02prc), nag_ode_ivp_rkts_diag (d02ptc) and nag_ode_ivp_rkts_errass (d02puc)) 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 (see Brankin *et al.* (1991)), integrate

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

where y is the vector of n solution components and t is the independent variable.

nag_ode_ivp_rkts_onestep (d02pfc) computes the solution at the end of an integration step. Using the information computed on that step nag_ode_ivp_rkts_interp (d02psc) 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 setup function nag_ode_ivp_rkts_setup (d02pqc).

4 References

Brankin R W, Gladwell I and Shampine L F (1991) RKSUITE: A suite of Runge–Kutta codes for the initial value problems for ODEs *SoftReport 91-S1* Southern Methodist University

5 Arguments

1: **n** – Integer *Input*

On entry: n, the number of ordinary differential equations in the system to be solved by the integration function.

Constraint: $\mathbf{n} \geq 1$.

2: **twant** – double *Input*

On entry: t, the value of the independent variable where a solution is desired.

3: **reqest** – Nag_SolDeriv *Input*

On entry: determines whether the solution and/or its first derivative are to be computed

reqest = Nag_Sol
compute approximate solution.

reqest = Nag_Der
compute approximate first derivative.

reqest = Nag_SolDer
compute approximate solution and first derivative.

Constraint: **reqest** = Nag_Sol, Nag_Der or Nag_SolDer.

4: **nwant** – Integer *Input*

On entry: the number of components of the solution to be computed. The first **nwant** components are evaluated.

Constraint: $1 \leq$ **nwant** $\leq$ **n**.

5: **ywant**[**nwant**] – double *Output*

On exit: an approximation to the first **nwant** components of the solution at **twant** if **reqest** = Nag_Sol or Nag_SolDer. Otherwise **ywant** is not defined.

6: **ypwant**[**nwant**] – double *Output*

On exit: an approximation to the first **nwant** components of the first derivative at **twant** if **reqest** = Nag_Der or Nag_SolDer. Otherwise **ypwant** is not defined.

7: **f** – function, supplied by the user *External Function*

f 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_rkts_onestep (d02pfc).

The specification of **f** is:

```
void f (double t, Integer n, const double y[], double yp[],
     Nag_Comm *comm)
```

1: **t** – double *Input*

On entry: t, the current value of the independent variable.

2: **n** – Integer *Input*

On entry: n, the number of ordinary differential equations in the system to be solved.

3: **y**[**n**] – const double *Input*

On entry: the current values of the dependent variables, y_i, for $i = 1, 2, \ldots, n$.

4: **yp**[**n**] – double *Output*

On exit: the values of f_i, for $i = 1, 2, \ldots, n$.

5: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **f**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling nag_ode_ivp_rkts_interp (d02psc) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_ode_ivp_rkts_interp (d02psc) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

8: **wcomm**[**lwcomm**] – double *Communication Array*

On entry: this array stores information that can be utilized on subsequent calls to nag_ode_ivp_rkts_interp (d02psc).

9: **lwcomm** – Integer *Input*

On entry: length of **wcomm**.

If in a previous call to nag_ode_ivp_rkts_setup (d02pqc):

method = Nag_RK_2_3 then **lwcomm** must be at least 1.

method = Nag_RK_4_5 then **lwcomm** must be at least $\mathbf{n} + \max(\mathbf{n}, 5 \times \mathbf{nwant})$.

method = Nag_RK_7_8 then **wcomm** and **lwcomm** are not referenced.

10: **comm** – Nag_Comm *

The NAG communication argument (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

11: **iwsav**[**130**] – Integer *Communication Array*
12: **rwsav**[$\mathbf{32} \times \mathbf{n} + \mathbf{350}$] – double *Communication Array*

On entry: these must be the same arrays supplied in a previous call nag_ode_ivp_rkts_onestep (d02pfc). They must remain unchanged between calls.

On exit: information about the integration for use on subsequent calls to nag_ode_ivp_rkts_one step (d02pfc), nag_ode_ivp_rkts_interp (d02psc) or other associated functions.

13: **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_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_INT

On entry, **lwcomm** = ⟨*value*⟩.
Constraint: for **method** = Nag_RK_2_3, $\mathbf{lwcomm} \geq 1$.

NE_INT_2

On entry, **nwant** = ⟨*value*⟩ and **n** = ⟨*value*⟩.
Constraint: $1 \leq$ **nwant** $\leq$ **n**.

NE_INT_3

On entry, **lwcomm** = ⟨*value*⟩, **n** = ⟨*value*⟩ and **nwant** = ⟨*value*⟩.
Constraint: for **method** = Nag_RK_4_5, **lwcomm** $\geq$ **n** + max(**n**, 5 × **nwant**).

NE_INT_CHANGED

On entry, **n** = ⟨*value*⟩, but the value passed to the setup function was **n** = ⟨*value*⟩.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in How to Use the NAG Library and its Documentation for further information.

NE_MISSING_CALL

You cannot call this function before you have called the step integrator.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_PREV_CALL

On entry, a previous call to the setup function has not been made or the communication arrays have become corrupted, or a catastrophic error has already been detected elsewhere.
You cannot continue integrating the problem.

NE_PREV_CALL_INI

You cannot call this function after the integrator has returned an error.

NE_RK_INVALID_CALL

You cannot call this function when you have specified, in the setup function, that the range integrator will be used.

NE_RK_NO_INTERP

method = Nag_RK_7_8 in setup, but interpolation is not available for this method. Either use **method** = Nag_RK_4_5 in setup or use reset function to force the integrator to step to particular points.

7 Accuracy

The computed values will be of a similar accuracy to that computed by nag_ode_ivp_rkts_onestep (d02pfc).

8 Parallelism and Performance

nag_ode_ivp_rkts_interp (d02psc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

None.

10 Example

This example solves the equation

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

reposed as

$$y'_1 = y_2$$
$$y'_2 = -y_1$$

over the range $[0, 2\pi]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$. Relative error control is used with threshold values of 1.0e−8 for each solution component. nag_ode_ivp_rkts_onestep (d02pfc) is used to integrate the problem one step at a time and nag_ode_ivp_rkts_interp (d02psc) 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. A low order Runge–Kutta method (**method** = Nag_RK_2_3) is also used with tolerances **tol** = 1.0e−4 and **tol** = 1.0e−5 in turn so that solutions may be compared.

10.1 Program Text

```
/* nag_ode_ivp_rkts_interp (d02psc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd02.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL f(double t, Integer n, const double *y,
                         double *yp, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

#define N 2

int main(void)
{
  /* Scalars */
  double tol0 = 1.0e-3;
  Integer npts = 16, exit_status = 0;
  Integer liwsav, lrwsav, lwcomm, n;
  double hnext, hstart, tend, tgot, tinc, tol, tstart, twant, waste;
  Integer fevals, i, j, k, stepcost, stepsok;
  /* Arrays */
  static double ruser[1] = { -1.0 };
  double *rwsav = 0, *thresh = 0, *ygot = 0, *yinit = 0, *ypgot = 0;
```

```
  double *ywant = 0, *ypwant = 0, *wcomm = 0;
  Integer *iwsav = 0;
  char nag_enum_arg[40];
  /* NAG types */
  NagError fail;
  Nag_RK_method method;
  Nag_ErrorAssess errass;
  Nag_SolDeriv reqest = Nag_SolDer;
  Nag_Comm comm;

  INIT_FAIL(fail);

  n = N;
  liwsav = 130;
  lrwsav = 350 + 32 * n;
  lwcomm = 6 * n;
  printf("nag_ode_ivp_rkts_interp (d02psc) Example Program Results\n\n");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  if (!(thresh = NAG_ALLOC(n, double)) ||
      !(ygot = NAG_ALLOC(n, double)) ||
      !(yinit = NAG_ALLOC(n, double)) ||
      !(ypgot = NAG_ALLOC(n, double)) ||
      !(ywant = NAG_ALLOC(n, double)) ||
      !(ypwant = NAG_ALLOC(n, double)) ||
      !(iwsav = NAG_ALLOC(liwsav, Integer)) ||
      !(rwsav = NAG_ALLOC(lrwsav, double)) ||
      !(wcomm = NAG_ALLOC((lwcomm), double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Set initial conditions for ODE and parameters for the integrator. */

#ifdef _WIN32
  scanf_s(" %39s%*[^\n] ", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n] ", nag_enum_arg);
#endif
  /* nag_enum_name_to_value (x04nac) Converts NAG enum member name to value. */
  method = (Nag_RK_method) nag_enum_name_to_value(nag_enum_arg);

#ifdef _WIN32
  scanf_s(" %39s%*[^\n] ", nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf(" %39s%*[^\n] ", nag_enum_arg);
#endif
  errass = (Nag_ErrorAssess) nag_enum_name_to_value(nag_enum_arg);

#ifdef _WIN32
  scanf_s("%lf%lf%*[^\n] ", &tstart, &tend);
#else
  scanf("%lf%lf%*[^\n] ", &tstart, &tend);
#endif

  for (j = 0; j < n; j++)
#ifdef _WIN32
    scanf_s("%lf", &yinit[j]);
#else
    scanf("%lf", &yinit[j]);
```

```
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &hstart);
#else
  scanf("%lf%*[^\n] ", &hstart);
#endif
  for (j = 0; j < n; j++)
#ifdef _WIN32
    scanf_s("%lf", &thresh[j]);
#else
    scanf("%lf", &thresh[j]);
#endif
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  tinc = (tend - tstart) / (double) (npts);
  tol = tol0;
  for (i = 1; i <= 2; i++) {
    tol = tol * 0.1;
    /* Initialize Runge-Kutta method for integrating ODE using
     * nag_ode_ivp_rkts_setup (d02pqc).
     */
    nag_ode_ivp_rkts_setup(n, tstart, tend, yinit, tol, thresh, method,
                           errass, hstart, iwsav, rwsav, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ode_ivp_rkts_setup (d02pqc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }

    printf(" Calculation with tol = %8.1e\n", tol);
    printf("    t          y1        y1'\n");
    printf("%6.3f", tstart);
    for (k = 0; k < n; k++)
      printf("   %8.4f", yinit[k]);
    printf("\n");

    /* Set up first point at which solution is desired. */
    twant = tstart + tinc;
    tgot = tstart;
    /* Integrate by by single steps until tend is reached or error is
     * encountered. Solution is required at regular increments, requiring
     * interpolation on those steps that pass over the regulat grid values
     * of t.
     */
    while (tgot < tend) {
      /* Solve ODE by Runge-Kutta method by a sequence of single steps using
       * nag_ode_ivp_rkts_onestep (d02pfc).
       */
      nag_ode_ivp_rkts_onestep(f, n, &tgot, ygot, ypgot, &comm,
                               iwsav, rwsav, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_ode_ivp_rkts_onestep (d02pfc).\n%s\n",
               fail.message);
        exit_status = 2;
        goto END;
      }

      /* Interpolate onto those grid values passed over in by last step. */
      while (twant <= tgot) {
        /* Interpolate at t = twant, given solution by
```

```
           * nag_ode_ivp_rkts_onestep (d02pfc), using
           * nag_ode_ivp_rkts_interp (d02psc).
           */
          nag_ode_ivp_rkts_interp(n, twant, reqest, n, ywant, ypwant, f,
                                  wcomm, lwcomm, &comm, iwsav, rwsav, &fail);
          if (fail.code != NE_NOERROR) {
            printf("Error from nag_ode_ivp_rkts_interp (d02psc).\n%s\n",
                   fail.message);
            exit_status = 3;
            goto END;
          }
          printf("%6.3f   %8.4f   %8.4f\n", twant, ywant[0], ypwant[0]);
          /* Set next required solution point. */
          twant = twant + tinc;
        }
      }
      /* Get diagnostics on whole integration using
       * nag_ode_ivp_rkts_diag (d02ptc).
       */
      nag_ode_ivp_rkts_diag(&fevals, &stepcost, &waste, &stepsok, &hnext,
                            iwsav, rwsav, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_ode_ivp_rkts_diag (d02ptc).\n%s\n",
               fail.message);
        exit_status = 4;
        goto END;
      }
      printf("Cost of the integration in evaluations of f is %6" NAG_IFMT
             "\n\n", fevals);
  }
END:
  NAG_FREE(thresh);
  NAG_FREE(yinit);
  NAG_FREE(ygot);
  NAG_FREE(ypgot);
  NAG_FREE(ywant);
  NAG_FREE(ypwant);
  NAG_FREE(rwsav);
  NAG_FREE(iwsav);
  NAG_FREE(wcomm);
  return exit_status;
}

static void NAG_CALL f(double t, Integer n, const double *y, double *yp,
                       Nag_Comm *comm)
{
  if (comm->user[0] == -1.0) {
    printf("(User-supplied callback f, first invocation.)\n");
    comm->user[0] = 0.0;
  }
  yp[0] = y[1];
  yp[1] = -y[0];
}
```

10.2 Program Data

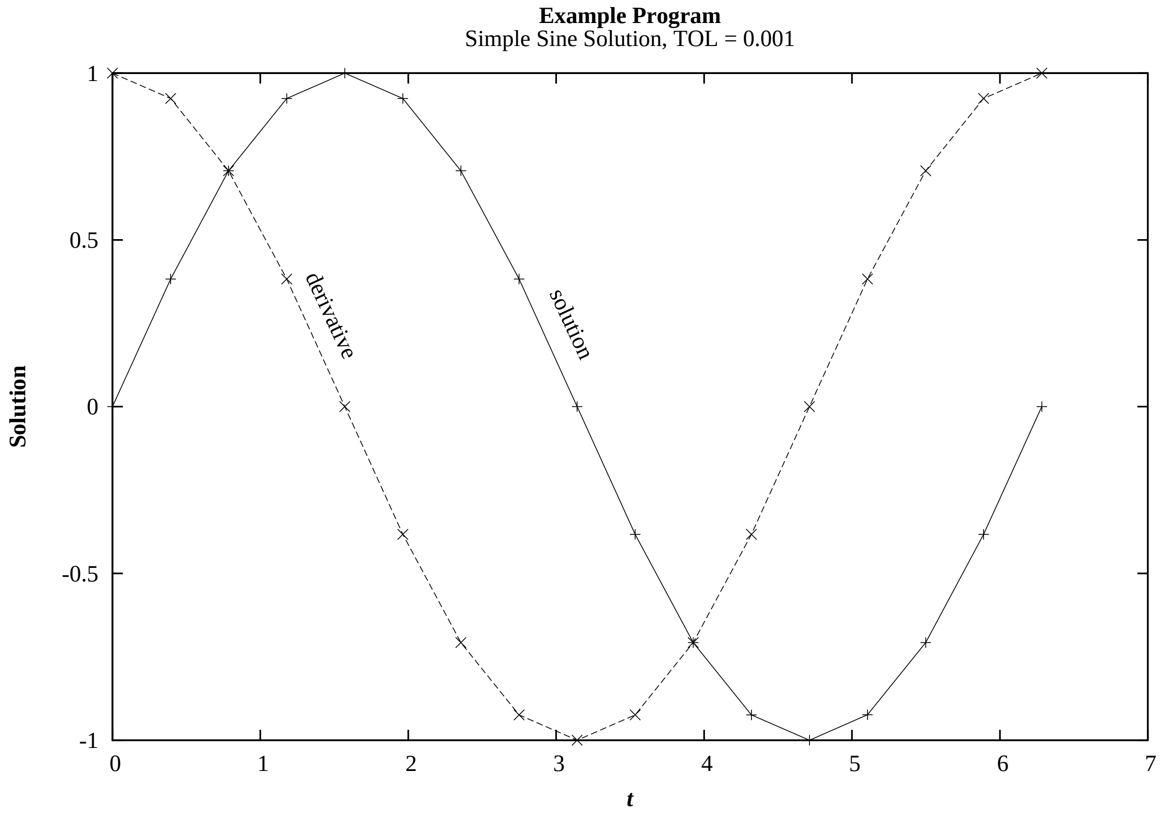
```
nag_ode_ivp_rkts_interp (d02psc) Example Program Data
   Nag_RK_2_3                          : method
   Nag_ErrorAssess_off                 : errass
   0.0     6.28318530717958647692 : tstart, tend
   0.0     1.0                         : yinit(1:n)
   0.0                                 : hstart
   1.0E-8  1.0E-8                      : thresh(1:n)
```

10.3 Program Results

```
nag_ode_ivp_rkts_interp (d02psc) Example Program Results

 Calculation with tol =  1.0e-04
    t          y1         y1'
 0.000     0.0000     1.0000
(User-supplied callback f, first invocation.)
 0.393     0.3827     0.9238
 0.785     0.7071     0.7070
 1.178     0.9238     0.3826
 1.571     0.9999    -0.0000
 1.963     0.9238    -0.3827
 2.356     0.7070    -0.7071
 2.749     0.3826    -0.9238
 3.142    -0.0000    -0.9998
 3.534    -0.3826    -0.9237
 3.927    -0.7070    -0.7069
 4.320    -0.9237    -0.3826
 4.712    -0.9998     0.0000
 5.105    -0.9236     0.3827
 5.498    -0.7069     0.7070
 5.890    -0.3825     0.9236
Cost of the integration in evaluations of f is    235

 Calculation with tol =  1.0e-05
    t          y1         y1'
 0.000     0.0000     1.0000
 0.393     0.3827     0.9239
 0.785     0.7071     0.7071
 1.178     0.9239     0.3827
 1.571     1.0000    -0.0000
 1.963     0.9239    -0.3827
 2.356     0.7071    -0.7071
 2.749     0.3827    -0.9239
 3.142    -0.0000    -1.0000
 3.534    -0.3827    -0.9239
 3.927    -0.7071    -0.7071
 4.320    -0.9239    -0.3827
 4.712    -1.0000     0.0000
 5.105    -0.9239     0.3827
 5.498    -0.7071     0.7071
 5.890    -0.3827     0.9239
Cost of the integration in evaluations of f is    493
```


Example Program
Simple Sine Solution, TOL = 0.001
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
Solution
t
derivative
solution