NAG Library Function Document

nag_ode_ivp_rk_range (d02pcc)

1 Purpose

nag_ode_ivp_rk_range (d02pcc) is a function for solving the initial value problem for a first order system of ordinary differential equations using Runge–Kutta methods.

2 Specification

```
#include <nag.h>
#include <nagd02.h>
void nag_ode_ivp_rk_range (Integer neq,
     void  (*f)(Integer neq, double t, const double y[], double yp[],
           Nag_User *comm),
     double twant, double *tgot, double ygot[], double ypgot[],
     double ymax[], Nag_ODE_RK *opt, Nag_User *comm, NagError *fail)
```

3 Description

nag_ode_ivp_rk_range (d02pcc) and its associated functions (nag_ode_ivp_rk_setup (d02pvc), 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.

This function is designed for the usual task, namely to compute an approximate solution at a sequence of points. You must first call nag_ode_ivp_rk_setup (d02pvc) to specify the problem and how it is to be solved. Thereafter you call nag_ode_ivp_rk_range (d02pcc) repeatedly with successive values of **twant**, the points at which you require the solution, in the range from **tstart** to **tend** (as specified in nag_ode_ivp_rk_setup (d02pvc)). In this manner nag_ode_ivp_rk_range (d02pcc) returns the point at which it has computed a solution **tgot** (usually **twant**), the solution there **ygot** and its derivative **ypgot**. If nag_ode_ivp_rk_range (d02pcc) encounters some difficulty in taking a step toward **twant**, then it returns the point of difficulty **tgot** and the solution and derivative computed there **ygot** and **ypgot**.

In the call to nag_ode_ivp_rk_setup (d02pvc) you can specify the first step size for nag_ode_ivp_rk_range (d02pcc) to attempt or that it compute automatically an appropriate value. Thereafter nag_ode_ivp_rk_range (d02pcc) estimates an appropriate step size for its next step. This value and other details of the integration can be obtained after any call to nag_ode_ivp_rk_range (d02pcc) by examining the contents of the structure **opt**, see Section 5. The local error is controlled at every step as specified in nag_ode_ivp_rk_setup (d02pvc). If you wish to assess the true error, you must set **errass** = Nag_ErrorAssess_on in the call to nag_ode_ivp_rk_setup (d02pvc). This assessment can be obtained after any call to nag_ode_ivp_rk_range (d02pcc) by a call to the function nag_ode_ivp_rk_errass (d02pzc).

For more complicated tasks, you are referred to functions nag_ode_ivp_rk_onestep (d02pdc), nag_ode_ivp_rk_interp (d02pxc) and nag_ode_ivp_rk_reset_tend (d02pwc).

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: **neq** – Integer *Input*

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

Constraint: **neq** ≥ 1.

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

f must evaluate the first derivatives y_i' (that is the functions f_i) for given values of the arguments t, y_i.

The specification of **f** is:

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

1: **neq** – Integer *Input*

On entry: the number of differential equations.

2: **t** – double *Input*

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

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

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

4: **yp**[**neq**] – double *Output*

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

5: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: 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.)

3: **twant** – double *Input*

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

Constraint: **twant** must be closer to **tend** than the previous of **tgot** (or **tstart** on the first call to nag_ode_ivp_rk_range (d02pcc)); see nag_ode_ivp_rk_setup (d02pvc) for a description of **tstart** and **tend**. **twant** must not lie beyond **tend** in the direction of integration.

4: **tgot** – double * *Output*

On exit: the value of the independent variable t at which a solution has been computed. On successful exit with **fail.code** = NE_NOERROR, **tgot** will equal **twant**. For non-trivial values of **fail** (i.e., those not related to an invalid call of nag_ode_ivp_rk_range (d02pcc)) a solution has still been computed at the value of **tgot** but in general **tgot** will not equal **twant**.

5: **ygot**[**neq**] – double *Input/Output*

On entry: on the first call to nag_ode_ivp_rk_range (d02pcc), **ygot** need not be set. On all subsequent calls **ygot** must remain unchanged.

On exit: an approximation to the true solution at the value of **tgot**. At each step of the integration to **tgot**, the local error has been controlled as specified in nag_ode_ivp_rk_setup (d02pvc). The local error has still been controlled even when **tgot** $\neq$ **twant**, that is after a return with a non-trivial error.

6: **ypgot**[**neq**] – double *Output*

On exit: an approximation to the first derivative of the true solution at **tgot**.

7: **ymax**[**neq**] – double *Input/Output*

On entry: on the first call to nag_ode_ivp_rk_range (d02pcc), **ymax** need not be set. On all subsequent calls **ymax** must remain unchanged.

On exit: **ymax**$[i-1]$ contains the largest value of $|y_i|$ computed at any step in the integration so far.

8: **opt** – Nag_ODE_RK *

Pointer to a structure of type Nag_ODE_RK as initialized by the setup function nag_ode_ivp_rk_setup (d02pvc) with the following members:

totfcn – Integer *Output*

On exit: the total number of evaluations of f used in the primary integration so far; this does not include evaluations of f for the secondary integration specified by a prior call to nag_ode_ivp_rk_setup (d02pvc) with **errass** = Nag_ErrorAssess_on.

stpcst – Integer *Output*

On exit: the cost in terms of number of evaluations of f of a typical step with the method being used for the integration. The method is specified by the argument **method** in a prior call to nag_ode_ivp_rk_setup (d02pvc).

waste – double *Output*

On exit: the number of attempted steps that failed to meet the local error requirement divided by the total number of steps attempted so far in the integration. A 'large' fraction indicates that the integrator is having trouble with the problem being solved. This can happen when the problem is 'stiff' and also when the solution has discontinuities in a low order derivative.

stpsok – Integer *Output*

On exit: the number of accepted steps.

hnext – double *Output*

On exit: the step size the integrator plans to use for the next step.

9: **comm** – Nag_User *

Pointer to a structure of type Nag_User with the following member:

p – Pointer

On entry/exit: the pointer **comm**→**p**, of type Pointer, allows you to communicate information to and from **f**. An object of the required type should be declared, e.g., a structure, and its address assigned to the pointer **comm**→**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.

10: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.6 in the Essential Introduction).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

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.

NE_MEMORY_FREED

Internally allocated memory has been freed by a call to nag_ode_ivp_rk_free (d02ppc) without a subsequent call to the setup function nag_ode_ivp_rk_setup (d02pvc).

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_NO_SETUP

The setup function nag_ode_ivp_rk_setup (d02pvc) has not been called.

NE_PREV_CALL

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

NE_PREV_CALL_INI

The previous call to the function nag_ode_ivp_rk_range (d02pcc) had resulted in a severe error. You must call nag_ode_ivp_rk_setup (d02pvc) to start another problem.

NE_RK_INVALID_CALL

The function to be called as specified in the setup function nag_ode_ivp_rk_setup (d02pvc) was nag_ode_ivp_rk_onestep (d02pdc). However the actual call was made to nag_ode_ivp_rk_range (d02pcc). This is not permitted.

NE_RK_PCC_METHOD

The efficiency of the integration has been degraded. Consider calling the setup function nag_ode_ivp_rk_setup (d02pvc) to re-initialize the integration at the current point with **method** = Nag_RK_4_5. Alternatively nag_ode_ivp_rk_range (d02pcc) can be called again to resume at the current point.

NE_RK_PDC_GLOBAL_ERROR_S

The global error assessment algorithm failed at the start of the integration.

NE_RK_PDC_GLOBAL_ERROR_T

The global error assessment may not be reliable for **t** past **tgot**. **tgot** = ⟨*value*⟩.

NE_RK_PDC_POINTS

More than 100 output points have been obtained by integrating to **tend**. They have been sufficiently close to one another that the efficiency of the integration has been degraded. It would probably be (much) more efficient to obtain output by interpolating with nag_ode_ivp_rk_interp (d02pxc) (after changing to **method** = Nag_RK_4_5 if you are using **method** = Nag_RK_7_8).

NE_RK_PDC_STEP

In order to satisfy the error requirements nag_ode_ivp_rk_range (d02pcc) would have to use a step size of ⟨*value*⟩ at current **t** = ⟨*value*⟩. This is too small for the ***machine precision***.

NE_RK_PDC_TEND

tend = ⟨*value*⟩ has been reached already. To integrate further with same problem the function nag_ode_ivp_rk_reset_tend (d02pwc) must be called with a new value of **tend**.

NE_RK_TGOT_EQ_TEND

The call to nag_ode_ivp_rk_range (d02pcc) has been made after reaching **tend**. The previous call to nag_ode_ivp_rk_range (d02pcc) resulted in **tgot** (**tstart** on the first call) = **tend**. You must call nag_ode_ivp_rk_setup (d02pvc) to start another problem.

NE_RK_TGOT_RANGE_TEND

The call to nag_ode_ivp_rk_range (d02pcc) has been made with a **twant** that does not lie between the previous value of **tgot** (**tstart** on the first call) and **tend**. This is not permitted.

NE_RK_TGOT_RANGE_TEND_CLOSE

The call to nag_ode_ivp_rk_range (d02pcc) has been made with a **twant** that does not lie between the previous value of **tgot** (**tstart** on the first call) and **tend**. This is not permitted. However **twant** is very close to **tend**, so you may have meant it to be **tend** exactly. Check your program.

NE_RK_TWANT_CLOSE_TGOT

The call to nag_ode_ivp_rk_range (d02pcc) has been made with a **twant** that is not sufficiently different from the last value of **tgot** (**tstart** on the first call). When using **method** = Nag_RK_7_8, it must differ by at least ⟨*value*⟩.

NE_STIFF_PROBLEM

The problem appears to be stiff.

NW_RK_TOO_MANY

Approximately ⟨*value*⟩ function evaluations have been used to compute the solution since the integration started or since this message was last printed.

7 Accuracy

The accuracy of integration is determined by the arguments **tol** and **thres** in a prior call to nag_ode_ivp_rk_setup (d02pvc). Note that only the local error at each step is controlled by these arguments. The error estimates obtained are not strict bounds but are usually reliable over one step. Over a number of steps the overall error may accumulate in various ways, depending on the properties of the differential system.

8 Parallelism and Performance

Not applicable.

9 Further Comments

If nag_ode_ivp_rk_range (d02pcc) returns with **fail.code** = NE_RK_PDC_STEP and the accuracy specified by **tol** and **thres** is really required then you should consider whether there is a more fundamental difficulty. For example, the solution may contain a singularity. In such a region the solution components will usually be of a large magnitude. Successive output values of **ygot** and **ymax** should be monitored (or the function nag_ode_ivp_rk_onestep (d02pdc) should be used since this takes one integration step at a time) with the aim of trapping the solution before the singularity. In any case numerical solution cannot be continued through a singularity, and analytical treatment may be necessary.

Performance statistics are available after any return from nag_ode_ivp_rk_range (d02pcc) by examining the structure **opt**, see Section 5. If **errass** = Nag_ErrorAssess_on in the call to nag_ode_ivp_rk_setup (d02pvc), global error assessment is available after any return from nag_ode_ivp_rk_range (d02pcc) (except when the error is due to incorrect input arguments or incorrect set up) by a call to the function nag_ode_ivp_rk_errass (d02pzc). The approximate extra number of evaluations of f used is given by $2 \times$ **opt→stpsok** $\times$ **opt→stpcst** for **method** = Nag_RK_4_5 or Nag_RK_7_8 and $3 \times$ **opt→stpsok** $\times$ **opt→stpcst** for **method** = Nag_RK_2_3.

After a failure with **fail.code** = NE_RK_PDC_STEP, NE_RK_PDC_GLOBAL_ERROR_T or NE_RK_PDC_GLOBAL_ERROR_S the diagnostic function nag_ode_ivp_rk_errass (d02pzc) may be called only once.

If nag_ode_ivp_rk_range (d02pcc) returns with **fail.code** = NE_STIFF_PROBLEM then it is advisable to change to another code more suited to the solution of stiff problems. nag_ode_ivp_rk_range (d02pcc) will not return with **fail.code** = NE_STIFF_PROBLEM if the problem is actually stiff but it is estimated that integration can be completed using less function evaluations than already computed.

10 Example

We solve the equation

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

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 and compute the solution at intervals of length $\pi/4$ across the range. We use a low order Runge–Kutta method (**method** = Nag_RK_2_3) with tolerances **tol** = 1.0e−3 and **tol** = 1.0e−4 in turn so that we may compare the solutions. The value of π is obtained by using nag_pi (X01AAC).

See also Section 10 in nag_ode_ivp_rk_errass (d02pzc).

10.1 Program Text

```
/* nag_ode_ivp_rk_range (d02pcc) Example Program.
 *
 * Copyright 1992 Numerical Algorithms Group.
 *
 * Mark 3, 1992.
 * Mark 7 revised, 2001.
 * Mark 8 revised, 2004.
 *
 */

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

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL f(Integer neq, double t1, const double y[], double yp[],
                       Nag_User *comm);
#ifdef __cplusplus
}
#endif

#define NEQ  2
```

```
#define ZERO 0.0
#define ONE  1.0
#define TWO  2.0
#define FOUR 4.0

int main(void)
{
  static Integer use_comm[1] = {1};
  Integer         exit_status = 0, i, j, neq, nout;
  NagError        fail;
  Nag_ErrorAssess errass;
  Nag_ODE_RK      opt;
  Nag_RK_method   method;
  Nag_User        comm;
  double          hstart, pi, tend, tgot, *thres = 0, tinc, tol, tstart, twant,
  *ygot = 0;
  double          *ymax = 0, *ypgot = 0, *ystart = 0;

  INIT_FAIL(fail);

  printf("nag_ode_ivp_rk_range (d02pcc) Example Program Results\n");

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

  /* Set initial conditions and input for nag_ode_ivp_rk_setup (d02pvc) */
  neq = NEQ;
  if (neq >= 1)
    {
      if (!(thres = NAG_ALLOC(neq, double)) ||
          !(ygot = NAG_ALLOC(neq, double)) ||
          !(ymax = NAG_ALLOC(neq, double)) ||
          !(ypgot = NAG_ALLOC(neq, double)) ||
          !(ystart = NAG_ALLOC(neq, double)))
        {
          printf("Allocation failure\n");
          exit_status = -1;
          goto END;
        }
    }
  else
    {
      exit_status = 1;
      return exit_status;
    }

  /* nag_pi (x01aac).
   * pi
   */
  pi = nag_pi;
  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;
  method = Nag_RK_2_3;

  /*
   * Set control for output
   */
  nout = 8;
  tinc = (tend-tstart)/nout;

  for (i = 1; i <= 2; i++)
    {
      if (i == 1)
        tol = 1.0e-3;
      else
```

```
        tol = 1.0e-4;
      /* nag_ode_ivp_rk_setup (d02pvc).
       * Setup function for use with nag_ode_ivp_rk_range (d02pcc)
       * and/or nag_ode_ivp_rk_onestep (d02pdc)
       */
      nag_ode_ivp_rk_setup(neq, tstart, ystart, tend, tol, thres, method,
                           Nag_RK_range, errass, hstart, &opt, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_ode_ivp_rk_setup (d02pvc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }
      printf("\nCalculation with tol = %10.1e\n\n", tol);
      printf("    t          y1         y2\n\n");
      printf("%8.3f   %8.3f   %8.3f\n", tstart, ystart[0], ystart[1]);
      for (j = nout-1; j >= 0; j--)
        {
          twant = tend - j*tinc;
          /* nag_ode_ivp_rk_range (d02pcc).
           * Ordinary differential equations solver, initial value
           * problems over a range using Runge-Kutta methods
           */
          nag_ode_ivp_rk_range(neq, f, twant, &tgot, ygot, ypgot, ymax, &opt,
                               &comm, &fail);
          if (fail.code != NE_NOERROR)
            {
              printf("Error from nag_ode_ivp_rk_range (d02pcc).\n%s\n",
                      fail.message);
              exit_status = 1;
              goto END;
            }

          printf("%8.3f   %8.3f   %8.3f\n", tgot, ygot[0], ygot[1]);
        }
      printf("\nCost of the integration in evaluations of f is"
              " %ld\n\n", opt.totfcn);
      /* nag_ode_ivp_rk_free (d02ppc).
       * Freeing function for use with the Runge-Kutta suite (d02p
       * functions)
       */
      nag_ode_ivp_rk_free(&opt);
    }
 END:
  NAG_FREE(thres);
  NAG_FREE(ygot);
  NAG_FREE(ymax);
  NAG_FREE(ypgot);
  NAG_FREE(ystart);
  return exit_status;
}
static void NAG_CALL f(Integer neq, double t, const double y[], double yp[],
                       Nag_User *comm)
{
  Integer *use_comm = (Integer *)comm->p;

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

  yp[0] = y[1];
  yp[1] = -y[0];
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_ode_ivp_rk_range (d02pcc) Example Program Results

Calculation with tol =    1.0e-03

    t          y1         y2

   0.000      0.000      1.000
(User-supplied callback f, first invocation.)
   0.785      0.707      0.707
   1.571      0.999     -0.000
   2.356      0.706     -0.706
   3.142     -0.000     -0.999
   3.927     -0.706     -0.706
   4.712     -0.998      0.000
   5.498     -0.705      0.706
   6.283      0.001      0.997

Cost of the integration in evaluations of f is 124


Calculation with tol =    1.0e-04

    t          y1         y2

   0.000      0.000      1.000
   0.785      0.707      0.707
   1.571      1.000     -0.000
   2.356      0.707     -0.707
   3.142     -0.000     -1.000
   3.927     -0.707     -0.707
   4.712     -1.000      0.000
   5.498     -0.707      0.707
   6.283      0.000      1.000

Cost of the integration in evaluations of f is 235
```