NAG Library Function Document

nag_ode_ivp_rk_onestep (d02pdc)

1 Purpose

nag_ode_ivp_rk_onestep (d02pdc) is a one-step 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_onestep (Integer neq,
     void (*f)(Integer neq, double t, const double y[], double yp[],
          Nag_User *comm),
     double *tnow, double ynow[], double ypnow[], Nag_ODE_RK *opt,
     Nag_User *comm, NagError *fail)
```

3 Description

nag_ode_ivp_rk_onestep (d02pdc) and its associated functions (nag_ode_ivp_rk_setup (d02pvc), nag_ode_ivp_rk_reset_tend (d02pwc), nag_ode_ivp_rk_interp (d02pxc) and 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 to be used in complicated tasks when solving systems of ordinary differential equations. You must first call nag_ode_ivp_rk_setup (d02pvc) to specify the problem and how it is to be solved. Thereafter you (repeatedly) call nag_ode_ivp_rk_onestep (d02pdc) to take one integration step at a time from **tstart** in the direction of **tend** (as specified in nag_ode_ivp_rk_setup (d02pvc)). In this manner nag_ode_ivp_rk_onestep (d02pdc) returns an approximation to the solution **ynow** and its derivative **ypnow** at successive points **tnow**. If nag_ode_ivp_rk_onestep (d02pdc) encounters some difficulty in taking a step, the integration is not advanced and the function returns with the same values of **tnow**, **ynow** and **ypnow** as returned on the previous successful step. nag_ode_ivp_rk_onestep (d02pdc) tries to advance the integration as far as possible subject to passing the test on the local error and not going past **tend**. In the call to nag_ode_ivp_rk_setup (d02pvc) you can specify either the first step size for nag_ode_ivp_rk_onestep (d02pdc) to attempt or that it compute automatically an appropriate value. Thereafter nag_ode_ivp_rk_onestep (d02pdc) 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_onestep (d02pdc) 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_onestep (d02pdc) by a call to the subroutine nag_ode_ivp_rk_errass (d02pzc).

If you want answers at specific points there are two ways to proceed:

The more efficient way is to step past the point where a solution is desired, and then call nag_ode_ivp_rk_interp (d02pxc) to get an answer there. Within the span of the current step, you can get all the answers you want at very little cost by repeated calls to nag_ode_ivp_rk_interp (d02pxc). This is very valuable when you want to find where something happens, e.g., where a particular solution component vanishes. You cannot proceed in this way with **method** = Nag_RK_7_8.

The other way to get an answer at a specific point is to set **tend** to this value and integrate to **tend**. nag_ode_ivp_rk_onestep (d02pdc) will not step past **tend**, so when a step would carry it past, it will reduce the step size so as to produce an answer at **tend** exactly. After getting an answer there (**tnow** = **tend**), you can reset **tend** to the next point where you want an answer, and repeat. **tend** could be reset by a call to nag_ode_ivp_rk_setup (d02pvc), but you should not do this. You should use nag_ode_ivp_rk_reset_tend (d02pwc) because it is both easier to use and much more efficient. This way of getting answers at specific points can be used with any of the available methods, but it is the only way with **method** = Nag_RK_7_8. It can be inefficient. Should this be the case, the code will bring the matter to your attention.

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: **tnow** – double * *Output*

On exit: the value of the independent variable t at which a solution has been computed.

4: **ynow**[**neq**] – double *Output*

On exit: an approximation to the solution at **tnow**. The local error of the step to **tnow** was no greater than permitted by the specified tolerances (see nag_ode_ivp_rk_setup (d02pvc)).

5: **ypnow**[**neq**] – double *Output*

On exit: an approximation to the derivative of the solution at **tnow**.

6: **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.

7: **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.

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

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

6 Error Indicators and Warnings

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_onestep (d02pdc) 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_range (d02pcc). However the actual call was made to nag_ode_ivp_rk_onestep (d02pdc). This is not permitted.

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 **tnow**. **tnow** = ⟨*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_onestep (d02pdc) 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_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_onestep (d02pdc) 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 **ynow** should be monitored with the aim of trapping the solution before the singularity. In any case numerical integration cannot be continued through a singularity, and analytical treatment may be necessary.

If nag_ode_ivp_rk_onestep (d02pdc) returns with a non-trivial value of **fail** (i.e., those not related to an invalid call) then performance statistics are available by examining the structure **opt** (see Section 5). Furthermore if **errass** = Nag_ErrorAssess_on then global error assessment is available by a call to the function nag_ode_ivp_rk_errass (d02pzc). The approximate extra number of evaluations of f used is given by $2 \times$ **stpsok** $\times$ **stpcst** for **method** = Nag_RK_4_5 or Nag_RK_7_8 and $3 \times$ **stpsok** $\times$ **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_onestep (d02pdc) 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_onestep (d02pdc) 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 print the solution at each integration step across the range. We use a medium order Runge–Kutta method (**method** = Nag_RK_4_5) with tolerances **tol** = 1.0e−4 and **tol** = 1.0e−5 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_reset_tend (d02pwc) and nag_ode_ivp_rk_interp (d02pxc).

10.1 Program Text

```
/* nag_ode_ivp_rk_onestep (d02pdc) Example Program.
 *
 * Copyright 2014 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, neq;
  NagError        fail;
  Nag_ErrorAssess errass;
  Nag_ODE_RK      opt;
  Nag_RK_method   method;
  Nag_User        comm;
  double          hstart, pi, tend, *thres = 0, tnow, tol, tstart, *ynow = 0,
  *ypnow = 0;
  double          *ystart = 0;

  INIT_FAIL(fail);

  printf("nag_ode_ivp_rk_onestep (d02pdc) Example Program Results\n");

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

  /* Set initial conditions and input for nag_ode_ivp_rk_setup (d02pvc) */
  neq = NEQ;
  if (neq >= 1)
    {
      if (!(thres = NAG_ALLOC(neq, double)) ||
          !(ynow = NAG_ALLOC(neq, double)) ||
          !(ypnow = 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_4_5;

for (i = 1; i <= 2; i++)
  {
    if (i == 1)
      tol = 1.0e-4;
    else
      tol = 1.0e-5;
    /* 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_onestep, 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]);
    do
      {
        /* nag_ode_ivp_rk_onestep (d02pdc).
         * Ordinary differential equations solver, initial value
         * problems, one time step using Runge-Kutta methods
         */
        nag_ode_ivp_rk_onestep(neq, f, &tnow, ynow, ypnow, &opt, &comm,
                               &fail);
        if (fail.code != NE_NOERROR)
          {
            printf(
                    "Error from nag_ode_ivp_rk_onestep (d02pdc).\n%s\n",
                    fail.message);
            exit_status = 1;
            goto END;
          }
        printf("%8.3f   %8.3f   %8.3f\n", tnow, ynow[0], ynow[1]);
      } while (tnow < tend);

    printf("\nCost of the integration in evaluations of f is"
            " %"NAG_IFMT"\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(ynow);
 NAG_FREE(ypnow);
```

```
  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_onestep (d02pdc) Example Program Results

Calculation with tol =    1.0e-04

    t          y1         y2

   0.000      0.000      1.000
(User-supplied callback f, first invocation.)
   0.785      0.707      0.707
   1.519      0.999      0.051
   2.282      0.757     -0.653
   2.911      0.229     -0.974
   3.706     -0.535     -0.845
   4.364     -0.940     -0.341
   5.320     -0.821      0.571
   5.802     -0.463      0.886
   6.283      0.000      1.000

Cost of the integration in evaluations of f is 78


Calculation with tol =    1.0e-05

    t          y1         y2

   0.000      0.000      1.000
   0.393      0.383      0.924
   0.785      0.707      0.707
   1.416      0.988      0.154
   1.870      0.956     -0.294
   2.204      0.806     -0.592
   2.761      0.371     -0.929
   3.230     -0.088     -0.996
   3.587     -0.430     -0.903
   4.022     -0.771     -0.637
   4.641     -0.997     -0.072
   5.152     -0.905      0.426
   5.521     -0.690      0.724
   5.902     -0.372      0.928
   6.283      0.000      1.000

Cost of the integration in evaluations of f is 118
```