

# NAG Library Function Document

## nag\_ode\_ivp\_rkts\_range (d02pec)

### 1 Purpose

nag\_ode\_ivp\_rkts\_range (d02pec) solves an 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_rkts_range (
    void (*f)(double t, Integer n, const double y[], double yp[],
              Nag_Comm *comm),
    Integer n, double twant, double *tgot, double ygot[], double ypgot[],
    double ymax[], Nag_Comm *comm, Integer iwsav[], double rwsav[],
    NagError *fail)
```

### 3 Description

nag\_ode\_ivp\_rkts\_range (d02pec) and its associated functions (nag\_ode\_ivp\_rkts\_setup (d02pqc), nag\_ode\_ivp\_rkts\_diag (d02ptc) and nag\_ode\_ivp\_rkts\_errass (d02puc)) solve an 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\_range (d02pec) is designed for the usual task, namely to compute an approximate solution at a sequence of points. You must first call nag\_ode\_ivp\_rkts\_setup (d02pqc) to specify the problem and how it is to be solved. Thereafter you call nag\_ode\_ivp\_rkts\_range (d02pec) 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\_rkts\_setup (d02pqc)). In this manner nag\_ode\_ivp\_rkts\_range (d02pec) 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\_rkts\_range (d02pec) 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**, respectively).

In the call to nag\_ode\_ivp\_rkts\_setup (d02pqc) you can specify either the first step size for nag\_ode\_ivp\_rkts\_range (d02pec) to attempt or that it computes automatically an appropriate value. Thereafter nag\_ode\_ivp\_rkts\_range (d02pec) 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\_rkts\_range (d02pec) by a call to nag\_ode\_ivp\_rkts\_diag (d02ptc). The local error is controlled at every step as specified in nag\_ode\_ivp\_rkts\_setup (d02pqc). If you wish to assess the true error, you must set **errass** = Nag\_ErrorAssess\_on in the call to nag\_ode\_ivp\_rkts\_setup (d02pqc). This assessment can be obtained after any call to nag\_ode\_ivp\_rkts\_range (d02pec) by a call to nag\_ode\_ivp\_rkts\_errass (d02puc).

For more complicated tasks, you are referred to functions nag\_ode\_ivp\_rkts\_onestep (d02pfc), nag\_ode\_ivp\_rkts\_reset\_tend (d02prc) and nag\_ode\_ivp\_rkts\_interp (d02psc), all of which are used by nag\_ode\_ivp\_rkts\_range (d02pec).

## 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: **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  $t, y_i$ .

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  $i = 1, 2, \dots, n$ .

4: **yp[n]** – double *Output*  
*On exit:* the values of  $f_i$ , for  $i = 1, 2, \dots, n$ .

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\_range (d02pec) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag\_ode\_ivp\_rkts\_range (d02pec) (see Section 2.3.1.1 in How to Use the NAG Library and its Documentation).

2: **n** – Integer *Input*  
*On entry:*  $n$ , the number of ordinary differential equations in the system to be solved.  
*Constraint:*  $n \geq 1$ .

3: **twant** – double *Input*  
*On entry:*  $t$ , the next value of the independent variable where a solution is desired.  
*Constraint:* **twant** must be closer to **tend** than the previous value of **tgot** (or **tstart** on the first call to nag\_ode\_ivp\_rkts\_range (d02pec)); see nag\_ode\_ivp\_rkts\_setup (d02pqc) for a description of **tstart** and **tend**. **twant** must not lie beyond **tend** in the direction of integration.

4: **tgot** – double \* *Output*  
*On exit:*  $t$ , the value of the independent variable at which a solution has been computed. On successful exit with **fail.code** = NE\_NOERROR, **tgot** will equal **twant**. On exit with **fail.code** =

NE\_RK\_GLOBAL\_ERROR\_S, NE\_RK\_GLOBAL\_ERROR\_T, NE\_RK\_POINTS, NE\_RK\_STEP\_TOO\_SMALL, NE\_STIFF\_PROBLEM or NW\_RK\_TOO\_MANY, a solution has still been computed at the value of **tgot** but in general **tgot** will not equal **twant**.

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

*On entry:* on the first call to nag\_ode\_ivp\_rkts\_range (d02pec), **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\_rkts\_setup (d02pec). The local error has still been controlled even when **tgot**  $\neq$  **twant**, that is after a return with **fail.code** = NE\_RK\_GLOBAL\_ERROR\_S, NE\_RK\_GLOBAL\_ERROR\_T, NE\_RK\_POINTS, NE\_RK\_STEP\_TOO\_SMALL, NE\_STIFF\_PROBLEM or NW\_RK\_TOO\_MANY.

6: **ypgot**[**n**] – double *Output*

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

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

*On entry:* on the first call to nag\_ode\_ivp\_rkts\_range (d02pec), **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: **comm** – Nag\_Comm \*

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

9: **iwsav**[130] – Integer *Communication Array*

10: **rwsav**[ $32 \times \mathbf{n} + 350$ ] – double *Communication Array*

*On entry:* these must be the same arrays supplied in a previous call to nag\_ode\_ivp\_rkts\_setup (d02pec). They must remain unchanged between calls.

*On exit:* information about the integration for use on subsequent calls to nag\_ode\_ivp\_rkts\_range (d02pec) or other associated functions.

11: **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 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

On entry, argument  $\langle value \rangle$  had an illegal value.

### NE\_INT\_CHANGED

On entry,  $\mathbf{n} = \langle value \rangle$ , but the value passed to the setup function was  $\mathbf{n} = \langle value \rangle$ .

**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 2.7.6 in How to Use the NAG Library and its Documentation for further information.

**NE\_MISSING\_CALL**

On entry, a previous call to the setup function has not been made or the communication arrays have become corrupted.

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.  
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

**NE\_PREV\_CALL**

On entry, 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 it has returned an error.  
You must call the setup function to start another problem.

**NE\_RK\_GLOBAL\_ERROR\_S**

The global error assessment algorithm failed at start of integration.  
The integration is being terminated.

**NE\_RK\_GLOBAL\_ERROR\_T**

The global error assessment may not be reliable for times beyond  $\langle value \rangle$ .  
The integration is being terminated.

**NE\_RK\_INVALID\_CALL**

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

**NE\_RK\_POINTS**

This function is being used inefficiently because the step size has been reduced drastically many times to obtain answers at many points. Using the order 4 and 5 pair method at setup is more appropriate here. You can continue integrating this problem.

**NE\_RK\_STEP\_TOO\_SMALL**

In order to satisfy your error requirements the solver has to use a step size of  $\langle value \rangle$  at the current time,  $\langle value \rangle$ . This step size is too small for the *machine precision*, and is smaller than  $\langle value \rangle$ .

**NE\_RK\_TGOT\_EQ\_TEND**

**tend** (setup) had already been reached in a previous call.  
To start a new problem, you will need to call the setup function.

**NE\_RK\_TGOT\_RANGE\_TEND**

**twant** does not lie in the direction of integration. **twant** =  $\langle value \rangle$ .  
**twant** lies beyond **tend** (setup) in the direction of integration.  
**twant** =  $\langle value \rangle$  and **tend** =  $\langle value \rangle$ .

**NE\_RK\_TGOT\_RANGE\_TEND\_CLOSE**

**twant** lies beyond **tend** (setup) in the direction of integration, but is very close to **tend**. You may have intended **twant = tend**.  
 $|\mathbf{twant} - \mathbf{tend}| = \langle \mathit{value} \rangle$ .

**NE\_RK\_TWANT\_CLOSE\_TGOT**

**twant** is too close to the last value of **tgot** (**tstart** on setup). When using the method of order 8 at setup, these must differ by at least  $\langle \mathit{value} \rangle$ . Their absolute difference is  $\langle \mathit{value} \rangle$ .

**NE\_STIFF\_PROBLEM**

Approximately  $\langle \mathit{value} \rangle$  function evaluations have been used to compute the solution since the integration started or since this message was last printed. Your problem has been diagnosed as stiff. If the situation persists, it will cost roughly  $\langle \mathit{value} \rangle$  times as much to reach **tend** (setup) as it has cost to reach the current time. You should probably call functions intended for stiff problems. However, you can continue integrating the problem.

**NW\_RK\_TOO\_MANY**

Approximately  $\langle \mathit{value} \rangle$  function evaluations have been used to compute the solution since the integration started or since this message was last printed. However, you can continue integrating the problem.

**7 Accuracy**

The accuracy of integration is determined by the arguments **tol** and **thresh** in a prior call to `nag_ode_ivp_rkts_setup` (d02pqc) (see the function document for `nag_ode_ivp_rkts_setup` (d02pqc) for further details and advice). 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**

`nag_ode_ivp_rkts_range` (d02pec) 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' Note for your implementation for any additional implementation-specific information.

**9 Further Comments**

If `nag_ode_ivp_rkts_range` (d02pec) returns with **fail.code** = `NE_RK_STEP_TOO_SMALL` and the accuracy specified by **tol** and **thresh** 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 large in magnitude. Successive output values of **ygot** and **ymax** should be monitored (or `nag_ode_ivp_rkts_onestep` (d02pfc) 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 integration cannot be continued through a singularity, and analytical treatment may be necessary.

Performance statistics are available after any return from `nag_ode_ivp_rkts_range` (d02pec) by a call to `nag_ode_ivp_rkts_diag` (d02ptc). If **errass** = `Nag_ErrorAssess_on` in the call to `nag_ode_ivp_rkts_setup` (d02pqc), global error assessment is available after a return from `nag_ode_ivp_rkts_range` (d02pec) with **fail.code** = `NE_NOERROR`, `NE_RK_GLOBAL_ERROR_S`, `NE_RK_GLOBAL_ERROR_T`,

NE\_RK\_POINTS, NE\_RK\_STEP\_TOO\_SMALL, NE\_STIFF\_PROBLEM or NW\_RK\_TOO\_MANY by a call to nag\_ode\_ivp\_rkts\_errass (d02puc).

After a failure with **fail.code** = NE\_RK\_GLOBAL\_ERROR\_S, NE\_RK\_GLOBAL\_ERROR\_T or NE\_RK\_STEP\_TOO\_SMALL each of the diagnostic functions nag\_ode\_ivp\_rkts\_diag (d02ptc) and nag\_ode\_ivp\_rkts\_errass (d02puc) may be called only once.

If nag\_ode\_ivp\_rkts\_range (d02pec) 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\_rkts\_range (d02pec) 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

This example solves the equation

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

reposed as

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

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 and compute the solution at intervals of length  $\pi/4$  across the range. A low-order Runge–Kutta method (see nag\_ode\_ivp\_rkts\_setup (d02pqc)) is also used with tolerances **tol** =  $1.0e-3$  and **tol** =  $1.0e-4$  in turn so that the solutions can be compared.

See also Section 10 in nag\_ode\_ivp\_rkts\_errass (d02puc).

### 10.1 Program Text

```

/* nag_ode_ivp_rkts_range (d02pec) 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 = 8, exit_status = 0;
    Integer liwsav, lrwsav, 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, *ymax = 0;
double *ypgot = 0;
Integer *iwsav = 0;
char nag_enum_arg[40];
/* NAG types */
NagError fail;
Nag_RK_method method;
Nag_ErrorAssess errass;
Nag_Comm comm;

INIT_FAIL(fail);

n = N;
liwsav = 130;
lrwsav = 350 + 32 * n;

printf("nag_ode_ivp_rkts_range (d02pec) 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)) ||
    !(ymax = NAG_ALLOC(n, double)) ||
    !(iwsav = NAG_ALLOC(liwsav, Integer)) ||
    !(rwsav = NAG_ALLOC(lrwsav, 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
#endif

```

```

    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

/* Set control for output */
tinc = (tend - tstart) / (double) (npts);
tol = 10.0 * 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          y2\n");
    printf("%6.3f", tstart);
    for (k = 0; k < n; k++)
        printf("    %7.3f", yinit[k]);
    printf("\n");

    twant = tstart;
    for (j = 0; j < npts; j++) {
        twant = twant + tinc;
        /* Solve ODE by Runge-Kutta method up to next time increment using
         * nag_ode_ivp_rkts_range (d02pec).
         */
        nag_ode_ivp_rkts_range(f, n, twant, &tgot, ygot, ypgot, ymax, &comm,
                              iwsav, rwsav, &fail);
        if (fail.code != NE_NOERROR) {
            printf("Error from nag_ode_ivp_rkts_range (d02pec).\n%s\n",
                  fail.message);
            exit_status = 2;
            goto END;
        }

        printf("%6.3f", tgot);
        for (k = 0; k < n; k++)
            printf("    %7.3f", ygot[k]);
        printf("\n");
    }
    /* 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 = 3;
        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(ymax);
    NAG_FREE(rwsav);
    NAG_FREE(iwsav);
    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_range (d02pec) 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_range (d02pec) 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

**Example Program**  
 First-order ODEs using Runge-Kutta  
 Low-order Method using Two Tolerances

