

NAG Library Function Document

nag_ode_ivp_rk_step_revcomm (d02pgc)

1 Purpose

nag_ode_ivp_rk_step_revcomm (d02pgc) is a reverse communication one-step function for solving an initial value problem for a first-order system of ordinary differential equations using Runge–Kutta methods. The direct communication version of this function is nag_ode_ivp_rkts_onestep (d02pfc). See Section 2.3.2 in How to Use the NAG Library and its Documentation for the difference between forward and reverse communication.

2 Specification

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

void nag_ode_ivp_rk_step_revcomm (Integer *irevcm, Integer n, double *t,
    double y[], const double yp[], Integer iwsav[], double rwsav[],
    NagError *fail)
```

3 Description

nag_ode_ivp_rk_step_revcomm (d02pgc) and its associated functions (nag_ode_ivp_rk_interp_setup (d02phc), nag_ode_ivp_rk_interp_eval (d02pjc), 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 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_rk_step_revcomm (d02pgc) is designed to be used in complicated tasks when solving systems of ordinary differential equations. You must first call nag_ode_ivp_rkts_setup (d02pqc) to specify the problem and how it is to be solved. Thereafter you (repeatedly) call nag_ode_ivp_rk_step_revcomm (d02pgc) in reverse communication loops to take one integration step at a time from **tstart** in the direction of **tend** (as specified in nag_ode_ivp_rkts_setup (d02pqc)). In this manner nag_ode_ivp_rk_step_revcomm (d02pgc) returns an approximation to the solution **y** and its derivative **yp** at successive points **t**. If nag_ode_ivp_rk_step_revcomm (d02pgc) encounters some difficulty in taking a step, the integration is not advanced and the function returns with the same values of **t**, **y** and **yp** as returned on the previous successful step. nag_ode_ivp_rk_step_revcomm (d02pgc) 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_rkts_setup (d02pqc) you can specify either the first step size for nag_ode_ivp_rk_step_revcomm (d02pgc) to attempt or it computes automatically an appropriate value. Thereafter nag_ode_ivp_rk_step_revcomm (d02pgc) estimates an appropriate step size for its next step. This value and other details of the integration can be obtained after a completed step by nag_ode_ivp_rk_step_revcomm (d02pgc) 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_rk_step_revcomm (d02pgc) by a call to nag_ode_ivp_rkts_errass (d02puc).

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

- (i) The more efficient way is to step past the point where a solution is desired, and then call nag_ode_ivp_rk_interp_setup (d02phc) and nag_ode_ivp_rk_interp_eval (d02pjc) 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_eval` (d02pjc). This is very valuable when you want to find where something happens, e.g., where a particular solution component vanishes.

- (ii) Alternatively, set **tend** to the desired value and integrate to **tend**. `nag_ode_ivp_rk_step_revcomm` (d02pgc) 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 (**t = 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_rkts_setup` (d02pgc), but you should not do this. You should use `nag_ode_ivp_rkts_reset_tend` (d02prc) instead 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 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

Note: this function uses **reverse communication**. Its use involves an initial entry, intermediate exits and re-entries, and a final exit, as indicated by the argument **irevcm**. Between intermediate exits and re-entries, **all arguments other than those specified by the value of irevcm must remain unchanged**.

1: **irevcm** – Integer * *Input/Output*

On initial entry: **irevcm** must be set to zero to indicate that a new step is being taken.

On intermediate re-entry: **irevcm** should remain unchanged.

On intermediate exit: **irevcm** returns a value > 0 to indicate that a function evaluation is required prior to re-entry; the value of the derivatives $y' = f(t, y)$ must be returned in **yp** where the value of t is supplied in **t** and the values $y(t)$ are supplied in the array **y**. The value of **irevcm** indicates the reason for the function evaluation as follows:

irevcm = 1

For initial entry values of **t** and **y**.

irevcm = 2

To determine stiffness of system.

irevcm = 3

For the stages of the primary step.

irevcm = 4

A final stage of the primary step.

irevcm = 5

For the stages of a secondary step (if global error assessment is required).

On final exit:

irevcm = -1

Successful exit; **t**, **y** and **yp** contain the solution at the end of a successful integration step.

irevcm = -2

Error exit; **fail** should be interrogated to determine the nature of the error.

2: **n** – Integer *Input*

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

Constraint: $n \geq 1$. This must be the same value as supplied in a previous call to `nag_ode_ivp_rkts_setup` (d02pgc).

- 3: **t** – double * *Output*
On intermediate exit: **t** contains the value of the independent variable t at which the derivatives y' are to be evaluated.
On final exit: the value of t at which a solution has been computed following a successful step.
- 4: **y[n]** – double *Output*
On intermediate exit: **y** contains the value of the solution y at which the derivatives y' are to be evaluated.
On final exit: the approximation to the solution computed following a successful step.
- 5: **yp[n]** – const double *Input*
On initial entry: **yp** need not be set.
On intermediate re-entry: **yp** must contain the value of the derivatives $y' = f(t, y)$ where t is supplied in **t** and y is supplied in the array **y**.
- 6: **iwsav[130]** – Integer *Communication Array*
7: **rwsav[32 × n + 350]** – double *Communication Array*
On entry: these must be the same arrays supplied in a previous call to nag_ode_ivp_rkts_setup (d02pgc). They must remain unchanged between calls.
On exit: information about the integration for use on subsequent calls to nag_ode_ivp_rk_step_revcomm (d02pgc) or other associated functions.
- 8: **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

irevcm < 0 on entry.

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

NE_INT_CHANGED

On entry, **n** = $\langle value \rangle$, but the value passed to the setup function was **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 3.6.6 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 3.6.5 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

A call to this function cannot be made after it has returned an error.
The setup function must be called 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_POINTS

More than 100 output points have been obtained by integrating to **tend** (as specified in the setup function). They have been so clustered that it would probably be (much) more efficient to use the interpolation function. However, you can continue integrating the 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, as specified in the setup function, has already been reached. To start a new problem, you will need to call the setup function. To continue integration beyond **tend** then `nag_ode_ivp_rkts_reset_tend (d02prc)` must first be called to reset **tend** to a new end value.

NE_STIFF_PROBLEM

Approximately $\langle value \rangle$ 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 $\langle value \rangle$ 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 $\langle value \rangle$ 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_rk_step_revcomm` (d02pgc) 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_rk_step_revcomm` (d02pgc) returns 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` then the values returned in **t** and **y** are for the last successful step, or the initial conditions supplied if no successful step has been taken.

If `nag_ode_ivp_rk_step_revcomm` (d02pgc) 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 **y** 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.

Performance statistics are available after any return from `nag_ode_ivp_rk_step_revcomm` (d02pgc) (except when **fail.code** = `NE_BAD_PARAM`, `NE_INT_CHANGED`, `NE_MISSING_CALL`, `NE_PREV_CALL`, `NE_PREV_CALL_INI` or `NE_RK_TGOT_EQ_TEND`) 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 any return from `nag_ode_ivp_rk_step_revcomm` (d02pgc) (except when **fail.code** = `NE_BAD_PARAM`, `NE_INT_CHANGED`, `NE_MISSING_CALL`, `NE_PREV_CALL`, `NE_PREV_CALL_INI` or `NE_RK_TGOT_EQ_TEND`) 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_rk_step_revcomm` (d02pgc) 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_step_revcomm` (d02pgc) 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 $y_2 = 1.0$. We use relative error control with threshold values of $1.0e-8$ for each solution component and print the solution at regular intervals using the interpolation functions `nag_ode_ivp_rk_interp_setup` (d02phc) and `nag_ode_ivp_rk_interp_eval` (d02pje) within integration steps across the range; points on the range at which y_1 or y_2 change sign are

also evaluated using a combination of the root finding function `nag_zero_cont_func_brent_rcomm` (c05azc) and the interpolation functions. We use a medium order Runge–Kutta method (`method = Nag_RK_4_5`) with tolerance `tol = 1.0e-5`.

10.1 Program Text

```

/* nag_ode_ivp_rk_step_revcomm (d02pgc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagc05.h>
#include <nagd02.h>

int main(void)
{
    /* Constants */
    double      const tol = 1.0e-5;
    Integer     const n = 2;
    Integer     const liwsav = 130;

    /* Scalars */
    double      hnext, hstart, t, t1, t2, tend, tnow, tout, tprev, waste;
    Integer     ind, irevcm, j, k, nchange, stepcost,
                stepsok, totf, lrwsav, lwcomm, exit_status = 0;

    /* Arrays */
    double      c[17];
    double      *rwsav = 0, *thresh = 0, *troot = 0, *wcomm = 0, *y = 0,
                *ynow = 0, *yout = 0, *yp = 0, *ypnow = 0, *yprev = 0;
    Integer     *iroot = 0, *iwsav = 0;
    char        nag_enum_arg[40];
    /* Nag Types */
    Nag_Boolean  icode;
    Nag_Error    fail, fail2;
    Nag_RK_method method;
    Nag_ErrorAssess errass;

    INIT_FAIL(fail);
    INIT_FAIL(fail2);

    printf("nag_ode_ivp_rk_step_revcomm (d02pgc) Example Program Results\n\n");

    lrwsav = 350 + 32 * n;
    lwcomm = 6 * n;

    if (!(thresh = NAG_ALLOC((n), double)) ||
        !(iwsav = NAG_ALLOC((liwsav), Integer)) ||
        !(rwsav = NAG_ALLOC((lrwsav), double)) ||
        !(ynow = NAG_ALLOC((n), double)) ||
        !(ypnow = NAG_ALLOC((n), double)) ||
        !(yprev = NAG_ALLOC((n), double)) ||
        !(wcomm = NAG_ALLOC((lwcomm), double)) ||
        !(yout = NAG_ALLOC((n), double)) ||
        !(iroot = NAG_ALLOC((n), Integer)) ||
        !(y = NAG_ALLOC((n), double)) ||
        !(yp = NAG_ALLOC((n), double)) || !(troot = NAG_ALLOC((n), 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] ", &t, &tend);
#else
    scanf("%lf%lf%*[\n] ", &t, &tend);
#endif
for (j = 0; j < n; j++)
#ifdef _WIN32
    scanf_s("%lf", &ynow[j]);
#else
    scanf("%lf", &ynow[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

/* Initialize Runge-Kutta method for integrating ODE using
 * nag_ode_ivp_rkts_setup (d02pgc).
 */
nag_ode_ivp_rkts_setup(n, t, tend, ynow, tol, thresh, method,
                      errass, hstart, iwsav, rwsav, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_ivp_rkts_setup (d02pgc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

printf(" Calculation with tol = %8.1e\n", tol);
printf("      t          y1          y2\n");
printf("%7.3f", t);

```

```

for (k = 0; k < n; k++)
    printf("%11.4f", ynow[k]);
printf("\n");

tout = 0.1;
tnow = t;

while (tnow < tend) {
    tprev = tnow;
    for (k = 0; k < n; ++k)
        yprev[k] = ynow[k];

    /* Solve ODE by Runge-Kutta method by a sequence of single steps.
     * Each step requires a reverse communication loop around
     * nag_ode_ivp_rk_step_revcomm (d02pgc).
     */
    irevcm = 0;
    while (irevcm >= 0) {
        nag_ode_ivp_rk_step_revcomm(&irevcm, n, &tnow, ynow, yprev, iwsav,
                                   rwsav, &fail);

        if (irevcm > 0) {
            yprev[0] = ynow[1];
            yprev[1] = -ynow[0];
        }
    }
    if (irevcm == -2) {
        if (fail.code != NE_RK_POINTS && fail.code != NE_STIFF_PROBLEM &&
            fail.code != NW_RK_TOO_MANY) {
            printf("Error from nag_ode_ivp_rk_step (d02pgc).\n%s\n", fail.message);
            exit_status = 2;
            goto END;
        }
    }
}

/* Detect sign changes in last step */
for (k = 0; k < n; ++k)
    iroot[k] = 0;
nchange = 0;
for (k = 0; k < n; ++k) {
    if (ynow[k] * yprev[k] < 0.0) {
        iroot[nchange] = k + 1;
        nchange++;
    }
}
if (tnow >= tout || nchange > 0) {
    /* nag_ode_ivp_rk_interp_setup (d02phc).
     * Compute interpolant for the last step taken by the Runge-Kutta
     * integrator nag_ode_ivp_rk_step_revcomm (d02pgc).
     */
    irevcm = 0;
    while (irevcm >= 0) {
        nag_ode_ivp_rk_interp_setup(&irevcm, n, n, &t, y, yp, wcomm, lwcomm,
                                   iwsav, rwsav, &fail);

        if (irevcm > 0) {
            yp[0] = y[1];
            yp[1] = -y[0];
        }
    }
    if (fail.code != NE_NOERROR) {
        printf("Error from nag_ode_ivp_rk_interp_setup (d02phc).\n%s\n",
              fail.message);
        exit_status = 3;
        goto END;
    }
}
icheck = Nag_TRUE;
for (k = 0; k < nchange; ++k) {
    j = iroot[k] - 1;
    t1 = tprev;
    t2 = tnow;
    ind = 1;
    /* nag_zero_cont_func_brent_rcomm (c05azc).

```



```

    * Locates a simple zero of a continuous function.
    * Reverse communication.
    */
while (ind != 0) {
    nag_zero_cont_func_brent_rcomm(&t1, &t2, y[j], tol,
                                   Nag_Mixed, c, &ind, &fail);

    if (ind > 1) {
        /* nag_ode_ivp_rk_interp_eval (d02pjc).
        * Evaluate interpolant at a point in the last integrated step
        * as computed by nag_ode_ivp_rk_interp_setup (d02phc).
        */
        nag_ode_ivp_rk_interp_eval(icheck, n, n, t1, 0, y, wcomm,
                                   lwcomm, iwsav, rwsav, &fail2);

        icheck = Nag_FALSE;
    }
}
if (fail.code != NE_NOERROR) {
    printf("Error from nag_zero_cont_func_brent_rcomm (c05azc).\n%s\n",
          fail.message);
    exit_status = 4;
    goto END;
}
troot[k] = t1;
}
while (tnow >= tout) {
    for (k = 0; k < nchange; k++) {
        if (troot[k] < tout && iroot[k] > 0) {
            printf("Component %2" NAG_IFMT " has a root at t = %7.4f\n",
                  iroot[k], troot[k]);
            iroot[k] = -iroot[k];
        }
    }
    nag_ode_ivp_rk_interp_eval(icheck, n, n, tout, 0, yout, wcomm,
                              lwcomm, iwsav, rwsav, &fail2);

    icheck = Nag_FALSE;
    printf("%7.3f", tout);
    for (k = 0; k < n; k++) {
        printf("%11.4f", yout[k]);
    }
    printf("\n");
    tout = tout + 0.1;
}
for (k = 0; k < nchange; k++) {
    if (iroot[k] > 0) {
        printf("Component %2" NAG_IFMT " has a root at t = %7.4f\n",
              iroot[k], troot[k]);
    }
}
}
}
/* Get diagnostics on whole integration using
 * nag_ode_ivp_rkts_diag (d02ptc).
 */
nag_ode_ivp_rkts_diag(&totf, &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 = 5;
    goto END;
}
printf("\nCost of the integration in evaluations of f is %6" NAG_IFMT
       "\n\n", totf);

END:
    NAG_FREE(thresh);
    NAG_FREE(ynow);
    NAG_FREE(ypnow);
    NAG_FREE(yprev);
    NAG_FREE(yout);
    NAG_FREE(y);
    NAG_FREE(yp);

```

```

NAG_FREE(wcomm);
NAG_FREE(rwsav);
NAG_FREE(iwsav);
NAG_FREE(iroot);
NAG_FREE(troot);
return exit_status;
}

```

10.2 Program Data

```

nag_ode_ivp_rk_step_revcomm (d02pgc) Example Program Data
  Nag_RK_4_5           : method
  Nag_ErrorAssess_on  : errass
  0.0 6.28318530717958647692 : tstart, tend
  0.0 1.0              : yinit(1:n)
  0.0                  : hstart
  1.0e-8 1.0e-8       : thres(1:n)

```

10.3 Program Results

```

nag_ode_ivp_rk_step_revcomm (d02pgc) Example Program Results

```

```

Calculation with tol = 1.0e-05
  t          y1          y2
  0.000      0.0000      1.0000
  0.100      0.0998      0.9950
  0.200      0.1987      0.9801
  0.300      0.2955      0.9553
  0.400      0.3894      0.9211
  0.500      0.4794      0.8776
  0.600      0.5646      0.8253
  0.700      0.6442      0.7648
  0.800      0.7174      0.6967
  0.900      0.7833      0.6216
  1.000      0.8415      0.5403
  1.100      0.8912      0.4536
  1.200      0.9320      0.3624
  1.300      0.9636      0.2675
  1.400      0.9854      0.1700
  1.500      0.9975      0.0707
Component 2 has a root at t = 1.5708
  1.600      0.9996      -0.0292
  1.700      0.9917      -0.1288
  1.800      0.9738      -0.2272
  1.900      0.9463      -0.3233
  2.000      0.9093      -0.4161
  2.100      0.8632      -0.5048
  2.200      0.8085      -0.5885
  2.300      0.7457      -0.6663
  2.400      0.6755      -0.7374
  2.500      0.5985      -0.8011
  2.600      0.5155      -0.8569
  2.700      0.4274      -0.9041
  2.800      0.3350      -0.9422
  2.900      0.2392      -0.9710
  3.000      0.1411      -0.9900
  3.100      0.0416      -0.9991
Component 1 has a root at t = 3.1416
  3.200      -0.0584      -0.9983
  3.300      -0.1577      -0.9875
  3.400      -0.2555      -0.9668
  3.500      -0.3508      -0.9365
  3.600      -0.4425      -0.8968
  3.700      -0.5298      -0.8481
  3.800      -0.6119      -0.7910
  3.900      -0.6878      -0.7259
  4.000      -0.7568      -0.6536
  4.100      -0.8183      -0.5748
  4.200      -0.8716      -0.4903
  4.300      -0.9162      -0.4008

```

```

4.400    -0.9516    -0.3073
4.500    -0.9775    -0.2108
4.600    -0.9937    -0.1122
4.700    -0.9999    -0.0124
Component 2 has a root at t = 4.7124
4.800    -0.9962     0.0875
4.900    -0.9825     0.1865
5.000    -0.9589     0.2837
5.100    -0.9258     0.3780
5.200    -0.8835     0.4685
5.300    -0.8323     0.5544
5.400    -0.7728     0.6347
5.500    -0.7055     0.7087
5.600    -0.6313     0.7756
5.700    -0.5507     0.8347
5.800    -0.4646     0.8855
5.900    -0.3739     0.9275
6.000    -0.2794     0.9602
6.100    -0.1822     0.9833
6.200    -0.0831     0.9965
Component 1 has a root at t = 6.2832
    
```

Cost of the integration in evaluations of f is 356

