NAG Library Function Document

nag_ode_ivp_rkts_onestep (d02pfc)

1 Purpose

nag_ode_ivp_rkts_onestep (d02pfc) is a one-step function for solving 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_onestep (
     void (*f)(double t, Integer n, const double y[], double yp[],
          Nag_Comm *comm),
     Integer n, double *tnow, double ynow[], double ypnow[], Nag_Comm *comm,
     Integer iwsav[], double rwsav[], NagError *fail)
```

3 Description

nag_ode_ivp_rkts_onestep (d02pfc) and its associated functions (nag_ode_ivp_rkts_setup (d02pqc), nag_ode_ivp_rkts_reset_tend (d02prc), nag_ode_ivp_rkts_interp (d02psc), 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_onestep (d02pfc) 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_rkts_onestep (d02pfc) 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_rkts_onestep (d02pfc) returns an approximation to the solution **ynow** and its derivative **ypnow** at successive points **tnow**. If nag_ode_ivp_rkts_onestep (d02pfc) 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_rkts_onestep (d02pfc) 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_rkts_onestep (d02pfc) to attempt or that it computes automatically an appropriate value. Thereafter nag_ode_ivp_rkts_onestep (d02pfc) 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_onestep (d02pfc) 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_onestep (d02pfc) 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_rkts_interp (d02psc) 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_rkts_interp (d02psc). 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.

(ii) 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_rkts_onestep (d02pfc) 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_rkts_setup (d02pqc), 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 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: **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 $i = 1, 2, \ldots, n$.

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_onestep (d02pfc) you may allocate memory and initialize these pointers with various quantities for use by **f** when called from nag_ode_ivp_rkts_onestep (d02pfc) (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: $\mathbf{n} \geq 1$.

3: **tnow** – double * *Output*

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

4: **ynow**[**n**] – 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_rkts_setup (d02pqc)).

5: **ypnow**[**n**] – double *Output*

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

6: **comm** – Nag_Comm *

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

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

On entry: these must be the same arrays supplied in a previous call to nag_ode_ivp_rkts_setup (d02pqc). 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) or other associated functions.

9: **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 ⟨*value*⟩ had an illegal value.

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 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

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 ⟨*value*⟩.
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 (if **method** = Nag_RK_7_8, switch to **method** = Nag_RK_4_5 at setup). 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 ⟨*value*⟩ at the current time, ⟨*value*⟩. This step size is too small for the ***machine precision***, and is smaller than ⟨*value*⟩.

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 ⟨*value*⟩ 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 ⟨*value*⟩ 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 ⟨*value*⟩ 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_onestep (d02pfc) 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_onestep (d02pfc) 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 **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.

Performance statistics are available after any return from nag_ode_ivp_rkts_onestep (d02pfc) (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_rkts_onestep (d02pfc) (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_rkts_onestep (d02pfc) 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_one step (d02pfc) 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

$$y_1' = y_2$$
$$y_2' = -y_1$$

over the range $[0, 2\pi]$ with initial conditions $y_1 = 0.0$ and $y_2 = 1.0$. 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.

10.1 Program Text

```
/* nag_ode_ivp_rkts_onestep (d02pfc) 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 exit_status = 0;
  Integer liwsav, lrwsav, n;
  double hnext, hstart, tend, tgot, tol, tstart, 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;
  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);

  printf("nag_ode_ivp_rkts_onestep (d02pfc) Example Program Results\n\n");

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

  n = N;
  liwsav = 130;
  lrwsav = 350 + 32 * n;
  if (!(thresh = NAG_ALLOC(n, double)) ||
      !(ygot = NAG_ALLOC(n, double)) ||
      !(yinit = NAG_ALLOC(n, double)) ||
      !(ypgot = 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
#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

  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        y2\n");
    printf("%6.3f", tstart);
    for (k = 0; k < n; k++)
      printf("   %7.3f", yinit[k]);
    printf("\n");

    tgot = tstart;
    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;
      }

      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(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

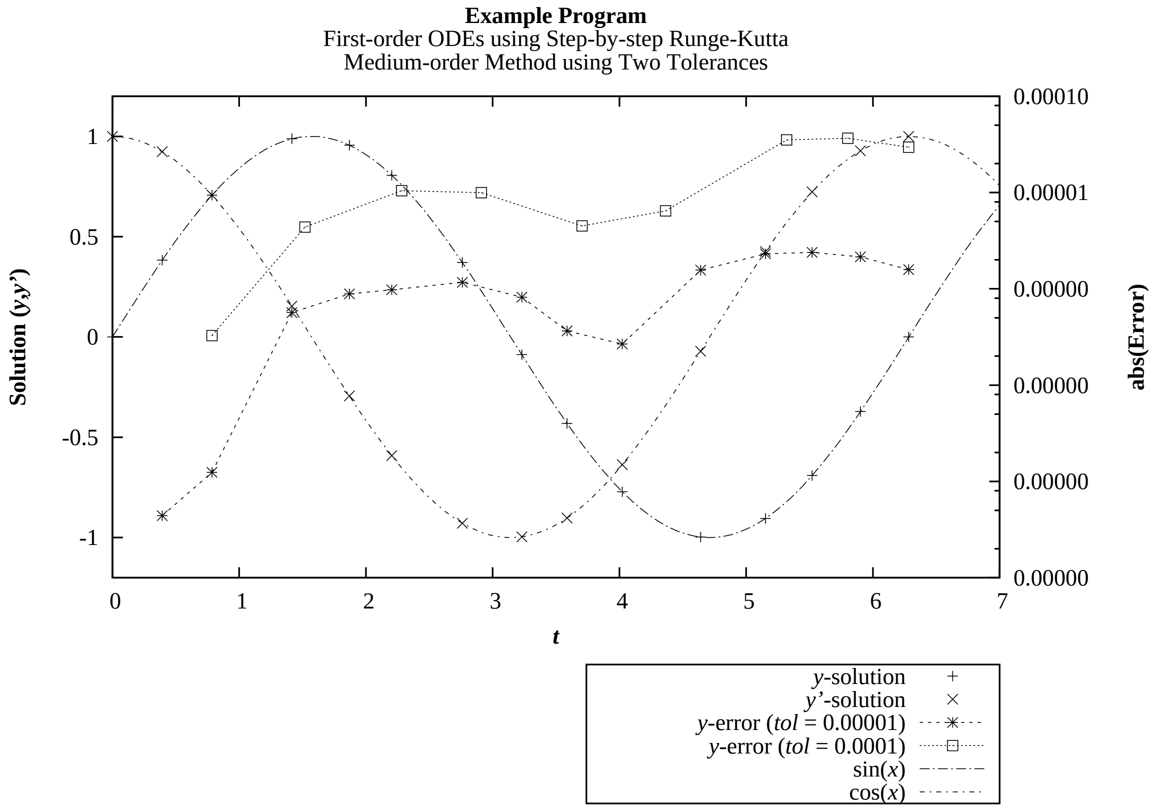
```
nag_ode_ivp_rkts_onestep (d02pfc) Example Program Data
   Nag_RK_4_5                       : method
   Nag_ErrorAssess_off              : errass
   0.0     6.28318530717958647692 : tstart, tend
   0.0     1.0                      : yinit(1:n)
   0.0                              : hstart
   1.0E-8  1.0E-8                   : thresh(1:n)
```

10.3 Program Results

```
nag_ode_ivp_rkts_onestep (d02pfc) 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
```


Example Program
First-order ODEs using Step-by-step Runge-Kutta
Medium-order Method using Two Tolerances
Solution (y,y')
abs(Error)
t
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
7
0.00010
0.00001
0.00000
0.00000
0.00000
0.00000
y-solution
y'-solution
y-error (tol = 0.00001)
y-error (tol = 0.0001)
sin(x)
cos(x)