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, 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

This function 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

\[ y' = f(t, y) \] given \[ y(t_0) = y_0 \]

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 repeatedly with successive values of \( \text{twant} \), the points at which you require the solution, in the range from \( \text{tstart} \) to \( \text{tend} \) (as specified in nag_ode_ivp_rk_setup (d02pvc)). In this manner nag_ode_ivp_rk_range returns the point at which it has computed a solution \( \text{tgot} \) (usually \( \text{twant} \)), the solution there \( \text{ygot} \) and its derivative \( \text{ypgot} \). If nag_ode_ivp_rk_range encounters some difficulty in taking a step toward \( \text{twant} \), then it returns the point of difficulty \( \text{tgot} \) and the solution and derivative computed there \( \text{ygot} \) and \( \text{ypgot} \).

In the call to nag_ode_ivp_rk_setup (d02pvc) you can specify the first step size for nag_ode_ivp_rk_range to attempt or that it compute automatically an appropriate value. Thereafter nag_ode_ivp_rk_range 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 by examining the contents of the structure \( \text{opt} \), see Section 4. 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 \( \text{errass} = \text{Nag_ErrorAssess_on} \) in the call to nag_ode_ivp_rk_setup (d02pvc). This assessment can be obtained after any call to nag_ode_ivp_rk_range by a call to the function nag_ode_ivp_rk_errass (d02pzc).

For more complicated tasks, you are referred to functions nag_ode_ivp_rkonestep (d02pdc), nag_ode_ivp_rk_interp (d02pzc) and nag_ode_ivp_rk_reset_tend (d02pvc).

4. Parameters

\( \text{neq} \)

Input: the number of ordinary differential equations in the system to be solved.
Constraint: \( \text{neq} \geq 1 \).

\( \text{f} \)

This function must evaluate the first derivatives \( y'_i \) (that is the functions \( f_i \)) for given values of the arguments \( t, y \).

[NP3275/5/pdf] 3.d02pcc.1
void f (Integer neq, double t, double y[], double yp[], Nag_User *comm)

    neq
    Input: the number of differential equations.

    t
    Input: the current value of the independent variable, t.

    y[neq]
    Input: the current values of the dependent variables, y_i for i = 1, 2, ..., neq.

    yp[neq]
    Output: the values of f_i for i = 1, 2, ..., neq.

    comm
    Input/Output: pointer to a structure of type Nag_User with the following member:

        p - Pointer
        Input/Output: 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.)

**twant**

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

Constraints: _twant_ must be closer to _tend_ than the previous of _tgot_ (or _tstart_ on the first call to nag_ode_ivp_rk_range); see nag_ode_ivp_rksetup (d02pvc) for a description of _tstart_ and _tend_. _twant_ must not lie beyond _tend_ in the direction of integration.

**tgot**

Output: 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.code (i.e., those not related to an invalid call of nag_ode_ivp_rk_range) a solution has still been computed at the value of _tgot_ but in general _tgot_ will not equal _twant_.

**ygot[neq]**

Input: on the first call to nag_ode_ivp_rk_range, _ygot_ need not be set. On all subsequent calls _ygot_ must remain unchanged.

Output: 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_rksetup (d02pvc). The local error has still been controlled even when _tgot_ ≠ _twant_, that is after a return with a non-trivial error.

**ypgot[neq]**

Output: an approximation to the first derivative of the true solution at _tgot_.

**ymax[neq]**

Input: on the first call to nag_ode_ivp_rk_range, _ymax_ need not be set. On all subsequent calls _ymax_ must remain unchanged.

Output: _ymax[i-1]_ contains the largest value of | _y_i_ | computed at any step in the integration so far.

**opt**

Input: pointer to a structure of type Nag_ODE_RK as initialised by the setup function nag_ode_ivp_rksetup (d02pvc).

Output: the following structure members hold information as follows:

- _totfcn_ - Integer
  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_rksetup (d02pvc) with errass = Nag_ErrorAssess_on.

- _stpcst_ - Integer
  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 parameter `method` in a prior call to `nag_ode_ivp_rk_setup (d02pvc).

- `waste` - double
  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
  The number of accepted steps.

- `hnext` - double
  The step size the integrator plans to use for the next step.

**comm**

Input/Output: pointer to a structure of type Nag_User with the following member:

- `p` - Pointer
  Input/Output: The pointer `p`, of type Pointer, allows the user to communicate information to and from the user-defined function `f()`. An object of the required type should be declared by the user, e.g. a structure, and its address assigned to the pointer `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.

- `fail` - Integer
  The NAG error parameter, see the Essential Introduction to the NAG C Library.

5. **Error Indications and Warnings**

- **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_NO_SETUP**
  The setup function `nag_ode_ivp_rk_setup (d02pvc)` has not been called.

- **NE_RK_INVALID_CALL**
  The function to be called as specified in the setup routine `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`. This is not permitted.

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

- **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_RK_TGOT_EQ_TEND**
  The call to `nag_ode_ivp_rk_range` has been made after reaching `tend`. The previous call to `nag_ode_ivp_rk_range` 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` 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` 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.
6. Further Comments

If nag_ode_ivp_rk_range 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 yg ot and ymax should be monitored (or the routine nag_ode_ivp_rkonestep (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 by examining the structure opt see Section 4. If errass was set to 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 (except when the error is due to incorrect input arguments or incorrect set-up) by a call to the routine nag_ode_ivp_rk_errass (d02pzc). The approximate extra number of evaluations of f used is given by \(2 \times stpsok \times stpect\) for method NAG_RK_4.5 or NAG_RK_7.8 and \(3 \times stpsok \times stpect\) 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 routine nag_ode_ivp_rk_errass (d02pzc) may be called only once.

If nag_ode_ivp_rk_range 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 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.
6.1. Accuracy

The accuracy of integration is determined by the parameters 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 parameters. 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.

6.2. References

Brankin R W, Gladwell I and Shampine L F (1991) RKSUITE: a suite of Runge-Kutta codes for the initial value problem for ODEs SoftReport 91-S1, Department of Mathematics, Southern Methodist University, Dallas, TX 75275, U.S.A.

7. See Also

nag_ode_ivp_adams_gen (d02cjc)
nag_ode_ivp_adams_roots (d02qfc)
nag_ode_ivp_rk_setup (d02pvc)
nag_ode_ivp_rk_errass (d02pzc)

8. 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 \(1.0 \times 10^{-8}\) 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_23) with tolerances \(\text{tol} = 1.0 \times 10^{-3}\) and \(\text{tol} = 1.0 \times 10^{-4}\) in turn so that we may compare the solutions. The value of \(\pi\) is obtained by using X01AAC.

See also the example program for nag_ode_ivp_rk_errass (d02pzc).

8.1. Program Text

```c
/* nag_ode_ivp_rk_range(d02pcc) Example Program */
/* Copyright 1994 Numerical Algorithms Group. */
/* Mark 3, 1994. */

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

#ifdef NAG_PROTO
static void f(Integer neq, double t1, double y[], double yp[], Nag_User *comm);
#else
static void f();
#endif

#define NEQ 2
#define ZERO 0.0
#define ONE 1.0
#define TWO 2.0
```

See also the example program for nag_ode_ivp_rk_errass (d02pzc).
#define FOUR 4.0

main()
{
  Integer neq;
  Nag_RK_method method;
  double hstart, pi, tgot, tend, tinc;
  double tol, tstart, twant;
  Integer i, j, nout;
  double thres[NEQ], ygot[NEQ], ymax[NEQ], ypgot[NEQ], ystart[NEQ];
  Nag_ErrorAssess errass;
  Nag_ODE_RK opt;
  Nag_User comm;

  Vprintf("d02pcc Example Program Results\n");

  /* Set initial conditions and input for d02pvc */
  neq = NEQ;
  pi = X01AAC;
  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;
    if (i==2) tol = 1.0e-4;
    d02pvc(neq, tstart, ystart, tend, tol, thres, method,
      Nag_RK_range, errass, hstart, &opt, NAGERR_DEFAULT);
    Vprintf("Calculation with tol = %8.1e\n\n", tol);
    Vprintf(" t y1 y2\n\n");
    Vprintf(" %8.3f %8.3f %8.3f\n", tstart, ystart[0], ystart[1]);
    for (j=nout-1; j>=0; j--)
      {
        twant = tend - j*tinc;
        d02pcc(neq, f, twant, &tgot, ygot, ypgot, ymax, &opt, &comm,
          Nag_RK_DEFAULT);
        Vprintf("%8.3f %8.3f %8.3f\n", twant, ygot[0], ygot[1]);
      }
    Vprintf("Cost of the integration in evaluations of f is %ld\n\n", opt.totfcn);
    d02ppc(&opt);
  }
  exit(EXIT_SUCCESS);
}
#endif NAG_PROTO
static void f(Integer neq, double t, double y[], double yp[], Nag_User *comm)
#else
static void f(neq, t, y, yp, comm)
  Integer neq;
  double t;
  double y[], yp[];
  Nag_User *comm;
#endif
{
  yp[0] = y[1];
yp[1] = -y[0];

8.2. Program Data

None.

8.3. Program Results

d02pcc Example Program Results

Calculation with tol = 1.0e-03

<table>
<thead>
<tr>
<th>t</th>
<th>y1</th>
<th>y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>0.785</td>
<td>0.707</td>
<td>0.707</td>
</tr>
<tr>
<td>1.571</td>
<td>0.999</td>
<td>0.000</td>
</tr>
<tr>
<td>2.356</td>
<td>0.706</td>
<td>-0.706</td>
</tr>
<tr>
<td>3.142</td>
<td>0.000</td>
<td>-0.999</td>
</tr>
<tr>
<td>3.927</td>
<td>-0.706</td>
<td>-0.706</td>
</tr>
<tr>
<td>4.712</td>
<td>-0.998</td>
<td>0.000</td>
</tr>
<tr>
<td>5.498</td>
<td>-0.705</td>
<td>0.706</td>
</tr>
<tr>
<td>6.283</td>
<td>0.001</td>
<td>0.997</td>
</tr>
</tbody>
</table>

Cost of the integration in evaluations of f is 124

Calculation with tol = 1.0e-04

<table>
<thead>
<tr>
<th>t</th>
<th>y1</th>
<th>y2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>0.000</td>
<td>1.000</td>
</tr>
<tr>
<td>0.785</td>
<td>0.707</td>
<td>0.707</td>
</tr>
<tr>
<td>1.571</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>2.356</td>
<td>0.707</td>
<td>-0.707</td>
</tr>
<tr>
<td>3.142</td>
<td>0.000</td>
<td>-1.000</td>
</tr>
<tr>
<td>3.927</td>
<td>-0.707</td>
<td>-0.707</td>
</tr>
<tr>
<td>4.712</td>
<td>-1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>5.498</td>
<td>-0.707</td>
<td>0.707</td>
</tr>
<tr>
<td>6.283</td>
<td>0.000</td>
<td>1.000</td>
</tr>
</tbody>
</table>

Cost of the integration in evaluations of f is 235