NAG Library Function Document

nag_ode_ivp_rk_errass (d02pzc)

1 Purpose

nag_ode_ivp_rk_errass (d02pzc) provides details about global error assessment computed during an integration with either nag_ode_ivp_rk_range (d02pcc) or nag_ode_ivp_rk_onestep (d02pdc).

2 Specification

```
#include <nag.h>
#include <nagd02.h>
void nag_ode_ivp_rk_errass (Integer neq, double rmserr[], double *errmax,
     double *terrmx, Nag_ODE_RK *opt, NagError *fail)
```

3 Description

nag_ode_ivp_rk_errass (d02pzc) and its associated functions (nag_ode_ivp_rk_range (d02pcc), nag_ode_ivp_rk_onestep (d02pdc), nag_ode_ivp_rk_setup (d02pvc), nag_ode_ivp_rk_reset_tend (d02pwc), nag_ode_ivp_rk_interp (d02pxc)) 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.

After a call to nag_ode_ivp_rk_range (d02pcc) or nag_ode_ivp_rk_onestep (d02pdc), nag_ode_ivp_rk_errass (d02pzc) can be called for information about error assessment, if this assessment was specified in the setup function nag_ode_ivp_rk_setup (d02pvc). A more accurate ''true'' solution $\hat{y}$ is computed in a secondary integration. The error is measured as specified in nag_ode_ivp_rk_setup (d02pvc) for local error control. At each step in the primary integration, an average magnitude σ_i of component y_i is computed, and the error in the component is

$$\frac{|y_i - \hat{y}_i|}{\max(\sigma_i, thres, (i))}.$$

where $thres(i)$ denotes the threshold value used in the error requirement, see nag_ode_ivp_rk_setup (d02pvc).

It is difficult to estimate reliably the true error at a single point. For this reason the RMS (root-mean-square) average of the estimated global error in each solution component is computed. This average is taken over all steps from the beginning of the integration through to the current integration point. If all has gone well, the average errors reported will be comparable to **tol** (see nag_ode_ivp_rk_setup (d02pvc)). The maximum error seen in any component in the integration so far and the point where the maximum error first occurred are also reported.

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.

Constraint: **neq** ≥ 1.

2: **rmserr**[**neq**] – double *Output*

On exit: **rmserr**$[i-1]$ approximates the RMS average of the true error of the numerical solution for the ith solution component y_i, for $i = 1, 2, \ldots,$ **neq**. The average is taken over all steps from the beginning of the integration to the current integration point.

3: **errmax** – double * *Output*

On exit: the maximum weighted approximate true error taken over all solution components and all steps.

4: **terrmx** – double * *Output*

On exit: the first value of the independent variable where an approximate true error attains the maximum value, **errmax**.

5: **opt** – Nag_ODE_RK * *Input/Output*

On entry: the structure of type Nag_ODE_RK as output from nag_ode_ivp_rk_range (d02pcc) or nag_ode_ivp_rk_onestep (d02pdc). You must not change this structure.

On exit: some members of **opt** are changed internally.

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

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

6 Error Indicators and Warnings

NE_ERRASS_REQ

No error assessment is available as it was not requested in the call to nag_ode_ivp_rk_setup (d02pvc).

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_MISSING_CALL

Previous call to nag_ode_ivp_rk_range (d02pcc) has not been made, hence nag_ode_ivp_rk_errass (d02pzc) must not be called.

Previous call to nag_ode_ivp_rk_onestep (d02pdc) has not been made, hence nag_ode_ivp_rk_errass (d02pzc) must not be called.

NE_NEQ

The value of **neq** supplied is not the same as that given to the setup function.

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_errass (d02pzc) had resulted in a severe error. You must call nag_ode_ivp_rk_setup (d02pvc) to start another problem.

NE_RK_NOSTEP

The integrator has not actually taken any successful steps. This function must not be called in this circumstance.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

If the integration has proceeded ‘‘well’’ and the problem is smooth enough, stable and not too difficult then the values returned in the arguments **rmserr** and **errmax** should be comparable to the value of **tol** specified in the prior call to nag_ode_ivp_rk_setup (d02pvc).

10 Example

We integrate a two body problem. The equations for the coordinates $(x(t), y(t))$ of one body as functions of time t in a suitable frame of reference are

$$x'' = \frac{-x}{r^3} \quad y'' = \frac{-y}{r^3}, r = \sqrt{(x^2 + y^2)}.$$

The intial conditions

$$x(0) = 1 - \epsilon, x'(0) = 0 \quad y(0) = 0, y'(0) = \sqrt{\frac{1+\epsilon}{1-\epsilon}}$$

lead to elliptic motion with $0 < \epsilon < 1$. We select $\epsilon = 0.7$ and repose as

$$\begin{aligned} y_1' &= y_2 \\ y_2' &= y_4 \\ y_3' &= \tfrac{-y_1}{r^3} \\ y_4' &= \tfrac{-y_1}{r^3} \end{aligned}$$

over the range $[0, 3\pi]$. We use relative error control with threshold values of 1.0e−10 for each solution component and a high order Runge–Kutta method (**method** = Nag_RK_7_8) with tolerance **tol** = 1.0e−6. The value of π is obtained by using nag_pi (X01AAC).

Note, for illustration purposes we select to integrate to the end of the range regardless of efficiency concerns.

10.1 Program Text

```
/* nag_ode_ivp_rk_errass (d02pzc) 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   4
#define ZERO  0.0
#define ONE   1.0
#define THREE 3.0
#define ECC   0.7

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          errmax, hstart, pi, *rmserr = 0, tend, terrmx, tgot,
  *thres = 0, tol;
  double          tstart, twant, *ygot = 0, *ymax = 0, *ypgot = 0, *ystart = 0;

  INIT_FAIL(fail);

  printf("nag_ode_ivp_rk_errass (d02pzc) 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)) ||
          !(ygot = NAG_ALLOC(neq, double)) ||
          !(ypgot = NAG_ALLOC(neq, double)) ||
          !(ystart = NAG_ALLOC(neq, double)) ||
          !(ymax = NAG_ALLOC(NEQ, double)) ||
          !(rmserr = 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] = ONE - ECC;
  ystart[1] = ZERO;
```

```
  ystart[2] = ZERO;
  ystart[3] = sqrt((ONE+ECC)/(ONE-ECC));
  tend = THREE*pi;
  for (i = 0; i < neq; i++)
    thres[i] = 1.0e-10;
  errass = Nag_ErrorAssess_on;
  hstart = ZERO;
  tol = 1.0e-6;
  method = Nag_RK_7_8;
  /* 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_range, 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          y3          y4\n\n");
  printf("%8.3f    %8.4f    %8.4f    %8.4f    %8.4f\n", tstart,
          ystart[0],
          ystart[1], ystart[2], ystart[3]);

  twant = tend;
  do
    {
      /* nag_ode_ivp_rk_range (d02pcc).
       * Ordinary differential equations solver, initial value
       * problems over a range using Runge-Kutta methods
       */
      nag_ode_ivp_rk_range(neq, f, twant, &tgot, ygot, ypgot, ymax, &opt,
                           &comm,
                           &fail);
    } while (fail.code == NE_RK_PDC_POINTS || fail.code == NE_STIFF_PROBLEM);

  if (fail.code != NE_NOERROR)
    {
      printf("Error from nag_ode_ivp_rk_range (d02pcc).\n%s\n",
              fail.message);
      exit_status = 1;
      goto END;
    }
  else
    {
      printf("%8.3f    %8.4f    %8.4f    %8.4f    %8.4f\n\n", tgot,
              ygot[0],
              ygot[1], ygot[2],
              ygot[3]);
      /* nag_ode_ivp_rk_errass (d02pzc).
       * A function to provide global error assessment during an
       * integration with either nag_ode_ivp_rk_range (d02pcc) or
       * nag_ode_ivp_rk_onestep (d02pdc)
       */
      nag_ode_ivp_rk_errass(neq, rmserr, &errmax, &terrmx, &opt, &fail);
      if (fail.code != NE_NOERROR)
        {
          printf("Error from nag_ode_ivp_rk_errass (d02pzc).\n%s\n",
                  fail.message);
          exit_status = 1;
          goto END;
        }

      printf("Componentwise error assessment\n          ");
      for (i = 0; i < neq; i++)
        printf("%11.2e ", rmserr[i]);
```

```
      printf("\n\n");
      printf("Worst global error observed was %11.2e - "
              "it occurred at t = %6.3f\n\n", errmax, terrmx);
      printf("Cost of the integration in evaluations of f is %"NAG_IFMT"\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(ygot);
  NAG_FREE(ypgot);
  NAG_FREE(ystart);
  NAG_FREE(ymax);
  NAG_FREE(rmserr);
  return exit_status;
}

static void NAG_CALL f(Integer neq, double t, const double y[], double yp[],
                       Nag_User *comm)

{
  double r, rp3;
  Integer *use_comm = (Integer *)comm->p;

  if (use_comm[0])
    {
      printf("(User-supplied callback f, first invocation.)\n");
      use_comm[0] = 0;
    }

  r = sqrt(y[0]*y[0]+y[1]*y[1]);
  rp3 = pow(r, 3.0);
  yp[0] = y[2];
  yp[1] = y[3];
  yp[2] = -y[0]/rp3;
  yp[3] = -y[1]/rp3;
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_ode_ivp_rk_errass (d02pzc) Example Program Results

Calculation with tol =    1.0e-06

    t          y1          y2          y3          y4

   0.000      0.3000      0.0000      0.0000      2.3805
(User-supplied callback f, first invocation.)
   9.425     -1.7000      0.0000     -0.0000     -0.4201

Componentwise error assessment
             3.81e-06    7.10e-06    6.92e-06    2.10e-06

Worst global error observed was    3.43e-05 - it occurred at t =  6.302

Cost of the integration in evaluations of f is 1361
```