NAG Library Function Document

nag_ode_bvp_coll_nlin_diag (d02tzc)

1 Purpose

nag_ode_bvp_coll_nlin_diag (d02tzc) returns information about the solution of a general two-point boundary value problem computed by nag_ode_bvp_coll_nlin_solve (d02tlc).

2 Specification

```
#include <nag.h>
#include <nagd02.h>
void nag_ode_bvp_coll_nlin_diag (Integer mxmesh, Integer *nmesh,
     double mesh[], Integer ipmesh[], double *ermx, Integer *iermx,
     Integer *ijermx, const double rcomm[], const Integer icomm[],
     NagError *fail)
```

3 Description

nag_ode_bvp_coll_nlin_diag (d02tzc) and its associated functions (nag_ode_bvp_coll_nlin_solve (d02tlc), nag_ode_bvp_coll_nlin_setup (d02tvc), nag_ode_bvp_coll_nlin_contin (d02txc) and nag_ode_bvp_coll_nlin_interp (d02tyc)) solve the two-point boundary value problem for a nonlinear mixed order system of ordinary differential equations

$$\begin{aligned} y_1^{(m_1)}(x) &= f_1\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right) \\ y_2^{(m_2)}(x) &= f_2\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right) \\ &\vdots \\ y_n^{(m_n)}(x) &= f_n\left(x, y_1, y_1^{(1)}, \ldots, y_1^{(m_1-1)}, y_2, \ldots, y_n^{(m_n-1)}\right) \end{aligned}$$

over an interval $[a,b]$ subject to p (>0) nonlinear boundary conditions at a and q (>0) nonlinear boundary conditions at b, where $p+q=\sum_{i=1}^{n} m_i$. Note that $y_i^{(m)}(x)$ is the mth derivative of the ith solution component. Hence $y_i^{(0)}(x)=y_i(x)$. The left boundary conditions at a are defined as

$$g_i(z(y(a)))=0, \quad i=1,2,\ldots,p,$$

and the right boundary conditions at b as

$$\bar{g}_j(z(y(b)))=0, \quad j=1,2,\ldots,q,$$

where $y=(y_1,y_2,\ldots,y_n)$ and

$$z(y(x))=\left(y_1(x), y_1^{(1)}(x), \ldots, y_1^{(m_1-1)}(x), y_2(x), \ldots, y_n^{(m_n-1)}(x)\right).$$

First, nag_ode_bvp_coll_nlin_setup (d02tvc) must be called to specify the initial mesh, error requirements and other details. Then, nag_ode_bvp_coll_nlin_solve (d02tlc) can be used to solve the boundary value problem. After successful computation, nag_ode_bvp_coll_nlin_diag (d02tzc) can be used to ascertain details about the final mesh. nag_ode_bvp_coll_nlin_interp (d02tyc) can be used to compute the approximate solution anywhere on the interval $[a,b]$ using interpolation.

The functions are based on modified versions of the codes COLSYS and COLNEW (see Ascher *et al.* (1979) and Ascher and Bader (1987)). A comprehensive treatment of the numerical solution of boundary value problems can be found in Ascher *et al.* (1988) and Keller (1992).

4 References

Ascher U M and Bader G (1987) A new basis implementation for a mixed order boundary value ODE solver *SIAM J. Sci. Stat. Comput.* **8** 483–500

Ascher U M, Christiansen J and Russell R D (1979) A collocation solver for mixed order systems of boundary value problems *Math. Comput.* **33** 659–679

Ascher U M, Mattheij R M M and Russell R D (1988) *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* Prentice–Hall

Cole J D (1968) *Perturbation Methods in Applied Mathematics* Blaisdell, Waltham, Mass.

Keller H B (1992) *Numerical Methods for Two-point Boundary-value Problems* Dover, New York

5 Arguments

1: **mxmesh** – Integer *Input*

On entry: the maximum number of points allowed in the mesh.

Constraint: this must be identical to the value supplied for the argument **mxmesh** in the prior call to nag_ode_bvp_coll_nlin_setup (d02tvc).

2: **nmesh** – Integer * *Output*

On exit: the number of points in the mesh last used by nag_ode_bvp_coll_nlin_solve (d02tlc).

3: **mesh**[**mxmesh**] – double *Output*

On exit: **mesh**$[i-1]$ contains the ith point of the mesh last used by nag_ode_bvp_coll_nlin_solve (d02tlc), for $i = 1, 2, \ldots,$ **nmesh**. **mesh**$[0]$ will contain a and **mesh**[**nmesh** $- 1$] will contain b. The remaining elements of **mesh** are not initialized.

4: **ipmesh**[**mxmesh**] – Integer *Output*

On exit: **ipmesh**$[i-1]$ specifies the nature of the point **mesh**$[i-1]$, for $i = 1, 2, \ldots,$ **nmesh**, in the final mesh computed by nag_ode_bvp_coll_nlin_solve (d02tlc).

ipmesh$[i-1] = 1$
Indicates that the ith point is a fixed point and was used by the solver before an extrapolation-like error test.

ipmesh$[i-1] = 2$
Indicates that the ith point was used by the solver before an extrapolation-like error test.

ipmesh$[i-1] = 3$
Indicates that the ith point was used by the solver only as part of an extrapolation-like error test.

The remaining elements of **ipmesh** are initialized to -1.

See Section 9 for advice on how these values may be used in conjunction with a continuation process.

5: **ermx** – double * *Output*

On exit: an estimate of the maximum error in the solution computed by nag_ode_bvp_coll_nlin_solve (d02tlc), that is

$$\textbf{ermx} = \max\left(\frac{\|y_i - v_i\|}{(1.0 + \|v_i\|)}\right)$$

where v_i is the approximate solution for the ith solution component. If nag_ode_bvp_coll_nlin_solve (d02tlc) returned successfully with **fail.code** = NE_NOERROR,

then **ermx** will be less than **tols**[**ijermx** -1] where **tols** contains the error requirements as specified in Sections 3 and 5 in nag_ode_bvp_coll_nlin_setup (d02tvc).

If nag_ode_bvp_coll_nlin_solve (d02tlc) returned with **fail.code** $=$ NW_MAX_SUBINT, then **ermx** will be greater than **tols**[**ijermx** -1].

If nag_ode_bvp_coll_nlin_solve (d02tlc) returned any other value for **fail.code** then an error estimate is not available and **ermx** is initialized to 0.0.

6: **iermx** – Integer * *Output*

On exit: indicates the mesh sub-interval where the value of **ermx** has been computed, that is [**mesh**[**iermx** -1], **mesh**[**iermx**]].

If an estimate of the error is not available then **iermx** is initialized to 0.

7: **ijermx** – Integer * *Output*

On exit: indicates the component i ($=$ **ijermx**) of the solution for which **ermx** has been computed, that is the approximation of y_i on [**mesh**[**iermx** -1], **mesh**[**iermx**]] is estimated to have the largest error of all components y_i over mesh sub-intervals defined by **mesh**.

If an estimate of the error is not available then **ijermx** is initialized to 0.

8: **rcomm**[dim] – const double *Communication Array*

Note: the dimension, dim, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **rcomm** in the previous call to nag_ode_bvp_coll_nlin_solve (d02tlc).

On entry: this must be the same array as supplied to nag_ode_bvp_coll_nlin_solve (d02tlc) and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated functions.

9: **icomm**[dim] – const Integer *Communication Array*

Note: the dimension, dim, of this array is dictated by the requirements of associated functions that must have been previously called. This array MUST be the same array passed as argument **icomm** in the previous call to nag_ode_bvp_coll_nlin_solve (d02tlc).

On entry: this must be the same array as supplied to nag_ode_bvp_coll_nlin_solve (d02tlc) and **must** remain unchanged between calls.

On exit: contains information about the solution for use on subsequent calls to associated functions.

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

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

6 Error Indicators and Warnings

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_CONVERGENCE_SOL

The solver function did not produce any results suitable for interpolation.

NE_INT_CHANGED

On entry, **mxmesh** $= \langle value \rangle$ and **mxmesh** $= \langle value \rangle$ in nag_ode_bvp_coll_nlin_setup (d02tvc).
Constraint: **mxmesh** = **mxmesh** in nag_ode_bvp_coll_nlin_setup (d02tvc).

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.

NE_MISSING_CALL

The solver function does not appear to have been called.

NW_NOT_CONVERGED

The solver function did not converge to a suitable solution.
A converged intermediate solution has been used.
Error estimate information is not available.

NW_TOO_MUCH_ACC_REQUESTED

The solver function did not satisfy the error requirements.
Information has been supplied on the last mesh used.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

Note that:

if nag_ode_bvp_coll_nlin_solve (d02tlc) returned **fail.code** = NE_NOERROR, NW_MAX_SUBINT or NW_NOT_CONVERGED then it will always be the case that **ipmesh**$[0]$ = **ipmesh**$[$**nmesh** $- 1] = 1$;

if nag_ode_bvp_coll_nlin_solve (d02tlc) returned **fail.code** = NE_NOERROR or NW_MAX_SUBINT then it will always be the case that **ipmesh**$[i-1] = 3$, for $i = 2, 4, \ldots,$ **nmesh** $- 1$ (even i) and **ipmesh**$[i-1] = 1$ or 2, for $i = 3, 5, \ldots,$ **nmesh** $- 2$ (odd i);

if nag_ode_bvp_coll_nlin_solve (d02tlc) returned **fail.code** = NW_NOT_CONVERGED then it will always be the case that **ipmesh**$[i-1] = 1$ or 2, for $i = 2, 3, \ldots,$ **nmesh** $- 1$.

If nag_ode_bvp_coll_nlin_diag (d02tzc) returns **fail.code** = NE_NOERROR, then examination of the mesh may provide assistance in determining a suitable starting mesh for nag_ode_bvp_coll_nlin_setup (d02tvc) in any subsequent attempts to solve similar problems.

If the problem being treated by nag_ode_bvp_coll_nlin_solve (d02tlc) is one of a series of related problems (for example, as part of a continuation process), then the values of **ipmesh** and **mesh** may be suitable as input arguments to nag_ode_bvp_coll_nlin_contin (d02txc). Using the mesh points not involved in the extrapolation error test is usually appropriate. **ipmesh** and **mesh** should be passed unchanged to nag_ode_bvp_coll_nlin_contin (d02txc) but **nmesh** should be replaced by $(\mathbf{nmesh} + 1)/2$.

If nag_ode_bvp_coll_nlin_diag (d02tzc) returns **fail.code** = NE_CONVERGENCE_SOL, NE_MISSING_CALL, NW_NOT_CONVERGED or NW_TOO_MUCH_ACC_REQUESTED, nothing can be said regarding the quality of the mesh returned. However, it may be a useful starting mesh for nag_ode_bvp_coll_nlin_setup (d02tvc) in any subsequent attempts to solve the same problem.

If nag_ode_bvp_coll_nlin_solve (d02tlc) returns **fail.code** = NW_MAX_SUBINT, this corresponds to the solver requiring more than **mxmesh** mesh points to satisfy the error requirements. If **mxmesh** can be increased and the preceding call to nag_ode_bvp_coll_nlin_solve (d02tlc) was not part, or was the first part, of a continuation process then the values in **mesh** may provide a suitable mesh with which to initialize a subsequent attempt to solve the same problem. If it is not possible to provide more mesh points then relaxing the error requirements by setting **tols**[**ijermx** − 1] to **ermx** might lead to a successful solution. It may be necessary to reset the other components of **tols**. Note that resetting the tolerances can lead to a different sequence of meshes being computed and hence to a different solution being computed.

10 Example

The following example is used to illustrate the use of fixed mesh points, simple continuation and numerical approximation of a Jacobian. See also nag_ode_bvp_coll_nlin_solve (d02tlc), nag_ode_bvp_coll_nlin_setup (d02tvc), nag_ode_bvp_coll_nlin_contin (d02txc) and nag_ode_bvp_coll_nlin_interp (d02tyc), for the illustration of other facilities.

Consider the Lagerstrom–Cole equation

$$y'' = (y - yy')/\epsilon$$

with the boundary conditions

$$y(0) = \alpha \quad y(1) = \beta, \tag{1}$$

where ϵ is small and positive. The nature of the solution depends markedly on the values of α, β. See Cole (1968).

We choose $\alpha = -\frac{1}{3}, \beta = \frac{1}{3}$ for which the solution is known to have corner layers at $x = \frac{1}{3}, \frac{2}{3}$. We choose an initial mesh of seven points $[0.0, 0.15, 0.3, 0.5, 0.7, 0.85, 1.0]$ and ensure that the points $x = 0.3, 0.7$ near the corner layers are fixed, that is the corresponding elements of the array **ipmesh** are set to 1. First we compute the solution for $\epsilon = 1.0\text{e}{-4}$ using in **guess** the initial approximation $y(x) = \alpha + (\beta - \alpha)x$ which satisfies the boundary conditions. Then we use simple continuation to compute the solution for $\epsilon = 1.0\text{e}{-5}$. We use the suggested values for **nmesh**, **ipmesh** and **mesh** in the call to nag_ode_bvp_coll_nlin_contin (d02txc) prior to the continuation call, that is only every second point of the preceding mesh is used and the fixed mesh points are retained.

Although the analytic Jacobian for this system is easy to evaluate, for illustration the procedure **fjac** uses central differences and calls to **ffun** to compute a numerical approximation to the Jacobian.

10.1 Program Text

```
/* nag_ode_bvp_coll_nlin_diag (d02tzc) Example Program.
 *
 * Copyright 2013 Numerical Algorithms Group.
 *
 * Mark 24, 2013.
 */

#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagd02.h>
#include <nagx02.h>

typedef struct {
  double  alpha, beta, eps;
  Integer mmax;
} func_data;

#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL ffun(double x, const double y[], Integer neq,
                          const Integer m[], double f[], Nag_Comm *comm);
static void NAG_CALL fjac(double x, const double y[], Integer neq,
```

```
                          const Integer m[], double dfdy[], Nag_Comm *comm);
static void NAG_CALL gafun(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double ga[], Nag_Comm *comm);
static void NAG_CALL gbfun(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double gb[], Nag_Comm *comm);
static void NAG_CALL gajac(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double dgady[], Nag_Comm *comm);
static void NAG_CALL gbjac(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double dgbdy[], Nag_Comm *comm);
static void NAG_CALL guess(double x, Integer neq, const Integer m[], double y[],
                           double dym[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{

  /* Scalars */
  Integer     exit_status = 0, neq = 1, mmax = 2, nlbc = 1, nrbc = 1;
  Integer     i, iermx, ijermx, j, licomm, lrcomm, mxmesh, ncol, nmesh;
  double      alpha, beta, eps, ermx;
  /* Arrays */
  static double ruser[7] = {-1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0};
  double      *mesh = 0, *rcomm = 0, *tol = 0, *y = 0;
  double      rdum[1];
  Integer     *ipmesh = 0, *icomm = 0, *m = 0;
  Integer     idum[2];
  /* Nag Types */
  Nag_Boolean failed = Nag_FALSE;
  func_data   fd;
  Nag_Comm    comm;
  NagError    fail;

  INIT_FAIL(fail);

  printf("nag_ode_bvp_coll_nlin_diag (d02tzc) Example Program Results\n\n");

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

  /* Skip heading in data file*/
  scanf("%*[^\n] ");
  scanf("%"NAG_IFMT "%"NAG_IFMT "%"NAG_IFMT "%*[^\n] ", &ncol, &nmesh, &mxmesh);
  if (!(mesh   = NAG_ALLOC(mxmesh, double)) ||
      !(m      = NAG_ALLOC(neq, Integer)) ||
      !(tol    = NAG_ALLOC(neq, double)) ||
      !(y      = NAG_ALLOC(neq*mmax, double)) ||
      !(ipmesh = NAG_ALLOC(mxmesh, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  /* Set problem orders */
  m[0] = 2;
  scanf("%lf%lf%lf%*[^\n] ", &alpha, &beta, &eps);
  for (i = 0; i < nmesh; i++) {
    scanf("%lf", &mesh[i]);
  }
  scanf("%*[^\n] ");
  for (i = 0; i < nmesh; i++) {
    scanf("%"NAG_IFMT "", &ipmesh[i]);
  }
  scanf("%*[^\n] ");
  for (i = 0; i < neq; i++) {
    scanf("%lf", &tol[i]);
  }
  scanf("%*[^\n] ");

  /* Communication space query to get size of rcomm and icomm,
```

```
   * by setting lrcomm=0 in call to
   * nag_ode_bvp_coll_nlin_setup (d02tvc):
   * Ordinary differential equations, general nonlinear boundary value problem,
   * setup for nag_ode_bvp_coll_nlin_solve (d02tlc).
   */
  nag_ode_bvp_coll_nlin_setup(neq, m, nlbc, nrbc, ncol, tol, mxmesh, nmesh,
                              mesh, ipmesh, rdum, 0, idum, 2, &fail);
  if (fail.code == NE_NOERROR) {
    lrcomm = idum[0];
    licomm = idum[1];

    if (!(rcomm = NAG_ALLOC(lrcomm, double)) ||
        !(icomm = NAG_ALLOC(licomm, Integer))) {
      printf("Allocation failure\n");
      exit_status = -2;
      goto END;
    }

    /* Initialize again using nag_ode_bvp_coll_nlin_setup (d02tvc). */
    nag_ode_bvp_coll_nlin_setup(neq, m, nlbc, nrbc, ncol, tol, mxmesh, nmesh,
                                mesh, ipmesh, rcomm, lrcomm, icomm, licomm,
                                &fail);
  }
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_bvp_coll_nlin_setup (d02tvc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  eps = 0.1 * eps;
  /* Set data required for the user-supplied functions */
  fd.alpha = alpha;
  fd.beta = beta;
  fd.eps = eps;
  fd.mmax = mmax;
  /* Associate the data structure with comm.p */
  comm.p = (Pointer)

  for (j = 0; j < 2; j++) {
    printf("\n Tolerance = %8.1e  eps = %10.3e\n", tol[0], eps);
    /* Solve*/

    /* nag_ode_bvp_coll_nlin_solve (d02tlc).
     * Ordinary differential equations, general nonlinear boundary value
     * problem, collocation technique.
     */
    nag_ode_bvp_coll_nlin_solve(ffun, fjac, gafun, gbfun, gajac, gbjac, guess,
                                rcomm, icomm, &comm, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ode_bvp_coll_nlin_solve (d02tlc).\n%s\n",
             fail.message);
      failed = Nag_TRUE;
      goto END;
    }

    /* Extract mesh.*/

    /* nag_ode_bvp_coll_nlin_diag (d02tzc).
     * Ordinary differential equations, general nonlinear boundary value
     * problem, diagnostics for nag_ode_bvp_coll_nlin_solve (d02tlc).
     */
    nag_ode_bvp_coll_nlin_diag(mxmesh, &nmesh, mesh, ipmesh, &ermx, &iermx,
                               &ijermx, rcomm, icomm, &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ode_bvp_coll_nlin_diag (d02tzc).\n%s\n",
             fail.message);
      exit_status = 2;
      goto END;
    }
```

```
    /* Print mesh statistics.*/
    printf("\n Used a mesh of %4ld  points\n", nmesh);
    printf(" Maximum error = %10.2e  in interval %4ld", ermx, iermx);
    printf("  for component %4"NAG_IFMT " \n", ijermx);
    if (failed) {
      goto END;
    }

    /* Print solution at every second point on final mesh. */
    printf("\n Solution and derivative at every second point:\n");
    printf("   x           u          u'\n");
    for (i = 0; i < nmesh; i += 2) {

      /* nag_ode_bvp_coll_nlin_interp (d02tyc).
       * Ordinary differential equations, general nonlinear boundary value
       * problem, interpolation for nag_ode_bvp_coll_nlin_solve (d02tlc).
       */
      nag_ode_bvp_coll_nlin_interp(mesh[i], y, neq, mmax, rcomm, icomm, &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_ode_bvp_coll_nlin_interp (d02tyc).\n%s\n",
               fail.message);
        exit_status = 3;
        goto END;
      }

      printf("%8.4f %11.5f %11.5f \n", mesh[i], y[0], y[neq]);
    }
    if (j == 0) {
      /* Halve final mesh for new initial mesh and set up for continuation.*/
      nmesh = (nmesh + 1)/2;

      /* nag_ode_bvp_coll_nlin_contin (d02txc).
       * Ordinary differential equations, general nonlinear boundary value
       * problem, continuation facility for
       * nag_ode_bvp_coll_nlin_solve (d02tlc).
       */
      nag_ode_bvp_coll_nlin_contin(mxmesh, nmesh, mesh, ipmesh, rcomm, icomm,
                                   &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_ode_bvp_coll_nlin_contin (d02txc).\n%s\n",
               fail.message);
        exit_status = 4;
        goto END;
      }

      /* Reduce continuation parameter.*/
      eps = 0.1 * eps;
      fd.eps = eps;
    }
  }

 END :
  NAG_FREE(mesh);
  NAG_FREE(m);
  NAG_FREE(tol);
  NAG_FREE(rcomm);
  NAG_FREE(y);
  NAG_FREE(ipmesh);
  NAG_FREE(icomm);
  return exit_status;
}

static void NAG_CALL ffun(double x, const double y[], Integer neq,
                          const Integer m[], double f[], Nag_Comm *comm)
{
  func_data *fd = (func_data *)comm->p;

  if (comm->user[0] == -1.0)
    {
      printf("(User-supplied callback ffun, first invocation.)\n");
      comm->user[0] = 0.0;
```

```
    }
  f[0] = (y[0] - y[0] * y[1])/fd->eps;
}

static void NAG_CALL fjac(double x, const double y[], Integer neq,
                          const Integer m[], double dfdy[], Nag_Comm *comm)
{
  double    epsh, fac, ptrb;
  Integer   j;
  double    f1[1], f2[1], yp[2];

  if (comm->user[1] == -1.0)
    {
      printf("(User-supplied callback fjac, first invocation.)\n");
      comm->user[1] = 0.0;
    }
  /* nag_machine_precision (x02ajc).
   * The machine precision.
   */
  epsh = 100.0 * nag_machine_precision;
  fac = sqrt(nag_machine_precision);
  for (j = 0; j < m[0]; j++) {
    yp[j] = y[j];
  }
  for (j = 0; j < m[0]; j++) {
    ptrb = MAX(epsh, fac * fabs(y[j]));
    yp[j] = y[j] + ptrb;
    ffun(x, yp, neq, m, f1, comm);
    yp[j] = y[j] - ptrb;
    ffun(x, yp, neq, m, f2, comm);
    dfdy[j] = 0.5 * (f1[0] - f2[0])/ptrb;
    yp[j] = y[j];
  }
}

static void NAG_CALL gafun(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double ga[], Nag_Comm *comm)
{
  func_data *fd = (func_data *)comm->p;

  if (comm->user[2] == -1.0)
    {
      printf("(User-supplied callback gafun, first invocation.)\n");
      comm->user[2] = 0.0;
    }
  ga[0] = ya[0] - fd->alpha;
}

static void NAG_CALL gbfun(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double gb[], Nag_Comm *comm)
{
  func_data *fd = (func_data *)comm->p;

  if (comm->user[3] == -1.0)
    {
      printf("(User-supplied callback gbfun, first invocation.)\n");
      comm->user[3] = 0.0;
    }
  gb[0] = yb[0] - fd->beta;
}

static void NAG_CALL gajac(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double dgady[], Nag_Comm *comm)
{
  if (comm->user[4] == -1.0)
    {
      printf("(User-supplied callback gajac, first invocation.)\n");
      comm->user[4] = 0.0;
    }
  dgady[0] = 1.0;
}
```

```
static void NAG_CALL gbjac(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double dgbdy[], Nag_Comm *comm)
{
  if (comm->user[5] == -1.0)
    {
      printf("(User-supplied callback gbjac, first invocation.)\n");
      comm->user[5] = 0.0;
    }
  dgbdy[0] = 1.0;
}

static void NAG_CALL guess(double x, Integer neq, const Integer m[], double y[],
                           double dym[], Nag_Comm *comm)
{
  func_data *fd = (func_data *)comm->p;
  double    alpha, beta;

  if (comm->user[6] == -1.0)
    {
      printf("(User-supplied callback guess, first invocation.)\n");
      comm->user[6] = 0.0;
    }
  alpha  = fd->alpha;
  beta   = fd->beta;
  y[0]   = alpha + (beta - alpha) * x;
  y[1]   = beta - alpha;
  dym[0] = 0.0;
}
```

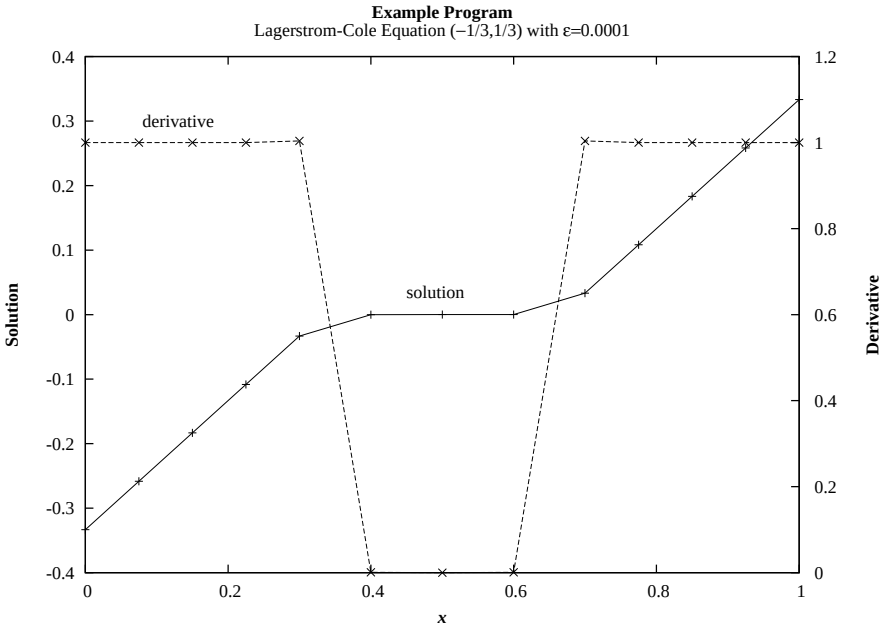
10.2 Program Data

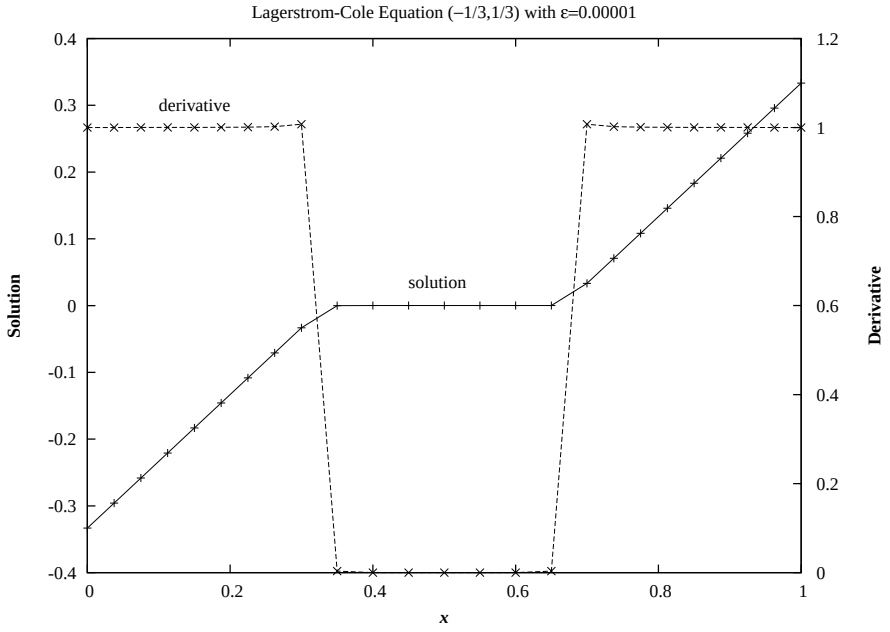
```
nag_ode_bvp_coll_nlin_diag (d02tzc) Example Program Data
  5   7    50                        : ncol, nmesh, mxmesh
 -0.33333333333333333333
  0.33333333333333333333   0.001    : alpha, beta, eps
  0.0 0.15 0.3 0.5 0.7 0.85 1.0      : mesh(1:nmesh)
  1   2    1   2   1   2    1        : ipmesh(1:nmesh)
  1.0E-5                            : tol
```

10.3 Program Results

```
nag_ode_bvp_coll_nlin_diag (d02tzc) Example Program Results


 Tolerance =  1.0e-05  eps =  1.000e-04
(User-supplied callback guess, first invocation.)
(User-supplied callback gafun, first invocation.)
(User-supplied callback gajac, first invocation.)
(User-supplied callback gbfun, first invocation.)
(User-supplied callback gbjac, first invocation.)
(User-supplied callback ffun, first invocation.)
(User-supplied callback fjac, first invocation.)

 Used a mesh of   25  points
 Maximum error =   2.15e-06  in interval   16  for component    1

 Solution and derivative at every second point:
   x           u          u'
  0.0000    -0.33333     1.00000
  0.0750    -0.25833     1.00000
  0.1500    -0.18333     1.00000
  0.2250    -0.10833     1.00002
  0.3000    -0.03332     1.00372
  0.4000    -0.00001     0.00084
  0.5000    -0.00000     0.00000
  0.6000     0.00001     0.00084
  0.7000     0.03332     1.00372
  0.7750     0.10833     1.00002
  0.8500     0.18333     1.00000
```

```
 0.9250     0.25833     1.00000
 1.0000     0.33333     1.00000

Tolerance =  1.0e-05  eps =  1.000e-05

Used a mesh of   49  points
Maximum error =   2.11e-06  in interval   32  for component    1

Solution and derivative at every second point:
  x           u          u'
 0.0000    -0.33333     1.00014
 0.0375    -0.29583     1.00018
 0.0750    -0.25833     1.00022
 0.1125    -0.22083     1.00029
 0.1500    -0.18333     1.00040
 0.1875    -0.14583     1.00059
 0.2250    -0.10833     1.00098
 0.2625    -0.07083     1.00202
 0.3000    -0.03333     1.00745
 0.3500    -0.00001     0.00354
 0.4000    -0.00000     0.00000
 0.4500    -0.00000     0.00000
 0.5000    -0.00000     0.00000
 0.5500     0.00000     0.00000
 0.6000     0.00000     0.00000
 0.6500     0.00001     0.00354
 0.7000     0.03333     1.00745
 0.7375     0.07083     1.00202
 0.7750     0.10833     1.00098
 0.8125     0.14583     1.00059
 0.8500     0.18333     1.00040
 0.8875     0.22083     1.00029
 0.9250     0.25833     1.00022
 0.9625     0.29583     1.00018
 1.0000     0.33333     1.00014
```




Lagerstrom-Cole Equation (−1/3,1/3) with ε=0.00001
derivative
solution
Solution
Derivative
x
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1