NAG Library Function Document

nag_ode_bvp_coll_nlin_interp (d02tyc)

1 Purpose

nag_ode_bvp_coll_nlin_interp (d02tyc) interpolates on 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_interp (double x, double y[], Integer neq,
     Integer mmax, double rcomm[], const Integer icomm[], NagError *fail)
```

3 Description

nag_ode_bvp_coll_nlin_interp (d02tyc) 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_diag (d02tzc)) solve the two-point boundary value problem for a nonlinear mixed order system of ordinary differential equations

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

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

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

and the right boundary conditions at b as

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

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

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

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 and other details of the solution procedure, and 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

Grossman C (1992) Enclosures of the solution of the Thomas–Fermi equation by monotone discretization *J. Comput. Phys.* **98** 26–32

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

5 Arguments

1: **x** – double *Input*

On entry: x, the independent variable.

Constraint: $a \leq \mathbf{x} \leq b$, i.e., not outside the range of the original mesh specified in the initialization call to nag_ode_bvp_coll_nlin_setup (d02tvc).

2: **y**[**neq** × **mmax**] – double *Output*

Note: where $\mathbf{Y}(i,j)$ appears in this document, it refers to the array element $\mathbf{y}[j \times \mathbf{neq} + i - 1]$.

On exit: $\mathbf{Y}(i,j)$ contains an approximation to $y_i^{(j)}(x)$, for $i = 1, 2, \ldots, \mathbf{neq}$ and $j = 0, 1, \ldots, m_i - 1$. The remaining elements of **y** (where $m_i < \mathbf{mmax}$) are initialized to 0.0.

3: **neq** – Integer *Input*

On entry: the number of differential equations.

Constraint: **neq** must be the same value as supplied to nag_ode_bvp_coll_nlin_setup (d02tvc).

4: **mmax** – Integer *Input*

On entry: the maximal order of the differential equations, $\max(m_i)$, for $i = 1, 2, \ldots, \mathbf{neq}$.

Constraint: **mmax** must contain the maximum value of the components of the argument **m** as supplied to nag_ode_bvp_coll_nlin_setup (d02tvc).

5: **rcomm**[*dim*] – 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.

6: **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.

7: **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 $\langle value\rangle$ had an illegal value.

NE_CONVERGENCE_SOL

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

NE_INT_2

On entry, **mmax** $= \langle value\rangle$ and $\max(\mathbf{m}[i]) = \langle value\rangle$.
Constraint: **mmax** $= \max(\mathbf{m}[i])$.

NE_INT_CHANGED

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

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

The solver function does not appear to have been called.

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_REAL

On entry, $\mathbf{x} = \langle value\rangle$.
Constraint: $\mathbf{x} \leq \langle value\rangle$.

On entry, $\mathbf{x} = \langle value\rangle$.
Constraint: $\mathbf{x} \geq \langle value\rangle$.

NW_NOT_CONVERGED

The solver function did not converge to a suitable solution.
A converged intermediate solution has been used.
Interpolated values should be treated with caution.

NW_TOO_MUCH_ACC_REQUESTED

The solver function did not satisfy the error requirements.
Interpolated values should be treated with caution.

7 Accuracy

If nag_ode_bvp_coll_nlin_interp (d02tyc) returns the value **fail.code** $=$ NE_NOERROR, the computed values of the solution components y_i should be of similar accuracy to that specified by the argument **tols** of nag_ode_bvp_coll_nlin_setup (d02tvc). Note that during the solution process the error in the derivatives $y_i^{(j)}$, for $j = 1, 2, \ldots, m_i - 1$, has not been controlled and that the derivative values returned by nag_ode_bvp_coll_nlin_interp (d02tyc) are computed via differentiation of the piecewise polynomial approximation to y_i. See also Section 9.

8 Parallelism and Performance

nag_ode_bvp_coll_nlin_interp (d02tyc) 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_bvp_coll_nlin_interp (d02tyc) returns the value **fail.code**$=$ NW_NOT_CONVERGED, then the accuracy of the interpolated values may be proportional to the quantity **ermx** as returned by nag_ode_bvp_coll_nlin_diag (d02tzc).

If nag_ode_bvp_coll_nlin_solve (d02tlc) returned a value for **fail.code** other than **fail.code** $=$ NE_NOERROR, then nothing can be said regarding either the quality or accuracy of the values computed by nag_ode_bvp_coll_nlin_interp (d02tyc).

10 Example

The following example is used to illustrate that a system with singular coefficients can be treated without modification of the system definition. 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_diag (d02tzc), for the illustration of other facilities.

Consider the Thomas–Fermi equation used in the investigation of potentials and charge densities of ionized atoms. See Grossman (1992), for example, and the references therein. The equation is

$$y'' = x^{-1/2} y^{3/2}$$

with boundary conditions

$$y(0) = 1, \quad y(a) = 0, \quad a > 0.$$

The coefficient $x^{-1/2}$ implies a singularity at the left-hand boundary $x = 0$.

We use the initial approximation $y(x) = 1 - x/a$, which satisfies the boundary conditions, on a uniform mesh of six points. For illustration we choose $a = 1$, as in Grossman (1992). Note that in **ffun** and **fjac** (see nag_ode_bvp_coll_nlin_solve (d02tlc)) we have taken the precaution of setting the function value and Jacobian value to 0.0 in case a value of y becomes negative, although starting from our initial solution profile this proves unnecessary during the solution phase. Of course the true solution $y(x)$ is positive for all $x < a$.

10.1 Program Text

```
/* nag_ode_bvp_coll_nlin_interp (d02tyc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

#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, nmesh_out = 11;
  Integer neq, mmax, nlbc, nrbc;
  Integer i, iermx, ijermx, licomm, lrcomm, mxmesh, ncol, nmesh;
  double a, ainc, ermx, x;
  /* Arrays */
  static Integer iuser[7] = { -1, -1, -1, -1, -1, -1, -1 };
  double *mesh = 0, *rcomm = 0, *tol = 0, *y = 0;
  double rdum[1], ruser[1];
  Integer *ipmesh = 0, *icomm = 0, *m = 0;
  Integer idum[2];
  /* Nag Types */
  Nag_Boolean failed = Nag_FALSE;
  Nag_Comm comm;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_ode_bvp_coll_nlin_interp (d02tyc) Example Program Results\n\n");
  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
```

```
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "", &neq, &mmax);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "", &neq, &mmax);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &nlbc, &nrbc);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &nlbc, &nrbc);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &ncol, &nmesh,
          &mxmesh);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &ncol, &nmesh,
        &mxmesh);
#endif
  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;
  }

  comm.user = ruser;
  comm.iuser = iuser;

  for (i = 0; i < neq; i++) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &m[i]);
#else
    scanf("%" NAG_IFMT "", &m[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

#ifdef _WIN32
  scanf_s("%lf%*[^\n] ", &a);
#else
  scanf("%lf%*[^\n] ", &a);
#endif

  for (i = 0; i < neq; i++) {
#ifdef _WIN32
    scanf_s("%lf", &tol[i]);
#else
    scanf("%lf", &tol[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  ainc = a / (double) (nmesh - 1);
  mesh[0] = 0.0;
  for (i = 1; i < nmesh - 1; i++) {
    mesh[i] = mesh[i - 1] + ainc;
  }
  mesh[nmesh - 1] = a;

  ipmesh[0] = 1;
  for (i = 1; i < nmesh - 1; i++) {
    ipmesh[i] = 2;
  }
  ipmesh[nmesh - 1] = 1;

  /* For communication with function guess store a in comm.user[0]. */
  ruser[0] = a;

  /* 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;
  }

  printf(" Tolerance = %8.1e, a = %8.2f\n", tol[0], a);
  /* 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;
  }

  /* 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 %4" NAG_IFMT "  points\n", nmesh);

  printf(" Maximum error = %10.2e  in interval %4" NAG_IFMT "", ermx, iermx);
  printf("  for component %4" NAG_IFMT " \n\n\n", ijermx);
  printf(" Mesh points:\n");
  for (i = 0; i < nmesh; i++) {
    printf("%4" NAG_IFMT "(%1" NAG_IFMT ")%13.4e%s", i + 1, ipmesh[i],
           mesh[i], (i + 1) % 4 ? "" : "\n");
  }
  printf("\n");

  if (!failed) {
    /* Print solution on output mesh. */
    printf("\n\n Computed solution\n     x       solution   derivative\n");
    x = 0.0;
    ainc = a / (double) (nmesh_out - 1);
    for (i = 0; i < nmesh_out; i++) {

      /* 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(x, 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("%8.2f %11.5f %11.5f\n", x, y[0], y[neq]);
      x = x + ainc;
    }
  }

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)
{
  if (comm->iuser[0] == -1) {
    printf("(User-supplied callback ffun, first invocation.)\n");
    comm->iuser[0] = 0;
  }
  if (y[0] <= 0.0) {
    f[0] = 0.0;
  }
  else {
    f[0] = pow(y[0], 1.5) / sqrt(x);
  }
}

static void NAG_CALL fjac(double x, const double y[], Integer neq,
                          const Integer m[], double dfdy[], Nag_Comm *comm)
{
  if (comm->iuser[1] == -1) {
    printf("(User-supplied callback fjac, first invocation.)\n");
    comm->iuser[1] = 0;
  }
  if (y[0] <= 0.0) {
    dfdy[0] = 0.0;
  }
  else {
    dfdy[0] = 1.5 * sqrt(y[0]) / sqrt(x);
  }
```

```
}

static void NAG_CALL gafun(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double ga[], Nag_Comm *comm)
{
  if (comm->iuser[2] == -1) {
    printf("(User-supplied callback gafun, first invocation.)\n");
    comm->iuser[2] = 0;
  }
  ga[0] = ya[0] - 1.0;
}

static void NAG_CALL gbfun(const double yb[], Integer neq, const Integer m[],
                           Integer nrbc, double gb[], Nag_Comm *comm)
{
  if (comm->iuser[3] == -1) {
    printf("(User-supplied callback gbfun, first invocation.)\n");
    comm->iuser[3] = 0;
  }
  gb[0] = yb[0];
}

static void NAG_CALL gajac(const double ya[], Integer neq, const Integer m[],
                           Integer nlbc, double dgady[], Nag_Comm *comm)
{
  if (comm->iuser[4] == -1) {
    printf("(User-supplied callback gajac, first invocation.)\n");
    comm->iuser[4] = 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->iuser[5] == -1) {
    printf("(User-supplied callback gbjac, first invocation.)\n");
    comm->iuser[5] = 0;
  }
  dgbdy[0] = 1.0;
}

static void NAG_CALL guess(double x, Integer neq, const Integer m[],
                           double y[], double dym[], Nag_Comm *comm)
{
  double a = comm->user[0];
  if (comm->iuser[6] == -1) {
    printf("(User-supplied callback guess, first invocation.)\n");
    comm->iuser[6] = 0;
  }
  y[0] = 1.0 - x / (a);
  y[neq] = -1.0 / (a);
  dym[0] = 0.0;
}
```

10.2 Program Data

```
nag_ode_bvp_coll_nlin_interp (d02tyc) Example Program Data
   1  2  1  1     : neq, mmax, nlbc, nrbc
   4  6  100      : ncol, nmesh, mxmesh
   2              : m
   1.0            : a
   1.0e-5         : tol
```

10.3 Program Results

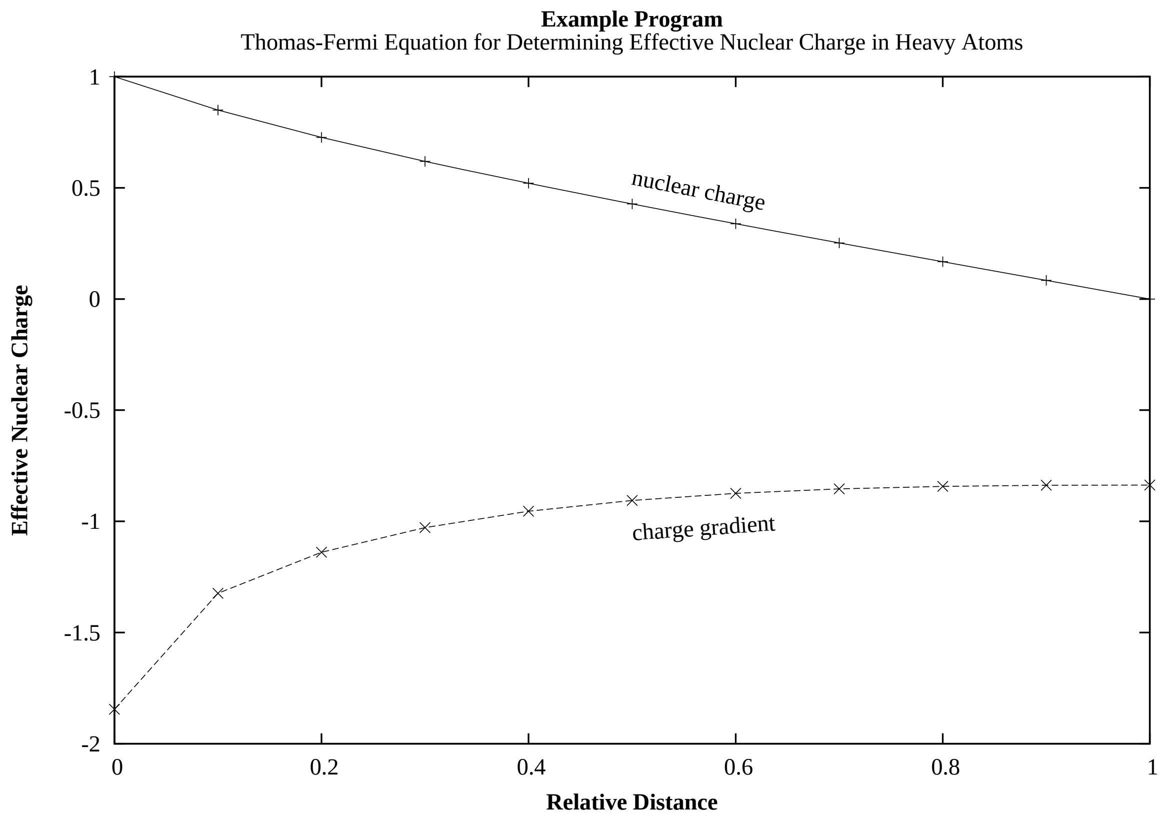
```
nag_ode_bvp_coll_nlin_interp (d02tyc) Example Program Results

 Tolerance =  1.0e-05, a =     1.00
(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   11  points
 Maximum error =   3.09e-06  in interval    1  for component    1


 Mesh points:
   1(1)   0.0000e+00   2(3)   1.0000e-01   3(2)   2.0000e-01   4(3)   3.0000e-01
   5(2)   4.0000e-01   6(3)   5.0000e-01   7(2)   6.0000e-01   8(3)   7.0000e-01
   9(2)   8.0000e-01  10(3)   9.0000e-01  11(1)   1.0000e+00


 Computed solution
     x       solution   derivative
    0.00     1.00000    -1.84496
    0.10     0.84944    -1.32330
    0.20     0.72721    -1.13911
    0.30     0.61927    -1.02776
    0.40     0.52040    -0.95468
    0.50     0.42754    -0.90583
    0.60     0.33867    -0.87372
    0.70     0.25239    -0.85369
    0.80     0.16764    -0.84248
    0.90     0.08368    -0.83756
    1.00     0.00000    -0.83655
```


Example Program
Thomas-Fermi Equation for Determining Effective Nuclear Charge in Heavy Atoms
1
0.5
0
-0.5
-1
-1.5
-2
0
0.2
0.4
0.6
0.8
1
Effective Nuclear Charge
Relative Distance
nuclear charge
charge gradient