NAG Library Function Document

nag_ode_bvp_ps_lin_solve (d02uec)

1 Purpose

nag_ode_bvp_ps_lin_solve (d02uec) finds the solution of a linear constant coefficient boundary value problem by using the Chebyshev integration formulation on a Chebyshev Gauss–Lobatto grid.

2 Specification

```
#include <nag.h>
#include <nagd02.h>
void nag_ode_bvp_ps_lin_solve (Integer n, double a, double b, Integer m,
     const double c[], double bmat[], const double y[], const double bvec[],
     double f[], double uc[], double *resid, NagError *fail)
```

3 Description

nag_ode_bvp_ps_lin_solve (d02uec) solves the constant linear coefficient ordinary differential problem

$$\sum_{j=0}^{m} f_{j+1} \frac{d^j u}{dx^j} = f(x), \quad x \in [a,b]$$

subject to a set of m linear constraints at points $y_i \in [a,b]$, for $i = 1,2,\ldots,m$:

$$\sum_{j=0}^{m} B_{i,j+1} \left(\frac{d^j u}{dx^j} \right)_{(x=y_i)} = \beta_i,$$

where $1 \leq m \leq 4$, B is an $m \times (m+1)$ matrix of constant coefficients and β_i are constants. The points y_i are usually either a or b.

The function $f(x)$ is supplied as an array of Chebyshev coefficients c_j, $j = 0,1,\ldots,n$ for the function discretized on $n+1$ Chebyshev Gauss–Lobatto points (as returned by nag_ode_bvp_ps_lin_cgl_grid (d02ucc)); the coefficients are normally obtained by a previous call to nag_ode_bvp_ps_lin_coeffs (d02uac). The solution and its derivatives (up to order m) are returned, in the form of their Chebyshev series representation, as arrays of Chebyshev coefficients; subsequent calls to nag_ode_bvp_ps_lin_cgl_vals (d02ubc) will return the corresponding function and derivative values at the Chebyshev Gauss–Lobatto discretization points on $[a,b]$. Function and derivative values can be obtained on any uniform grid over the same range $[a,b]$ by calling the interpolation function nag_ode_bvp_ps_lin_grid_vals (d02uwc).

4 References

Clenshaw C W (1957) The numerical solution of linear differential equations in Chebyshev series *Proc. Camb. Phil. Soc.* **53** 134–149

Coutsias E A, Hagstrom T and Torres D (1996) An efficient spectral method for ordinary differential equations with rational function coefficients *Mathematics of Computation* **65(214)** 611–635

Greengard L (1991) Spectral integration and two-point boundary value problems *SIAM J. Numer. Anal.* **28(4)** 1071–80

Lundbladh A, Hennigson D S and Johannson A V (1992) An efficient spectral integration method for the solution of the Navier–Stokes equations *Technical report FFA–TN* 1992–28 Aeronautical Research Institute of Sweden

Muite B K (2010) A numerical comparison of Chebyshev methods for solving fourth-order semilinear initial boundary value problems *Journal of Computational and Applied Mathematics* **234(2)** 317–342

5 Arguments

1: **n** – Integer *Input*

On entry: n, where the number of grid points is $n+1$.

Constraint: $\mathbf{n} \geq 8$ and **n** is even.

2: **a** – double *Input*

On entry: a, the lower bound of domain $[a,b]$.

Constraint: $\mathbf{a} < \mathbf{b}$.

3: **b** – double *Input*

On entry: b, the upper bound of domain $[a,b]$.

Constraint: $\mathbf{b} > \mathbf{a}$.

4: **m** – Integer *Input*

On entry: the order, m, of the boundary value problem to be solved.

Constraint: $1 \leq \mathbf{m} \leq 4$.

5: **c**[**n** + **1**] – const double *Input*

On entry: the Chebyshev coefficients c_j, $j = 0,1,\ldots,n$, for the right hand side of the boundary value problem. Usually these are obtained by a previous call of nag_ode_bvp_ps_lin_coeffs (d02uac).

6: **bmat**[**m** × (**m** + **1**)] – double *Input/Output*

Note: the (i,j)th element of the matrix is stored in **bmat**$[(j-1)\times \mathbf{m} + i - 1]$.

On entry: **bmat**$[j \times \mathbf{m} + i - 1]$ must contain the coefficients $B_{i,j+1}$, for $i = 1,2,\ldots,m$ and $j = 0,1,\ldots,m$, in the problem formulation of Section 3.

On exit: the coefficients have been scaled to form an equivalent problem defined on the domain $[-1,1]$.

7: **y**[**m**] – const double *Input*

On entry: the points, y_i, for $i = 1,2,\ldots,m$, where the boundary conditions are discretized.

8: **bvec**[**m**] – const double *Input*

On entry: the values, β_i, for $i = 1,2,\ldots,m$, in the formulation of the boundary conditions given in Section 3.

9: **f**[**m** + **1**] – double *Input/Output*

On entry: the coefficients, f_j, for $j = 1,2,\ldots,m+1$, in the formulation of the linear boundary value problem given in Section 3. The highest order term, **f**[**m**], needs to be nonzero to have a well posed problem.

On exit: the coefficients have been scaled to form an equivalent problem defined on the domain $[-1,1]$.

10: **uc**[(**n** + **1**) × (**m** + **1**)] – double *Output*

Note: the (i, j)th element of the matrix is stored in **uc**$[(j-1) \times (\mathbf{n}+1) + i - 1]$.

On exit: the Chebyshev coefficients in the Chebyshev series representations of the solution and derivatives of the solution to the boundary value problem. The coefficients of U are stored as the first $n+1$ elements of **uc**, the first derivative coefficients are stored as the next $n+1$ elements of **uc**, and so on.

11: **resid** – double * *Output*

On exit: the maximum residual resulting from substituting the solution vectors returned in **uc** into both linear equations of Section 3 representing the linear boundary value problem and associated boundary conditions. That is

$$\max\left\{\max_{i=1,m}\left(\left|\sum_{j=0}^{m} B_{i,j+1}\left(\frac{d^j u}{dx^j}\right)_{(x=y_i)} - \beta_i\right|\right), \max_{i=1,n+1}\left(\left|\sum_{j=0}^{m} f_{j+1}\left(\frac{d^j u}{dx^j}\right)_{(x=x_i)} - f(x)\right|\right)\right\}.$$

12: **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 3.2.1.2 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_CONVERGENCE

During iterative refinement, convergence was achieved, but the residual is more than 100 × ***machine precision***. Residual achieved on convergence = ⟨*value*⟩.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: $1 \leq \mathbf{m} \leq 4$.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** is even.

On entry, **n** = ⟨*value*⟩.
Constraint: $\mathbf{n} \geq 8$.

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 3.6.6 in How to Use the NAG Library and its Documentation for further information.

Internal error while unpacking matrix during iterative refinement.
Please contact NAG.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

NE_REAL_2

On entry, $\mathbf{a} = \langle value \rangle$ and $\mathbf{b} = \langle value \rangle$.
Constraint: $\mathbf{a} < \mathbf{b}$.

NE_REAL_ARRAY

On entry, $\mathbf{f}[\mathbf{m}] = 0.0$.

NE_SINGULAR_MATRIX

Singular matrix encountered during iterative refinement.
Please check that your system is well posed.

NE_TOO_MANY_ITER

During iterative refinement, the maximum number of iterations was reached.
Number of iterations $= \langle value \rangle$ and residual achieved $= \langle value \rangle$.

7 Accuracy

The accuracy should be close to ***machine precision*** for well conditioned boundary value problems.

8 Parallelism and Performance

nag_ode_bvp_ps_lin_solve (d02uec) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_ode_bvp_ps_lin_solve (d02uec) 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' Notefor your implementation for any additional implementation-specific information.

9 Further Comments

The number of operations is of the order $n\log(n)$ and the memory requirements are $O(n)$; thus the computation remains efficient and practical for very fine discretizations (very large values of n). Collocation methods will be faster for small problems, but the method of nag_ode_bvp_ps_lin_solve (d02uec) should be faster for larger discretizations.

10 Example

This example solves the third-order problem $4U_{xxx} + 3U_{xx} + 2U_x + U = 2\sin x - 2\cos x$ on $[-\pi/2, \pi/2]$ subject to the boundary conditions $U[-\pi/2] = 0$, $3U_{xx}[-\pi/2] + 2U_x[-\pi/2] + U[-\pi/2] = 2$, and $3U_{xx}[\pi/2] + 2U_x[\pi/2] + U[\pi/2] = -2$ using the Chebyshev integration formulation on a Chebyshev Gauss–Lobatto grid of order 16.

10.1 Program Text

```
/* nag_ode_bvp_ps_lin_solve (d02uec) 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>
#include <nagx01.h>
#include <nagx02.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static double NAG_CALL exact(double x, Integer q);
  static void NAG_CALL bndary(Integer m, double a, double b, double y[],
                              double bmat[], double bvec[]);
  static void NAG_CALL pdedef(Integer m, double f[]);
#ifdef __cplusplus
}
#endif

int main(void)
{
  /* Scalars */
  Integer exit_status = 0;
  double a = -nag_pi / 2.0, b = nag_pi / 2.0, resid;
  Integer i, j, n, m = 3;
  double teneps = 10.0 * nag_machine_precision;
  /* Arrays */
  double *bmat = 0, *bvec = 0, *f = 0, *uerr = 0, *y = 0, *c = 0, *f0 = 0,
         *u = 0, *uc = 0, *x = 0;
  /* NAG types */
  Nag_Boolean reqerr = Nag_FALSE;
  NagError fail;

  INIT_FAIL(fail);

  printf("nag_ode_bvp_ps_lin_solve (d02uec) 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 "%*[^\n] ", &n);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &n);
#endif
  if (!(u = NAG_ALLOC((n + 1) * (m + 1), double)) ||
      !(f0 = NAG_ALLOC((n + 1), double)) ||
      !(c = NAG_ALLOC((n + 1), double)) ||
      !(uc = NAG_ALLOC((n + 1) * (m + 1), double)) ||
      !(x = NAG_ALLOC((n + 1), double)) ||
      !(bmat = NAG_ALLOC(m * (m + 1), double)) ||
      !(bvec = NAG_ALLOC(m, double)) ||
      !(f = NAG_ALLOC((m + 1), double)) ||
      !(uerr = NAG_ALLOC((m + 1), double)) || !(y = NAG_ALLOC(m, double))
         )
  {
    printf("Allocation failure\n");
```

```
    exit_status = -1;
    goto END;
  }

  /* Set up domain, boundary conditions and definition. */
  bndary(m, a, b, y, bmat, bvec);
  pdedef(m, f);

  /* nag_ode_bvp_ps_lin_cgl_grid (d02ucc).
   * Generate Chebyshev Gauss-Lobatto solution grid.
   */
  nag_ode_bvp_ps_lin_cgl_grid(n, a, b, x, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_bvp_ps_lin_cgl_grid (d02ucc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Set up problem right-hand sides for grid and transform. */
  for (i = 0; i < n + 1; i++)
    f0[i] = 2.0 * sin(x[i]) - 2.0 * cos(x[i]);

  /* nag_ode_bvp_ps_lin_coeffs (d02uac).
   * Coefficients of Chebyshev interpolating polynomial from function values f0
   * on Chebyshev grid.
   */
  nag_ode_bvp_ps_lin_coeffs(n, f0, c, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_bvp_ps_lin_coeffs (d02uac).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_ode_bvp_ps_lin_solve (d02uec).
   * Solve given boundary value problem on Chebyshev grid, in coefficient space
   * using an integral formulation of the pseudospectral method.
   */
  nag_ode_bvp_ps_lin_solve(n, a, b, m, c, bmat, y, bvec, f, uc, &resid,
                           &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_bvp_ps_lin_solve (d02uec).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_ode_bvp_ps_lin_cgl_vals (d02ubc).
   * Obtain function values from coefficients of Chebyshev polynomial.
   * Also obtain first- to third-derivative values.
   */
  for (i = 0; i < m + 1; i++) {
    nag_ode_bvp_ps_lin_cgl_vals(n, a, b, i, &uc[(n + 1) * i], &u[(n + 1) * i],
                                &fail);
    if (fail.code != NE_NOERROR) {
      printf("Error from nag_ode_bvp_ps_lin_cgl_vals (d02ubc).\n%s\n",
             fail.message);
      exit_status = 1;
      goto END;
    }
  }
  /* Print solution. */
  printf("Numerical Solution U and its first three derivatives\n\n");
  printf("%7s%12s%12s%11s%11s\n", "x", "U", "Ux", "Uxx", "Uxxx");
  for (i = 0; i < n + 1; i++)
    printf("%10.4f %10.4f %10.4f %10.4f %10.4f\n", x[i], u[i], u[(n + 1) + i],
           u[(n + 1) * 2 + i], u[(n + 1) * 3 + i]);

  if (reqerr) {
    for (i = 0; i < m + 1; i++)
```

```
      uerr[i] = 0.0;

    for (i = 0; i < n + 1; i++)
      for (j = 0; j <= m; j++)
        uerr[j] = MAX(uerr[j], fabs(u[(n + 1) * j + i] - exact(x[i], j)));

    for (i = 0; i <= m; i++) {
      printf("Error in the order %1" NAG_IFMT " derivative of U is < %8"
             NAG_IFMT " * machine precision.\n", i,
             10 * ((Integer) (uerr[i] / teneps) + 1));
    }
  }
END:
  NAG_FREE(c);
  NAG_FREE(f0);
  NAG_FREE(u);
  NAG_FREE(uc);
  NAG_FREE(x);
  NAG_FREE(bmat);
  NAG_FREE(bvec);
  NAG_FREE(f);
  NAG_FREE(uerr);
  NAG_FREE(y);
  return exit_status;
}

static double NAG_CALL exact(double x, Integer q)
{
  switch (q) {
  case 0:
    return cos(x);
    break;
  case 1:
    return -sin(x);
    break;
  case 2:
    return -cos(x);
    break;
  case 3:
    return sin(x);
    break;
  }
  return 0.0;
}

static void NAG_CALL bndary(Integer m, double a, double b, double y[],
                            double bmat[], double bvec[])
{
  Integer i;
  /* Boundary condition on left side of domain. */
  for (i = 0; i < 2; i++)
    y[i] = a;

  y[2] = b;
  /* Set up Dirichlet condition using exact solution at x = a. */
  for (i = 0; i < m * (m + 1); i++)
    bmat[i] = 0.0;
  for (i = 0; i < 3; i++)
    bmat[i] = 1.0;
  for (i = 1; i < 3; i++)
    bmat[m + i] = 2.0;
  for (i = 1; i < 3; i++)
    bmat[m * 2 + i] = 3.0;

  bvec[0] = 0.0;
  bvec[1] = 2.0;
  bvec[2] = -2.0;
}

static void NAG_CALL pdedef(Integer m, double f[])
{
```

```
  f[0] = 1.0;
  f[1] = 2.0;
  f[2] = 3.0;
  f[3] = 4.0;
}
```

10.2 Program Data

```
nag_ode_bvp_ps_lin_solve (d02uec) Example Program Data
  16              : n
```

10.3 Program Results

```
nag_ode_bvp_ps_lin_solve (d02uec) Example Program Results

Numerical Solution U and its first three derivatives

      x           U          Ux         Uxx        Uxxx
   -1.5708    -0.0000     1.0000     0.0000    -1.0000
   -1.5406     0.0302     0.9995    -0.0302    -0.9995
   -1.4512     0.1193     0.9929    -0.1193    -0.9929
   -1.3061     0.2616     0.9652    -0.2616    -0.9652
   -1.1107     0.4440     0.8960    -0.4440    -0.8960
   -0.8727     0.6428     0.7661    -0.6428    -0.7661
   -0.6011     0.8247     0.5656    -0.8247    -0.5656
   -0.3064     0.9534     0.3017    -0.9534    -0.3017
   -0.0000     1.0000     0.0000    -1.0000    -0.0000
    0.3064     0.9534    -0.3017    -0.9534     0.3017
    0.6011     0.8247    -0.5656    -0.8247     0.5656
    0.8727     0.6428    -0.7661    -0.6428     0.7661
    1.1107     0.4440    -0.8960    -0.4440     0.8960
    1.3061     0.2616    -0.9652    -0.2616     0.9652
    1.4512     0.1193    -0.9929    -0.1193     0.9929
    1.5406     0.0302    -0.9995    -0.0302     0.9995
    1.5708    -0.0000    -1.0000     0.0000     1.0000
```