NAG Library Function Document

nag_ode_bvp_ps_lin_cgl_deriv (d02udc)

1 Purpose

nag_ode_bvp_ps_lin_cgl_deriv (d02udc) differentiates a function discretized on Chebyshev Gauss–Lobatto points. The grid points on which the function values are to be provided are normally returned by a previous call to nag_ode_bvp_ps_lin_cgl_grid (d02ucc).

2 Specification

```
#include <nag.h>
#include <nagd02.h>
```

```
void nag_ode_bvp_ps_lin_cgl_deriv (Integer n, const double f[], double fd[],
     NagError *fail)
```

3 Description

nag_ode_bvp_ps_lin_cgl_deriv (d02udc) differentiates a function discretized on Chebyshev Gauss–Lobatto points on $[-1,1]$. The polynomial interpolation on Chebyshev points is equivalent to trigonometric interpolation on equally spaced points. Hence the differentiation on the Chebyshev points can be implemented by the Fast Fourier transform (FFT).

Given the function values $f(x_i)$ on Chebyshev Gauss–Lobatto points $x_i = -\cos((i-1)\pi/n)$, for $i = 1, 2, \ldots, n+1$, f is differentiated with respect to x by means of forward and backward FFTs on the function values $f(x_i)$. nag_ode_bvp_ps_lin_cgl_deriv (d02udc) returns the computed derivative values $f'(x_i)$, for $i = 1, 2, \ldots, n+1$. The derivatives are computed with respect to the standard Chebyshev Gauss–Lobatto points on $[-1,1]$; for derivatives of a function on $[a,b]$ the returned values have to be scaled by a factor $2/(b-a)$.

4 References

Canuto C, Hussaini M Y, Quarteroni A and Zang T A (2006) *Spectral Methods: Fundamentals in Single Domains* Springer

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

Trefethen L N (2000) *Spectral Methods in MATLAB* SIAM

5 Arguments

1: **n** – Integer *Input*

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

Constraint: $\mathbf{n} > 0$ and $\mathbf{n}$ is even.

2: $\mathbf{f}[\mathbf{n}+\mathbf{1}]$ – const double *Input*

On entry: the function values $f(x_i)$, for $i = 1, 2, \ldots, n+1$

3: $\mathbf{fd}[\mathbf{n}+\mathbf{1}]$ – double *Output*

On exit: the approximations to the derivatives of the function evaluated at the Chebyshev Gauss–Lobatto points. For functions defined on $[a,b]$, the returned derivative values (corresponding to the domain $[-1, 1]$) must be multiplied by the factor $2/(b - a)$ to obtain the correct values on $[a, b]$.

4: **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_INT

On entry, **n** $= \langle$*value*$\rangle$.
Constraint: **n** > 0.

On entry, **n** $= \langle$*value*$\rangle$.
Constraint: **n** is even.

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_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.

7 Accuracy

The accuracy is close to ***machine precision*** for small numbers of grid points, typically less than 100. For larger numbers of grid points, the error in differentiation grows with the number of grid points. See Greengard (1991) for more details.

8 Parallelism and Performance

nag_ode_bvp_ps_lin_cgl_deriv (d02udc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

nag_ode_bvp_ps_lin_cgl_deriv (d02udc) 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

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).

10 Example

The function $2x + \exp(-x)$, defined on $[0, 1.5]$, is supplied and then differentiated on a grid.

10.1 Program Text

```
/* nag_ode_bvp_ps_lin_cgl_deriv (d02udc) 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 <nagx02.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static double NAG_CALL fcn(double x);
  static double NAG_CALL deriv(double x);
#ifdef __cplusplus
}
#endif

int main(void)
{
  /*  Scalars */
  Integer exit_status = 0;
  Integer i, n;
  double a = 0.0, b = 1.5, scale;
  double teneps = 100.0 * nag_machine_precision;
  double uxerr = 0.0;
  /*  Arrays */
  double *f = 0, *fd = 0, *x = 0;
  /* NAG types */
  Nag_Boolean reqerr = Nag_FALSE;
  NagError fail;

  INIT_FAIL(fail);
  printf("nag_ode_bvp_ps_lin_cgl_deriv (d02udc) 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 (!(f = NAG_ALLOC((n + 1), double)) ||
      !(fd = NAG_ALLOC((n + 1), double)) || !(x = NAG_ALLOC((n + 1), double))
        )
  {
```

```
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
  }

  /* 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;
  }

  /* Evaluate the function on Chebyshev grid. */
  for (i = 0; i < n + 1; i++)
    f[i] = fcn(x[i]);

  /* nag_ode_bvp_ps_lin_cgl_deriv (d02udc).
   * Differentiate a function using function values on Chebyshev grid.
   */
  nag_ode_bvp_ps_lin_cgl_deriv(n, f, fd, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_ode_bvp_ps_lin_cgl_deriv (d02udc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  scale = 2.0 / (b - a);
  for (i = 0; i < n + 1; i++)
    fd[i] = scale * fd[i];

  /* Print function and its derivative. */
  printf("Original function f and numerical derivative fx\n\n");
  printf("%8s%11s%11s\n", "x", "f", "fx");
  for (i = 0; i < n + 1; i++)
    printf("%10.4f %10.4f %10.4f\n", x[i], f[i], fd[i]);

  if (reqerr) {
    for (i = 0; i < n + 1; i++)
      uxerr = MAX(uxerr, fabs(fd[i] - deriv(x[i])));
    printf("fx is within a multiple %8" NAG_IFMT
           " of machine precision.\n",
           100 * ((Integer) (uxerr / teneps) + 1));
  }
END:
  NAG_FREE(f);
  NAG_FREE(fd);
  NAG_FREE(x);
  return exit_status;
}

static double NAG_CALL fcn(double x)
{
  return 2.0 * x + exp(-x);
}

static double NAG_CALL deriv(double x)
{
  return 2.0 - exp(-x);
}
```

10.2 Program Data

```
nag_ode_bvp_ps_lin_cgl_deriv (d02udc) Example Program Data
  16              : n
```

10.3 Program Results

```
nag_ode_bvp_ps_lin_cgl_deriv (d02udc) Example Program Results

Original function f and numerical derivative fx

       x          f          fx
    0.0000     1.0000     1.0000
    0.0144     1.0145     1.0143
    0.0571     1.0587     1.0555
    0.1264     1.1341     1.1187
    0.2197     1.2421     1.1972
    0.3333     1.3832     1.2835
    0.4630     1.5554     1.3706
    0.6037     1.7542     1.4532
    0.7500     1.9724     1.5276
    0.8963     2.2007     1.5919
    1.0370     2.4285     1.6455
    1.1667     2.6448     1.6886
    1.2803     2.8386     1.7221
    1.3736     3.0004     1.7468
    1.4429     3.1221     1.7638
    1.4856     3.1975     1.7736
    1.5000     3.2231     1.7769
```