NAG Library Function Document

nag_4d_shep_eval (e01tlc)

1 Purpose

nag_4d_shep_eval (e01tlc) evaluates the four-dimensional interpolating function generated by nag_4d_shep_interp (e01tkc) and its first partial derivatives.

2 Specification

```
#include <nag.h>
#include <nage01.h>
void nag_4d_shep_eval (Integer m, const double x[], const double f[],
     const Integer iq[], const double rq[], Integer n, const double xe[],
     double q[], double qx[], NagError *fail)
```

3 Description

nag_4d_shep_eval (e01tlc) takes as input the interpolant $Q(\mathbf{x})$, $x \in \mathbb{R}^4$ of a set of scattered data points $(\mathbf{x}_r, f_r)$, for $r = 1, 2, \ldots, m$, as computed by nag_4d_shep_interp (e01tkc), and evaluates the interpolant and its first partial derivatives at the set of points $\mathbf{x}_i$, for $i = 1, 2, \ldots, n$.

nag_4d_shep_eval (e01tlc) must only be called after a call to nag_4d_shep_interp (e01tkc).

nag_4d_shep_eval (e01tlc) is derived from the new implementation of QS3GRD described by Renka (1988). It uses the modification for high-dimensional interpolation described by Berry and Minser (1999).

4 References

Berry M W, Minser K S (1999) Algorithm 798: high-dimensional interpolation using the modified Shepard method *ACM Trans. Math. Software* **25** 353–366

Renka R J (1988) Algorithm 661: QSHEP3D: Quadratic Shepard method for trivariate interpolation of scattered data *ACM Trans. Math. Software* **14** 151–152

5 Arguments

1: **m** – Integer *Input*

On entry: **must** be the same value supplied for argument **m** in the preceding call to nag_4d_shep_interp (e01tkc).

Constraint: $\mathbf{m} \geq 16$.

2: **x**[**4** × **m**] – const double *Input*

Note: the coordinates of x_r are stored in $\mathbf{x}[(r-1) \times 4] \ldots \mathbf{x}[(r-1) \times 4 + 3]$.

On entry: **must** be the same array supplied as argument **x** in the preceding call to nag_4d_shep_interp (e01tkc). It **must** remain unchanged between calls.

3: **f**[**m**] – const double *Input*

On entry: **must** be the same array supplied as argument **f** in the preceding call to nag_4d_shep_interp (e01tkc). It **must** remain unchanged between calls.

4: **iq**[**2** × **m** + **1**] – const Integer *Input*

On entry: **must** be the same array returned as argument **iq** in the preceding call to nag_4d_shep_interp (e01tkc). It **must** remain unchanged between calls.

5: **rq**[**15** × **m** + **9**] – const double *Input*

On entry: **must** be the same array returned as argument **rq** in the preceding call to nag_4d_shep_interp (e01tkc). It **must** remain unchanged between calls.

6: **n** – Integer *Input*

On entry: n, the number of evaluation points.

Constraint: **n** ≥ 1.

7: **xe**[**4** × **n**] – const double *Input*

Note: the (i,j)th element of the matrix is stored in **xe**$[(j-1) \times 4 + i - 1]$.

On entry: **xe**$[(r-1) \times 4], \ldots,$ **xe**$[(r-1) \times 4 + 3]$ must be set to the evaluation point $\mathbf{x}_i$, for $i = 1, 2, \ldots, n$.

8: **q**[**n**] – double *Output*

On exit: **q**$[i-1]$ contains the value of the interpolant, at $\mathbf{x}_i$, for $i = 1, 2, \ldots, n$. If any of these evaluation points lie outside the region of definition of the interpolant the corresponding entries in **q** are set to the largest machine representable number (see nag_real_largest_number (X02ALC)), and nag_4d_shep_eval (e01tlc) returns with **fail**.**code** = NE_BAD_POINT.

9: **qx**[**4** × **n**] – double *Output*

Note: the (i,j)th element of the matrix is stored in **qx**$[(j-1) \times 4 + i - 1]$.

On exit: **qx**$[(i-1) \times 4 + j - 1]$ contains the value of the partial derivatives with respect to $\mathbf{x}_j$ of the interpolant $Q(\mathbf{x})$ at $\mathbf{x}_i$, for $i = 1, 2, \ldots, n$, and for each of the four partial derivatives $j = 1, 2, 3, 4$. If any of these evaluation points lie outside the region of definition of the interpolant, the corresponding entries in **qx** are set to the largest machine representable number (see nag_real_largest_number (X02ALC)), and nag_4d_shep_eval (e01tlc) returns with **fail**.**code** = NE_BAD_POINT.

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_BAD_POINT

On entry, at least one evaluation point lies outside the region of definition of the interpolant. At all such points the corresponding values in **q** and **qx** have been set to nag_real_largest_number: nag_real_largest_number = ⟨*value*⟩.

NE_INT

On entry, **m** = ⟨*value*⟩.
Constraint: **m** ≥ 16.

On entry, **n** = ⟨*value*⟩.
Constraint: **n** ≥ 1.

NE_INT_ARRAY

On entry, values in **iq** appear to be invalid. Check that **iq** has not been corrupted between calls to nag_4d_shep_interp (e01tkc) and nag_4d_shep_eval (e01tlc).

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_REAL_ARRAY

On entry, values in **rq** appear to be invalid. Check that **rq** has not been corrupted between calls to nag_4d_shep_interp (e01tkc) and nag_4d_shep_eval (e01tlc).

7 Accuracy

Computational errors should be negligible in most practical situations.

8 Parallelism and Performance

nag_4d_shep_eval (e01tlc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

The time taken for a call to nag_4d_shep_eval (e01tlc) will depend in general on the distribution of the data points. If the data points are approximately uniformly distributed, then the time taken should be only $O(n)$. At worst $O(mn)$ time will be required.

10 Example

This program evaluates the function

$$f(x) = \frac{(1.25 + \cos(5.4x_4))\cos(6x_1)\cos(6x_2)}{6 + 6(3x_3 - 1)^2}$$

at a set of 30 randomly generated data points and calls nag_4d_shep_interp (e01tkc) to construct an interpolating function $Q(\mathbf{x})$. It then calls nag_4d_shep_eval (e01tlc) to evaluate the interpolant at a set of random points.

To reduce the time taken by this example, the number of data points is limited to 30. Increasing this value improves the interpolation accuracy at the expense of more time.

See also Section 10 in nag_4d_shep_interp (e01tkc).

10.1 Program Text

```
/* nag_4d_shep_eval (e01tlc) Example Program.
 *
 * Copyright 2011 Numerical Algorithms Group.
 *
 * Mark 23, 2010.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage01.h>
#include <nagg05.h>
```

```
#include <math.h>

#ifdef __cplusplus
extern "C" {
#endif
static double NAG_CALL funct(double x[]);
#ifdef __cplusplus
}
#endif

#define X(I, J) x[I *4 + J]
#define XE(I, J) xe[I *4 + J]

int main(void)
{
  /* Scalars */
  Integer     exit_status, i, m, n, nq, nw, liq, lrq, lstate, subid;
  Integer     lseed = 1;
  double      fun;
  Nag_BaseRNG genid;
  NagError    fail;
  /* Arrays */
  double      *f = 0, *q = 0, *qx = 0, *rq = 0, *xe = 0, *x = 0;
  Integer     *iq = 0, *state = 0;
  Integer     seed[1], seed2[1];

  exit_status = 0;

  INIT_FAIL(fail);

  printf("nag_4d_shep_eval (e01tlc) Example Program Results\n");

  /* Skip heading in data file */
  scanf("%*[^\n] ");

  /* Input the seeds. */
  scanf("%ld%ld%*[^\n] ", &seed[0], &seed2[0]);

  /* Choose the base generator */
  genid = Nag_Basic;
  subid = 0;

  /* Get the length of the state array */
  lstate = -1;
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
            fail.message);
    exit_status = 1;
    goto END;
  }

  /* Input the number of nodes. */
  scanf("%ld%*[^\n] ", &m);

  /* Allocate memory */
  lrq = 21 * m + 11;
  liq = 2 * m + 1;
  if (!(f = NAG_ALLOC(m, double)) ||
      !(x = NAG_ALLOC(m*4, double)) ||
      !(rq = NAG_ALLOC(lrq, double)) ||
      !(iq = NAG_ALLOC(liq, Integer)) ||
      !(state = NAG_ALLOC(lstate, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Initialise the generator to a repeatable sequence */
  nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
```

```
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",fail.message);
    exit_status = 1;
    goto END;
  }

  /* Generate the data points X */
  nag_rand_basic(m*4, state, x, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_basic (g05sac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Evaluate F */
  for (i = 0; i < m; ++i) {
    f[i] = funct(&X(i,0));
  }

  /* Generate the interpolant. */
  nq = 0;
  nw = 0;

  /* nag_4d_shep_interp (e01tkc).
   * Interpolating functions, modified Shepard's method, four
   * variables
   */
  nag_4d_shep_interp(m, x, f, nw, nq, iq, rq, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_4d_shep_interp (e01tkc).\n%s\n",fail.message);
    exit_status = 1;
    goto END;
  }

  /* Input the number of evaluation points. */
  scanf("%ld%*[^\n] ", &n);

  /* Allocate memory for nag_4d_shep_eval (e01tlc) */
  if (!(q = NAG_ALLOC(n, double)) ||
      !(qx = NAG_ALLOC(n*4, double)) ||
      !(xe = NAG_ALLOC(n*4, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }

  /* Generate repeatable evaluation points. */
  nag_rand_init_repeatable(genid, subid, seed2, lseed, state, &lstate, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",fail.message);
    exit_status = 1;
    goto END;
  }
  nag_rand_basic(n*4, state, xe, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_basic (g05sac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* nag_4d_shep_eval (e01tlc).
   * Interpolated values, evaluate interpolant and first derivatives
   * computed by nag_4d_shep_interp (e01tkc).
   */
  fail.print = Nag_TRUE;
  nag_4d_shep_eval(m, x, f, iq, rq, n, xe, q, qx, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_4d_shep_eval (e01tlc).\n%s\n",fail.message);
    exit_status = 1;
    goto END;
```

```
  }

  printf("\n     i     f(x)      q(x)   |f(x)-q(x)|\n");
  for (i = 0; i < n; ++i) {
    fun = funct(&XE(i,0));
    printf("%6ld%10.4f%10.4f%10.4f\n", i, fun, q[i], fabs(fun-q[i]));
  }

 END:
  NAG_FREE(f);
  NAG_FREE(q);
  NAG_FREE(qx);
  NAG_FREE(rq);
  NAG_FREE(xe);
  NAG_FREE(x);
  NAG_FREE(iq);
  NAG_FREE(state);

  return exit_status;
}

static double NAG_CALL funct(double x[])
{
  /* Scalars */
  double ret_val;

  ret_val = ((1.25+cos(5.4*x[3]))*cos(6.0*x[0])*cos(6.0*x[1]))/
    (6.0*(1.0+pow((3.0*x[2]-1.0),2.0)));
  return ret_val;
}
```

10.2 Program Data

```
nag_4d_shep_eval (e01tlc) Example Program Data
1762543  43331            : random seeds
30                        : m the number of data points
8                         : n the number of evaluation points
```

10.3 Program Results

```
nag_4d_shep_eval (e01tlc) Example Program Results

     i     f(x)      q(x)   |f(x)-q(x)|
     0   -0.0189   -0.0394    0.0205
     1   -0.0186    0.0967    0.1153
     2    0.1147    0.0606    0.0541
     3    0.0096   -0.1313    0.1409
     4   -0.1354   -0.1878    0.0524
     5    0.0022   -0.1595    0.1617
     6   -0.0095   -0.1179    0.1084
     7    0.0113   -0.3950    0.4063
```