NAG Library Function Document

nag_2d_triang_interp (e01sjc)

1 Purpose

nag_2d_triang_interp (e01sjc) generates a two-dimensional surface interpolating a set of scattered data points, using the method of Renka and Cline.

2 Specification

```
#include <nag.h>
#include <nage01.h>
void nag_2d_triang_interp (Integer m, const double x[], const double y[],
     const double f[], Integer triang[], double grads[], NagError *fail)
```

3 Description

nag_2d_triang_interp (e01sjc) constructs an interpolating surface $F(x,y)$ through a set of m scattered data points (x_r, y_r, f_r), for $r = 1,2,\ldots,m$, using a method due to Renka and Cline. In the (x,y) plane, the data points must be distinct. The constructed surface is continuous and has continuous first derivatives.

The method involves firstly creating a triangulation with all the (x,y) data points as nodes, the triangulation being as nearly equiangular as possible (see Cline and Renka (1984)). Then gradients in the x- and y-directions are estimated at node r, for $r = 1,2,\ldots,m$, as the partial derivatives of a quadratic function of x and y which interpolates the data value f_r, and which fits the data values at nearby nodes (those within a certain distance chosen by the algorithm) in a weighted least squares sense. The weights are chosen such that closer nodes have more influence than more distant nodes on derivative estimates at node r. The computed partial derivatives, with the f_r values, at the three nodes of each triangle define a piecewise polynomial surface of a certain form which is the interpolant on that triangle. See Renka and Cline (1984) for more detailed information on the algorithm, a development of that by Lawson (1977). The code is derived from Renka (1984).

The interpolant $F(x,y)$ can subsequently be evaluated at any point (x,y) inside or outside the domain of the data by a call to nag_2d_triang_eval (e01skc). Points outside the domain are evaluated by extrapolation.

4 References

Cline A K and Renka R L (1984) A storage-efficient method for construction of a Thiessen triangulation *Rocky Mountain J. Math.* **14** 119–139

Lawson C L (1977) Software for C^1 surface interpolation *Mathematical Software III* (ed J R Rice) 161–194 Academic Press

Renka R L (1984) Algorithm 624: triangulation and interpolation of arbitrarily distributed points in the plane *ACM Trans. Math. Software* **10** 440–442

Renka R L and Cline A K (1984) A triangle-based C^1 interpolation method *Rocky Mountain J. Math.* **14** 223–237

5 Arguments

1: **m** – Integer *Input*

On entry: m, the number of data points.

Constraint: $\mathbf{m} \geq 3$.

2: **x**[**m**] – const double *Input*
3: **y**[**m**] – const double *Input*
4: **f**[**m**] – const double *Input*

On entry: the coordinates of the rth data point, for $r = 1, 2, \ldots, m$. The data points are accepted in any order, but see Section 9.

Constraint: the (x, y) nodes must not all be collinear, and each node must be unique.

5: **triang**[$7 \times \mathbf{m}$] – Integer *Output*

On exit: a data structure defining the computed triangulation, in a form suitable for passing to nag_2d_triang_eval (e01skc).

6: **grads**[$2 \times \mathbf{m}$] – double *Output*

Note: the (i, j)th element of the matrix is stored in **grads**$[(j-1) \times 2 + i - 1]$.

On exit: the estimated partial derivatives at the nodes, in a form suitable for passing to nag_2d_triang_eval (e01skc). The derivatives at node r with respect to x and y are contained in **grads**$[(r-1) \times 2]$ and **grads**$[(r-1) \times 2 + 1]$ respectively, for $r = 1, 2, \ldots, m$.

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_ALL_DATA_COLLINEAR

All nodes are collinear. There is no unique solution.

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 ⟨*value*⟩ had an illegal value.

NE_DATA_NOT_UNIQUE

On entry, $(\mathbf{x}[I-1], \mathbf{y}[I-1]) = (\mathbf{x}[J-1], \mathbf{y}[J-1])$, for $I, J = \langle value \rangle \langle value \rangle$, $\mathbf{x}[I-1]$, $\mathbf{y}[I-1] = \langle value \rangle \langle value \rangle$.

NE_INT

On entry, $\mathbf{m} = \langle value \rangle$.
Constraint: $\mathbf{m} \geq 3$.

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

On successful exit, the computational errors should be negligible in most situations but you should always check the computed surface for acceptability, by drawing contours for instance. The surface always interpolates the input data exactly.

8 Parallelism and Performance

nag_2d_triang_interp (e01sjc) is not threaded in any implementation.

9 Further Comments

The time taken for a call of nag_2d_triang_interp (e01sjc) is approximately proportional to the number of data points, m. The function is more efficient if, before entry, the values in **x**, **y** and **f** are arranged so that the **x** array is in ascending order.

10 Example

This example reads in a set of 30 data points and calls nag_2d_triang_interp (e01sjc) to construct an interpolating surface. It then calls nag_2d_triang_eval (e01skc) to evaluate the interpolant at a sample of points on a rectangular grid.

Note that this example is not typical of a realistic problem: the number of data points would normally be larger, and the interpolant would need to be evaluated on a finer grid to obtain an accurate plot, say.

10.1 Program Text

```
/* nag_2d_triang_interp (e01sjc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

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

int main(void)
{

  /* Scalars */
  double xhi, xlo, yhi, ylo;
  Integer exit_status, i, j, m, nx, ny;

  /* Arrays */
  double *f = 0, *grads = 0, *pf = 0, *px = 0, *py = 0, *x = 0, *y = 0;
  Integer *triang = 0;
```

```
  /* Nag Types */
  NagError fail;

  exit_status = 0;
  INIT_FAIL(fail);

  printf("nag_2d_triang_interp (e01sjc) Example Program Results\n");

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Input the number of nodes. */
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%*[^\n] ", &m);
#else
  scanf("%" NAG_IFMT "%*[^\n] ", &m);
#endif
  if (m >= 3) {
    /* Allocate memory */
    if (!(f = NAG_ALLOC(m, double)) ||
        !(grads = NAG_ALLOC(2 * m, double)) ||
        !(x = NAG_ALLOC(m, double)) ||
        !(y = NAG_ALLOC(m, double)) || !(triang = NAG_ALLOC(7 * m, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Invalid m.\n");
    exit_status = 1;
    goto END;
  }
  /* Input the nodes (X,Y) and heights, F. */
  for (i = 1; i <= m; ++i) {
#ifdef _WIN32
    scanf_s("%lf%lf%lf%*[^\n] ", &x[i - 1], &y[i - 1], &f[i - 1]);
#else
    scanf("%lf%lf%lf%*[^\n] ", &x[i - 1], &y[i - 1], &f[i - 1]);
#endif
  }
  /* Generate the triangulation and gradients. */

  /* nag_2d_triang_interp (e01sjc).
   * A function to generate a two-dimensional surface
   * interpolating a set of data points, using either the
   * method of Renka and Cline or the modified Shepard's
   * method
   */
  nag_2d_triang_interp(m, x, y, f, triang, grads, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_2d_triang_interp (e01sjc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Evaluate the interpolant on a rectangular grid at NX*NY points */
  /* over the domain (XLO to XHI) x (YLO to YHI). */

#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%lf%lf%*[^\n] ", &nx, &xlo, &xhi);
#else
  scanf("%" NAG_IFMT "%lf%lf%*[^\n] ", &nx, &xlo, &xhi);
#endif
#ifdef _WIN32
```

```
  scanf_s("%" NAG_IFMT "%lf%lf%*[^\n] ", &ny, &ylo, &yhi);
#else
  scanf("%" NAG_IFMT "%lf%lf%*[^\n] ", &ny, &ylo, &yhi);
#endif

  if (nx > 0 && ny > 0) {
    /* Allocate memory */
    if (!(pf = NAG_ALLOC(nx, double)) ||
        !(px = NAG_ALLOC(nx, double)) || !(py = NAG_ALLOC(ny, double)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Invalid nx or ny.\n");
    exit_status = 1;
    goto END;
  }

  for (i = 1; i <= nx; ++i) {
    px[i - 1] = (double) (nx - i) / (nx - 1) * xlo +
           (double) (i - 1) / (nx - 1) * xhi;
  }
  for (i = 1; i <= ny; ++i) {
    py[i - 1] = (double) (ny - i) / (ny - 1) * ylo +
           (double) (i - 1) / (ny - 1) * yhi;
  }
  printf("\n");
  printf("%s", "          X");
  for (i = 1; i <= nx; ++i) {
    printf("%8.2f", px[i - 1]);
  }
  printf("\n");
  printf("%s", "     Y");
  printf("\n");
  for (i = ny; i >= 1; --i) {
    for (j = 1; j <= nx; ++j) {
      /* nag_2d_triang_eval (e01skc).
       * A function to evaluate, at a set of points, the
       * two-dimensional interpolant function generated by
       * nag_2d_triang_interp (e01sjc).
       */
      nag_2d_triang_eval(m, x, y, f, triang, grads, px[j - 1],
                         py[i - 1], &pf[j - 1], &fail);
      if (fail.code != NE_NOERROR) {
        printf("Error from nag_2d_triang_eval (e01skc).\n%s\n", fail.message);
        exit_status = 1;
        goto END;
      }
    }
    printf("%8.2f", py[i - 1]);
    printf("%3s", "");
    for (j = 1; j <= nx; ++j) {
      printf("%8.2f", pf[j - 1]);
    }
    printf("\n");
  }
END:
  NAG_FREE(f);
  NAG_FREE(grads);
  NAG_FREE(pf);
  NAG_FREE(px);
  NAG_FREE(py);
  NAG_FREE(x);
  NAG_FREE(y);
  NAG_FREE(triang);

  return exit_status;
}
```

10.2 Program Data

```
nag_2d_triang_interp (e01sjc) Example Program Data
      30                      M, the number of data points
     11.16    1.24   22.15    X, Y, F data point definition
     12.85    3.06   22.11
     19.85   10.72    7.97
     19.72    1.39   16.83
     15.91    7.74   15.30
      0.00   20.00   34.60
     20.87   20.00    5.74
      3.45   12.78   41.24
     14.26   17.87   10.74
     17.43    3.46   18.60
     22.80   12.39    5.47
      7.58    1.98   29.87
     25.00   11.87    4.40
      0.00    0.00   58.20
      9.66   20.00    4.73
      5.22   14.66   40.36
     17.25   19.57    6.43
     25.00    3.87    8.74
     12.13   10.79   13.71
     22.23    6.21   10.25
     11.52    8.53   15.74
     15.20    0.00   21.60
      7.54   10.69   19.31
     17.32   13.78   12.11
      2.14   15.03   53.10
      0.51    8.37   49.43
     22.69   19.63    3.25
      5.47   17.13   28.63
     21.67   14.36    5.52
      3.31    0.33   44.08    End of the data points
   7    3.0   21.0            Grid definition, X axis
   6    2.0   17.0            Grid definition, Y axis
```

10.3 Program Results

```
nag_2d_triang_interp (e01sjc) Example Program Results

          X    3.00    6.00    9.00   12.00   15.00   18.00   21.00
     Y
   17.00       41.25   27.62   18.03   12.29   11.68    9.09    5.37
   14.00       47.61   36.66   22.87   14.02   13.44   11.20    6.46
   11.00       38.55   25.25   16.72   13.83   13.08   10.71    6.88
    8.00       37.90   23.97   16.79   16.43   15.46   13.02    9.30
    5.00       40.49   29.26   22.51   20.72   19.30   16.72   12.87
    2.00       43.52   33.91   26.59   22.23   21.15   18.67   14.88
```