NAG Library Function Document

nag_2d_triangulate (e01eac)

1 Purpose

nag_2d_triangulate (e01eac) generates a triangulation for a given set of two-dimensional points using the method of Renka and Cline.

2 Specification

```
#include <nag.h>
#include <nage01.h>
void nag_2d_triangulate (Integer n, const double x[], const double y[],
     Integer triang[], NagError *fail)
```

3 Description

nag_2d_triangulate (e01eac) creates a Thiessen triangulation with a given set of two-dimensional data points as nodes. This triangulation will be as equiangular as possible (Cline and Renka (1984)). 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 computed triangulation is returned in a form suitable for passing to nag_2d_triang_bary_eval (e01ebc) which, for a set of nodal function values, computes interpolated values at a set of points.

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: **n** – Integer *Input*

On entry: n, the number of data points.

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

2: **x**[**n**] – const double *Input*

On entry: the x coordinates of the n data points.

3: **y**[**n**] – const double *Input*

On entry: the y coordinates of the n data points.

4: **triang**[$7 \times$ **n**] – Integer *Output*

On exit: a data structure defining the computed triangulation, in a form suitable for passing to nag_2d_triang_bary_eval (e01ebc). Details of how the triangulation is encoded in **triang** are given in Section 9. These details are most likely to be of use when plotting the computed triangulation which is demonstrated in Section 10.

5: **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

On entry, all the (x, y) pairs are collinear.

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** ≥ 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

Not applicable.

8 Parallelism and Performance

nag_2d_triangulate (e01eac) is not threaded in any implementation.

9 Further Comments

The time taken for a call of nag_2d_triangulate (e01eac) is approximately proportional to the number of data points, n. The function is more efficient if, before entry, the (x, y) pairs are arranged in **x** and **y** such that the x values are in ascending order.

The triangulation is encoded in **triang** as follows:

set $j_0 = 0$; for each node, $k = 1, 2, \ldots, n$, (using the ordering inferred from **x** and **y**)

$i_k = j_{k-1} + 1$

$j_k =$ **triang**$[6 \times \mathbf{n} + k - 1]$

triang$[j-1]$, for $j = i_k, \ldots, j_k$, contains the list of nodes to which node k is connected. If **triang**$[j_k - 1] = 0$ then node k is on the boundary of the mesh.

10 Example

In this example, nag_2d_triangulate (e01eac) creates a triangulation from a set of data points. nag_2d_triang_bary_eval (e01ebc) then evaluates the interpolant at a sample of points using this triangulation. Note that this example is not typical of a realistic problem: the number of data points would normally be larger, so that interpolants can be more accurately evaluated at the fine triangulated grid.

10.1 Program Text

```
/* nag_2d_triangulate (e01eac) 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 */
  Integer exit_status, i, m, n;

  /* Arrays */
  double *f = 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_triangulate (e01eac) Example Program Results\n\n");

  /* Skip heading in data file and read array lengths */
#ifdef _WIN32
  scanf_s("%*[^\n]");
  scanf_s("%" NAG_IFMT "%*[^\n]", &n);
  scanf_s("%" NAG_IFMT "%*[^\n]", &m);
#else
  scanf("%*[^\n]");
  scanf("%" NAG_IFMT "%*[^\n]", &n);
  scanf("%" NAG_IFMT "%*[^\n]", &m);
#endif

  if (!(x = NAG_ALLOC(n, double)) ||
      !(y = NAG_ALLOC(n, double)) ||
      !(f = NAG_ALLOC(n, double)) ||
      !(triang = NAG_ALLOC(7 * n, Integer)) ||
      !(px = NAG_ALLOC(m, double)) ||
      !(py = NAG_ALLOC(m, double)) || !(pf = NAG_ALLOC(m, double)))
  {
    printf("Allocation failure\n");
```

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

  /* Read scattered 2d data points and function values. */
  for (i = 0; i < n; i++) {
#ifdef _WIN32
    scanf_s("%lf%lf%lf%*[^\n]", &x[i], &y[i], &f[i]);
#else
    scanf("%lf%lf%lf%*[^\n]", &x[i], &y[i], &f[i]);
#endif
  }

  /* Obtain triangulation of scattered points (x,y) using
   * nag_2d_triangulate (e01eac).
   */
  nag_2d_triangulate(n, x, y, triang, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_2d_triangulate (e01eac).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
  }

  /* Read points at which interpolated values required. */
  for (i = 0; i < m; i++) {
#ifdef _WIN32
    scanf_s("%lf%lf%*[^\n]", &px[i], &py[i]);
#else
    scanf("%lf%lf%*[^\n]", &px[i], &py[i]);
#endif
  }
  /* Use triangulation to perform barycentric interpolation on to the
   * set of m points (px,py) using nag_2d_triang_bary_eval (e01ebc).
   */
  nag_2d_triang_bary_eval(m, n, x, y, f, triang, px, py, pf, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_2d_triang_bary_eval (e01ebc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

  /* Display results */
  printf("    %4s   %7s   %19s\n", "px", "py", "Interpolated Value");
  for (i = 0; i < m; i++) {
    printf("   %7.4f   %7.4f       %7.4f\n", px[i], py[i], pf[i]);
  }

END:
  NAG_FREE(f);
  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_triangulate (e01eac) Example Program Data
     30                       : n, the number of data points
     5                        : m, number of interpolation points

     0.00     0.00    58.20
     0.00    20.00    34.60
     0.51     8.37    49.43
     2.14    15.03    53.10
```

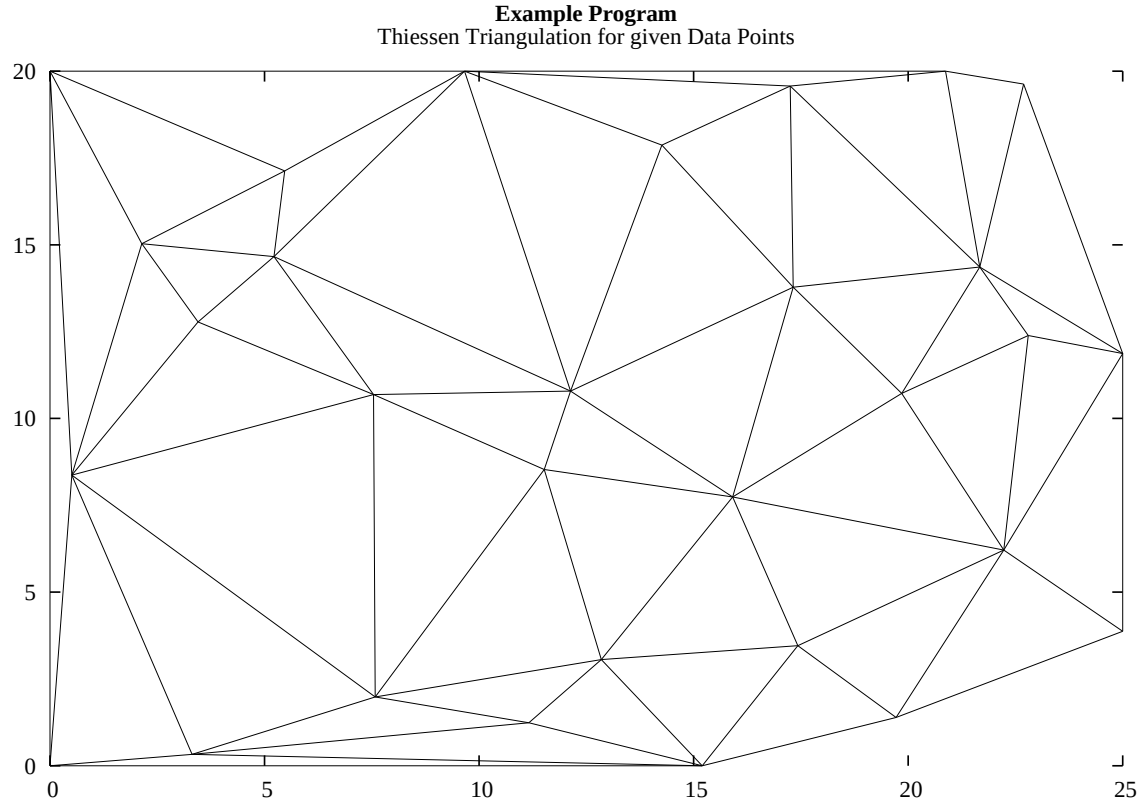
```
 3.31     0.33    44.08
 3.45    12.78    41.24
 5.22    14.66    40.36
 5.47    17.13    28.63
 7.54    10.69    19.31
 7.58     1.98    29.87
 9.66    20.00     4.73
11.16     1.24    22.15
11.52     8.53    15.74
12.13    10.79    13.71
12.85     3.06    22.11
14.26    17.87    10.74
15.20     0.00    21.60
15.91     7.74    15.30
17.25    19.57     6.43
17.32    13.78    12.11
17.43     3.46    18.60
19.72     1.39    16.83
19.85    10.72     7.97
20.87    20.00     5.74
21.67    14.36     5.52
22.23     6.21    10.25
22.69    19.63     3.25
22.80    12.39     5.47
25.00    11.87     4.40
25.00     3.87     8.74   : (x,y,f)[i] for i=0,1,...,n-1

 2.05     1.775
 3.75     3.25
 5.00     5.00
 8.54     2.05
 9.14     4.45           : (px,py)[i] for i=0,1,...,m-1
```

10.3 Program Results

```
nag_2d_triangulate (e01eac) Example Program Results

      px        py    Interpolated Value
  2.0500    1.7750       48.2100
  3.7500    3.2500       41.4195
  5.0000    5.0000       36.1613
  8.5400    2.0500       28.2458
  9.1400    4.4500       24.4543
```


Example Program
Thiessen Triangulation for given Data Points
20
15
10
5
0
0
5
10
15
20
25