NAG Library Routine Document

E01EAF

Note: before using this routine, please read the Users' Note for your implementation to check the interpretation of ***bold italicised*** terms and other implementation-dependent details.

1 Purpose

E01EAF generates a triangulation for a given set of two-dimensional points using the method of Renka and Cline.

2 Specification

```
SUBROUTINE E01EAF (N, X, Y, TRIANG, IFAIL)
INTEGER            N, TRIANG(7*N), IFAIL
REAL (KIND=nag_wp) X(N), Y(N)
```

3 Description

E01EAF 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 E01EBF 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: N $\geq$ 3.

2: X(N) – REAL (KIND=nag_wp) array *Input*

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

3: Y(N) – REAL (KIND=nag_wp) array *Input*

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

4: TRIANG($7 \times \mathrm{N}$) – INTEGER array *Output*

On exit: a data structure defining the computed triangulation, in a form suitable for passing to E01EBF. 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: IFAIL – INTEGER *Input/Output*

On entry: IFAIL must be set to 0, -1 or 1. If you are unfamiliar with this argument you should refer to Section 3.4 in How to Use the NAG Library and its Documentation for details.

For environments where it might be inappropriate to halt program execution when an error is detected, the value -1 or 1 is recommended. If the output of error messages is undesirable, then the value 1 is recommended. Otherwise, if you are not familiar with this argument, the recommended value is 0. **When the value -1 or 1 is used it is essential to test the value of IFAIL on exit.**

On exit: IFAIL $= 0$ unless the routine detects an error or a warning has been flagged (see Section 6).

6 Error Indicators and Warnings

If on entry IFAIL $= 0$ or -1, explanatory error messages are output on the current error message unit (as defined by X04AAF).

Errors or warnings detected by the routine:

IFAIL $= 1$

On entry, N $= \langle value \rangle$.
Constraint: N ≥ 3.

IFAIL $= 2$

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

IFAIL $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

See Section 3.9 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -399$

Your licence key may have expired or may not have been installed correctly.

See Section 3.8 in How to Use the NAG Library and its Documentation for further information.

IFAIL $= -999$

Dynamic memory allocation failed.

See Section 3.7 in How to Use the NAG Library and its Documentation for further information.

7 Accuracy

Not applicable.

8 Parallelism and Performance

E01EAF is not threaded in any implementation.

9 Further Comments

The time taken for a call of E01EAF is approximately proportional to the number of data points, n. The routine 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)

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

$j_k = \text{TRIANG}(6 \times \text{N} + k)$

TRIANG(j), for $j = i_k, \ldots, j_k$, contains the list of nodes to which node k is connected. If TRIANG$(j_k) = 0$ then node k is on the boundary of the mesh.

10 Example

In this example, E01EAF creates a triangulation from a set of data points. E01EBF 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

```
    Program e01eafe

!     E01EAF Example Program Text
!     Mark 26 Release. NAG Copyright 2016.

!     .. Use Statements ..
      Use nag_library, Only: e01eaf, e01ebf, nag_wp
!     .. Implicit None Statement ..
      Implicit None
!     .. Parameters ..
      Integer, Parameter               :: nin = 5, nout = 6
      Logical, Parameter               :: pr_tr = .False.
!     .. Local Scalars ..
      Integer                          :: i, ifail, m, n
!     .. Local Arrays ..
      Real (Kind=nag_wp), Allocatable  :: f(:), pf(:), px(:), py(:), x(:),     &
                                          y(:)
      Integer, Allocatable             :: triang(:)
!     .. Executable Statements ..

      Write (nout,*) 'E01EAF Example Program Results'

!     Skip heading in data file
      Read (nin,*)

      Read (nin,*) n
      Allocate (x(n),y(n),f(n),triang(7*n))
      Read (nin,*)(x(i),y(i),f(i),i=1,n)

!     Triangulate data
      ifail = 0
      Call e01eaf(n,x,y,triang,ifail)

      Read (nin,*) m
      Allocate (px(m),py(m),pf(m))
      Read (nin,*)(px(i),py(i),i=1,m)

!     Interpolate data
      ifail = 0
      Call e01ebf(m,n,x,y,f,triang,px,py,pf,ifail)

!     Display results
```

```
      Write (nout,*)
      Write (nout,99999) 'px', 'py', 'Interpolated Value'
      Write (nout,99998)(px(i),py(i),pf(i),i=1,m)

      If (pr_tr) Then
        Call print_triang
      End If

99999 Format (2X,A4,4X,A4,4X,A19)
99998 Format (1X,F7.4,1X,F7.4,8X,F7.4)

    Contains
      Subroutine print_triang

!       .. Implicit None Statement ..
        Implicit None
!       .. Local Scalars ..
        Integer                        :: i_k, j, j_k, k
!       .. Executable Statements ..

!       Print a sequence of unique line segments for plotting triangulation
        Write (nout,*)
        Write (nout,*) '  Triangulation as a set of line segments'
        Write (nout,*)
        j_k = 0
        Do k = 1, n
          i_k = j_k + 1
          j_k = triang(6*n+k)
          Do j = i_k, j_k
            If (triang(j)>k) Then
              Write (nout,99999) x(k), y(k)
              Write (nout,99999) x(triang(j)), y(triang(j))
              Write (nout,*)
            End If
          End Do
        End Do
        Return
99999   Format (1X,F7.4,1X,F7.4)
      End Subroutine print_triang

    End Program e01eafe
```

10.2 Program Data

```
E01EAF Example Program Data
     30                        :   n, data 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)(1:n)

  5                         : m, interpolation points
  2.05     1.775
  3.75     3.25
  5.00     5.00
  8.54     2.05
  9.14     4.45             : (px,py)(1:m)
```

10.3 Program Results

```
 E01EAF 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
```