An implementation of Watson's algorithm for computing 2-dimensional Delaunay triangulations

S. W. SLOAN

Department of Civil Engineering, University of Newcastle, New South Wales 2308, Australia

G. T. HOULSBY

Department of Engineering Science, Parks Road, University of Oxford, Oxford OX1 3PJ, UK

A FORTRAN 77 implementation of Watson's algorithm for computing two-dimensional Delaunay triangulations is described. The algorithm is shown to have an asymptotic time complexity bound which is better than $O(N^{1.5})$ by applying it to collections of N points generated randomly within the unit square. The computer code obeys strict FORTRAN 77 syntax. Excluding the memory needed to store the co-ordinates of the points, it requires slightly greater than $9N$ integer words of memory to assemble and store the Delaunay triangulation.

INTRODUCTION

The problem of triangulating an arbitrary collection of points in the plane is one which has received considerable attention in the literature Two areas where triangulation algorithms find wide application are mesh generation for finite element analysis and the construction of contour plots

In order to illustrate the geometric basis of the Delaunay triangulation it is convenient to consider its geometric dual, the Dirichlet tessellation Consider the collection of five points in the plane shown in Fig 1 The Dirichlet tessellation, which is also commonly known as the Voronoi or Theissen tessellation, associated with these points is shown by the heavy lines This tessellation breaks up the plane into a series of polygonal regions, the boundaries of which are the perpendicular bisectors of the lines joining the points Each region is known as a tile and is associated with a single interior generating point Any point inside a given tile is closer to the tile generating point than to any other generating point The Delaunay triangulation associated with the Dirichlet tessellation is constructed by connecting all generating points which share a common tile boundary These generating points are said to be contiguous The Delaunay triangulation for the five points shown in Fig 1 is indicated by the feint lines

Any triangulation algorithm should attempt to produce a grid which is 'well conditioned', i e it should try and avoid forming triangles which contain very small included angles A useful criterion for assessing the quality of a triangulation has been given by Lawson [1] This requires that in every convex quadrilateral formed by two adjacent triangles, the minimum of the six angles in the two triangles should be greater than it would have been if the alternative diagonal had been drawn and the other pair of triangles chosen A grid which satisfies Lawson's criterion is said to be locally equiangular Recently both Lawson[2] and Sibson[3] have shown that the Delaunay triangulation automatically satisfies the Lawson[1] criterion and, except for special cases, is also unique

An algorithm for computing planar Delaunay triangulations has been given by Green and Sibson [4] If N is the number of points to be triangulated, Green and Sibson proved that their method has a time complexity bound of $O(N^{1.5})$ More recently Bowyer[5] and Watson[6] have described algorithms for computing Delaunay triangulations in k dimensional Euclidean space (where $k > 1$) For two-dimensional space, both of these algorithms have time complexity bounds of $O(N^{1.5})$ The procedure given by Watson[6] has the advantage of being particularly simple In this paper an efficient implementation of this algorithm is described

THEORY

Except in special cases, which are known as degeneracies, the vertices of the Dirichlet tessellation occur at points

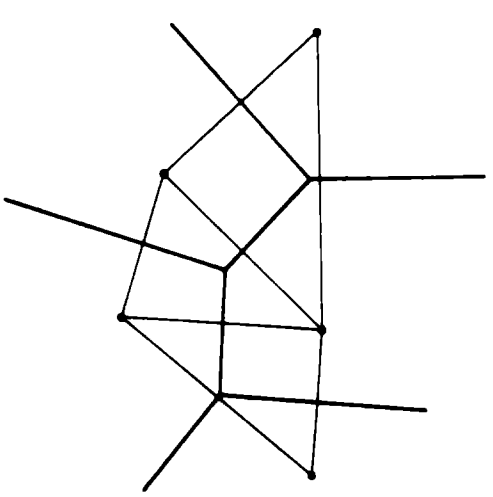

Figure 1 The Dirichlet tessellation and Delaunay triangulation

Accepted June 1984 Written discussion closes December 1984

0141-1195/84/040192-07 $2 00

where three tiles meet The three interior generating points associated with each of these tiles form a Delaunay triangle By definition, each vertex of a Dirichlet tessellation is equidistant from its three generating points and is thus the circumcentre of the corresponding Delaunay triangle Thus, each Delaunay triangle is associated with a unique vertex in the Dirichlet tessellation

In some cases, the Delaunay triangulation for a collection of points in the plane may not be unique Such cases are said to be degenerate An example of degeneracy is shown in Fig 2, where four points are located on the corners of a square The single vertex of the Dirichlet tessellation is located at the centroid of the square, and is associated with four tiles Two different Delaunay triangulations are possible with this tessellation and both are equally valid In practice, the problem of degeneracy may be dealt with by making an arbitrary choice between alternative triangulations

In the algorithm published by Watson,[6] the Delaunay triangulation is assembled by introducing each point one at a time To begin the process, three points are chosen to form a 'supertriangle' which completely encompasses all of the points to be triangulated Initially the Delaunay triangulation is thus comprised of a single triangle defined by the supertriangle vertices When a new point is introduced into an existing triangulation, the circumcentre of each existing triangle is computed together with its circumcircle radius Any triangles which contain the new point within their circumcircles are flagged as being intersected Since the supertriangle encompasses all of the points to be triangulated, each new point must intersect at least one of the existing triangle circumcircles When added together, the triangles that are intersected form a polygon, with all of their vertices lying on its boundary All of the intersected triangles are replaced, and the new point forms triangles with each pair of vertices on the boundary of the polygon (noting that the new point is always interior to the polygon) After each new point is inserted, the net gain in the total number of triangles must be exactly two This is because each polygon formed by the intersected triangles always has two more edges than triangles Thus, when the triangulation of N points is complete, there must always be $2N+1$ triangles (including the triangles formed with the vertices of the supertriangle)

When inserting each new point, the need to check all existing triangles for intersection may be avoided by pre-sorting all of the points in ascending sequence of one co-ordinate, say their x-co-ordinate In order to test whether a new point lies inside the circumcircle of an existing triangle, the x-component of the distance from the new point to the triangle circumcentre is computed first If this quantity is greater than or equal to the radius of the triangle circumcircle, then neither the current point, nor any of the remaining points, lie within the triangle circumcircle and the Delaunay triangle may be flagged as complete A triangle which is flagged as complete will remain unchanged as all of the remaining points are introduced In this manner the Delaunay triangulation is assembled as an 'advancing front', with all of the triangles behind the front being complete and all of the triangles ahead of the front being subject to modification

After all of the points have been dealt with, the final triangulation is computed by removing all of the triangles that contain one or more of the supertriangle vertices Any vertex which appears in these deleted triangles, but is not a supertriangle vertex, must lie on the boundary of the final triangulation

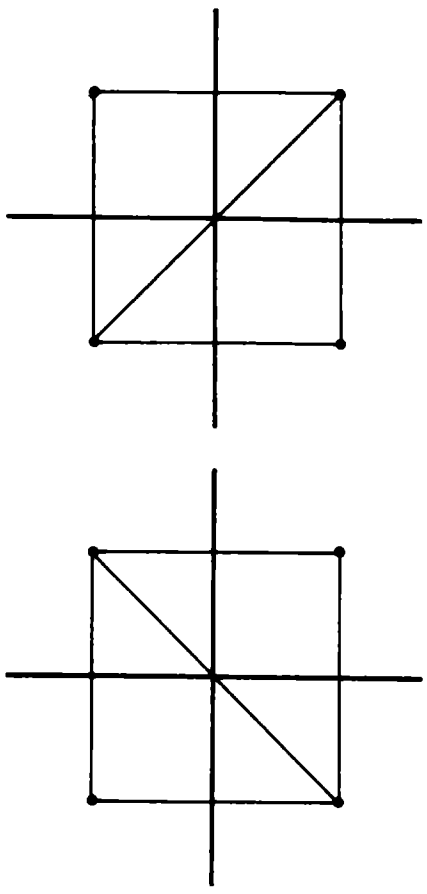

Figure 2 Degenerate Delaunay triangulations

WATSON'S ALGORITHM

The fundamental steps in Watson's algorithm are as follows

(1) Sort the N points to be triangulated in ascending sequence of their x-co-ordinate

(2) Define the vertices of the supertriangle in anticlockwise order It is convenient to number these vertices as $N+1$, $N+2$ and $N+3$ Set the co-ordinates of these vertices so that all of the points to be triangulated lie within the supertriangle Add the supertriangle to a list of triangles formed and flag it as incomplete

(3) Introduce a new point from the list of sorted points with co-ordinates (x_{new}, y_{new})

(4) Examine the list of all triangles formed so far For each triangle which is flagged as incomplete, do steps 5 to 9

(5) Compute the co-ordinates of the triangle circumcentre, (x_c, y_c), and the square of its circumcircle radius, R^2

(6) Compute the square of the x-distance from the new point to the triangle circumcentre, i e the quantity

$$D_x^2 = (x_c - x_{new})^2$$

(7) If $D_x^2 \geqslant R^2$, then the circumcircle for this triangle cannot be intersected by any of the remaining points Flag this triangle as complete and do not execute steps 8 and 9

(8) Compute the square of the distance from the new point to the triangle circumcentre, i e the quantity

$$D^2 = D_x^2 + (y_c - y_{new})^2$$

(9) If $D^2 < R^2$, then the new point intersects the circumcircle for this triangle Delete this triangle from the list of triangles formed and store the three pairs of vertices which define its edges on a list of edges If $D^2 \gg R^2$, then the new point lies on or outside the circumcircle for this triangle and the triangle remains unmodified

(10) Loop over the list of edges and delete all edges which occur twice in the list This removes all edges which are interior to the polygon formed by the intersected triangles Since the vertices defining the edges of each triangle are always recorded in an anticlockwise sequence, this step may be executed efficiently by searching for ordered pairs

(11) Form the new triangles by matching the new point with each pair of vertices in the list of edges The new point forms new triangles with each pair of vertices on the boundary of the polygon formed by the intersected triangles Define each new triangle such that its vertices are always listed in an anticlockwise sequence and flag it as incomplete

(12) Repeat steps 3 to 11 until the list of points to be triangulated is exhausted.

(13) Form the final triangulation by removing all triangles which contain one or more of the supertriangle vertices This may be achieved by scanning the list of triangles and deleting any of those which have vertex numbers which are greater than N

Since the triangulation algorithm has a time complexity bound which is better than $O(N^{1.5})$, an efficient sorting procedure is required for step 1 The Quicksort algorithm, which has an average time complexity bound of $O(N \log_2 N)$, is ideally suited for this purpose A lucid description of the Quicksort algorithm has been given by Dromey [7] FORTRAN 77 implementations of various sorting procedures, including Quicksort, may be found in Houlsby and Sloan [8] With reference to step 5, an efficient method for computing the circumcentre and circumcircle radius of a triangle may be found in Bowyer and Woodwark [9] Note that in steps 6 to 9, where checks are performed to determine the position of a point with respect to a triangle circumcircle, it is sufficient to compare squares of distances This avoids costly square root evaluations In step 9, a triangle is deleted only if the new points lies inside its circumcircle This allows degenerate cases to be dealt with in a consistent manner

IMPLEMENTATION

A FORTRAN 77 implementation of Watson's algorithm for computing two-dimensional Delaunay triangulations is given in Appendix 1 The algorithm is implemented in a subroutine named DELAUN. When calling this subroutine, NUMPTS is the number of points to be triangulated and MAXTRI is the maximum number of triangles permitted As discussed previously, the number of triangles created by the algorithm, including those formed with the supertriangle vertices, is equal to 2*NUMPTS+1 Thus, to conserve storage, MAXTRI should be set equal to this quantity The double precision number SMALLD is used to check for any ill-conditioning during execution of the algorithm This may occur if the program attempts to triangulate a set of points in which two or more of the points have identical co-ordinates (to within the accuracy of the machine) It may also arise if the algorithm attempts to create an extremely thin triangle whose vertices are nearly collinear If the overall dimensions of the triangulated area are of the order of unity, and double precision arithmetic is used on a 32-bit machine, a suitable value for SMALLD is 10^{-12} The x- and y-co-ordinates of the points to be triangulated are stored in the double precision vectors X and Y Both of these vectors must be of length $N+3$, as the last three storage locations are used to store the co-ordinates of the supertriangle vertices The numbers of the points to be triangulated, in ascending sequence of their x-co-ordinate, are stored in the integer vector LIST This vector is of length NUMPTS The three vertices defining each triangle as it is formed, modified and completed are stored in the linked lists V1, V2 and V3 The three vertices are listed in anticlockwise order and for the triangle in location I are given by V1(I), V2(I), V3(I) The pointer vector which serves these linked lists is POINTR Storing the triangle vertices as linked lists allows triangles to be added and deleted efficiently as the triangulation proceeds The address of the first triangle stored in V1, V2 and V3 is given by POINTR(0) The address of the triangle following the triangle in the Ith location is given by POINTR(I) The end of these lists, which corresponds to the last triangle formed in the grid, is marked by setting POINTR equal to zero for this entry The lengths of V1, V2 and V3 should be set equal to MAXTRI, whilst the vector POINTR should be assigned a length of MAXTRI+1 When a new point is found to intersect the circumcircle for a triangle, the address of this deleted triangle in the linked lists is stored in the locally declared vector STACK This integer vector acts as a last-in, first-out stack and has a length of MAXSTK Storing the addresses of all deleted triangles in this manner avoids the need to search the vertex lists for a vacant location when assembling a new triangle MAXSTK represents the maximum number of triangle circumcircles that a new point may intersect After the insertion of a new point, the three pairs of vertices that define the edges for each deleted triangle are stored in the locally declared vectors LIST1 and LIST2 Before assembling the new triangles, it is necessary to remove any edges that are interior to the polygon formed by all of the deleted triangles As the vertex pairs defining each deleted triangle are stored in LIST1 and LIST2 in anticlockwise order, this is achieved by searching these lists and removing all duplicate ordered pairs An edge will appear only once in these lists if it is on the boundary of the polygon

At the start of the triangulation algorithm, the vertices of the supertriangle are allocated the numbers $N+1$ to $N+3$ Their co-ordinates are found by first defining a rectangular window which encompasses all of the points to be triangulated, i e a window defined by (x_{min}, y_{min}), (x_{max}, y_{max}) where

$$x_{min} = \min\{x_i \quad i = 1, N\}$$
$$x_{max} = \max\{x_i \quad i = 1, N\}$$
$$y_{min} = \min\{y_i \quad i = 1, N\}$$
$$y_{max} = \max\{y_i \quad i = 1, N\}$$

The supertriangle is defined to be approximately equilateral with the co-ordinates of its vertices given by

$$x_{N+1} = x_{cen} - 0.866 d_{max}$$
$$x_{N+2} = x_{cen} + 0.866 d_{max}$$

$$x_{N+3} = x_{cen}$$

$$y_{N+1} = y_{cen} - 0\,5 d_{max}$$

$$y_{N+2} = y_{cen} - 0\,5 d_{max}$$

$$y_{N+3} = y_{cen} + d_{max}$$

where

$$d_{max} = 3 \max\{x_{max} - x_{min},\ y_{max} - y_{min}\}$$

and

$$x_{cen} = 0\,5 * (x_{min} + x_{max})$$

$$y_{cen} = 0\,5 * (y_{min} + y_{max})$$

The size and shape of the supertriangle may be chosen arbitrarily It is sufficient merely that the supertriangle should contain all of the points to be triangulated It is worth noting, however, that if the corners of the supertriangle are very close to the window enclosing the points, the boundary of the final triangulation may be locally concave (and, strictly speaking, the grid produced may not correspond exactly to the Delaunay triangulation, as some long thin triangles along the boundary may be omitted) On the other hand, if the supertriangle is made too large, ill-conditioning of the equations may occur The dimensions chosen are thought to be a reasonable compromise which usually avoid any numerical problems Slight concavity of the triangulated zone occasionally occurs, but this is not considered to be a serious disadvantage for most practical applications

After exiting from subroutine DELAUN, the vertices defining each triangle in the Delaunay triangulation are stored in the lists V1, V2 and V3 Included in these lists are the triangles formed with the supertriangle vertices To remove these triangles and thus produce the final triangulation, it is necessary to scan all of the triangles and delete any which contain one or more vertices with numbers greater than N

Appendix 2 lists one subroutine and two function subprograms which are called by subroutine DELAUN These are self-explanatory

The FORTRAN 77 implementation given in the Appendices is written in double precision arithmetic To convert the implementation to single precision arithmetic, all DOUBLE PRECISION type declarations should be replaced by REAL type declarations, and all double precision constants in PARAMETER statements should be replaced by single precision constants For single precision arithmetic on a 32-bit machine, a suitable value for SMALLD is 10^{-6} (assuming that the overall dimensions of the triangulated zone are of the order of unity) Excluding the $2N + 6$ double precision words of memory needed to store the co-ordinates of the points and supertriangle vertices, subroutine DELAUN requires $9N + 7*$MAXSTK $+ 5$ words of memory to compute and store the Delaunay triangulation for N points Setting MAXSTK equal to 50 has proved to be sufficient for triangulating up to 5000 points generated randomly within the unit square, but larger values than this may be required for triangulating regular grids in which a large number of triangles may be formed with the supertriangle vertices.

APPLICATIONS

To assess the efficiency of the implementation given in the Appendices, it has been used to triangulate collections of points generated randomly within the unit square Figure 3 illustrates the Delaunay triangulation for 20 such points Table 1 shows the CPU times required by the algorithm for different numbers of points up to a maximum of 5000 points These timing statistics are for the VAX 11/780 using the optimising FORTRAN 77 compiler They were obtained using the internal clock of the machine and are accurate to the nearest hundredth of a second The points were sorted in ascending sequence of their x-co-ordinate using the Quicksort subroutine QSORT described in Houlsby and Sloan.[8] In all cases, the time required by the sorting procedure is negligible in comparison with the time required by subroutine DELAUN to compute the triangulation The statistics given in Table 1 suggest that the triangulation algorithm has a time complexity bound of approximately $O(N^{1\,43})$ In addition to these examples the implementation has been applied to grids of points distributed on a square lattice These point patterns yield Delaunay triangulations which are non-unique (and thus degenerate) but were handled successfully

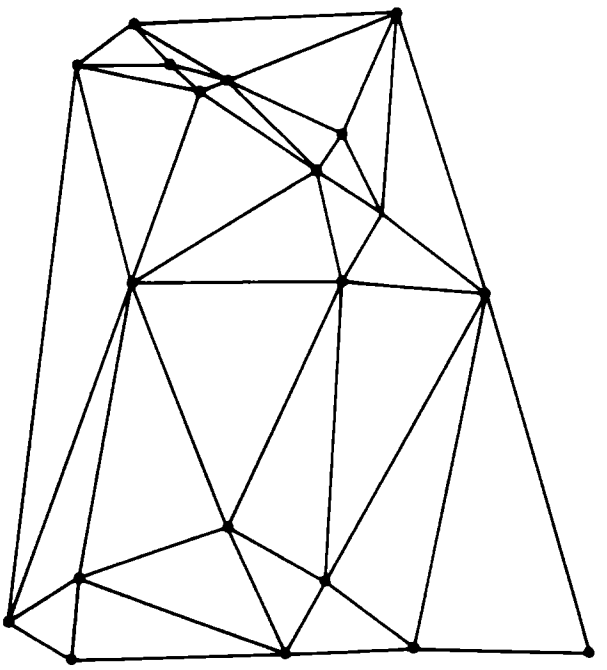

Figure 3 Delaunay triangulation for 20 points

Table 1 Timing statistics for example problems

N	T_s	T_t
100	0 02	0 77
500	0 09	5 70
1000	0 19	15 21
2000	0 45	43 19
3000	0 74	80 23
4000	1 00	124.28
5000	1 45	170 66

N = number of points to be triangulated, T_S = CPU time required for sorting, T_t = CPU time required for triangulation All times in seconds

The implementation given in the Appendices has been designed to be efficient and require a minimum amount of storage At the cost of extra storage, the execution time of subroutine DELAUN could be improved by saving the circumcentre co-ordinates and circumcircle radius for each

triangle as they are computed. This would require an additional $6N + 3$ double precision words of memory, but reduce the number of operations required to assemble the Delaunay triangulation. It is intended to examine an alternative algorithm which is faster, but requires more storage, in a future publication

REFERENCES

1 Lawson, C L Generation of a triangular grid with application to contour plotting. *California Institute of Technology Jet Propulsion Laboratory Technical Memorandum, Number 299*, 1972

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

3 Sibson, R Locally equiangular triangulations *The Computer Journal* 1978, **21** (3), 243

4 Green, P. J and Sibson, R Computing Dirichlet tessellations in the plane *The Computer Journal* 1978, **21** (2) 168

5 Bowyer, A Computing Dirichlet tessellations *The Computer Journal* 1981, **24** (2), 162

6 Watson D F Computing the *n*-dimensional Delaunay triangulation with application to Voronoi polytopes *The Computer Journal* 1981, **24** (2), 167

7 Dromey, R G *How To Solve It By Computer* Prentice-Hall, 1982

8 Houlsby, G T, and Sloan, S. W. *Efficient Sorting Routines in FORTRAN 77. Advances in Engineering Software* 1984, **6** (4), 198

9 Bowyer, A. and Woodwark, J. *A Programmer's Geometry*, Butterworths, 1983

APPENDIX 1

The Delaunay triangulation subroutine

```
      SUBROUTINE DELAUN(NUMPTS MAXTRI SMALLD X Y LIST,POINTR V1 V2,V3
     +                  NUMTRI )
C
C*****************************************************************
C     SUBPROGRAM DELAUN
C     COMPUTE DELAUNAY TRIANGULATION USING WATSON S ALGORITHM
C
C     INPUT PARAMETERS
C
C      NUMPTS  THE NUMBER OF POINTS TO BE TRIANGULATED
C      MAXTRI  THE MAXIMUM NUMBER OF TRIANGLES PERMITTED
C      SMALLD  A SMALL DOUBLE PRECISION NUMBER
C      X  A VECTOR OF X-COORDS FOR POINTS TO BE TRIANGULATED
C      Y  A VECTOR OF Y-COORDS FOR POINTS TO BE TRIANGULATED
C      X  AND  Y' ARE OF LENGTH NUMPTS+3
C      LIST  A LIST OF POINTS TO BE TRIANGULATED, ORDERED IN ASCENDING
C     SEQUENCE OF THEIR X-COORDS, OF LENGTH NUMPTS
C
C     INTERNAL PARAMETERS
C
C      POINTR  A POINTER VECTOR TO THE LINKED LISTS 'V1,V2 AND V3
C     OF LENGTH MAXTRI+1
C
C     OUTPUT PARAMETERS
C
C      V1,V2,V3  LINKED LISTS WHICH DEFINE VERTICES FOR EACH TRIANGLE
C     IN ANTICLOCKWISE ORDER
C      V1 V2,V3  ARE OF LENGTH MAXTRI
C      NUMTRI  THE NUMBER OF TRIANGLES IN THE DELAUNAY TRIANGULATION
C     (INCLUDING TRIANGLES WHICH CONTAIN SUPERTRIANGLE VERTICES)
C
C     NOTES
C
C     EACH POINT IS INTRODUCED INTO THE SUPERTRIANGLE ONE AT A
C     TIME AND CREATES TWO NEW TRIANGLES
C     THUS THE FINAL NUMBER OF TRIANGLES = 2*NUMPTS+1
C     TO CONSERVE SPACE SET MAXTRI = 2*NUMPTS+1
C     ERROR DIAGNOSTICS ARE OUTPUT TO UNIT NUMBER 6
C     TO OBTAIN FINAL DELAUNAY TRIANGULATION, DELETE ALL TRIANGLES
C     WHICH CONTAIN SUPERTRIANGLE VERTICES AFTER EXITING FROM
C     THIS SUBROUTINE (I E  DELETE ALL TRIANGLES WHICH CONTAIN
C     VERTICES WITH NUMBERS GREATER THAN NUMPTS)
C
C*****************************************************************
C
      INTEGER MAXTRI,V1(*),V2(*),V3(*),POINTR(0 *),NUMPTS,LIST(*),I,J,
     +        N1,N2,N3,X,NEDGES ADDRES,LSTADD,NUMTRI,NODE,NODE1,NODE2,
     +        MAXEDG,MAXSTK,TOPSTK,OUT
C
      DOUBLE PRECISION X(*),Y(*),XCENTR,YCENTR,RADISQ,SMALLD,XNEW,YNEW,
     +                 DISTSQ,X1,X2,X3,Y1,Y2,Y3,X21,Y21,X31,Y31,DET,R21,
     +                 R31,C000P5,CP8660,C00003,XMIN,XMAX,YMIN,YMAX,
     +                 XCEN,YCEN,DMAX,MINDV,MAXDV
C
      PARAMETER(MAXSTK=50,
     +          MAXEDG=3*MAXSTK,
     +          OUT   =6,
     +          C000P5=0 5D0
     +          CP8660=0 86603D0,
     +          C00003=3 0D0)
C
      INTEGER LIST1(MAXEDG),LIST2(MAXEDG),STACK(MAXSTK)
C
C
C     CHECK THAT SUFFICIENT SPACE EXISTS TO COMPLETE TRIANGULATION
C
      IF(MAXTRI LT (2*NUMPTS+1))THEN
        WRITE(OUT, (''0***ERROR IN SUBROUTINE DELAUN  )')
        WRITE(OUT,'(    ***TOO MANY TRIANGLES WILL BE GENERATED  )')
        WRITE(OUT,'(' ***MAX NO  OF TRIANGLES ALLOWED= ',I5) )MAXTRI
        WRITE(OUT,'( ' ***NO  OF TRIANGLES GENERATED= ',I5)')2*NUMPTS+1
        STOP
      END IF
C
C     INITIALISE
C
      CALL FILLI(V1 1 MAXTRI 0)
      CALL FILLI(V2 1 MAXTRI 0)
      CALL FILLI(V3 1 MAXTRI 0)
C
C     DEFINE SUPERTRIANGLE VERTICES
C
      N1=NUMPTS+1
      N2=NUMPTS+2
      N3=NUMPTS+3
      V1(1)=N1
      V2(1)=N2
      V3(1)=N3
C
C     COMPUTE COORDS OF SUPERTRIANGLE
C
      XMIN=X(LIST(1))
      XMAX=X(LIST(NUMPTS))
      YMIN=MINDV(Y,1 NUMPTS)
      YMAX=MAXDV(Y,1,NUMPTS)
      XCEN=C000P5*(XMIN+XMAX)
      YCEN=C000P5*(YMIN+YMAX)
      DMAX=C00003*MAX(XMAX-XMIN,YMAX-YMIN)
      X(N1)=XCEN-CP8660*DMAX
      X(N2)=XCEN+CP8660*DMAX
      X(N3)=XCEN
      Y(N1)=YCEN-C000P5*DMAX
      Y(N2)=YCEN-C000P5*DMAX
      Y(N3)=YCEN+DMAX
C
C     LOOP OVER EACH NODE
C
      NUMTRI=1
      POINTR(0)=1
      POINTR(1)=0
      DO 100 I=1,NUMPTS
        NODE=LIST(I)
        XNEW=X(NODE)
        YNEW=Y(NODE)
C
C       LOOP OVER EACH TRIANGLE
C
        TOPSTK=0
        NEDGES=1
        LSTADD=0
        ADDRES=POINTR(0)
   30   IF(ADDRES GT 0)THEN
C
C         FIND CIRCUMCENTRE OF TRIANGLE AND RADIUS OF CIRCUMCIRCLE
C         COMPUTE SQUARE OF RADIUS TO AVOID SQUARE ROOT EXTRACTION
C         USE BOWYER AND WOODWARK ALGORITHM
C
          X1=X(V1(ADDRES))
          X2=X(V2(ADDRES))
          X3=X(V3(ADDRES))
          Y1=Y(V1(ADDRES))
          Y2=Y(V2(ADDRES))
          Y3=Y(V3(ADDRES))
          X21=X2-X1
          Y21=Y2-Y1
          X31=X3-X1
          Y31=Y3-Y1
          DET=X21*Y31-X31*Y21
          IF(DET LT SMALLD)THEN
C
C           ABORT TRIANGULATION IF 3 VERTICES ARE COLLINEAR OR IF
C           AT LEAST TWO OF VERTICES ARE COINCIDENT
C
            WRITE(OUT,'(''0***ERROR IN TRIANGULATION ROUTINE' ) )
            WRITE(OUT,'(  ' ***UNABLE TO FIND TRIANGLE CIRCUMCENTRE  )')
            WRITE(OUT, (   X1=  ,F10 5, ' X2=' ,F10 5,   X3=' ,F10 5) )
     +                  X1,           X2,           X3
            WRITE(OUT, (' Y1=' ,F10 5   Y2=  ,F10 5, ' Y3=  ,F10 5)')
     +                  Y1,           Y2,           Y3
            STOP
                                ELSE
```

```
C             COORD. OF TRIANGLE CIRCUMCENTRE ARE (XCENTR,YCENTR)
C             SQUARE OF CIRCUMCIRCLE RADIUS = RADISQ
C
              DET=CHOOPS/DET
              R21=X21*X21+Y21*Y21
              R31=X31*X31+Y31*Y31
              XCENTR=DET*(R21*Y31-R31*Y21)
              YCENTR=DET*(X21*R31-X31*R21)
              RADISQ=XCENTR*XCENTR+YCENTR*YCENTR
              XCENTR=XCENTR+X1
              YCENTR=YCENTR+Y1
            END IF
C
C           FIND SQUARE OF X-DISTANCE FROM NEW POINT TO CIRCUMCENTRE
C
            DISTSQ=(XCENTR-XNEW)**2
            IF(DISTSQ GE RADISQ)THEN
C
C             TRIANGLE IS COMPLETE
C             SINCE POINTS ARE ORDERED IN ASCENDING SEQUENCE OF THEIR
C             X-COORDS, NONE OF THE REMAINING POINTS CAN BE INSIDE THE
C             CIRCUMCIRCLE FOR THIS TRIANGLE
C             RESET POINTERS TO REMOVE (BUT NOT DELETE) THESE VERTICES
C             FROM THE LINKED LISTS
C
              POINTR(LSTADD)=POINTR(ADDRES)
              POINTR(ADDRES)=0
              ADDRES=POINTR(LSTADD)
                                ELSE
              DISTSQ=DISTSQ+(YCENTR-YNEW)**2
              IF(DISTSQ LT RADISQ)THEN
C
C               POINT IS INSIDE TRIANGLE CIRCUMCIRCLE
C               STORE ADDRESS OF THIS TRIANGLE IN STACK
C               ENCODE EDGES OF TRIANGLE AND DELETE IT FROM THE LINKED
C               LISTS
C
                TOPSTK=TOPSTK+1
                IF(TOPSTK GT MAXSTK)THEN
                  WRITE(OUT, (' 0***ERROR IN SUBROUTINE DELAUN ')
                  WRITE(OUT, ('  ***STACK OVERFLOW ')')
                  STOP
                                  ELSE
                  STACK(TOPSTK)=ADDRES
                END IF
                LIST1(NEDGES)=V1(ADDRES)
                LIST2(NEDGES)=V2(ADDRES)
                LIST1(NEDGES+1)=V2(ADDRES)
                LIST2(NEDGES+1)=V3(ADDRES)
                LIST1(NEDGES+2)=V3(ADDRES)
                LIST2(NEDGES+2)=V1(ADDRES)
                NEDGES=NEDGES+3
                POINTR(LSTADD)=POINTR(ADDRES)
                POINTR(ADDRES)=0
                ADDRES=POINTR(LSTADD)
                                ELSE
C
C               POINT IS ON OR OUTSIDE TRIANGLE CIRCUMCIRCLE
C               TRIANGLE IS NOT MODIFIED
C
                LSTADD=ADDRES
                ADDRES=POINTR(ADDRES)
              END IF
            END IF
            GOTO 30
          END IF
C
C         REMOVE ANY EDGE THAT OCCURS TWICE IN LIST
C
          DO 50 J=1,NEDGES-2
            NODE1=LIST1(J)
            IF(NODE1 NE 0)THEN
              NODE2=LIST2(J)
              DO 40 K=J+1,NEDGES-1
                IF(LIST2(K) EQ NODE1)THEN
                  IF(LIST1(K) EQ NODE2)THEN
                    LIST1(K)=0
                    LIST1(J)=0
                    GOTO 50
                  END IF
                END IF
   40         CONTINUE
            END IF
   50     CONTINUE
C
C         FORM NEW TRIANGLES
C
          DO 60 J=1,NEDGES-1
            IF(LIST1(J) NE 0)THEN
C
C             FIND VACANT LOCATION FOR ASSEMBLY OF NEW TRIANGLE
C
              IF(TOPSTK GT 0)THEN
                ADDRES=STACK(TOPSTK)
                TOPSTK=TOPSTK-1
                              ELSE
                NUMTRI=NUMTRI+1
                ADDRES=NUMTRI
              END IF
C
C             INSERT NEW TRIANGLE INTO THE LINKED LISTS
C
                ADDRES=NODE
                ADDRES=LIST1
                ADDRES=LIST2
              POINTR(LSTADD)=ADDRES
              POINTR(ADDRES)=0
              LSTADD=ADDRES
            END IF
   60     CONTINUE
   70     CONTINUE
          END
```

APPENDIX 2

Utility subprograms

```
      FUNCTION MINDV(DOUBLE,ISTR    I I
C
C*****************************    *    ***********************
C     FUNCTION MINDV
C     FIND MINIMUM VALUE OF A DOUBLE PRECISION VECTOR
C
C     INPUT PARAMETERS
C
C      DOUBLE  INPUT VECTOR
C      ISTRT  ADDRESS IN DOUBLE FOR START OF SCAN TO FIND MINIMUM
C      ISTOP  ADDRESS IN DOUBLE FOR END OF SCAN TO FIND MINIMUM
C
C     OUTPUT PARAMETERS
C
C      MINDV  MINIMUM VALUE IN DOUBLE OVER RANGE SPECIFIED BY ISTRT
C     AND ISTOP
C
C*****************************  ***********************************
C
      INTEGER ISTOP,ISTRT,I
C
      DOUBLE PRECISION DOUBLE(*),MINDV
C
      MINDV=DOUBLE(ISTRT)
      DO 10 I=ISTRT+1,ISTOP
        MINDV=MIN(MINDV,DOUBLE(I))
   10 CONTINUE
      END


      FUNCTION MAXDV(DOUBLE,ISTRT,ISTOP)
C
C*****************************************************************
C     FUNCTION MAXDV
C     FIND MAXIMUM VALUE OF A DOUBLE PRECISION VECTOR
C
C     INPUT PARAMETERS
C
C      DOUBLE  INPUT VECTOR
C     'ISTRT' ADDRESS IN DOUBLE FOR START OF SCAN TO FIND MAXIMUM
C      ISTOP' ADDRESS IN DOUBLE FOR END OF SCAN TO FIND MAXIMUM
C
C     OUTPUT PARAMETERS
C
C      MAXDV  MAXIMUM VALUE IN VECTOR OVER RANGE SPECIFIED BY ISTRT
C     AND ISTOP
C
C*****************************************************************
C
      INTEGER ISTOP,ISTRT,I
C
      DOUBLE PRECISION DOUBLE(*),MAXDV
C
      MAXDV=DOUBLE(ISTRT)
      DO 10 I=ISTRT+1,ISTOP
        MAXDV=MAX(MAXDV,DOUBLE(I))
   10 CONTINUE
      END


      SUBROUTINE FILLI(INTGR,ISTRT,ISTOP,VALUE)
C
C*****************************************************************
C     SUBPROGRAM FILLI
C     FILL INTEGER VECTOR WITH PRESCRIBED VALUE
C
C     INPUT PARAMETERS:
C
C      INTGR  VECTOR TO BE INITIALISED
C     'ISTRT  ADDRESS IN INTGR FOR START OF INITIALISATION
C      ISTOP' ADDRESS IN INTGR FOR END OF INITIALISATION
C      VALUE' VALUE THAT INTGR IS INITIALISED TO
C
C     OUTPUT PARAMETERS
C
C      INTGR  VECTOR FILLED WITH PRESCRIBED VALUE
C
C*****************************************************************
C
      INTEGER INTGR(*),ISTRT,ISTOP,VALUE,I
C
      DO 10 I=ISTRT,ISTOP
        INTGR(I)=VALUE
   10 CONTINUE
      END
```