A Reproduced Copy

NASA-TM-81287 19810015278

NASA Technical Memorandum 81287

USAAVRADCOM TR 81-A-9

A General Algorithm for the Construction of Contour Plots

Wayne Johnson and Fred Silva

(NASA-TM-81287) A GENERAL ALGORITHM FOR THE CONSTRUCTION OF CONTOUR PLOTS (NASA) 107 p HC A06/MF A01 CSCL 12A

N81-23813

Unclas

G3/61 42271

April 1981

National Aeronautics and Space Administration

United States Army Aviation Research and Development Command

NASA Technical Memorandum 81287

USAAVRADCOM TR 81-A-9

A General Algorithm for the Construction of Contour Plots

Wayne Johnson, Aeromechanics Laboratory
AVRADCOM Research and Technology Laboratories
Ames Research Center, Moffett Field California

Fred Silva, Informatics, Inc., Palo Alto, California

NASA
National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California 94035

United States Army
Aviation Research and Development Command
St. Louis Missouri 63166

CONTENTS

PRECEDING PAGE BLANK NOT FILMED

A GENERAL ALGORITHM FOR THE CONSTRUCTION OF CONTOUR PLOTS

Wayne Johnson

Ames Research Center
and
Aeromechanics Laboratory
AVRADCOM Research and Technology Laboratories

Fred Silva
Informatics, Inc.

SUMMARY

An algorithm is described that performs the task of drawing equal-level contours on a plane, which requires interpolation in two dimensions based on data prescribed at points distributed irregularly over the plane. The approach is described in detail. The computer program that implements the algorithm is documented and listed.

1.0 INTRODUCTION

The graphical presentation of experimentally or theoretically generated data sets frequently involves the construction of contour plots. Consider a dependent variable z that is a function of two independent variables x and y: $z = f(x,y)$. The functional form f is not known. It is assumed that f is a single-valued function of x and y. By measurements or calculations, the value of z is obtained at a set of N discrete points. The data may be presented in graphical form in terms of contours of equal value of z on the x-y plane. To construct such contours, it is necessary to interpolate the values of z between the prescribed data points. In general, these data points may be distributed irregularly over the x-y plane. This report describes an algorithm developed to construct contour plots for such cases. The computer program that implements the algorithm is documented and listed.

1.1 Description of the Approach

The data are prescribed at a set of N points distributed irregularly over the x-y plane: z_n, x_n, y_n for n=1 to N. In order to perform the interpolation, the points on the x-y plane are connected by straight line segments, to form a set of triangles with a convex boundary (figure 1). Then the data can be

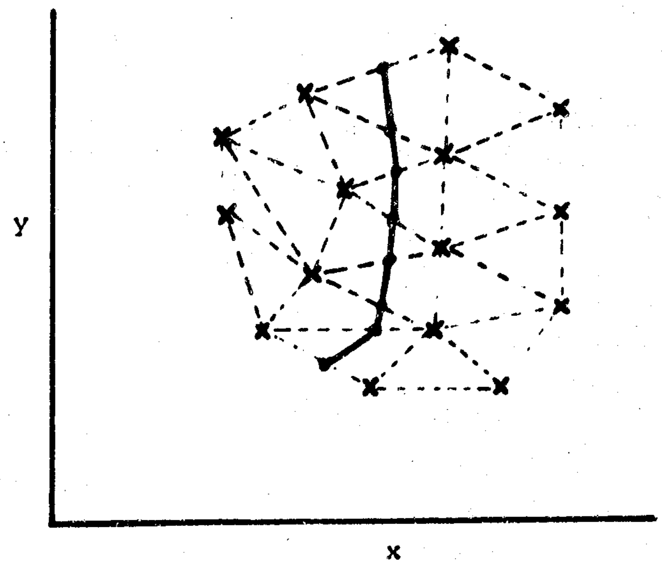


Figure 1. Construction of the contours

interpolated over the edges of the triangles. To construct the contour for the value Z, all points on the edges where z=Z are located. Finally, these points are connected to form the z=Z contour. With the triangulation algorithm described here, the interpolation along the edges often involves widely separated points on the x-y plane. In such a case, linear interpolation between the end points of the edge is unlikely to produce a smooth contour. Hence, it is usually necessary to smooth the data, by using a least-squared-error fit of the data to a bi-variable polynominal for z=f(x,y). Then the interpolation along the edges is performed using this functional form. It is also possible to use some standard technique to draw a smooth curve through the interpolated points on the edges of the triangles. In summary, the algorithm involves four basic steps: (a) triangulation of the plane; (b) smoothing of the data; (c) interpolation along the edges; and (d) drawing the contours.

The first step is triangulation of the plane. There are N data points x_n and y_n. The triangulation will be described by an array that identifies the two end points of each edge, and an array that identifies the three vertices of each triangle. At each stage in the procedure, there are a set of points that define (in order) a boundary, inside which the triangles have been identified. At the start, all the data points are outside the boundary, and no points on the boundary have been located. The last data point and the data point closest to it are used to start the procedure: they are the initial boundary points, and are no longer in the set of points outside the boundary: one edge has been identified. Thereafter, the algorithm proceeds by marching around the boundary, examining points outside the boundary relative to a boundary edge. The objective is to identify a point that together with the boundary edge will form a new triangle. The criteria for locating such a point are that it be closest to the boundary edge and that there be no other points within the resulting triangle. These criteria are satisfied by locating the point such that the parameter D is minimized, where D equals the sum of the distances from the point to the two end points of the boundary edge. The points examined in this manner are those on the boundary, immediately adjacent to the boundary edge being considered; as well as the points outside the boundary. From the points outside the boundary it is necessary to exclude any for which the resulting triangle would overlap the triangles already identified (within the boundary), which requires two tests. First, relative to the boundary edge there is a side within the boundary. The straight line formed by the boundary edge and its extensions to infinity divides the x-y plane into two half-planes. All points that are either on this line or in the half-plane corresponding to within the boundary are immediately excluded. Second, the point identified as closest to the boundary edge is examined to determine whether the two new edges of the resulting triangle would pass through any of the boundary, inside which the triangles have been identified. At the start, all the data points are outside the boundary, and no points on the boundary have been located. The last data point and the data point closest to it are used to start the procedure: they are the initial boundary points, and are no longer in the set of points outside the boundary; one edge has been identified. Thereafter, the algorithm proceeds by marching around the boundary, examining points outside the boundary relative to a boundary edge. The objective is to identify a point that together with the boundary edge will form a new triangle. The criteria for locating such a point are that it be closest to the boundary edge and that there be no other points within the resulting triangle. These criteria are satisfied by locating the point such that the parameter D is minimized, where D equals the sum of the distances from the point to the two end points of the boundary edge. The points examined in this manner are those on the boundary, immediately adjacent to the boundary edge being considered; as well as the points outside the boundary. From the points outside the boundary it is necessary to exclude any for which the resulting triangle would overlap the triangles already identified (within the boundary), which requires two tests. First, relative to the boundary edge there is a side within the boundary. The straight line formed by the boundary edge and its extensions to infinity divides the x-y plane into two half-planes. All points that are either on this line or in the half-plane corresponding to within the boundary are immediately excluded. Second, the point identified as closest to the boundary edge is examined to determine whether the two new edges of the resulting triangle would pass through any of the other edges on the boundary (which may happen if the boundary is concave). If so, the point is excluded. When a point has been successfully found from among the points outside the boundary, a new triangle and two new edges have been identified; a new boundary point is inserted between the two current boundary points being considered (hence two new boundary edges replace the old edge); and the point is no longer outside the boundary. When a point has been successfully found from among the adjacent boundary points, a new triangle and one new edge has been identified; and the middle boundary point is no longer on the boundary (hence the new boundary edge replaced the two old edges). This procedure continues, marching around the boundary until there are no more points outside the boundary. The boundary may be concave at this stage, however, so the procedure still continues, examining adjacent boundary points relative to each boundary edge until the boundary is completely convex, that completes the triangulation. The end points of all edges have been identified. For the interpolation procedure it is necessary then to identify those edges that form the boundary. To draw the contours, the four other edges that form the two triangles on either side of each edge must be identified as well.

The following relationships are useful. Let P = number of data points, E = number of edges, T = number of triangles, and B = number of boundary points or edges. Then

$$E = \frac{3}{2}T + \frac{1}{2}B$$

$$P = \frac{1}{2}T + (\frac{1}{2}B + 1)$$

so

$$T = 2(P - 1) - B$$

$$E = 3(P - 1) - B$$

$$E - T = P - 1$$

The minimum number of boundary points $B_{min} = 3$ gives the maximum number of triangles and edges: $T_{max} = 2P-5$ and $E_{max} = 3P-6$. The maximum number of boundary points is $B_{max} = P$, which gives: $T_{min} = P-2$ and $E_{min} = 2P-3$.

The triangulation depends only on the x and y coordinates of the data points, hence it is the same for all dependent variables. The remaining steps depend on the dependent variable as well.

The second step in the algorithm is smoothing of the data for z. This step is optional, and does not depend on the triangulation. The z-surface is fitted to a polynomial of the form:

$$\tilde{z} = \sum_{i=0}^{I} \sum_{j=0}^{L} c_{ij} x^i y^j$$

where

$$K = \text{maximum}\ (I,J)$$

$$L = \text{minimum}\ (K-i,J)$$

The input parameters I and J define the highest powers in the polynomial. The coefficients c_{ij} are obtained from a least-squared error fit of this function z to the actual data z, at the set of N data points. Then the polynomial is used to evaluate a new set of z values at the data point. This set of smoothed values of the dependent variable replaces the original data in the interpolation algorithm. The error of the smoothed data is defined as:

$$\epsilon = \frac{1}{N} \left[\sum_{n=1}^{N} (z_{n_{old}} - z_{n_{new}})^2 \right]^{\frac{1}{2}}$$

The third step is interpolation along the edges. The contour value Z is specified. Then each edge is examined to determine whether $z_1 \leq Z \leq z_2$ where z_1 and z_2 are the values of the dependent variable at the end points of the edge. If so, then there is a point on the edge where z = Z, hence this is a point on the required contour. This point is obtained by linear interpolation between the end points if the data has not been smoothed. If the data has been smoothed, the fitted polynomial is used to evaluate z along the edge and hence locate the point where z = Z. The result of the interpolation procedure is a set of points on the x-y plane where z = Z, and the edges on which these points are located.

The fourth step is drawing the contour for z = Z. The task is to convert the interpolated points in the proper order. The contour will consist of one or more lines that either start and end at a boundary edge, or are closed curves. There can only be one contour through a triangle. The procedure starts by searching the list of interpolated points for one that lies on a boundary edge. There are two outer edges that form a triangle with this boundary edge, which were identified in the triangulation algorithm. The contour must pass through one, and only one of these edges. So the list of interpolated points is searched for the point that lies on one of these two edges. There are four edges (identified in the triangulation algorithm) that form two triangles with one edge on which this second point lies. The list of interpolated points is searched for the point that lies on one of these four edges. (There will be only one such point in the list: one from each of the two triangles, and one of these will be the previous point on the contour.) The procedure continues searching for points in this fashion until another boundary point is reached. Then a contour line is drawn through these points, in the order located. The procedure is repeated until there are no more points in the list that lie on boundary edges. If there are still interpolated points that have not been used, there must be a contour segment that forms a closed curve. One of the remaining points is picked as a starting point, and the above procedure is followed until this starting point is encountered again. Then a contour line is drawn through these points, in the order located. The procedure is repeated until all the interpolated points have been used.

The desired contours are specified in terms of a base value z_o and an increment Δz, so the contour value is $Z = z_o + n\Delta z$ where n is any integer (positive, negative, or zero). The interpolation and contour drawing steps are repeated for every such Z that lies within the range of the data.

The computer program described here does not include the graphics software. The user must supply the subroutine that is called to draw the contour on the particular graphics device being used for the output.

1.2 Summary of Component Modules

The above procedures are computationally independent steps in the process. For this reason, each procedure is self-contained within separate subroutine modules. One master subroutine is called by the user program and it, in turn, controls and sequences the execution of the procedures described above. The master subroutine also accepts, by means of an argument list, the data and parameters that the user supplies for the procedure. In addition, the user supplies a subroutine for graphics output of the contour lines as they are generated.

The modular approach allows flexibility in modifying the algorithm for certain applications. In cases where the x-y data points define a regular or predetermined grid on the plane, it may be desirable to replace the triangulation subroutine with a specific procedure for the known distribution of points. This replacement will often increase the execution speed substantially. In other cases, there may be a large number of data points given and the function values may be regular enough to allow for a linear interpolation over many triangle edges. For such a case, the smoothing option would not be exercised and the procedure for the surface curve fitting could be deleted altogether. This would result in a substantial savings in object time program size.

There are other variations which may be used to modify the method for the purpose of reducing object time storage requirements or increasing execution speed. These modifications are discussed later in Section 7.

The remainder of this section is composed of a short description of each component module. Figure 2 presents a heirarchy diagram of the processing package.

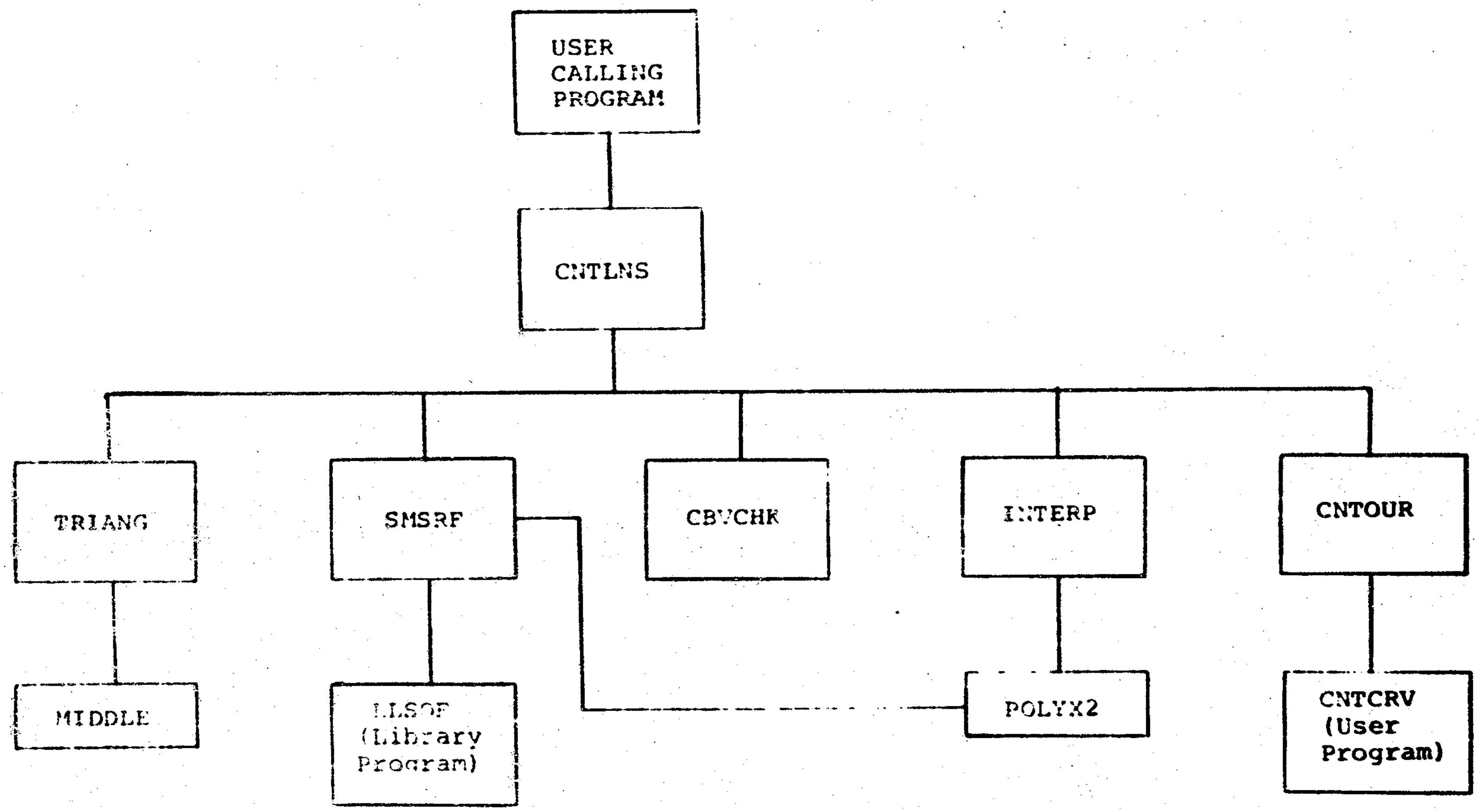


Figure 2. Program Heirarchy Diagram

CNTLNS

This is the subroutine accessed by the users calling program for drawing contour lines of constant z_c for some set of data defining $z = f(x,y)$. CNTLNS is supplied with the known values of x,y and z, several computational parameters, and a list of constant z values for which contours are to be calculated and drawn. There must be at least 3 triplets of x-y-z points and no duplicate points are allowed. The function $z = f(x,y)$ must be single valued.

TRIANG

Called by CNTLNS. This subroutine constructs the convex polygon of triangles from the x-y data.

MIDDLE

Function subprogram used by TRIANG. This routine finds the middle value of three known integer values.

SMSRF

Called by CNTLNS. Performs least-square smoothing of the z-surface. The smoothing is an optional procedure.

LLSQF

Called by SMSRF. This is a utility module taken from the International Mathematical and Statistical Library (IMSL). LLSQF is used to solve a linear least-squares problem. It solves for the solution vector X of the general problem AX = B, where A is the coefficient matrix and B is the right hand solution vector. LLSQF is a proprietary program; LLSQF or its equivalent must be obtained by the user.

INTERP

Called by CNTLNS. Performs linear or non-linear interpolation over the triangle edges for constant contour values.

POLYX2

Function subprogram used by INTERP to evaluate the polynomials obtained in SMSRF for values on triangle edges.

CNTOUR

Called by CNTLNS. Reorders interpolated points into proper contour lines. Both closed and open contours are accommodated. CNTOUR calls a user supplied subroutine to draw the contour line. The user subroutine must be named CNTCRV.

CBVCHK

Called by CNTLNS. If the user specifies a base value and increment scheme for defining Z_c (as described later), then this routine is used to verify that Z_o is within the range of the known data. If not, Z_o is incremented or decremented by ΔZ until Z_o is in the proper range.

CNTCRV

Called by CNTOUR. This is the user supplied subroutine used to draw the contour on the graphics device.

2.0 MASTER SUBROUTINE

The subroutine CNTLNS is the user's application program contact with the contour software. Its primary function is to check for errors and, based on user input parameters, control and properly sequence the calls to other modules which perform the computational tasks. After all requested contours have been processed, control is passed back to the application program.

2.1 Description of Argument List

```
CALL CNTLNS (X,Y,Z,N,ISMOPT,IEXP,JEXP,NCNTRS,CLIST,
            EPSLON,IERR)
```

Input arguments:

X_n = the list of independent variable values for the function $z = f(x,y)$ for n = 1 to N

Y_n = the list of independent variable values for the function $z = f(x,y)$ for n = 1 to N

Z_n = the list of dependent variable values for the function $z = f(x,y)$ for n = 1 to N

N = the range of N for the x,y and z lists

ISMOPT = smoothing option parameter

= 0 for no smoothing
≠ 0 then the function $z = f(x,y)$ is smoothed by means of a least squared error curve fit

IEXP = highest order of the smoothing polynomial for x if ISMOPT ≠ 0

JEXP = highest order of the smoothing polynomial for Y if ISMOPT ≠ 0

(The dimension C in the program must be at least $(K+1)(L+1-\frac{1}{2}K)$ where $K = \min(I,J)$ and $L = \max(I,J)$.)

NCNTRS = the number of contours of constant Z to be generated, and NCNTRS ≤ 50. If NCNTRS ≤ 0, then the program will determine constant Z values to process from the relation

$$Z_c = Z_o + n\Delta Z$$

where Z_c = constant Z value
Z_o = contour base value
ΔZ = increment value.

$CLIST_j$ = If 1 ≤ NCNTRS ≤ 50, then CLIST is the list of constant Z values (Z_c) for which contours will be generated, for j=1 to NCNTRS.

If NCNTRS ≤ 0, then CLIST(1) is taken to be Z_o and ΔZ = CLIST(2).

Return arguments:

EPSLON = the error ε, introduced by the smoothing if ISMOPT ≠ 0.

IERR = return error flag

= 0 for no errors
= 1 for N<3 or N>MAXPTS where MAXPTS is the maximum number of x,y,z triplets allowed
= 2 for invalid IEXP and/or JEXP values if ISMOPT ≠ 0

(Note -
IERR is 2 if the number of coefficients resulting from IEXP and JEXP is greater than MAXCOF or greater than N, the number of points under consideration)

(Where MAXPTS is the dimension N, and MAXCOF is the dimension C in the program.)

Required dimensions:

X(N)
Y(N)
Z(N)
CLIST(50)
ZNEW(N)
IE(E,2)
ITE(E,4)
XI(E)
ETA(E)
LAMBDA(E)
IBE(E)
IPOWR(C)
JPOWR(C)
COEF(C)

For the array dimensions given above, and for all array dimensions used in this document, the following definitions apply:

N = the maximum number of data points to be processed

C = the maximum number of coefficients to be used for smoothing

E = 3N-6 = the maximum number of triangle edges produced by the triangulation of N points

T = 2N-5 = the maximum number of triangles produced by the triangulation of N points.

2.2 Description of Algorithm

Figures 3a and 3b present a block diagram of the module CNTLNS. The functions of parts A to M are as follows:

Figure 3a. Block Diagram of CNTLNS, Parts A to F

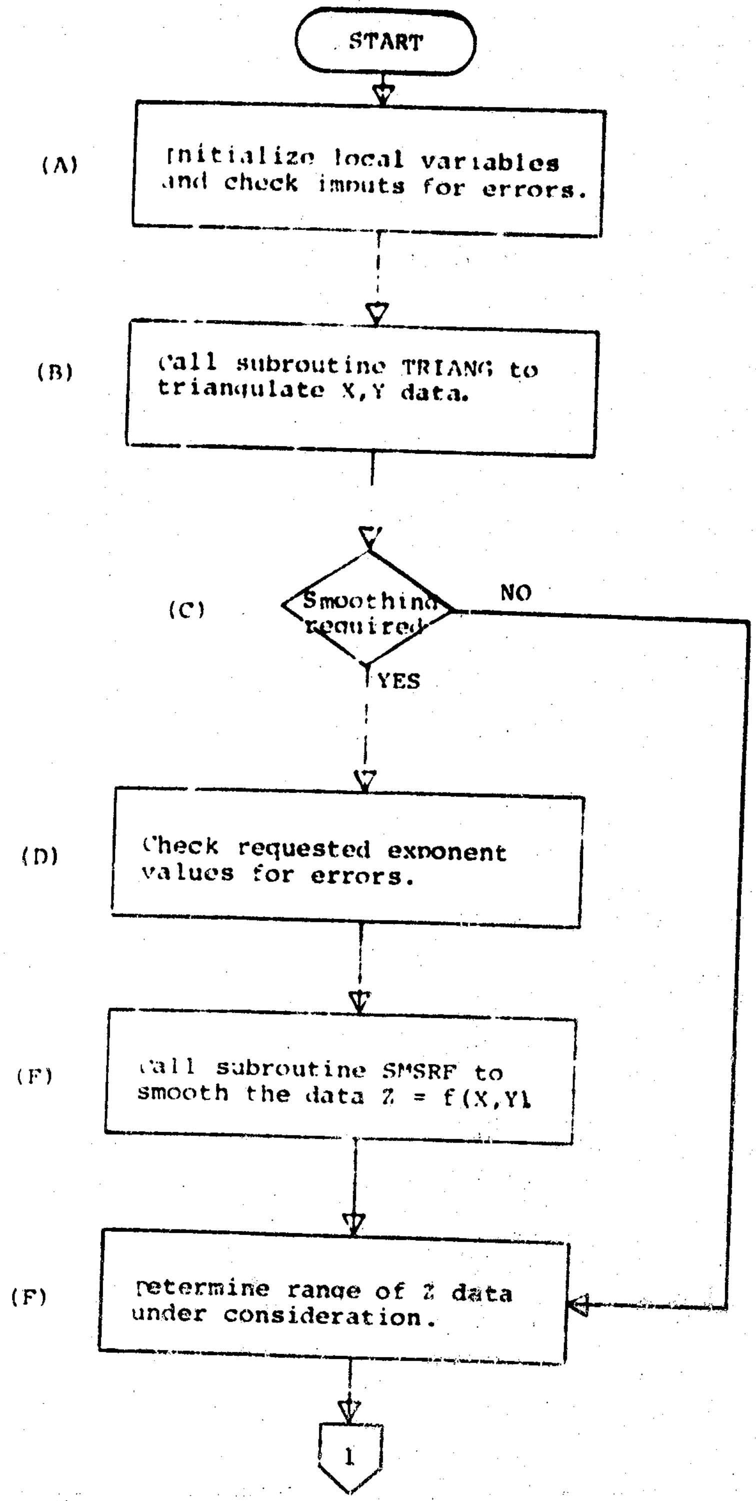
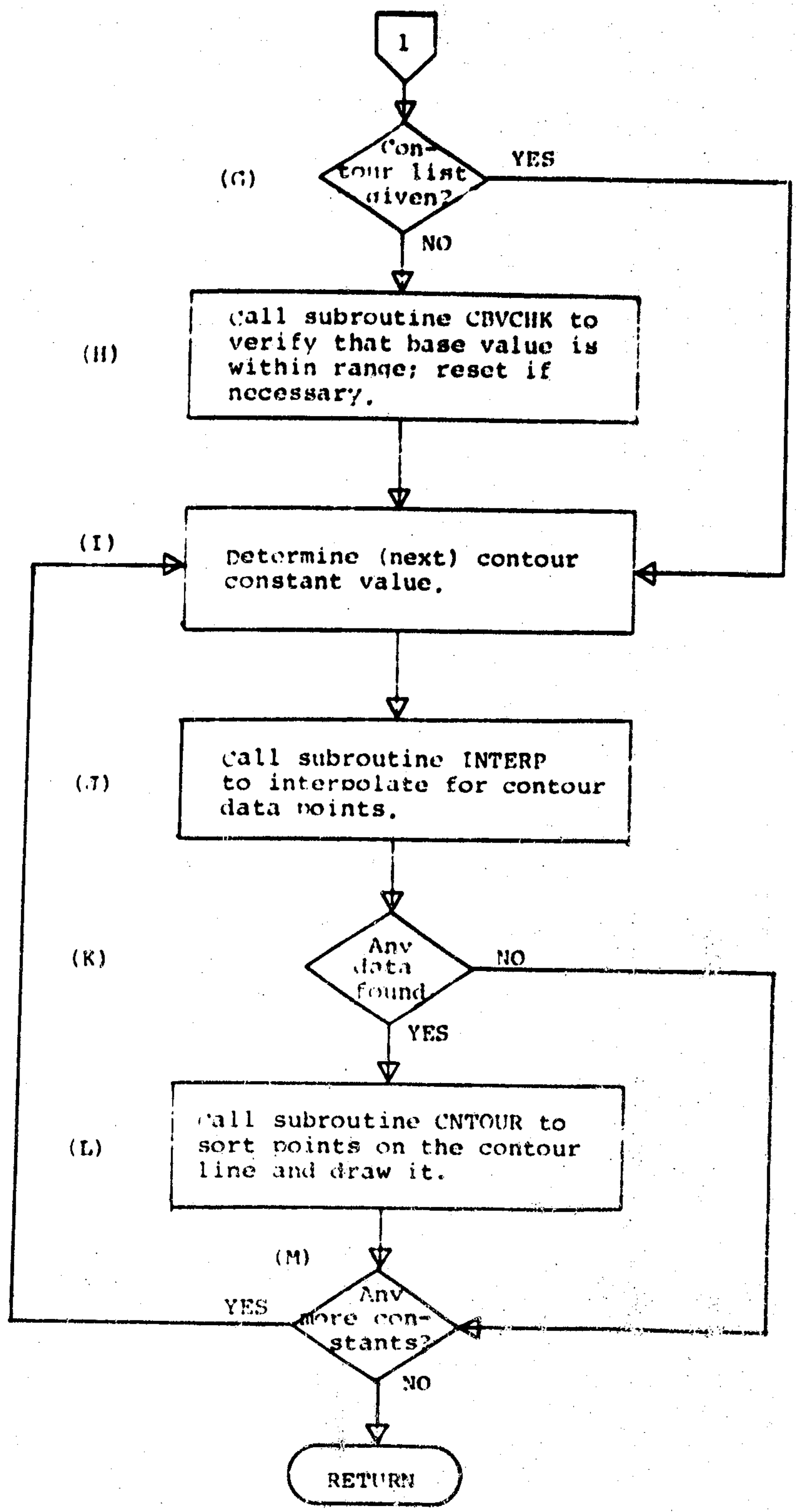


Figure 3b. Block Diagram of CNTLNS, parts G to M



(A) Initialize local variables and check input arguments for errors

```
MAXCOF = 23, MAXPTS = 500
IERR   = 0
E      = 0.0
if N<3 or N>MAXPTS goto (997)
```

(B) Call subroutine to triangulate the X-Y data

```
CALL TRIANG(X,Y,N,LEDGES,IE,IBE,ITE)
```

(C) If smoothing option is off (equal to zero) then skip around sections D and E

```
if ISMOPT = 0, goto (110)
```

(D) Else, check the requested exponents for errors

```
if IEXP<0 or JEXP<0, goto (998)
I1   = IEXP+1, J1 = JEXP+1
NMIN = minimum of I1, J1
NMAX = maximum of I1, J1
if J1>I1 then NC = (IEXP+1)*(JEXP+1) - IEXP/2)
if J1<I1 then NC = (JEXP+1)*(IEXP+1 - JEXP/2)
if NC>N or NC>MAXCOF, goto (998)

        [for K=1,MAXCOF
        [IPOWR(K) = 0
        [JPOWR(K) = 0
```

(E) Call subroutine to smooth the data for the function Z=f(X,Y)

```
Call SMSRF (X,Y,Z,ZNEW,N,IEXP,JEXP,NCOEF,COEF,IPOWR,JPOWR)
```

If there were no errors in the smoothing process, calculate the epsilon value -- the normalized error

```
if NCOEF<0 goto (120)

        for k = 1 to N
        ε = + (Z_k - ZNEW_k)

ε = (√ε)/FLOAT(N)
goto (120)

(110)  [for k = 1 to N
       [ZNEW_k = Z_k
```

(F) Determine the range of the Z data under consideration. The minimum and maximum values of Z determines the contour values which can be accommodated.

```
      ZMIN = Z(1)
      ZMAX = ZMIN

              [for k = 2 to N
              [ZMIN = minimum of ZMIN, Z_k
              [ZMAX = maximum of ZMAX, Z_k
```

(G) Branch around the next section if the contour list is given,
(H) otherwise call subroutine to range check the base value and reset it if necessary.

```
      K =0
      FN=1.0

(200) if NCNTRS = 0 goto (180)
      call CBVCHK (CLIST(1),CLIST(2),ZMIN,ZMAX,CLNEW)
      if CLIST(1) ≠ CLNEW then CLIST(1) = CLNEW
```

(I) Determine the (next) contour constant value.

```
(210) K = K+1
      ZCON = (K) (FN) (CLIST(2)) + CLIST(1)
      if ZCON>ZMIN and ZCON<ZMAX goto (150)
      If FN<0.0 goto (300)
      FN =-1.0
      K  = 0
      goto (210)

(180) K = K+1
      if K>NCNTRS goto (300)
      ZCON = CLIST(K)
      If ZCON<ZMIN or ZCON>ZMAX goto (200)
```

(J) Call subroutine INTERP to interpolate
(K) contour points for constant Z

```
(150) CALL INTERP (X,Y,ZNEW,ZCON,LEDGES,IE,IEXP,JEXP,ISMOPT,LAMBDA,
                   XI,ETA,J,COEF,IPOWR,JPOWR,NCOEF,N)

(L)   if J≠0, CALL CNTOUR (ZCON,XI,ETA,LAMBDA,J,IBE,ITE)

(M)   goto (200)

(300) RETURN

(997) IERR = 1
      RETURN

(998) IERR = 2
      RETURN
```

2.3 Description of Subroutine CBVCHK

This subroutine is called by CNTLNS after the Z data range has been determined. CBVCHK will check the given value of the contour base value (Z_o) to verify that it is within the range of the data. If not, the base value is shifted by the given increment (ΔZ) until ZMIN $\leq Z_o \leq$ ZMAX, and the shifted value of Z_o assumes the new reset value. This verification and resetting of Z_o is often useful for cases in which the given base value is only a guess by the user and the range of the Z data may not be known in advance. The argument list for CBVCHK is established as follows:

```
CALL CBVCHK (ZZERO,DELZ,ZMIN,ZMAX,ZZNEW)
```

Where Z_o and ΔZ are the selected base and increment values for selecting the constant values of Z for the contours, ZMIN and ZMAX are the data range as calculated in CNTLNS. Z_{new} is, on return, the base value which falls between ZMIN and ZMAX and may or may not be equal to Z_o.

3.0 TRIANGULATION SUBROUTINE

The subroutine TRIANG performs the triangulation, as described in Section 1. This subroutine uses the function middle.

3.1 Description of Argument List

CALL TRIANG (XD,YD,N,L,E,BE,TE)

The triangulation algorithm is supplied with a set of N data points (X_i,Y_i), i=1 to N. The coordinate pairs are to be connected by straight lines to form the triangles. The procedure input consists of:

XD(i) = the list of abscissa values

YD(i) = the list of corresponding ordinates

N = the range of i; the number of points in the x and y lists

The subroutine output consists of a set of index pointers defining each triangle edge, each boundary edge of the final polygon, and indices of adjacent edges to each triangle edge. The subroutine output is stored as:

$E(\ell,2)$ = index pointer of end points of a triangle edge in ascending order $(E(\ell,1)<E(\ell,2)$ for all ℓ for ℓ = 1 to L

BE(ℓ)	=	1 if the ℓ-th row of E defines a boundary edge; otherwise equal to zero; for ℓ = 1 to L.
TE(ℓ,4)	=	index of adjacent edges for each corresponding row of E; for ℓ = 1 to L.
L	=	total number of edges constructed by the triangulation procedure

An assortment of local variables are used during the triangulation process and are defined as follows:

P(j)	=	Index numbers of points lying outside the boundary of the triangulated points. P lists the indices of the remaining candidate points, for j = 1 to J.
J	=	Number of points remaining in array P.
B(k)	=	Index numbers of points defining the current boundary, in order, for k = 1 to K.
K	=	Number of values listed in array B.
T(m,3)	=	Indices of triangle vertices of each triangle, in ascending order, for m = 1 to M.
M	=	Total number of triangles; the same as the limit of m for array T.
X(i), Y(i)	=	Arrays of X and Y data after the XD and YD input values have been scaled by the range of data. Scaling of the data eliminates problems with machine precision while leaving the relative position of the data points unchanged.

3.2 Description of the Algorithm

Figures 4a to 4e present a block diagram of the module TRIANG. The functions of parts A to Y are as follows.

Figure 4a. Block Diagram of TRIANG, Parts A to F

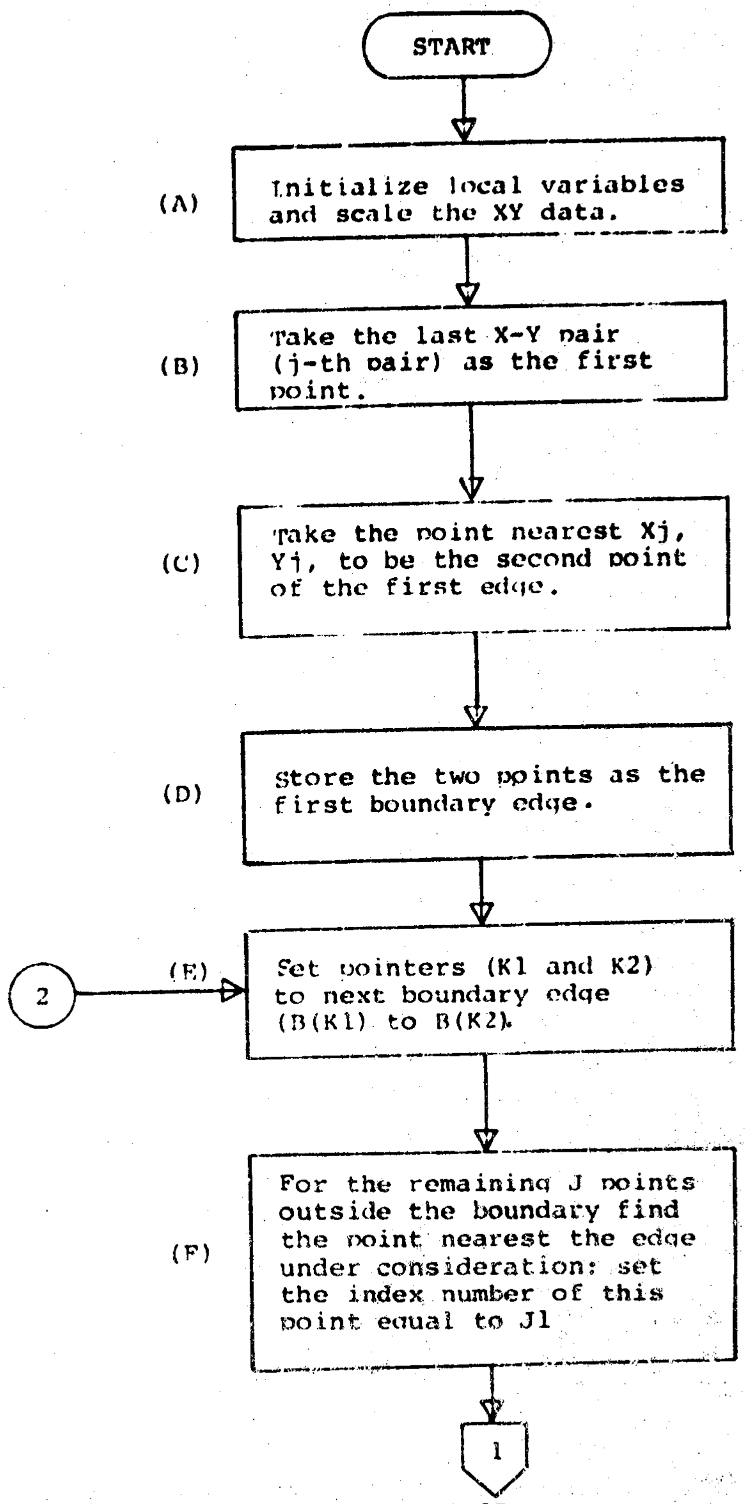


Figure 4b. Block Diagram of TRIANG, Parts G to L

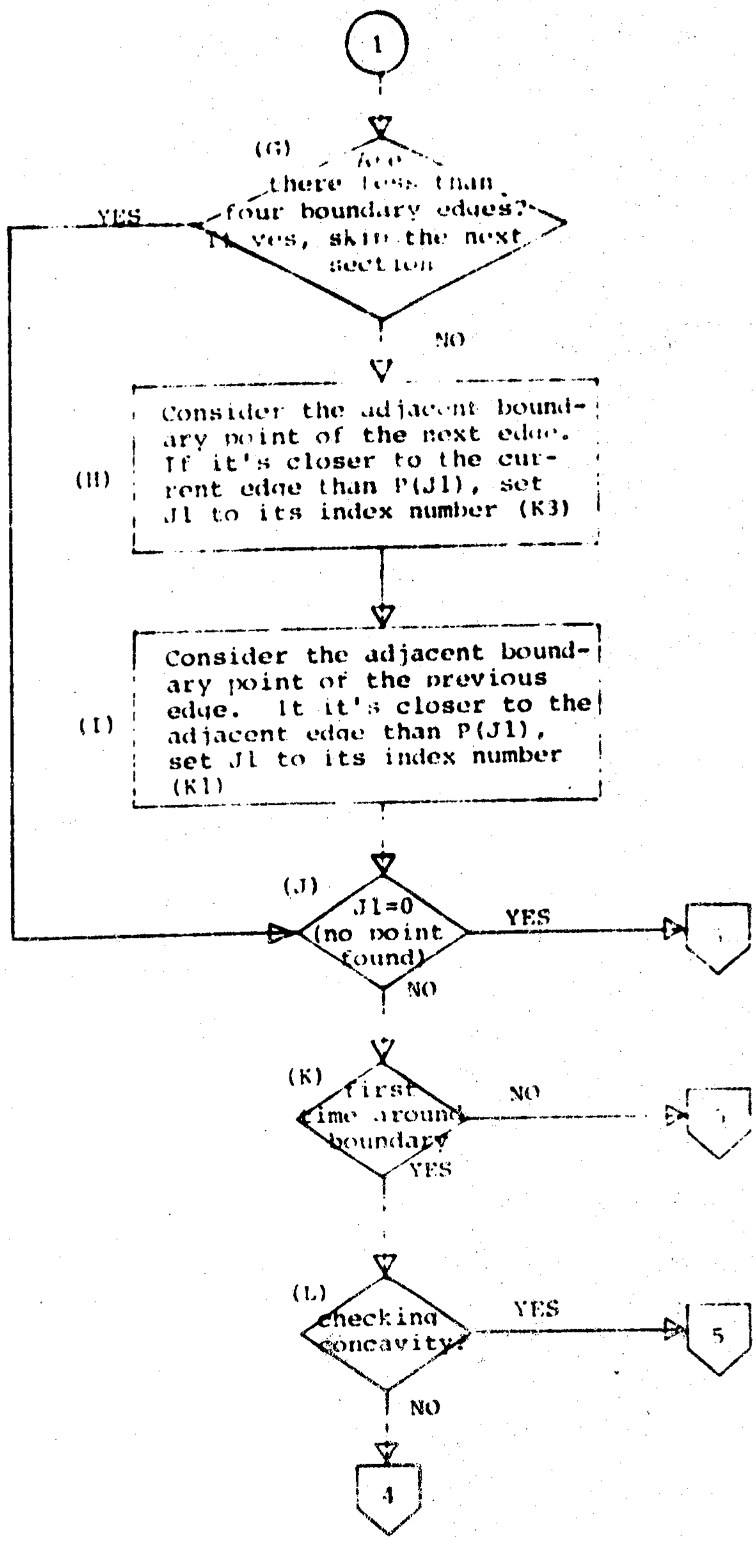


Figure 4c. Block Diagram of TRIANG, Parts M to O

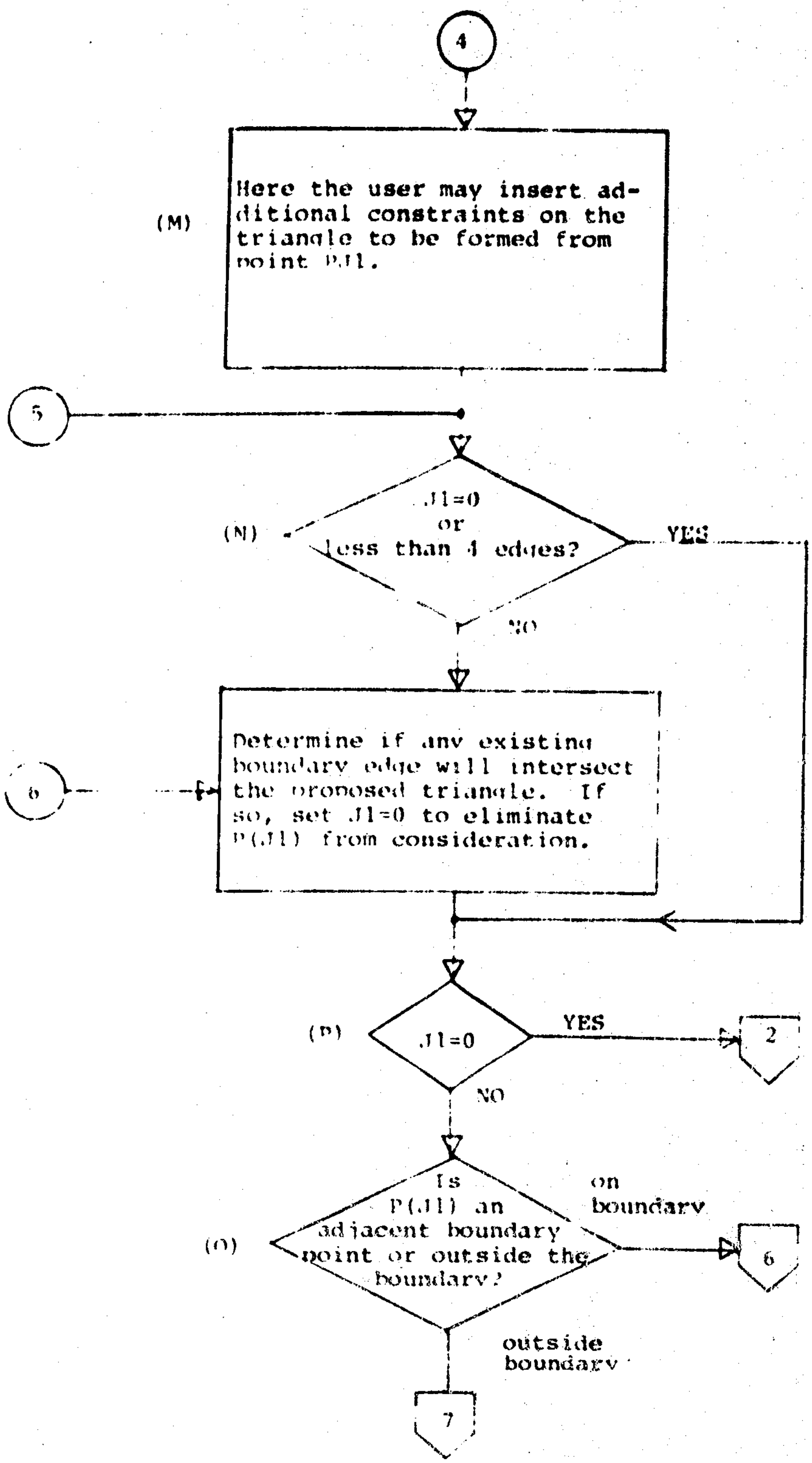


Figure 4d. Block Diagram of TRAING, Parts R to V

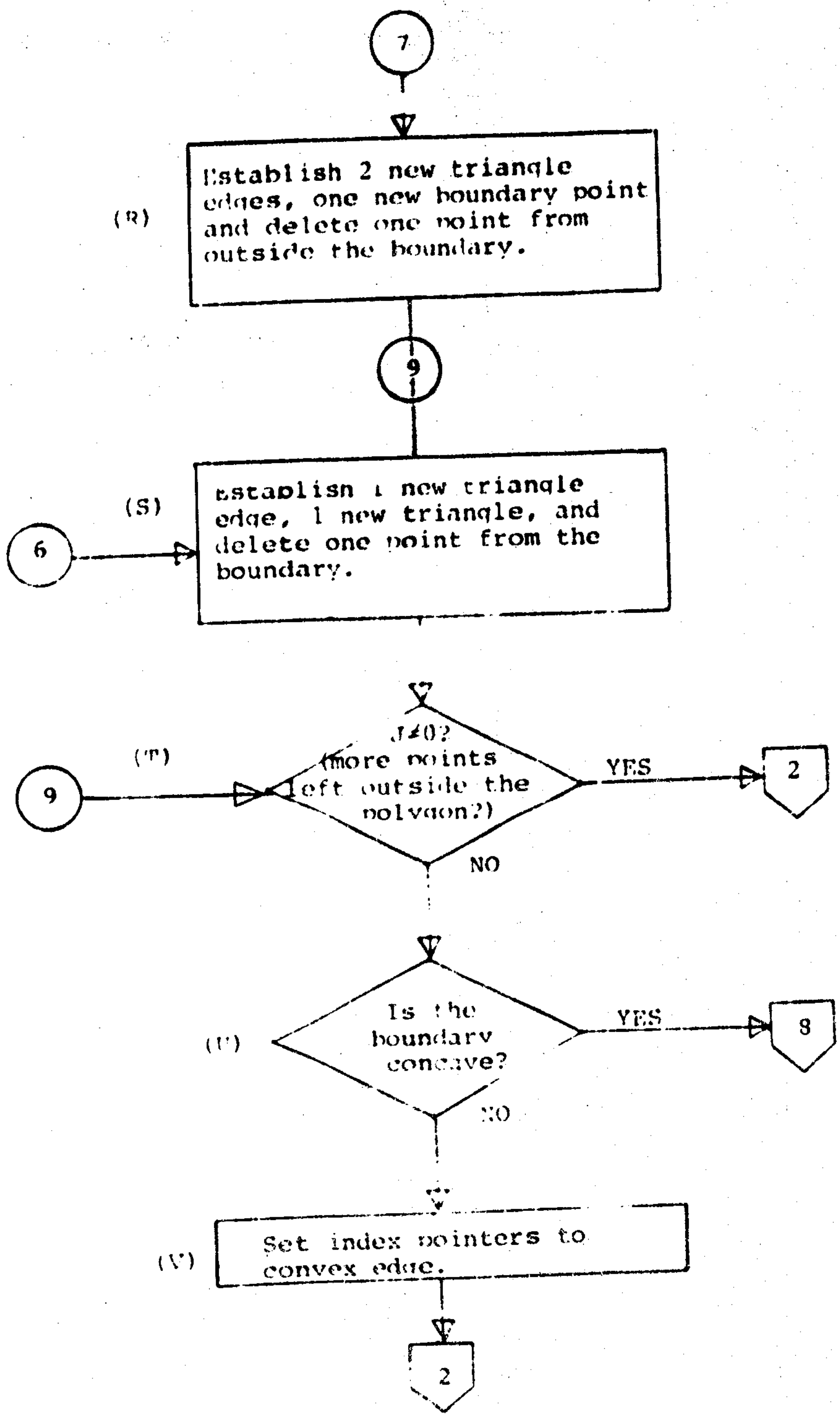
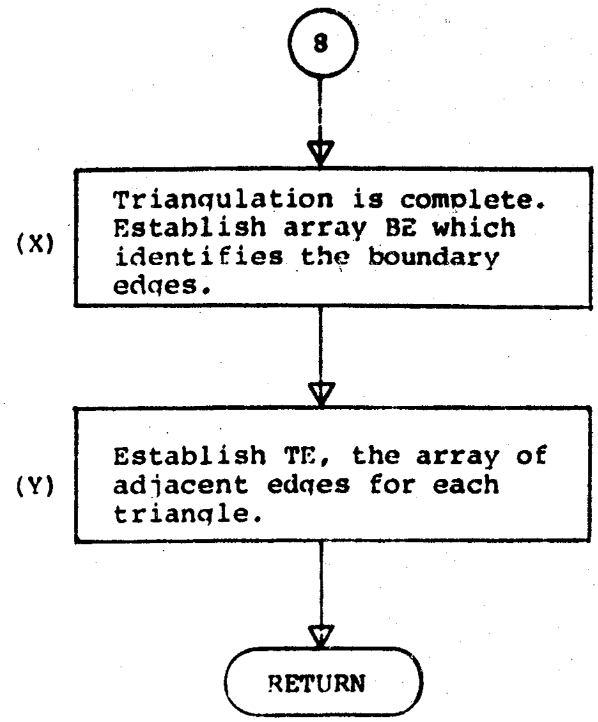


Figure 4e. Block Diagram of TRAING, Parts X to Y



(A) The procedure begins with no boundary, no edges, and all points under consideration. Initialize local variables and scale the X,Y data.

```
J=N,  K=L=M=0
P(j)=j for j=1 to J

XMAX=XMIN=XD(1)
YMAX=YMIN=YD(1)

    [ for k=2 to J
    | XMAX=maximum of XMAX,XD(k)
    | XMIN=minimum of XMIN,XD(k)
    | YMAX=maximum of YMAX,YD(k)
    [ YMIN=minimum of YMIN,YD(k)

DLXINV=1.0/(XMAX-XMIN)
DLYINV=1.0/(XMAX-XMIN)

    [ for k=1 to J
    | X(k)=(DLXINV)(XD(k))
    [ Y(k)=(DLYINV)(YD(k))
```

(B) Begin by taking the last pair of points, $(X,Y)_J$ in the list to be the first boundary point.

```
B(1)=J
J=J-1
```

(C) From the remaining points, find the point nearest the first

```
i1=B(1)
find i2≠i1 which minimizes
[(X(i1)-X(i2))² + (Y(i1)-Y(i2))²]
```

(D) Now, B(1) to B(i2) is the first edge. There is one edge and two boundary points.

```
B(2)=i2, K=2, L=1, J=J-1
E(ℓ,1)=i1, E(ℓ,2)=i2
if i2<J then P(j)=P(j+1)
                 for j=i2 to J
```

(E) Now begin circling around the boundary of the polygon, considering, in order, each boundary edge. The following indices are maintained --

K1=B array index of current edge - point 1
K2=B array index of current edge - point 2
B1,B2=index numbers of boundary point coordinates

```
      K1=KT=0
(11)  K1=K1+1      if K1>K then K1=1
(12)  K2=K1+1      if K2>K then K2=1
      B1=B(K1)
      B2=B(K2)
      KT=KT+1
```

(F) Consider the boundary edge from B1 to B2. For all points not yet triangulated (the J points remaining in P) find the point that, when triangulated with B1,B2, minimizes the length of the two new edges to be drawn.

J1=D1=0
BFLAG=0
if J=0 goto (6)

for LJ=1 to J

PJ=P(LJ)

if $\left[(Y_{PJ}-Y_{B1})(X_{B2}-X_{B1}) - (X_{PJ}-X_{B1})(Y_{B2}-Y_{B1})\right] \leq 0.0$

then goto (1)

$$D = \sqrt{(X_{PJ}-X_{B1})^2+(Y_{PJ}-Y_{B1})^2} + \sqrt{(X_{PJ}-X_{P2})^2+(Y_{PJ}-Y_{B2})^2}$$

if J1=0 or D1<D then J1=LJ, D1=D

(1) next LJ

(G) If less then three edges exist (no triangle defined yet) then there are no adjacent boundary points to be considered

if K≤3 goto (3)

(H) Consider the adjacent boundary point of the next edge of the polygon. Call its index number K3 and see if it's closer to the current edge then P(J1).

⑥ K3 = K2+1; <u>if</u> K3>K <u>then</u> K3=1

PK3 = B(K3)

<u>if</u> $\left[(Y_{PK3}-Y_{B1})(X_{B2}-X_{B1}) - (X_{PK3}-X_{B1})(Y_{B2}-Y_{B1})\right] \leq 0.0$

<u>then</u> goto ②

$$D = \sqrt{(X_{PK3}-X_{B1})^2 + (Y_{PK3}-Y_{B1})^2} + \sqrt{(X_{PK3}-X_{B2})^2 + (Y_{PK3}-Y_{B2})^2}$$

<u>if</u> J1=0 or D<D1 <u>then</u> J1=K3, D1=D, BFLAG=1

(I) Consider the adjacent boundary point of the previous edge of the polygon. Call its index number KØ and determine if it's closer to the current edge than P(J1) and B(K3)

② KØ = K1-1; <u>if</u> KØ<1 <u>then</u> KØ=K

PKØ = B(KØ)

<u>if</u> $\left[(Y_{PK\emptyset}-Y_{B1})(X_{B2}-X_{B1}) - (X_{PK0}-X_{B1})(Y_{B2}-Y_{B1})\right] \leq 0.0$

<u>then</u> goto ③

$$D = \sqrt{(X_{PK0}-X_{B1})^2 + (Y_P-Y_{B1})^2} + \sqrt{(X_{PK0}-X_{B2})^2 + (Y_{B2}-Y_{B1})^2}$$

<u>if</u> J1=0 or D D1 <u>then</u> J1=KØ, D1=D, BFLAG=-1

(J) Skip the next section if J1 is still zero, since a candidate point for triangulation with edge B1,B2 was not found.

③ <u>if</u> J1=0 goto ⑨

(K) (L) If the search for a candidate point has already considered each boundary edge at least once (KT>K) or if the boundary is being checked for concave edges (J=0), then the next section can be ommitted.

<u>if</u> KT>K <u>or</u> J=0 goto ⑨

(M) At this point the user may insert an additional constraint on the triangles, such as requiring that one interior angle be neither very small nor very large. If the triangle fails the test, it is deleted from consideration by setting J1=0.

(O) The next procedure checks all boundary edges of the polygon for intersection with the candidate triangle. If any existing boundary edge intersects any of the edges to be formed, then the candidate point is rejected. If BFLAG is not zero, then the edge defined by J1=KØ or J1=K3 is exempt from the test.

(N) If there are three or less existing boundary edges or if J1 has been set to zero, this test is omitted.

```
(9)   if K≤3 or J1=0  goto (7)

      if BFLAG=0   then NQ=P(J1)
      if BFLAG=1   then NQ=B(K3)
      if BFLAG=-1  then NQ=B(KØ)

      for KL=1 to K
      if KL=K1  goto (108)
      KN=KL+1;  if KL=K  then KN=1
      if BFLAG=-1 and (KL=KØ or KN=KØ) goto (108)
      if BFLAG=1  and (KL=K3 or KN=K3) goto (108)
      P1=B(KL)
      P2=B(KN)

          for JL=1 to 2
          if JL=1 and (BFLAG=0 or BFLAG=1) and KL=KØ goto (8)
          if JL=2 and (BFLAG=0 or BFLAG=-1) and KL=K2 goto (8)
          if JL=1 then BJ=B1
          if JL=2 then BJ=2

          XQB=X(NQ)-X(BJ)
          YQB=Y(NQ)-Y(BJ)
          X12=X(P1)-X(P2)
          Y12=Y(P1)-Y(P2)

          D=XQB*Y12-YQB*X12
          if D=0.0 goto (8)

          X1B=X(P1)-X(BJ)
          Y1B=Y(P1)-Y(BJ)

          S=(X1B*Y12-Y1B*X12)/D
          if S<0.0 or S>1.0 goto (8)

          TC=(XQB*Y1B-YQB*X1B)/D
          if TC<0.0 or TC>1.0 goto (8)

          J1=0
          goto (7)
      (8) next JL
(108) next KL
```

(7) continue

(P) If J1 is zero, then the candidate point did not pass the above tests or no point was found. If BFLAG is not zero, then a point on the boundary was found.

```
if J1=0 goto (10)
if BFLAG=1 goto (4)
if BFLAG=-1 goto (150)
```

(R) The triangulated point is outside the boundary. Establish two new edges, a new boundary point and delete one point from outside the boundary.

```
E(L+1,1) = minimum of P(J1), B(K1)
E(L+1,2) = maximum of P(J1), B(K1)
E(L+2,1) = minimum of P(J1), B(K2)
E(L+2,2) = maximum of P(J1), B(K2)

L=L+2
KT=0
M=M+1

T(M,1) = minimum of P(J1), B(K1), B(K2)
T(M,2) = middle  of P(J1), B(K1), B(K2)
T(M,3) = maximum of P(J1), B(K1), B(K2)

if K1≠K then B(k+1) = B(k) for k=K to (K1+1)

B(K1+1)=P(J1)
K=K+1
J=J-1

if J1<J then P(j)=P(j+1) for j=J1 to J
goto (10)
```

(S) The triangulated point is the next point on the boundary. Establish one new edge (from B(K1) to B(K3)), one new triangle (from B(K1) to B(K2) to B(K3)), and delete one point from the boundary (B(K2).

(4) KT=0, KK=0, KKNT=0

```
E(L+1,1)   minimum of B(K1), B(K3)
E(L+1,2)   maximum of B(K1), B(K3)
```

```
L=L+1
K=K-1
M=M+1

T(M,1) = minimum of B(K1), B(K2), B(K3)
T(M,2) = middle  of B(K1), B(K2), B(K3)
T(M,3) = maximum of B(K1), B(K2), B(K3)

if K2≤K then B(k)=B(k+1) for k=K2 to K

if K2=1 then K1=K1+1

goto (10)
```

(S) The triangulated point is the previous point on the boundary. Establish a new edge (from B(KØ) to B(K2)), one new triangle (from B(KØ) to B(K1) to B(K2)), and delete a point from the boundary (B(K1).

```
(150) KT=0, KK=0, KKNT=0

E(L+1,1) = minimum of B(KØ), B(K2)
E(L+1,2) = maximum of B(KØ), B(K2)

L=L+1
K=K+1
M=M+1

T(M,1) = minimum of B(KØ), B(K1), B(K2)
T(M,2) = middle  of B(KØ), B(K1), B(K2)
T(M,3) = maximum of B(KØ), B(K1), B(K2)

if K1≤K then B(k)=B(k+1) for k=K1 to K

K1=K1-1

if K1<1 then K1=K

goto (10)
```

(T) (U) If there are any points remaining outside the boundary, then repeat the procedure for the next edge.

```
(10) if J>0 and J1≠0 goto (12)
     if J>0 goto (11)
```

(V) All points have been triangulated. Check that all boundary edges form a concave polygon.

```
if KK≠0 goto (55)
KK=1, KL=0
```

```
(55)  KKNT=KKNT+1
      if KKNT N  goto (170)

(5)  KL=KL+1

          K2=KL+1,  if K2>K then K2=1
          K1=KL-1,  if K1<1 then K1=K

          PK=B(K),  B1=B(K1),  B2=B(B2)

          if [(Y_PK-Y_B1)(X_B2-X_B1) - (X_PK-X_B1)(Y_B2-Y_B1)]≤0 then
          goto (11)

     if KL<K  goto (5)
```

(X) The triangulation is complete and has been checked for a concave boundary. Now identify the boundary edges.

```
       for ℓ=1 to L
       BE(ℓ) = 0
(170)  KL = 0
   (21)     KL = KL+1

            if E(ℓ,1) ≠ B(KL) goto (22)
            K1 = K1+1
            if K1>K then K1=1

            if E(ℓ,2) ≠ B(K1) goto (24)
            BE(ℓ) = 1
            goto (23)
   (24)     K1 = KL-1
            if K1<1 then K1=K
            if E(ℓ,2) ≠ B(K1) goto (23)
            BE(ℓ) = 1
   (22)     if KL<K goto (21)
(23)   next ℓ
```

(Y) Finally, establish the indices of adjacent edges for each edge in the triangulation. Each boundary edge will have two adjacent edges: each interior edge will have four.

```
initialize TE

for ℓ = 1 to L
for i = 1 to 4
     TE(ℓ,i) = 0
```

```
establish TE
for m=1 to M
    for ℓ = to L
    if E(ℓ,1) = T(m,1) and E(ℓ,2) = T(m,2) then L1 = ℓ
    if E(ℓ,1) = T(m,2) and E(ℓ,2) = T(m,3) then L2 = ℓ
    if E(ℓ,1) = T(m,1) and E(ℓ,2) = T(m,3) then L3 = ℓ
λ=0;  if TE(L1,1)≠0 then λ=2
TE(L1,λ+1) = L2
TE(L1,λ+2) = L3
λ=0;  if TE(L2,1)≠0 then λ=2
TE(L2,λ+1) = L1
TE(L2,λ+2) = L3
λ=0;  if TE(L3,1)≠0 then λ=2
TE(L3,λ+1) = L1
TE(L3,λ+2) = L2
next m
RETURN
```

3.3 Description of Function MIDDLE

FUNCTION MIDDLE (I,J,K)

This function is used by the triangulation algorithm to find the middle value of three integer arguments (the value which is neither a minimum or maximum). I,J, and K are assumed to be discrete values, no two are equal.

4.0 SMOOTHING SUBROUTINE

The subroutine SMSRF performs the optional smoothing of the data for the dependent variable. This subroutine uses the library routine LLSQF and uses the function POLYX2.

The smoothing algorithm fits the surface $z=f(x,y)$ to a polynomial of the form:

$$z = \sum_{i=0}^{I} \sum_{j=0}^{L} c_{ij} x^i y^j \quad \text{where} \quad \begin{array}{l} K=\max(I,J) \\ L=\min(K-i,J) \\ I,J \text{ are selected parameters} \end{array}$$

$$= \sum_{k=1}^{M} c_k (x^i y^j)_k$$

$$\text{for } M = \begin{cases} (I+1)(J+1-I/2) & J \geq I \\ (J+1)(I+1-J/2) & I \geq J \end{cases}$$

The M terms of the polynomial are each evaluated for n=1 to N points, where $N \geq M$. This evaluation generates an N by M matrix denoted by [AM]. The AM matrix is scaled by column so that the magnitudes of the elements remain close. The scaling factor for each column is the average of the absolute values of all elements in the column. The N by 1 matrix of Z data is known. The task, then, is to solve the system

$$[AM][C] = [Z]$$

for the M by 1 matrix C of coefficients. This is accomplished by the International Mathematical and Statistical Library (IMSL) routine LLSQF, which solves the system by means of a linear least-squared error criteria. The LLSQF routine is the only Library procedure used in the contour calculation package. Installations which do not have the IMSL library available, would need to replace this function with a similar routine.

After obtaining [C] from the curve fit subroutine, the coefficients are normalized by the same scale factors originally used to condition [A']. The coefficients are then used to replace the original z data with new values acquired from evaluation of the polynomial. If the coefficients are not properly found, then no smoothing takes place and the original z data is retained.

4.1 Description of the Argument List

CALL SMSRF (X,Y,Z,ZNEW,N,I,J,NCOEF,C,IPOWR,JPOWR)

Input arguments:

X,Y,Z = arrays containing the function values for z=f(x,y)

N = the number of values stored in X,Y,Z,ZNEW

I,J = smoothing parameters selected by the user; used to define the K,L values of the polynomial described earlier

Return Arguments:

$ZNEW_n$ = array of new (smoothed) z data for n=1 to N; if the matrix computations fail, ZNEW=Z for all n

NCOEF = number of coefficients resulting from the values of I and J

C_i = array of calculated coefficients for i=1 to NCOEF

$IPOWR_i$= for the i=th term of the polynomial, the exponent of X and
$JPOWR_i$ Y respectively for i=1 to NCOEF

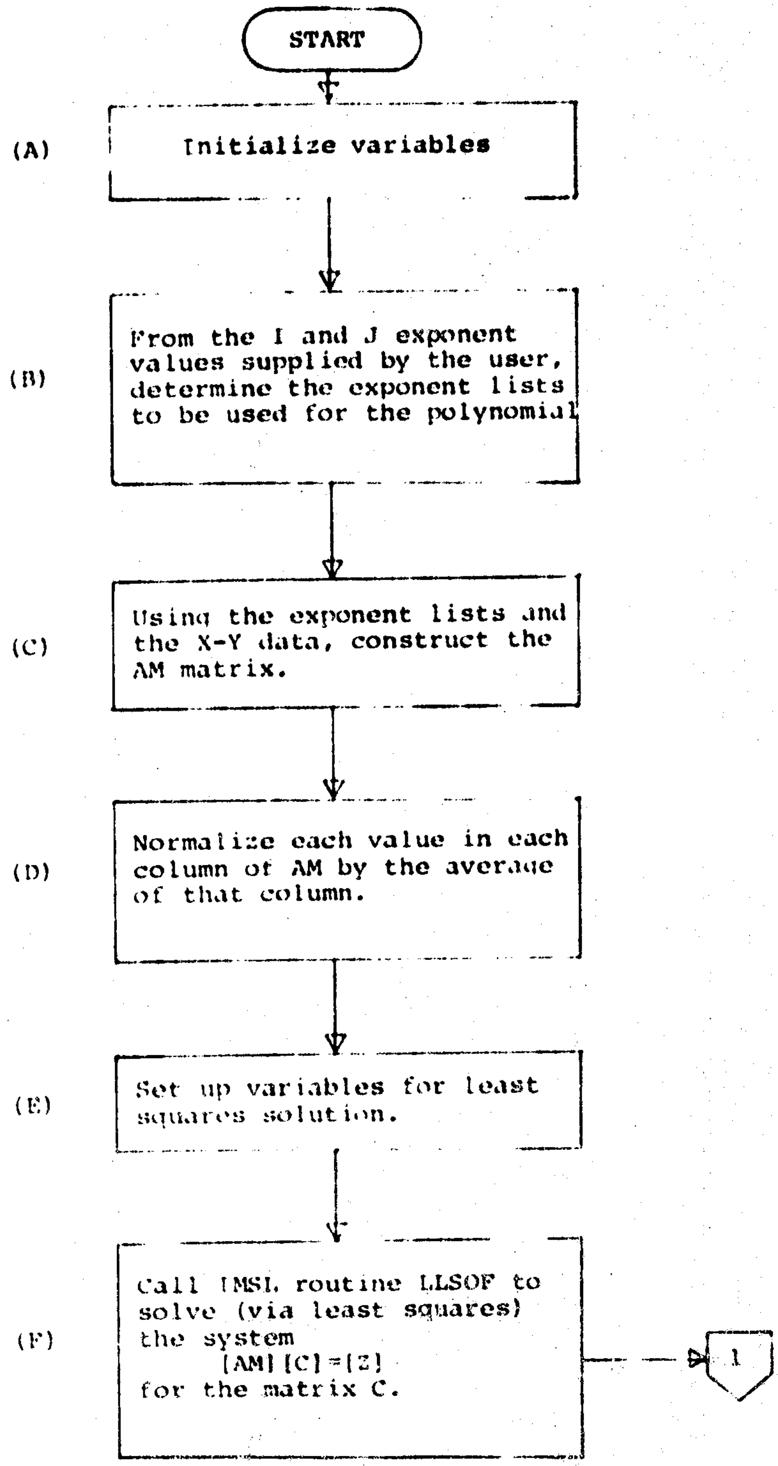
Required Dimensions:

X(N)	IPOWR(C)	XX(C)
Y(N)	JPOWR(C)	H(C)
Z(N)	C(C)	
ZNEW(N)	CNORM(C)	
B(N)	AVE(C)	
AM(C,N)	ID(C)	

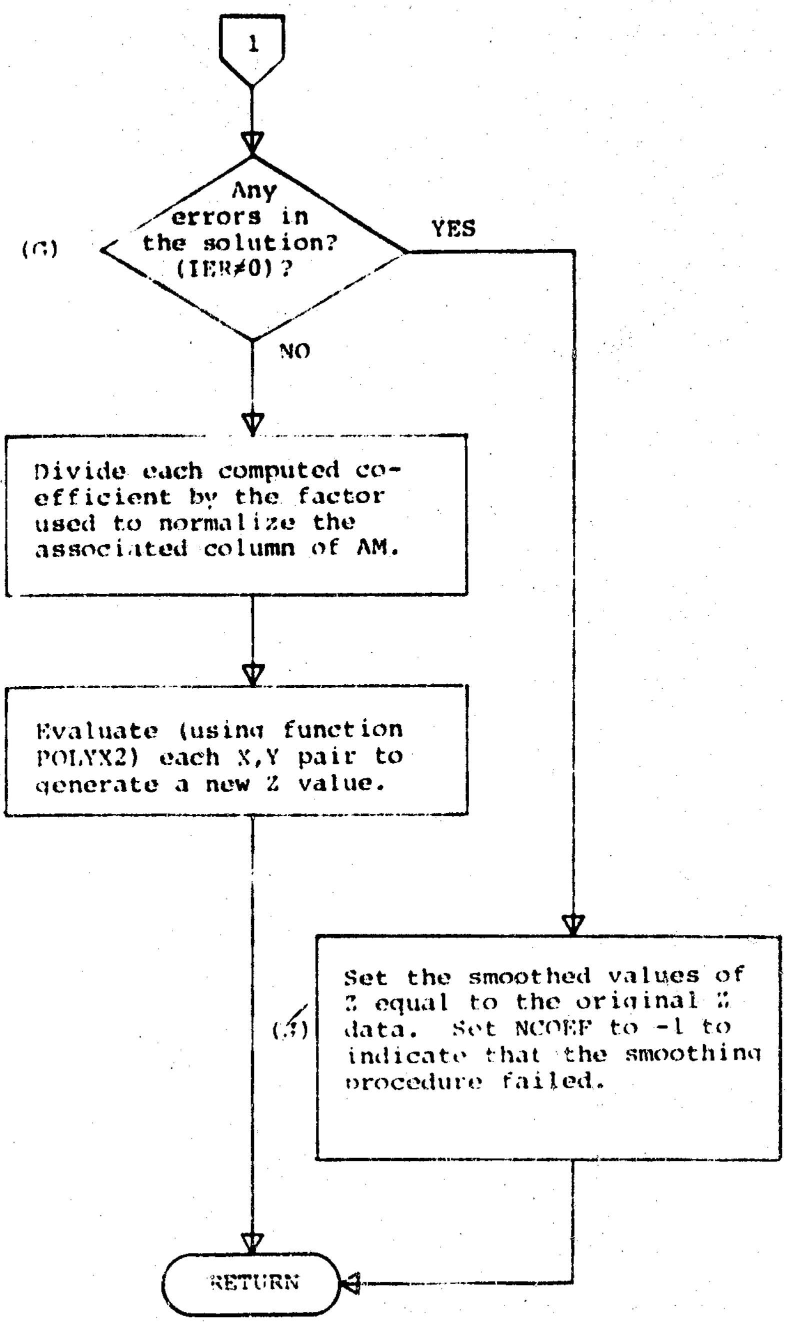
4.2 Description of Algorithm

Figures 5a and 5b present a block diagram of the module SMSRF. The functions of parts A to J are as follows.

Figure 5a. Block Diagram of SMSRF, Parts A to F



Fiqure 5b. Block Diagram of SMSRF, Parts G to J



(A) **Initialize local variables**

```
RN = FLOAT(N)
if I<1 then I=1
if J<1 then J=1
I1 = I+1
J1 = J+1
NCOEF = 0
IA = 500
```

(B) From the I and J exponent values provided by the calling program, determine the exponent lists. (IPOWR and JPOWR) to be used for the smoothing polynomial. The n-th entry in the lists is associated with the n-th term of the polynomial.

```
K = maximum of I1,J1

   for II = 1 to I1
   KI1 = K-III+1
   L = minimum of KI1, JI

         for JJ = 1 to L
         NCOEF = NCOEF + 1
         IPOWR(NCOEF) = II-1
         JPOWR(NCOEF) = JJ-1
         next JJ
   next II
```

(C) Using the exponent lists and the x-y data, construct the matrix AM.

```
for KCOL = 1 to NCOEF
IEX = IPOWR(KCOL)
JEX = JPOWR(KCOL)

      for KROW = 1 to N
      X2 = X(KROW)
      if X2 = 0.0 then X2 = 1.0
      XP = X2**IEX
      Y2 = Y(KROW)
      if Y2 = 0.0 then Y2 = 1.0
      YP = Y2**JEX
           AM(KROW,KCOL) = XP*YP
      next KROW

next KCOL
```

(D) Normalize each value in each column of AM by the average absolute value of that column. The average of column one is always one.

```
AVE(1) = 1.0
for L1 = 2 to NCOEF
AVE(L1) = 0.0
        for L2 = 1 to N
        AVE(L1) = AVE(L1) + |AM(L2,L1)|
AVE(L1) = AVE(L1)/RN, if AVE(L1) = 0, AVE(L1) = 1.0
        for L2 = 1 to N
        AM(L2,L1) = AM(L2,L1)/AVE(L1)

next L1
```

(E) Set up variables for least squares solution.

```
M=N
IER=0
KBASIS=NCOEF
TOL=0.0
for KK=1,N  B(KK)=Z(KK)
```

(F) (G) Call IMSL routine LLSQF to solve (by least squares) the system AM*C=Z for matrix C. If IER is zero on return, then the solution was found.

```
CALL LLSQF(AM,IA,M,NCOEF,B,TOL,KBASIS,XX,H,IP,IER)

if IER≠0 then goto (950)
```

(H) There were no errors. Transfer the calculated coefficients and divide out the normalization factor.

```
for L3 = 1,NCOEF
C(L3) = XX(L3)
CNORM(L3) = C(L3)/AVE(L3)
```

(I) Evaluate (using function POLYX2) each x-y pair to generate a new value for Z.

```
for L3 = 1,N
ZNEW(L3)= -POLYX2(0,X(L3),Y(L3),CNORM,IPOWR,JPOWR,NCOEF)
goto (999)
```

(J) **An error has occurred in the procedure. Set the smoothed values of Z equal to the original data Set NCOEF to negative as an error flag to be checked later.**

```
950  ZNEW(ℓ) = Z(ℓ) for ℓ = 1 to N
     NCOEF = -1
999  RETURN
```

4.3 Description of Function POLYX2

FUNCTION POLYX2 (Z,X,Y,X,IPOWR,JPOWR,N)

The polynomial evaluation function is used when the smoothing option has been invoked. X and Y are the known values of the independent variables for which the function value is required. Array C is the list of coefficients for each term of the polynomial. IPOWR and JPOWR are the exponents for each term and N is the number of terms. Z is an offset value when evaluating for a constant Z. The required dimensions are as follows:

C(N)
IPOWR(N)
JPOWR(N)

4.4 Description of Subroutine LLSQF

This is the Library routine taken from IMSL to compute the solution of a linear least squares problem. Detailed discussions of the argument list and the algorithm can be found in the second volume of the IMSL Library Reference Manual.

A summary of its use is as follows:

CALL LLSQF (A,IA,M,N,B,TOL,KBASIS,X,H,IP,IER)

Input Arguments:

A	M by N coefficient matrix. A is overwritten with information generated by LLSQF.
IA	Row dimension of matrix A as specified in the calling program.
M	Number of rows in matrices A and B.
N	Number of columns in matrix A.
B	On input, B is the right hand side of the least squares solution [A][X]=[B]. On return, B is overwritten with the residual R = B-A*X
TOL	Tolerance parameter to determine the number of columns of A to be included in the basis for the least squares fit of B. If TOL=0.0 is specified, pivoting is terminated only if the inclusion of the next column would result in a (numerically) rank deficient matrix.
KBASIS	On input, KBASIS=K implies that the first K columns of A are to be forced into the basis. Pivoting is performed on the last N-K columns of A. On output, KBASIS gives the number of columns included in the basis.

Return Arguments:

X	Solution vector of length N.
H	Work vector of length N.
IP	Work vector of length N.
IER	Error parameter =0 for normal execution =129 for M<0 or N≤0 =130 for TOL>1.0 (129 and 130 are terminal errors)

5.0 INTERPOLATION SUBROUTINE

The subroutine INTERP performs the interpolation of the data along the triangle edges. This subroutine uses the function POLYX2 if the smoothing option has been called.

The interpolation algorithm is supplied with a set of L edges (E(ℓ,1) and E(ℓ,2) for ℓ=1 to L) from the triangulation. At the endpoints of each edge the function value z_i and the independent variables x_i and y_i for i=1 to N are known. Additionally, if a function has been generated for the values of z_i (from the SMSRF subroutine), the coefficients and exponents are provided. The interpolation procedure will check each edge of the triangulation. If the constant value Z lies between the z function values at the end points, then the coordinates (ξ_J, η_J) of Z relative to the x,y coordinates of the endpoints will be calculated. ξ and η are the result of a linear interpolation if the data has not been smoothed; otherwise, the polynomial previously fitted to the surface is solved for the point.

5.1 Description of Argument List

```
CALL INTERP(X,Y,Z,ZCON,LEDGES,E,ISMOPT,LAMBDA,XI,ETA,J,C,
            IPOWR,JPOWR,NCOEF,N)
```

Input Arguments:

X_i = the X values of the function Z=f(x,y)

Y_i = the Y values of the function

Z_i = the Z values of the function

N = the range of i: the number of points in the X,Y and Z lists

ZCON = the constant value of Z for which the contour values are being interpolated

LEDGES= the number of triangle edges generated by the triangulation procedure

$E(\ell,2)$= index pointers of endpoints of each triangle edge: ℓ=1 to LEDGES

ISMOPT= smoothing option flag; 1 if SMSRF was called, Ø if not

C_k = coefficients of the polynomial terms as provided by SMSRF

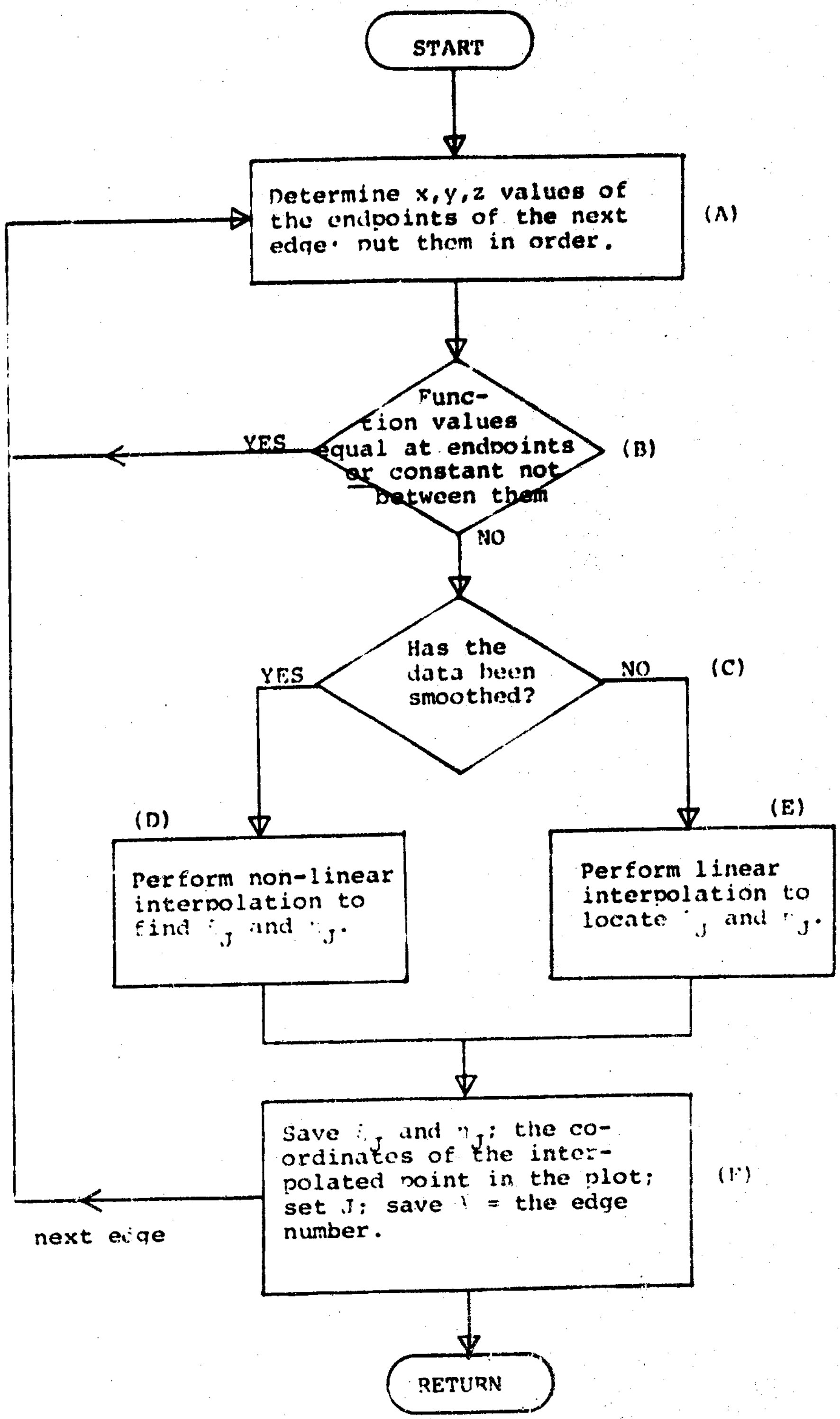
Required Dimensions:

X(N)	IE(E,2)	IPOWR(C)
Y(N)	XI(E)	JPOWR(C)
Z(N)	ETA(E)	C(C)
	LAMBDA(E)	

5.2 Description of Algorithm

Figure 6 presents a block diagram of the module INTERP. The functions of parts A to F are as follows:

Figure 6. Block Diagram of INTERP



```
J=0

for ℓ = 1 to LEDGES

   (A) determine x,y,z values of the endpoints of the next
       edge;  order them

       I1 = E(ℓ,1),   I2 = E(ℓ,2)

       X1 = X(I1),    Y1 = Y(I1),    Z1 = Z(I1)
       X2 = X(I2),    Y2 = Y(I2),    Z2 = Z(I2)

   (B) function values equal or contour value (constant) not
       between endpoints?

       if Z1 = Z2  goto (200)

       if Z1 > Z2  then  reverse X1 and X2
                                 Y1 and Y2
                                 Z1 and Z2

       if  Z1 ≥ ZCON or ZCON ≥ Z2   goto  (200)
       if  Z2 = ZCON  then Z2 = (1.00001)(ZCON)

       J = J + 1

   (C) has data been smoothed?
       if not, goto statement label 101

       if  ISMOPT = 0  goto (101)

   (D) non-linear interpolation is required
   (F) on this edge over the Z surface

       F1 = POLYX2(ZCON,X1,Y1,C,IPOWR,JPOWR,NCOEF)

          for k = 1 to 10   (.1% resolution)

          XN = (X1+X2)/2.0
          YN = (Y1+Y2)/2.0
          FN = POLYX2( CON,XN,YN,C,IPOWR,JPOWR,NCOEF)

          if  FN         = 0.0 goto  (132)

          if  sign (F1) = sign (FN)  then X1=XN, Y1=YN

          if  sign (F1) ≠ sign (FN)  then X2=XN, Y2=YN

          next k
```

```
132    XI(J)   = (X1+X2)/2.0
       ETA(J)  = (Y1+Y2)/2.0
       LAMBDA  = ℓ
       goto    200


(E)    linear interpolation is required
(F)    on this edge (no smoothing)
```

$$101 \quad X1(J) = \left(\frac{Z2-ZCON}{Z2-Z1}\right) X1 + \left(\frac{ZCON-Z1}{Z2-Z1}\right) X2$$

$$ETA(J) = \left(\frac{Z2-ZCON}{Z2-Z1}\right) Y1 + \left(\frac{ZCON-Z1}{Z2-Z1}\right) Y2$$

```
  LAMBDA(J) = ℓ

next ℓ

RETURN
```

6.0 CONTOUR SUBROUTINE

The subroutine CNTOUR draws the required contour for z=Z. This subroutine calls the user supplied program CNTCRV to draw the contour on the graphics device.

The contour algorithm makes use of the results of the triangulation and interpolation procedures in order to establish, for each contour to be drawn, the ordering of the ξ_j and η_j points (for j=1 to J). The coordinates of all interpolated points are known and the triangulation edge number associated with each coordinate pair is also known. For each edge, a list of adjacent edge numbers is provided. A contour line is constructed by choosing a boundary edge as a starting point (if any) for which an interpolated point exists. Then, the remaining points on the contour are ordered by means of searching adjacent edges for interpolated points, until another boundary edge is encountered. For closed contours, the iteration ends if the list of common edges ends. Then a graphics subroutine is called to draw the curve and perform any other user supplied application (for example, label the curve). The contour algorithm then continues to the next curve, if there are any points remaining. This process continues until all contours are drawn and the list of ξ and η coordinates is exhausted.

6.1 Description of the Argument List

CALL CNTOUR (ZCON,XI,ETA,LAMBDA,J,IBE,ITE)

Input Arguments:

ZCON	=	the constant value of Z for which the contours are being drawn
XIj	=	the x-coordinate of the interpolated point on the edge $E(\ell)$, ℓ = LAMBDA(j)
ETAj	=	the y-coordinate of the interpolated point on the edge $E(\ell)$, ℓ = LAMBDA(j)
LAMBDAj	=	the index number of each edge associated with XI and ETA values
J	=	the range of j; the number of interpolated points found for ZCON by the interpolation procedure
IBE(ℓ)	=	1 if the ℓ-th edge is a boundary edge; otherwise zero
ITE(ℓ,4)	=	indices of adjacent edges for the ℓ-th edge

Required Dimensions:

XI(E)
ETA(E)
LAMBDA(E)
IBE(E)
ITE(E,4)

6.2 Description of Algorithm

Figures 7a and 7c present a block diagram of the module CONTOUR. The functions for parts A to P are as follows.

Figure 7a. Block Diagram of CONTOUR, Parts A to G

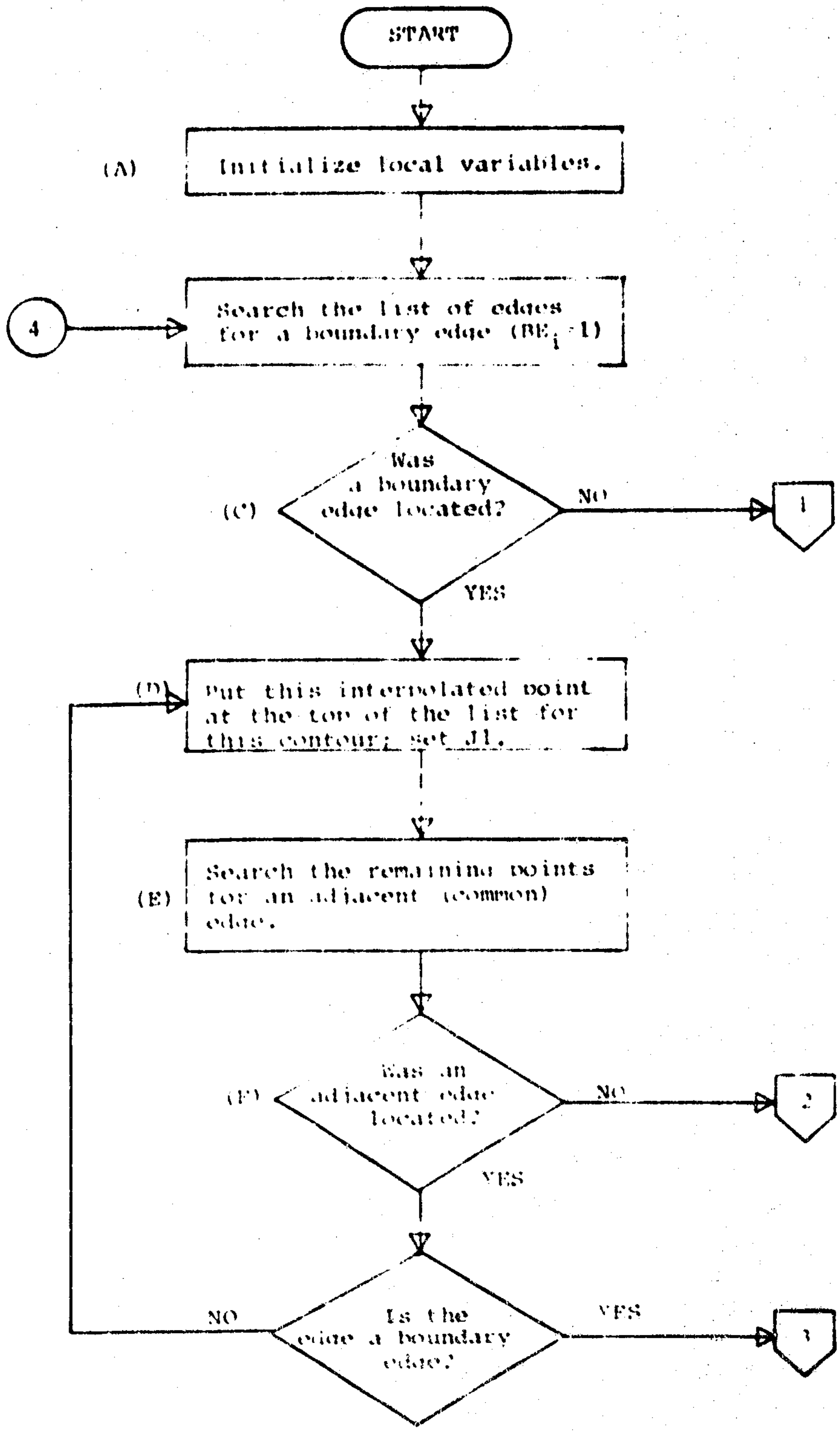


Figure 7b. Block Diagram of CONTCUR, Parts H to M

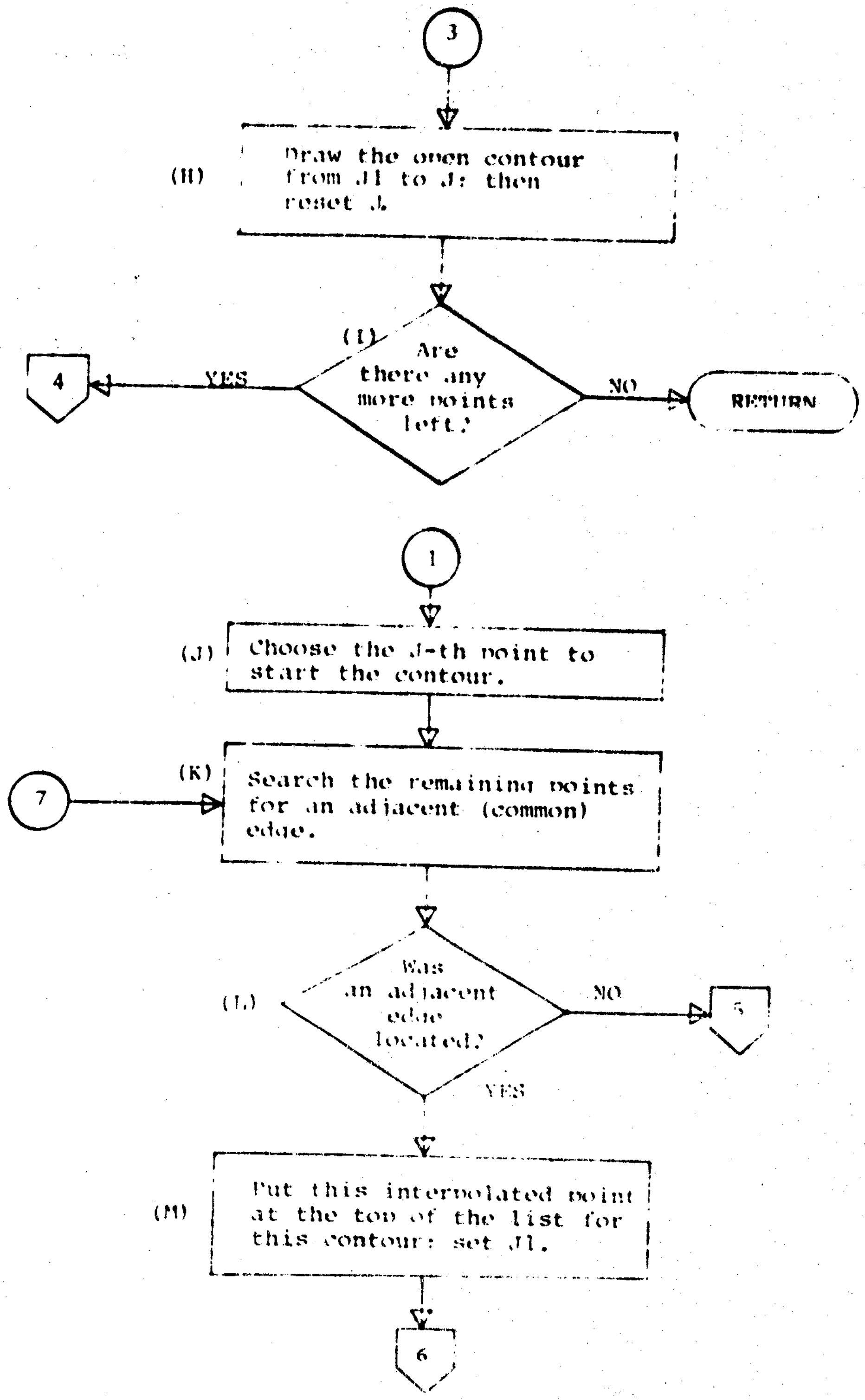
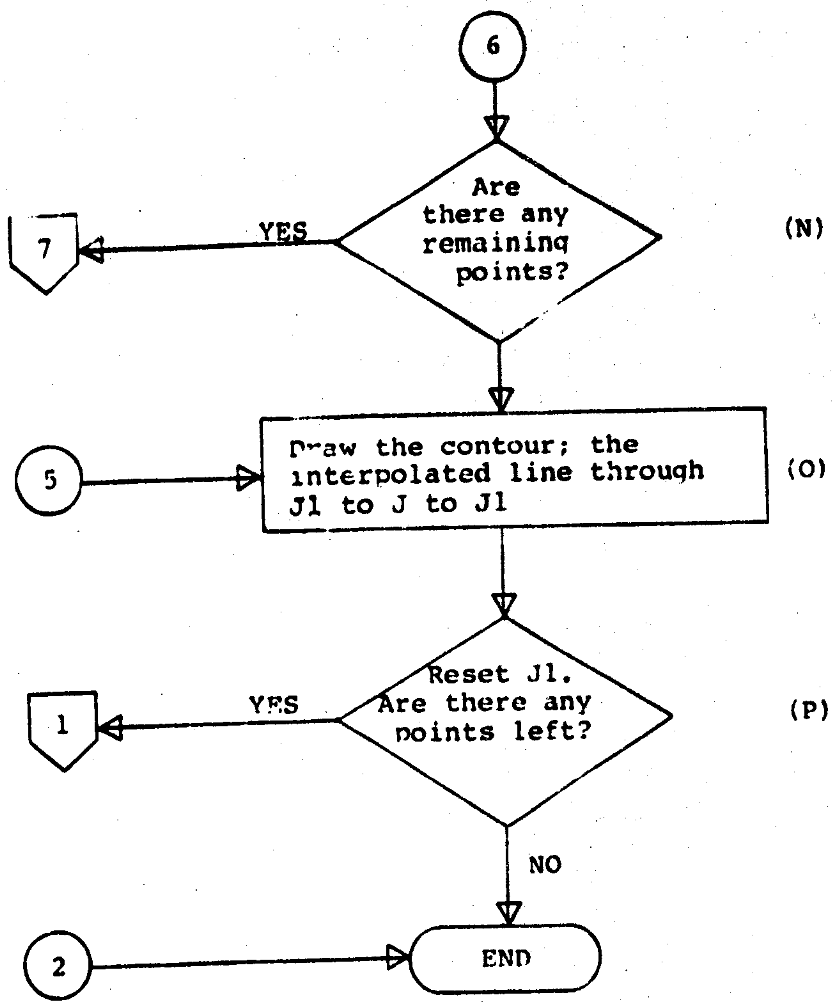


Figure 7c. Block Diagram of CONTOUR, Parts N to P



(A) Initialize local variable(s)

```
(10)    J1 = 0
```

(B) Search the list of edges for a
(C) boundary edge. If none is found, go to the procedure for closed contours.

```
(1)     J1 = J1 + 1
        L1 = LAMBDA(J1)

                if BE(L1) = 1 goto (2)
                if J1 < J goto (1)
                goto (11)
```

(D) Put this point at the top of the list and reset J1.

```
(2)     if J1 = J goto (3)

                XI(J+1)      = XI(J)
                ETA(J+1)     = ETA(J)
                LAMBDA(J+1)  = LAMBDA(J)

               [for JC = 1,J
               [XI(JC)       = XI(JC+1)
               [ETA(JC)      = ETA(JC+1)
               [LAMBDA(JC)   = LAMBDA(JC+1)

(3)     JB = J
         L = L1
```

(E) Search the remaining points for an adjacent (common) edge.

```
(6)     JB = JB-1

          J1 = 0
     (5)  J1 = J1 + 1
          L1 = LAMBDA(J1)

          if L1 = TE (L,i) for i = 1 to 4 goto (4)
          if J1   JB goto (5)
```

(F) An error has occurred. There is no next point.

```
          goto (800)
```

(G) Put this point at the top of the list. Continue if it's not a boundary edge.

```
(4)     XI(J+1)      = XI(J1)
        ETA(J+1)     = ETA(J1)
        LAMBDA(J+1)  = LAMBDA(J1)

        [for JJ=J1 to J
        [XI(JJ)       = XI(JJ+1)
        [ETA(JJ)      = ETA(JJ+1)
        [LAMBDA(JJ)   = LAMBDA(JJ+1)

        L = L1
        if BE(L1) ≠ 1 goto (6)
```

(H) Draw the open contour from J1 to J, then reset J.

```
NPOINT = J-JB+1
if NPOINT<1 goto (300)

Call CNTCRV (XI(JB),ETA(JB),NPOINT,ZCON)
```

(I) Are there any more points left?

```
(300)   J = JB-1
        if J<0 goto (10)
        If J=0 goto (300)
```

(J) Now draw internal lines (closed contours not starting or stopping at boundary edges). The point at JC = J in the list is chosen to start the contour.

```
(11)    JB = J+1
```

(K) (M) (P) Find the next point (on the edge with a common end point); put it at the top of the list; repeat until no more edges are left.

```
        L = LAMBDA (J)

(16)    JB = JB-1
            J1 = 0, if JB>J then J1 = 1
        (15) J1 = J1+1
            L1 = LAMBDA (J1)
            if L1 = TE (L,i) for i = 1 to 4, goto (14)
            If J1<JB goto (15)
```

(L) Otherwise, no adjacent edge was found; this contour line is complete; draw it.
go to (17)

```
(14)  XI(J+1)       = XI(J1)
      ETA(J+1)      = ETA(J1)
      LAMBDA(J+1)   = LAMBDA(J1)

      [for JJ=J1 to J
      [XI(JJ)       = XI(JJ+1)
      [ETA(JJ)      = ETA(JJ+1)
      [LAMBDA(JJ)   = LAMBDA(JJ+1)

      L = L1
      if JB ≠ 1 goto (16)

(17)  JJ + JB
      if JB ≠1 then JJ = JB+1

(O)   Draw the closed contour - the interpolated
      line through the points JJ to J to JJ

      KNT = 0

      [for KK + JJ to J
      [KNT = KNT+1
      [XX(KNT) = XI(KK)
      [YY(KNT) = ETA(KK)

      NPOINT = KNT+1
      XX(NPOINT) = XX(1)
      YY(NPOINT) = YY(1)

      Call CNTCRV (XX,YY,NPOINT,ZCON)

(P)   Reset J.  Establish next contour lines for
      remaining points or quit if J = 0.

      J = JB-1
      if J>0 goto (11)
(800) RETURN
```

6.3 Description of Subroutine CNTCRV

This module is supplied by the user and performs the graphical presentation of the contour to the device being used. Note that CNTOUR may call this routine several times for each constant value of Z_c, and a new contour line is provided with each call.

The argument list consists of the following items:

CALL CNTCRV (XX,YY,NPOINT,ZCON)

XX = (dimension NPOINT) is the array of X coordinates for each point on the contour

YY = (dimension NPOINT) is the array of Y coordinates for each point on the contour

NPOINT = is the number of values provided in the x,y coordinate lists

ZCON = is the constant value of Z associated with the provided contour line.

7.0 PROGRAMMING CONSIDERATIONS

The programs described in this document have been implemented in FORTRAN on both an IBM 360/67 (under TSS) and a CDC 7600 (under SCOPE). The subprogram packages were coded in such a way that as many machine dependent FORTRAN statements as possible were eliminated. In fact, the programs appear to be completely portable except for (1) the use of IMSL routine LLSQF in SMSRF would need to be replaced at installations where IMSL is not available and (2) the IBM version uses double precision statements in TRIANG that may need modification or deletion.

The execution time for the contour calculations increases with the number of points being processed. The following table illustrates typical execution times encountered on a CDC 7600. The test cases for this table all made use of the smoothing option (with parameters I and J both equal to 2), and were contrived so that three contour lines were generated, each consisting of about N/10 interpolated points. The N data points were generated at random for these tests.

N = Number of data points	CDC 7600 Execution time (CPU seconds)
50	0.20
100	0.47
150	0.91
200	1.52
300	2.66
400	4.53
500	6.95

So the execution time is approximately 1.5*(N/200)**1.67 seconds.

The algorithms require internal work areas that are used to store intermediate calculations during execution. The work areas required by the triangulation and smoothing procedures are the greatest contributing factors to the size of the total object time package. The amount of storage required by the triangulation is proportional to the number of data points to be processed, and is approximately equal to 30N. The amount of storage required by the least-squares curve fitting procedure for smooth data is proportional to both the value of N and the maximum number of coefficients to be computed (C), and is approximately equal to C(N+7)+N. The total work area required by all the routines is proportional to both C and N, and is approximately N(C+42).

For some applications, users may wish to reduce the program size. One method, already mentioned, is to eliminate the smoothing subroutines if linear interpolation is adequate for the data. Size reduction can also be accomplished by decreasing array dimensions to accommodate only the maximum number of points and coefficients to be processed. Conversely, the array dimensions can be enlarged to handle more points and/or coefficients if program size is not an imposing consideration.

Table 1 itemizes all array dimensions which may be given new dimensions for the purpose of increasing or decreasing program size as needed. For this table:

N = Number of data points to process

C = Number of coefficients to use during smoothing the Z data

E = 3N-6 = the maximum number of triangle edges which can result from the triangulation of N points

T = 2N-5 = the maximum number of triangles which can result from the triangulation of N points.

Table 1. Array Dimensions

Array Name(s)	Required Dimension	Appears in the Following Modules
ZNEW	(N)	CNTLNS
IE	(E,2)	CNTLNS, INTERP
IBE	(E)	CNTLNS, CNTOUR
ITE	(E,4)	CNTLNS, CNTOUR
XI, ETA, LAMBDA	(E)	CNTLNS, INTERP, CNTOUR
IP, XX, H	(C)	SMSRF
B	(N)	SMSRF
AM	(N,C)	SMSRF
IPOWR, JPOWR	(C)	SMSRF, POLYX2, CNTLNS, INTERP
C, CNORM	(C)	SMSRF, POLYX2, CNTLNS
XX, YY	(E)	CNTOUR
P, B, X, Y	(N)	TRIANG
E	(E,2)	TRIANG
BE	(E)	TRIANG
TE	(E,4)	TRIANG
T	(T,3)	TRIANG

Table 2 itemizes local variables that are initialized by means of data statements. These data values should be given new data assignments if any array dimensions are respecified.

Table 2. Internal Parameter Values

Data Statement Variable	Required Value	Module Name
IA	N	SMSRF
MAXCOF	C	CNTLNS
MAXPTS	N	CNTLNS

As a final note, it should be pointed out that for some applications the x and y coordinate values may be used repeatedly and only the values of Z will change. For such cases, the x-y plane triangulation is valid for each call after the first since the triangulation is not based on the Z data. Since the triangulation can be performed once and then saved, the master programs can be easily modified to bypass triangulation of the x-y data by inserting an extra parameter in the CNTLNS argument list. Such a scheme would result in a considerable savings in execution time.

The subroutine modules described in this report are listed in the Appendix.

APPENDIX

PROGRAM LISTINGS

```
      SUBROUTINE CNTLNS (X,Y,Z,N,ISMOPT,IEXP,JEXP,NCNTRS,CLIST,
     *                   EPSLON,IERR)
C
C     ------------------------------------------------------------------
C
C     DRIVER PROGRAM FOR COMPUTING AND DRAWING CONTOUR LINES OF
C     CONSTANT Z FOR THE FUNCTION. Z = F(X,Y).
C
C
C     ARGUMENT LIST DEFINITIONS -
C
C        X         = INPUT LIST OF X VALUES
C        Y         = INPUT LIST OF Y VALUES
C        Z         = INPUT LIST OF Z VALUES
C        N         = INPUT SPECIYING THE NUMBER OF VALUES IN X,Y AND Z
C        ISMOPT    = SMOOTHING OPTION FLAG (0=NO/OFF, 1=YES/ON)
C        IEXP      = I EXPONENT VALUE FOR SMOOTHING
C        JEXP      = J EXPONENT VALUE FOR SMOOTHING
C        NCNTRS    = NUMBER OF CONTOUR LINES TO BE DRAWN
C                    (SELF COMPUTING IF NCNTRS.LE.0)
C        CLIST     = LIST OF CONSTANT CONTOUR VALUES IF NCNTRS.GT.0
C        EPSLON    = ERROR FUNCTION (NORMALIZED VALUE) RETURNED TO
C                    CALLER IF ISMOPT IS NON-ZERO
C        IERR      = RETURN ERROR FLAG
C                    = 0 FOR NORMAL RETURN
C                    = 1 FOR INVALID VALUE FOR N
C                    = 2 FOR NUMBER OF ISMOPT COEFFICIENTS GREATER
C                         THAN 'MAXCOF' OR N
C
C        (NOTE / IF NCNTRS.LE.0, THEN CLIST(1) = BASE VALUE,
C         AND CLIST(2) = INCREMENT VALUE (DELTA) )
C
C
C     ------------------------------------------------------------------
C
C
C
      DIMENSION  X(N),Y(N),Z(N),CLIST(2)
      DIMENSION  ZNEW(500)
```

```
      DIMENSION  IE(1494,2),ITE(1494,4),XI(1494),ETA(1494),
     *           LAMBDA(1494),IBE(1494)
      DIMENSION  IPOWR(23),JPOWR(23),COEF(23)
C
      DATA       MAXCOF /23/
      DATA       MAXPTS /500/
C
C
C        (A)
C        INITIALIZE LOCAL VARIABLES
C        AND CHECK INPUTS FOR ERRORS
C
      IERR = 0
      EPSLON = 0.0
      IF (N.LT.3.OR.N.GT.MAXPTS) GOTO 997
C
C
C        (B)
C        CALL SUBROUTINE TRAING TO TRIANGULATE X-Y DATA POINTS
C
      CALL TRIANG (X,Y,N,LEDGES,IE,IBE,ITE)
C
C        (C)
C        SMOOTHING REQUIRED? . .
C
      IF (ISMOPT.EQ.0) GOTO 110
C
C
C        (D)
C        CHECK REQUESTED EXPONENT VALUES FOR ERRORS
C
      I1 = IEXP+1
      J1 = JEXP+1
      NMIN = MIN0(I1,J1)
      NMAX = MAX0(I1,J1)
      IF (J1.GE.I1) NC = (IEXP+1)*(JEXP+1-IEXP/2)
      IF (J1.LT.I1) NC = (JEXP+1)*(IEXP+1-JEXP/2)
      IF (NC.GT.N.OR.NC.GT.MAXCOF) GOTO 998
```

```
          DO 125 K=1,MAXCOF
          IPOWR(K) = 0
      125 JPOWR(K) = 0
C
C             (E)
C             CALL SUBROUTINE SMSRF TO SMOOTH THE DATA Z=F(X,Y)
C
          CALL SMSRF (X,Y,Z,ZNEW,N,IEXP,JEXP,NCOEF,COEF,IPOWR,JPOWR)
          IF (NCOEF.LT.0) GOTO 120
             DO 130 K=1,N
             EPSLON = EPSLON + (Z(K)-ZNEW(K))**2
      130    CONTINUE
             EPSLON = SQRT(EPSLON)/FLOAT(N)
          GOTO 120
      110 DO 100 K=1,N
      100 ZNEW(K) = Z(K)
C
C
C             (F)
C             DETERMINE THE RANGE OF THE Z DATA UNDER CONSIDERATION
C
      120 ZMIN = Z(1)
          ZMAX = ZMIN
             DO 50 K=2,N
             ZMIN = AMIN1(ZMIN,Z(K))
             ZMAX = AMAX1(ZMAX,Z(K))
       50    CONTINUE
C
C
C             (G,M)
C             HAS A CONTOUR LIST BEEN GIVEN? . .
C
          FN = 1.0
          FK = -1.0
      200 IF (NCNTRS.GT.0) GOTO 180
C
C
C             (H)
```

```
C        CALL SUBROUTINE CBVCHK TO VERIFY THAT THE SPECIFIED BASE
C          VALUE IS WITHIN RANGE OF DATA, RESET IF NEEDED
C
      CALL CBVCHK (CLIST(1),CLIST(2),ZMIN,ZMAX,CLNEW)
      IF (CLIST(1).NE.CLNEW) CLIST(1)=CLNEW
C
C          (I)
C          DETERMINE (NEXT) CONTOUR CONSTANT VALUE
C
  210 FK = FK+1.0
      ZCON = FK*FN*CLIST(2) + CLIST(1)
      IF (ZCON.GT.ZMIN.AND.ZCON.LT.ZMAX) GOTO 150
      IF (FN.LT.0.) GOTO 300
      FK = 0.0
      FN = -1.0
      GOTO 210
C
  180 K = K+1
      IF (K.GT.NCNTRS) GOTO 300
      ZCON = CLIST(K)
      IF (ZCON.LT.ZMIN.OR.ZCON.GT.ZMAX) GOTO 200
C
C          (J)
C          CALL SUBROUTINE INTERP TO
C          INTERPOLATE FOR CONTOUR LINE DATA POINTS
C
  150 CALL INTERP (X,Y,ZNEW,N,ZCON,LEDGES,IE,ISMOPT,LAMBDA,
     *                 XI,ETA,J,COEF,IPOWR,JPOWR,NCOEF)
C
C          (K,L)
C          ANY DATA POINTS FOUND? . .
C          CALL SUBROUTINE CNTOUR TO SORT THE INTERPOLATED POINTS
C          ON THE CONTOUR LINE AND DRAW IT
C
      IF (J.NE.0) CALL CNTOUR (ZCON,XI,ETA,LAMBDA,J,IBE,ITE)
      GOTO 200
C
  300 RETURN
```

```
  997 IERR = 1
      RETURN
  998 IERR = 2
      RETURN
      END
```

SMSRF*F ,11/05/80 09:29:41

```
      SUBROUTINE SMSRF (X,Y,Z,ZNEW,N,I,J,NCOEF,CNORM,IPOWR,JPOWR)
C
C     ------------------------------------------------------------------
C
C     SUBROUTINE SMSRF PERFORMS THE OPTIONAL SMOOTHING OF DATA BEFORE
C     TRIANGULATION OF THE PLANE IS INITIATED.  THE SURFACE DEFINED BY
C     Z = F(X,Y) IS SMOOTHED VIA A POLYNOMIAL CURVE FIT DEFINED BY A
C     LEAST SQUARES CRITERIA.
C
C
C     ARGUMENTS --
C     (INPUT)
C        X,Y,Z       ARRAYS OF VALUES DEFINING THE KNOWN SURFACE
C                    (POINTS IN SPACE FOR THE FUNCTION Z=F(X,Y))
C        N           THE NUMBER OF POINTS IN X,Y AND Z.
C        I,J         ARE THE EXPONENTS FOR THE SMOOTHING POLYNOMIAL
C                    AS SELECTED BY THE USER.
C     (RETURN)
C        ZNEW        IS THE ARRAY OF SMOOTHED VALUES FOR THE FUNCTION
C                    (ZNEW WILL CONTAIN THE ORIGINAL Z DATA ON RETURN
C                    IF THE SMOOTHING OPERATION FAILS, IN WHICH CASE
C                    NCOEF WILL BE SET TO -1).
C        NCOEF       IS THE NUMBER OF TERMS IN THE POLYNOMIAL RESULTING
C                    FROM THE VALUES OF I AND J.  NCOEF MUST BE LESS THAN
C                    OR EQUAL TO BOTH N AND MAXCOF.
C        C           IS THE ARRAY OF NCOEF COMPUTED COEFFICIENTS
C        IPOWR       THE ARRAY OF I EXPONENTS FOR EACH TERM
C        JPOWR       THE ARRAY OF J EXPONENTS FOR EACH TERM
C                    (EACH ELEMENT OF C, IPOWR AND JPOWR IS ASSOCIATED
C                    WITH THE NCOEF TERMS OF THE POLYNOMIAL, IN ORDER)
C
C     ------------------------------------------------------------------
C
C
C
      DIMENSION  X(N),Y(N),Z(N),ZNEW(N)
      DIMENSION  IPOWR(23),JPOWR(23),C(23),CNORM(23),AVE(23)
      DIMENSION  IP(23),XX(23),H(23)
```

```
      DIMENSION  B(500),AM(500,23)
C
      DATA       IA /500/
C
C
C        (A)
C        INITIALIZE LOCAL VARIABLES AND RANGE CHECK
C
      REALN = FLOAT(N)
      IF(I.LT.1) I = 1
      IF(J.LT.1) J = 1
      I1 = I+1
      J1 = J+1
      NCOEF = 0
C
C
C        (B)
C        DETERMINE THE X AND Y EXPONENTS TO BE USED
C        SAVE THEM IN ARRAYS IPOWR AND JPOWR
C
      NCOEF = 0
      K = MAX0(I1,J1)
      IF (K.EQ.0) GOTO 950
         DO 180 II=1,I1
         KI1 = K-II+1
         L = MIN0(KI1,J1)
            DO 181 JJ=1,L
            NCOEF = NCOEF+1
            IPOWR(NCOEF) = II-1
            JPOWR(NCOEF) = JJ-1
  181       CONTINUE
  180    CONTINUE
C
C
C        (C)
C        USING THE EXPONENT LISTS FROM ABOVE AND THE
C        KNOWN XY DATA POINTS, CONSTRUCT THE MATRIX AM
C
```

```
      DO 182 KCOL=1,NCOEF
      IEX = IPOWR(KCOL)
      JEX = JPOWR(KCOL)
          DO 284 KROW=1,N
          X2 = X(KROW)
          IF (X2.EQ.0.0) X2=1.0
          XP = X2**IEX
          Y2 = Y(KROW)
          IF (Y2.EQ.0.0) Y2=1.0
          YP = Y2**JEX
              AM(KROW,KCOL) = XP*YP
  284     CONTINUE
  182 CONTINUE
      KROW = NCOEF
C
C
C        (D)
C        NORMALIZE EACH VALUE IN EACH COLUMN OF AM BY THE COLUMN AVERAGE
C
      AVE(1) = 1.0
      DO 403 L1 = 2,NCOEF
      AVE(L1) = 0.0
      DO 402 L2 = 1,N
  402 AVE(L1) = AVE(L1) +  ABS(AM(L2,L1))
      AVE(L1) = AVE(L1)/REALN
      IF (AVE(L1) .EQ. 0.)  AVE(L1) = 1.0
      DO 404 L2 = 1,N
  404 AM(L2,L1) = AM(L2,L1)/AVE(L1)
  403 CONTINUE
C
C
C
C        (E,F,G)
C        USE IMSL ROUTINE LLSQF TO SOLVE (VIA LEAST-SQUARES)
C        THE SYSTEM  AM*C = Z  FOR MATRIX C
C
C
      M = N
```

```
      IER = 0
      KBASIS = NCOEF
      TOL = 0.0
      DO 222 KK=1,N
      B(KK) = Z(KK)
  222 CONTINUE
      CALL LLSQF (AM,IA,M,NCOEF,B,TOL,KBASIS,XX,H,IP,IER)
      IF (IER.NE.0) GOTO 950
C
C
C        (H)
C        DIVIDE OUT THE SCALE FACTOR FROM THE SOLUTION
C        MATRIX AND ESTABLISH THE COEFFICIENTS
C
      DO 905 L3 = 1,NCOEF
      C(L3) = XX(L3)
      CNORM(L3) = C(L3)/AVE(L3)
  905 CONTINUE
C
C
C        (I)
C        ESTABLISH THE NEW Z VALUES BY
C        EVALUATING THE POLYNOMIAL FOR EACH KNOWN X-Y PAIR
C
      DO 934 L3=1,N
      ZNEW(L3) = -1.0*POLYX2(0.0,X(L3),Y(L3),CNORM,IPOWR,JPOWR,NCOEF)
  934 CONTINUE
      RETURN
C
C
C
C        (J)
C        ERROR RETURN,  SET NCOEF TO -1 AND
C        SEND BACK OLD Z VALUES TO CALLING PROGRAM
C
  950 DO 960 L1=1,N
  960 ZNEW(L1) = Z(L1)
      NCOEF = -1
```

SMSPF>> ,11/05/80 09:29:41

```
      RETURN
C
C
      END
```

TRIANG$$,11/05/80 09:30:01

```
      SUBROUTINE TRIANG (XD,YD,N,L,E,BE,TE)
C
C     ------------------------------------------------------------------
C
C        A SET OF N DATA POINTS ARE KNOWN (X(I),Y(I),I=1,N)  THEY ARE TO
C        BE CONNECTED BY LINES TO FORM A SET OF TRIANGLES (FOR N.LE.
C        MAXPTS).  THE FINAL TRIANGULATION ESTABLISHES A CONVEX POLYGON
C        DEFINED BY LINKED LISTS OF EDGE NUMBERS, END POINTS AND
C        BOUNDARY EDGES.
C
C
C
C     SUBROUTINE INPUT
C       XD    = ARRAY OF ABSCISSAS
C       YD    = ARRAY OF ORDINATES
C       N    = NUMBER OF POINTS IN X AND Y
C
C     SUBROUTINE OUTPUT
C       L    = NUMBER OF EDGES LISTED IN E, BE AND TE
C       E    = LIST OF INDICES OF EACH TRAINGLE EDGE
C       BE   = 1 IF I OF E IS A BOUNDARY EDGE
C       TE   = INDICES OF NEIGHBORING EDGES FOR EACH TRAINGLE
C
C     LOCAL VARIABLES
C       P    = INDICES OF POINTS OUTSIDE THE BOUNDARY
C       J    = NO. OF VALUES IN LIST P
C       B    = INDEX OF POINTS ON THE BOUNDARY ..  INORDER
C       K    = NO. OF POINTS LISTED IN ARRAY B
C       T    = INDICES OF ADJACENT TRIANGLE EDGES
C       M    = NO. OF ROWS USED IN ARRAY T
C       X    = ARRAY OF SCALED X DATA
C       Y    = ARRAY OF SCALED Y DATA
C
C     ------------------------------------------------------------------
C
C
      IMPLICIT INTEGER (P,B)
      INTEGER  T,TE,E
```

```
C
      DIMENSION  XD(N),YD(N),X(500),Y(500)
      DIMENSION  P(500),B(500)
      DIMENSION  E(1494,2),BE(1494),TE(1494,4)
      DIMENSION  T(995,3)
C
C     ..DOUBLE PRECISION SPECIFICATION STATEMENTS FOR IBM360
      REAL*8     TERM,DCOMP,D,D1,S,TC
      REAL*8     XP1,X21,YP1,Y21,XP2,X12,YP2,Y12,X1P,Y1P,X2P,Y2P
C
C
C        (A)
C        THE PROCEDURE BEGINS WITH NO BOUNDARY, NO EDGES, AND
C        ALL X-Y DATA POINTS UNDER CONSIDERATION
C        SCALE THE X,Y DATA AND INITIALIZE LOCAL VARIABLES.
C
C
      J = N
      K = 0
      L = 0
      M = 0
      KKNT = 0
      DO 100 JCNT=1,J
  100 P(JCNT) = JCNT
      XMAX = XD(1)
      XMIN = XD(1)
      YMAX = YD(1)
      YMIN = YD(1)
         DO 98 K=2,N
         XMAX = AMAX1(XMAX,XD(K))
         XMIN = AMIN1(XMIN,XD(K))
         YMAX = AMAX1(YMAX,YD(K))
         YMIN = AMIN1(YMIN,YD(K))
   98    CONTINUE
      DLXINV = 1.0/(XMAX-XMIN)
      DLYINV = 1.0/(YMAX-YMIN)
         DO 99 K=1,N
         X(K) = XD(K)*DLXINV
```

```
            Y(K) = YD(K)*DLYINV
      99    CONTINUE
C
C
C
C           (B)
C           BEGIN BY TAKING THE LAST PAIR OF POINTS (X,Y(J)) IN THE
C           LIST TO BE THE FIRST BOUNDARY POINT
C
         B(1) = J
         J = J-1
C
C
C
C           (C)
C           FROM THE REAMINING POINTS, FIND THE POINT NEAREST THE FIRST
C
C
         I2 = 1
         I1 = B(1)
         DMIN = (X(I1)-X(1))**2 + (Y(I1)-Y(1))**2
         DO 270 J1=2,J
         DST = (X(I1)-X(J1))**2 + (Y(I1)-Y(J1))**2
         IF (DST.GE.DMIN) GOTO 270
         I2 = J1
         DMIN = DST
     270 CONTINUE
C
C           (D)
C           NOW B(1) TO B(I2) IS THE FIRST EDGE.
C           THERE IS ONE EDGE AND TWO BOUNDARY POINTS.
C
         J = J-1
         IF (I2.GT.J) GOTO 275
         DO 274 JCNT=I2,J
         P(JCNT) = P(JCNT+1)
     274 CONTINUE
     275 K = 2
```

```
      B(2) = 12
      L = 1
      E(1,1) = MIN0(B(1),B(2))
      E(1,2) = MAX0(B(1),B(2))
C
C
C
C         (E)
C         NOW BEGIN CIRCLING AROUND THE BOUNDARY OF THE POLYGON,
C         CONSIDERING, IN ORDER, EACH BOUNDARY EDGE.  MAINTAIN THE
C         FOLLOWING INDICES -
C           K1 = B ARRAY INDEX OF THE CURRENT EDGE - POINT 1
C           K2 = B ARRAY INDEX OF THE CURRENT EDGE - POINT 2
C           B1,B2 = INDICES OF BOUNDARY POINT COORDINATES
C
      K1 = 0
      KT = 0
   11 K1 = K1+1
      IF (K1.GT.K) K1=1
   12 K2 = K1+1
      IF (K2.GT.K) K2=1
      B1 = B(K1)
      B2 = B(K2)
      KT = KT+1
C
C         (F)
C         CONSIDER THE BOUNDARY EDGE FROM B1 TO B2.  FOR ALL POINTS NOT
C         YET TRIANGULATED (THE J POINTS REMAINING IN P), FIND THE
C         POINT THAT, WHEN TRAINGULATED WITH B1,B2, MINIMIZES THE LENGTH
C         OF THE TWO NEW EDGES TO BE DRAWN.
C
      D1 = 0.
      J1 = 0
      BFLAG = 0
      IF (J.EQ.0) GOTO 6
      DO 1 LJ=1,J
      PJ = P(LJ)
      TERM = (Y(PJ)-Y(B1))*(X(B2)-X(B1))-(X(PJ)-X(B1))*(Y(B2)-Y(B1))
```

```
          IF (TERM.LE.0.) GOTO 1
          D = SQRT((X(PJ)-X(B1))**2+(Y(PJ)-Y(B1))**2)
         2   +SQRT((X(PJ)-X(B2))**2+(Y(PJ)-Y(B2))**2)
          IF (J1.NE.0.AND.D1.LT.D) GOTO 1
          J1 = LJ
          D1 = D
        1 CONTINUE
C
C           (G)
C           IF LESS THAN THREE EDGES EXIST (NO TRIANGLE DEFINED YET),
C           THEN THERE ARE NO ADJACENT BOUNDARY POINTS TO BE CONSIDERED.
C           SO GO TO SECTION J.
C
          IF (K.LE.3) GOTO 3
C
C
C           (H)
C           CONSIDER THE ADJACENT BOUNDARY POINT OF THE NEXT EDGE OF THE
C           POLYGON.  CALL ITS INDEX NUMBER K3 AND SEE IF ITS CLOSER TO
C           THE CURRENT EDGE THAN P(J1).
C
        6 K3 = K2+1
          IF (K3.GT.K) K3=1
          PK3 = B(K3)
          TERM = (Y(PK3)-Y(B1))*(X(B2)-X(B1))-(X(PK3)-X(B1))*(Y(B2)-Y(B1))
          IF (TERM.LE.0.) GOTO 2
          D = SQRT((X(PK3)-X(B1))**2+(Y(PK3)-Y(B1))**2)
         2   +SQRT((X(PK3)-X(B2))**2+(Y(PK3)-Y(B2))**2)
          IF (J1.NE.0.AND.D1.LT.D) GOTO 2
          J1 = K3
          D1 = D
          BFLAG = 1
C
C           (I)
C           CONSIDER THE ADJACENT BOUNDARY POINT OF THE PREVIOUS EDGE OF
C           THE POLYGON.  CALL ITS INDEX NUMBER K0 AND SEE IF ITS CLOSER
C           TO THE CURRENT EDGE THAN P(J1) AND B(K3).
C
```

```
      2 CONTINUE
        KO = K1-1
        IF (KO.LT.1) KO=K
        PKO = B(KO)
        TERM = (Y(PKO)-Y(B1))*(X(B2)-X(B1))-(X(PKO)-X(B1))*(Y(B2)-Y(B1))
        IF (TERM.LE.0.) GOTO 3
        D = SQRT((X(PKO)-X(B1))**2+(Y(PKO)-Y(B1))**2)
     2    +SQRT((X(PKO)-X(B2))**2+(Y(PKO)-Y(B2))**2)
        IF (J1.NE.0.AND.D1.LT.D) GOTO 3
        J1 = KO
        D1 = D
        BFLAG = -1
      3 CONTINUE
C
C          (J)
C          SKIP THE NEXT SECTION IF J1 IS STILL ZERO, SINCE A CANDIDATE
C          POINT FOR TRIANGULATION WITH EDGE B1,B2 WAS NOT FOUND.
C
        IF (J1.EQ.0) GOTO 9
C
C
C
C
C          (K,L)
C          IF THE SEARCH FOR A CANDIDATE POINT HAS ALREADY CONSIDERED EACH
C          BOUNDARY EDGE AT LEAST ONCE (KT.GT.K) OR IF THE BOUNDARY IS
C          BEING CHECKED FOR CONCAVE EDGES (J=0), THEN THE NEXT SECTION
C          (SECTION M) CAN BE OMMITTED.
C
        IF (KT.GT.K.OR.J.EQ.0) GOTO 9
C
C          (M)
C          AT THIS POINT THE USER MAY INSERT ANY ADDITIONAL CONSTRAINT
C          ON THE TRIANGLE TO BE FORMED BY THE POINT PJ1.  IF THE
C          CANDIDATE TRIANGLE FAILS THE TEST, IT IS DELETED FROM
C          CONSIDERATION BY SETTING THE VARIABLE J1 TO ZERO.
C
      9 CONTINUE
```

```
C
C
C
C        (N,O)
C        THE NEXT PROCEDURE CHECKS ALL BOUNDARY EDGES OF THE POLYGON
C        FOR INTERSECTION WITH THE CANDIDATE TRIANGLE.  IF ANY EXISTING
C        BOUNDARY EDGE INTERSECTS ANY OF THE EDGES TO BE FORMED BY THE
C        CANDIDATE TRIANGLE, THEN THE CANDIDATE POINT IS REJECTED.  IF
C        BFLAG IS NOT ZERO, THEN THE EDGE DEFINED BY J1=KO OR J1=K3 IS
C        EXEMPT FORM THIS TEST.
C
C        IF THERE ARE THREE OR LESS EXISTING BOUNDARY EDGES OR IF
C        J1 HAS BEEN SET TO ZERO, THIS TEST IS OMMITTED.
C
      IF (K.LE.3.OR.J1.EQ.0) GOTO 7
      IF (BFLAG.EQ.0) NQ = P(J1)
      IF (BFLAG.EQ.1) NQ = B(K3)
      IF (BFLAG.EQ.-1) NQ = B(KO)
        DO 108 KCNT=1,K
        IF (KCNT.EQ.K1) GOTO 108
        KN = KCNT+1
        IF (KCNT.EQ.K) KN=1
        IF (BFLAG.EQ.-1.AND.(KCNT.EQ.KO.OR.KN.EQ.KO)) GOTO 108
        IF (BFLAG.EQ. 1.AND.(KCNT.EQ.K3.OR.KN.EQ.K3)) GOTO 108
        P1 = B(KCNT)
        P2 = B(KN)
          DO 8 JCNT=1,2
          IF (JCNT.EQ.1.AND.(BFLAG.EQ.0.OR.BFLAG.EQ.1).AND.KCNT.EQ.KO)  -
     *        GOTO 108
          IF (JCNT.EQ.2.AND.(BFLAG.EQ.0.OR.BFLAG.EQ.-1).AND.KCNT.EQ.K2) -
     *        GOTO 108
          BJ = B1
          IF (JCNT.EQ.2) BJ=B2
          XQB = X(NQ)-X(BJ)
          YQB = Y(NQ)-Y(BJ)
          X12 = X(P1)-X(P2)
          Y12 = Y(P1)-Y(P2)
          D = XQB*Y12-YQB*X12
```

```
              IF (D.EQ.0.) GOTO 8
              X1B = X(P1)-X(BJ)
              Y1B = Y(P1)-Y(BJ)
              S = (X1B*Y12-Y1B*X12)/D
              IF (S.LT.0..OR.S.GT.1.) GOTO 8
              TC = (XQB*Y1B-YQB*X1B)/D
              IF (TC.LT.0..OR.TC.GT.1.) GOTO 8
              J1 = 0
              GOTO 7
    8         CONTINUE
  108      CONTINUE
     7 CONTINUE
C
C
C            (P,Q)
C            IF J1 IS ZERO, THEN THE CANDIDATE POINT DID NOT PASS THE ABOVE
C            TESTS OR NO POINT WAS FOUND.  IF BFLAG IS NOT ZERO, THEN A
C            POINT ON THE BOUNDARY WAS FOUND.
C
         IF (J1.EQ.0) GOTO 10
         IF (BFLAG) 150,160,4
C
C
C            THE TRIANGULATED POINT IS OUTSIDE THE BOUNDARY.  ESTABLISH TWO
C            NEW EDGES, A NEW BOUNDARY POINT AND DELETE ONE POINT FROM
C            OUTSIDE THE BOUNDARY.
C
C
  160 E(L+1,1) = MINO(P(J1),B(K1))
      E(L+1,2) = MAXO(P(J1),B(K1))
      E(L+2,1) = MINO(P(J1),B(K2))
      E(L+2,2) = MAXO(P(J1),B(K2))
      KT = 0
      L = L+2
      M = M+1
      T(M,1) = MINO(P(J1),B(K1),B(K2))
      T(M,2) = MIDDLE(P(J1),B(K1),B(K2))
      T(M,3) = MAXO(P(J1),B(K1),B(K2))
```

```
      IF (K1.EQ.K) GOTO 140
         KM = K
         KP1 = K1+1
147      B(KM+1) = B(KM)
         KM = KM-1
         IF (KM.GE.KP1) GOTO 147
140   B(K1+1) = P(J1)
      K = K+1
      J = J-1
      IF (J1.GT.J) GOTO 10
      DO 144 JCNT=J1,J
144   P(JCNT) = P(JCNT+1)
      GOTO 10
C
C
C        (S)
C        THE TRIANGULATED POINT IS THE NEXT POINT ON THE BOUNDARY.
C        ESTABLISH ONE NEW EDGE (FROM B(K1) TO B(K3)), ONE NEW
C        TRIANGLE (FROM B(K1) TO B(K2) TO B(K3)), AND DELETE ONE POINT
C        FROM THE BOUNDARY (B(K2)).
C
C
    4 E(L+1,1) = MINO(B(K3),B(K1))
      E(L+1,2) = MAXO(B(K3),B(K1))
      KK = 0
      KKNT = 0
      KT = 0
      L=L+1
      K=K-1
      M = M+1
      T(M,1) = MINO(B(K1),B(K2),B(K3))
      T(M,2) = MIDDLE(B(K1),B(K2),B(K3))
      T(M,3) = MAXO(B(K1),B(K2),B(K3))
      IF (K2.GT.K) GOTO 155
      DO 151 KCNT=K2,K
151   B(KCNT) = B(KCNT+1)
155   IF (K2.EQ.1) K1=K1-1
      GOTO 10
```

```
C
C        (R)
C        THE TRIANGULATED POINT IS THE PREVIOUS POINT ON THE BOUNDARY.
C        ESTABLISH A NEW EDGE (FROM B(K0) TO B(K2)), ONE NEW TRIANGLE
C        (FROM B(K0) TO B(K1) TO B(K2)), AND DELETE ONE POINT FROM THE
C        BOUNDARY (B(K1))
C
  150 E(L+1,1) = MIN0(B(K0),B(K2))
      E(L+1,2) = MAX0(B(K0),B(K2))
      KK = 0
      KKNT = 0
      KT = 0
      L = L+1
      K = K-1
      M = M+1
      T(M,1) = MIN0(B(K0),B(K1),B(K2))
      T(M,2) = MIDDLE(B(K0),B(K1),B(K2))
      T(M,3) = MAX0(B(K0),B(K1),B(K2))
      IF (K1.GT.K) GOTO 157
      DO 158 KCNT=K1,K
  158 B(KCNT) = B(KCNT+1)
  157 K1 = K1-1
      IF (K1.LT.1) K1=K
C
C
C        (T)
C        IF THERE ARE ANY POINTS REMAINING OUTSIDE THE BOUNDARY, THEN
C        REPEAT THE PROCEDURE FOR THE NEXT EDGE.
C
C
   15 IF (J.GT.0.AND.J1.NE.0) GOTO 12
      IF (J.GT.0) GOTO 11
C
C
C        (U,V,W)
C        ALL POINTS HAVE BEEN TRIANGULATED.  CHECK THAT ALL BOUNDARY
C        EDGES FORM A CONCAVE POLYGON.
C
```

```
      IF (KK.NE.0) GOTO 55
      KK = 1
      KL = 0
   55 KKNT = KKNT+1
      IF (KKNT.GT.N) GOTO 170
    5 KL = KL+1
      K2 = KL+1
      IF (K2.GT.K) K2=1
      K1 = KL-1
      IF (K1.LT.1) K1=K
      PKL = B(KL)
      B1 = B(K1)
      B2 = B(K2)
      TERM = (Y(PKL)-Y(B1))*(X(B2)-X(B1))-(X(PKL)-X(B1))*(Y(B2)-Y(B1))
      IF (TERM.LT.0.) GOTO 11
      IF (KL.LT.K) GOTO 5
C
C
C        (X)
C        THE TRIANGULATION IS COMPLETE AND HAS BEEN CHECKED FOR A
C        CONCAVE BOUNDARY.  NOW IDENTIFY THE BOUNDARY EDGES.
C
C
  170 DO 23 LCNT=1,L
      BE(LCNT) = 0
      KL = 0
   21 KL = KL+1
      IF (E(LCNT,1).NE.B(KL)) GOTO 22
      K1 = KL+1
      IF (K1.GT.K) K1=1
      IF (E(LCNT,2).NE.B(K1)) GOTO 162
      BE(LCNT) = 1
      GOTO 23
  162 K1 = KL-1
      IF (K1.LT.1) K1=K
      IF (E(LCNT,2).NE.B(K1)) GOTO 22
      BE(LCNT) = 1
      GOTO 23
```

```
   22 IF (KL.LT.K) GOTO 21
   23 CONTINUE
C
C
C           (Y)
C           FINALLY, ESTABLISH THE INDICES OF ADJACENT EDGES FOR EACH
C           EDGE IN THE TRIANGULATION.  EACH BOUNDARY EDGE WILL HAVE TWO
C           ADJACENT EDGES - EACH INTERIOR EDGE WILL HAVE FOUR.
C
      DO 190 LL  =1,4
      DO 190 LCNT=1,L
  190 TE(LCNT,LL) = 0
      DO 191 MCNT=1,M
         DO 192 LL=1,L
         IF (E(LL,1).EQ.T(MCNT,1).AND.E(LL,2).EQ.T(MCNT,2)) L1=LL
         IF (E(LL,1).EQ.T(MCNT,2).AND.E(LL,2).EQ.T(MCNT,3)) L2=LL
         IF (E(LL,1).EQ.T(MCNT,1).AND.E(LL,2).EQ.T(MCNT,3)) L3=LL
  192    CONTINUE
      LAMBDA = 0
      IF (TE(L1,1).NE.0) LAMBDA=?
      TE(L1,LAMBDA+1) = L2
      TE(L1,LAMBDA+2) = L3
      LAMBDA = 0
      IF (TE(L2,1).NE.0) LAMBDA=2
      TE(L2,LAMBDA+1) = L1
      TE(L2,LAMBDA+2) = L3
      LAMBDA = 0
      IF (TE(L3,1).NE.0) LAMBDA = 2
      TE(L3,LAMBDA+1) = L1
      TE(L3,LAMBDA+2) = L2
  191 CONTINUE
C
      RETURN
      END
```

MIDDLE:S,11/05/80 09:30:59

```
      FUNCTION MIDDLE(I,J,K)
C
C
C        THIS FUNCTION SUBPROGRAM IS USED BY THE TRIANGULATION ALGORITHM
C        TO FIND THE MIDDLE VALUE OF THE THREE INTEGER ARGUMENTS (THE
C        VALUE WHICH IS NEITHER A MINIMUM OR A MAXIMUM).  I, J AND K ARE
C        ARE ASSUMED TO BE DISCRETE VALUES WITH NO TWO EQUAL.
C
C
      IF (J.LT.I.AND.I.LT.K) GOTO 100
      IF (K.LT.I.AND.I.LT.J) GOTO 100
      IF (I.LT.J.AND.J.LT.K) GOTO 200
      IF (K.LT.J.AND.J.LT.I) GOTO 200
      MIDDLE = K
      RETURN
  100 MIDDLE = I
      RETURN
  200 MIDDLE = J
      RETURN
      END
```

POLYX2,$,11/05/80 09:31:11

```
      FUNCTION POLYX2 (Z,X,Y,C,IPOWR,JPOWR,NCOEF)
C
C
C        POLYX2 IS THE POLYNOMIAL EVALUATION FUNCTION USED WHEN THE
C        SMOOTHING OPTION HAS BEEN INVOKED.  X AND Y LISTS ARE THE
C        KNOWN VALUES OF THE INDEPENDENT VARIABLES.  C IS THE LIST OF
C        COEFFICIENTS FOR EACH TERM.  IPOWR AND JPOWR ARE THE EXPONENTS
C        FOR EACH TERM AND N IS THE NUMBER OF TERMS IN THE POLYNOMIAL.
C        Z IS AN OFFSET TERM WHEN EVALUATING FOR A CONSTANT X VALUE.
C
C
      DIMENSION  IPOWR(23),JPOWR(23),C(23)
C
C
      POLYX2 = 0.0
      DO 120 II=1,NCOEF
      POLYX2 = POLYX2 + ((X**IPOWR(II)) * (Y**JPOWR(II))) * C(II)
  120 CONTINUE
      POLYX2 = Z - POLYX2
      RETURN
      END
```

CBVCHK:S,11/05/80 09:31:17

```
      SUBROUTINE CBVCHK (ZZERO,DELZ,ZMIN,ZMAX,ZZNEW)
C
C     ------------------------------------------------------------------
C        CONTOUR BASE VALUE CHECKING ROUTINE
C        *       *    *     **  *
C
C     THIS SUBROUTINE SHIFTS THE BASE VALUE (ZZERO) UNTIL IT FALLS
C     WITHIN THE RANGE OF DATA FOR THIS CONTOUR (I.E. BETWEEN ZMIN
C     AND ZMAX).  THE SHIFTED VALUE (THE NEW STARTING BASE VALUE) IS
C     RETURNED TO CALLER AS ZZNEW.  THE USER SHIFT INCREMENT COMES
C     INTO CBVCHK AS DELZ FOR Z CONTOURS.
C
C     ARGUMENTS -
C        ZZERO     = BASE VALUE (INPUT)
C        DELZ      = INCREMENT VALUE (INPUT)
C        ZMIN,ZMAX = RANGE OF Z DATA (INPUT)
C        ZZNEW     = NEW BASE VALUE, MAY OR MAY NOT BE
C                    THE SAME AS ZZERO (RETURN)
C
C
C     ------------------------------------------------------------------
C
C
      IF (ZMIN.EQ.ZMAX) GOTO 999
      ZZNEW = ZZERO
      IF (ZMIN.LE.ZZNEW.AND.ZZNEW.LE.ZMAX) GOTO 999
    2 IF (ZZNEW.GE.ZMAX) GOTO 1
      ZZNEW = ZZNEW + DELZ
      IF (ZMIN.LE.ZZNEW.AND.ZZNEW.LE.ZMAX) GOTO 999
      GOTO 2
C
    1 ZZNEW = ZZERO
    4 IF (ZZNEW.LE.ZMIN) GOTO 999
      ZZNEW = ZZNEW - DELZ
      IF (ZMIN.LE.ZZNEW.AND.ZZNEW.LE.ZMAX) GOTO 999
      GOTO 4
C
  999 RETURN
```

```
SOVLHR$$,11/05/80 09:31:17

 3900          END
```

```
      SUBROUTINE INTERP (X,Y,U,N,ZCON,LEDGES,IE,ISMOPT,LAMBDA,XI,
     *                   ETA,J,C,IPOWR,JPOWR,NCOEF)
C     ----------------------------------------------------------------
C
C     SUBROUTINE INTERP IS GIVEN A CONSTANT U VALUE (BIGU) FOR WHICH
C     THE CONTOUR LINE IS TO BE DRAWN.  CHECK ALL GIVEN TRIANGLE EDGES,
C     (ARRAY IE) AND CHECK THE VALUES OF U AT THE ENDPOINTS.
C     INTERPOLATE FOR ALL POSSIBLE VALUES ON THE TRIANGLE EDGES.
C     IF ISMOPT = 0, THEN USE A LINEAR INTERPOLATION, IF ISMOPT NOT ZERO
C     THEN EVALUATE FOR A NON-LINEAR SURFACE USING THE COEFFICIENTS
C     FROM SMSRF AND FUNCTION SUBROUTINE POLYX.
C
C        X,Y,U      = DEPENDENT AND INDEPENDENT VALUES FOR
C                     THE RELATION U=F(X,Y)  (INPUT)
C        ZCON       = CONSTANT VALUE OF Z FOR WHICH INTERPOLATION
C                     IS REQUIRED  (INPUT)
C        LEDGES     = NO. OF EDGES IN THE TRIANGULATION  (INPUT)
C        IE         = EDGE ENDPOINT INDICES FROM TRIANGULATION (INPUT)
C        ISMOPT     = SMOOTHING OPTION FLAG,  0=OFF, 1=ON,  (INPUT)
C        LAMBDA     = INDEX OF EDGES FOR INTERPOLATED POINTS  (RETURN)
C        XI         = LIST OF X-COORDINATES OF INTERPOLATED POINTS
C        ETA        = LIST OF Y-COORDINATES OF INTERPOLATED POINTS
C        J          = NUMBER OF VALUES IN XI, ETA LISTS
C                     (XI, ETA AND J ARE RETURNED)
C        C          = LIST OF COEFFICIENTS OF EACH TERM OF THE EQUATION.
C        IPOWR,JPOWR ARE THE LIST OF EXPONENTS FOR EACH TERM OF
C        THE POLYNOMIAL USED TO SMOOTH THE DATA  (INPUT).
C        NCOEF      = NUMBER OF TERMS IN THE POLYNOMIAL
C                     (IPOWR,JPOWR,C, AND NCOEF ARE INPUT)
C
C
C     ----------------------------------------------------------------
C
C
      DIMENSION  X(N),Y(N),U(N)
      DIMENSION  IE(1494,2),XI(1494),ETA(1494),LAMBDA(1494)
      DIMENSION  IPOWR(23),JPOWR(23),C(23)
C
```

```
C
      IF (NCEDF.LT.1) ISMCPT=0
      J = 0
C
      DO 1 LCNT=1,LEDGES
C
C        (A)
C        DETERMINE X,Y,Z FOR THE ENDPOINTS OF THE NEXTEDGE - ORDER THEM
C
      I1 = IE(LCNT,1)
      I2 = IE(LCNT,2)
      X1 = X(I1)
      X2 = X(I2)
      Y1 = Y(I1)
      Y2 = Y(I2)
      U1 = U(I1)
      U2 = U(I2)
C
C        (B)
C        FUNCTION VALUES EQUAL AT ENDPOINTS OR
C        CONSTANT ZC NOT BETWEEN THEM? . .
C
      IF (U1.EQ.U2) GOTO 1
      IF (U1.LT.U2) GOTO 100
      TEMP = U2
      U2 = U1
      U1 = TEMP
      TEMP = X2
      X2 = X1
      X1 = TEMP
      TEMP = Y2
      Y2 = Y1
      Y1 = TEMP
  100 IF (ZCON.LT.U1.OR.U2.LT.ZCON) GOTO 1
      IF (U2.EQ.ZCON) U2 = 1.000001 * ZCON
      J = J+1
C
C        (C)
```

```
C           HAS DATA BEEN SMOOTHED? . .
C           IF NOT, GOTO SECTION E (STATEMENT LABEL 101)
C
      IF (ISMOPT.EQ.0) GOTO 101
C
C           (D,F)
C           NON-LINEAR INTERPOLATION IS REQUIRED
C           ON THIS EDGE OVER THE Z-SURFACE
C
      F1 = POLYX2 (ZCON,X1,Y1,C,IPOWR,JPOWR,NCOEF)
C
      DO 220 K=1,10
      XN = (X1+X2)*0.5
      YN = (Y1+Y2)*0.5
      FN = POLYX2 (ZCON,XN,YN,C,IPOWR,JPOWR,NCOEF)
      IF (FN.EQ.0.) GOTO 132
      IF (FN.LT.0..AND.F1.LT.0.) GOTO 235
      IF (FN.GT.0..AND.F1.GT.0.) GOTO 235
      X2 = XN
      Y2 = YN
      GOTO 220
  235 X1 = XN
      Y1 = YN
  220 CONTINUE
  132 XI(J)  = (X1+X2)*0.5
      ETA(J) = (Y1+Y2)*0.5
      GOTO 200
C
C
C           (E,F)
C           LINEAR INTERPOLATION IS REQUIRED
C           FOR THIS EDGE OVER THE Z-SURFACE
C
  101 T1 = (U2-ZCON)/(U2-U1)
      T2 = (ZCON-U1)/(U2-U1)
      XI(J) = T1*X1+T2*X2
      ETA(J)= T1*Y1+T2*Y2
  200 LAMBDA(J) = LCNT
```

INTERF*$,11/05/80 09:31:24

```
    1 CONTINUE
      RETURN
      END
```

CNTOUR ,$,11/05/80 09:31:46

```
      SUBROUTINE CNTOUR (ZCON,XI,ETA,LAMBDA,J,IBE,ITE)
C
C     ------------------------------------------------------------------
C
C     A SET OF J INTERPOLATED POINTS FOR Z=ZCON  (XI(I),ETA(I) ON EDGE
C     LAMBDA(I) FOR I=1,J), THE CONTOUR LINES MUST NOW BE DRAWN.  THERE
C     MAY BE SEVERAL LINES, EITHER OPEN OR CLOSED CONTOURS.  THIS
C     ALGORITHM WILL USE THE TRIANGULATION RELATIONSHIPS TO SORT OUT
C     EACH LINE IN ORDER.  AS EACH CONTOUR LINE IS ESTABLISHED, USER
C     SUPPLIED PROGRAM CNTCRV IS CALLED TO OUTPUT IT TO THE GRAPHICS
C     DEVICE BEING USED.
C
C
C
C     ARGUMENTS (ALL ARE INPUTS) -
C        ZCON      = CONSTANT VALUE OF Z UNDER CONSIDERATION
C        XI(J)     = ARRAY OF X COORDIANTES OF INTERPOLATED POINTS
C        ETA(J)    = ARRAY OF Y COORDIANTES OF INTERPOLATED POINTS
C        LAMBDA(J) = ARRAY OF EDGE NUMBERS FOR J-TH INTERPOLATED POINT
C        J         = NUMBER OF POINTS IN THE LIST OF INTERPOLATED POINTS
C        IBE       = THE LIST OF BOUNDARY EDGES TAKEN FROM THE TRIANGULATION
C        ITE       = LINKED LIST OF INDICES OF ADJACENT EDGES PROVIDED
C                    BY THE TRIANGULATION PROCEDURE.
C
C
C     ------------------------------------------------------------------
C
C
      DIMENSION  XI(1494),ETA(1494),LAMBDA(1494),IBE(1494),XX(1494),
     *           YY(1494),ITE(1494,4)
C
C
C
C        (A)
C        INITIALIZE LOCAL VARIABLES
C
      IF (J.EQ.0) RETURN
   10 J1 = 0
```

```
C
C          (B,C)
C          SEARCH THE LIST OF EDGES FOR A BOUNDARY EDGE (BE(I)=1)
C
      1 J1 = J1+1
        L1 = LAMBDA(J1)
        IF (IBE(L1).EQ.1) GOTO 2
        IF (J1.LT.J) GOTO 1
        GOTO 11
C       SEARCH FOR A BOUNDARY EDGE AND PUT IT AT THE TOP OF THE LIST.
C
C          (D)
C          PUT THIS INTERPOLATED POINT AT THE TOP OF THE
C          LIST FOR THIS CONTOUR,  SET J1
C
      2 IF (J1.EQ.J) GOTO 3
        XI(J+1) = XI(J1)
        ETA(J+1) = ETA(J1)
        LAMBDA(J+1) = LAMBDA(J1)
        DO 101 JCNT = J1,J
        XI(JCNT) = XI(JCNT+1)
        ETA(JCNT) = ETA(JCNT+1)
    101 LAMBDA(JCNT) = LAMBDA(JCNT+1)
C
C          (E)
C          SEARCH THE REMAINING POINTS FOR AN ADJACENT (COMMON) EDGE
C
      3 J1BIG = J
        LCNT = L1
      6 J1BIG = J1BIG-1
        J1 = 0
      5 J1 = J1+1
        L1 = LAMBDA(J1)
        DO 102 I=1,4
        IF (L1.EQ.ITE(LCNT,I)) GOTO 4
    102 CONTINUE
C          (F)
C          ERROR - THERE IS NO NEXT POINT.
```

```
      IF (J1.LT.J1BIG) GOTO 5
      GOTO 800
C
C        (G)
C        PUT THIS POINT AT THE TOP OF THE
C        LIST.  CONTINUE IF ITS NOT A BOUNDARY EDGE.
C
    4 XI(J+1) = XI(J1)
      ETA(J+1) = ETA(J1)
      LAMBDA(J+1) = LAMBDA(J1)
      DO 103 JCNT = J1,J
      XI(JCNT) = XI(JCNT+1)
      ETA(JCNT) = ETA(JCNT+1)
  103 LAMBDA(JCNT) = LAMBDA(JCNT+1)
      LCNT = L1
      IF (IBE(L1).NE.1) GOTO 6
C
C        (H)
C        DRAW THE OPEN CONTOUR LINE THROUGH THE POINTS
C        XI(J1),ETA(J1) ...... XI(J1+1),ETA(J1+1) ...... XI(J),ETA(J)
C        THEN RESET J AND CONTINUE
C
C
C     ----------------------------------------------------------------
      NPOINT = J-J1BIG+1
      IF (NPOINT.LE.1) GOTO 300
      CALL CNTCRV (XI(J1BIG),ETA(J1BIG),NPOINT,ZCON)
C     ----------------------------------------------------------------
C
  300 J=J1BIG - 1
C
C        (I)
C        ARE THERE ANY MORE POINTS LEFT? . .
C
      IF (J) 800,800,10
C
C
C        (J)
```

```
C          NOW DRAW INTERNAL LINES (CLOSED CONTOURS THAT DO NOT START
C          OR STOP AT BOUNDARY EDGES).  THE POINT AT J1BIG=J IN
C          THE LIST IS CHOSEN TO START THE CONTOUR.
C
   11 J1BIG = J+1
      LCNT = LAMBDA(J)
C
C          (K,M,P)
C          FIND THE NEXT POINT FOR THIS CONTOUR (ON AN EDGE WITH A COMMON
C          END POINT).  PUT IT AT THE TOP OF THE LIST, AND REPEAT UNTIL
C          NO MORE COMMON EDGES REMAIN FOR THIS LINE.
C
   16 J1BIG = J1BIG-1
      J1 = 0
      IF (J1BIG.GT.J) J1=1
   15 J1 = J1+1
      L1 = LAMBDA(J1)
      DO 104 I=1,4
      IF (L1.EQ.ITE(LCNT,I)) GOTO 14
  104 CONTINUE
      IF (J1.LT.J1BIG) GOTO 15
C        (L)
C        OTHERWISE, NO ADJACENT EDGE WAS FOUND.
C        THIS CONTOUR LINE IS COMPLETE, GO DRAW IT.
      GOTO 17
C
   14 XI(J+1) = XI(J1)
      ETA(J+1) = ETA(J1)
      LAMBDA(J+1) = LAMBDA(J1)
      DO 105 JCNT = J1,J
      XI(JCNT) = XI(JCNT+1)
      ETA(JCNT) = ETA(JCNT+1)
  105 LAMBDA(JCNT) = LAMBDA(JCNT+1)
      LCNT = L1
      IF (J1BIG.NE.1) GOTO 16
C
C
C
```

102

```
C          (O)
C          DRAW THE CLOSED CONTOUR LINE, THE INTERPOLATED LINE THROUGH
C          XI(J1),ETA(J1) ...... XI(J),ETA(J) ...... XI(J1),ETA(J1)
C
C
C
C       ---------------------------------------------------------------
     17 JJ = J1BIG
        IF (J1BIG.NE.1) JJ = J1BIG+1
        KNT = 0
        DO 510 KK = JJ,J
        KNT = KNT+1
        XX(KNT) = XI(KK)
        YY(KNT) = ETA(KK)
    510 CONTINUE
        XX(KNT+1) = XX(1)
        YY(KNT+1) = YY(1)
        NPOINT = KNT+1
        CALL CNTCRV (XX(1),YY(1),NPOINT,ZCON)
C       ---------------------------------------------------------------
C
C
C          (P)
C          RESET J.  ESTABLISH THE NEXT CONTOUR LINE FOR REMAINING POINTS
C          OR QUIT THE PROCEDURE IF NO MORE POINTS REMAIN.
C
        J = J1BIG - 1
        IF (J) 800,800,11
    800 RETURN
        END
```

END

DATE

FILMED

JUL 13 19[illegible]