Report 013514-2-M

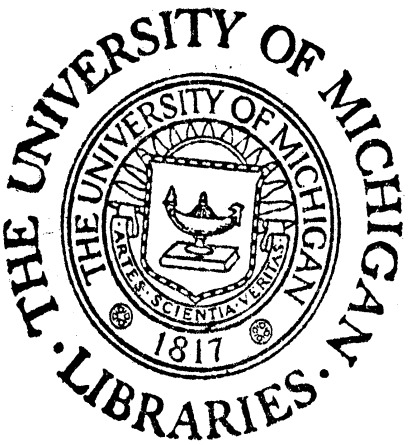
CONSYS: A Collection of FORTRAN Subroutines to Produce Contour Maps of Data Surfaces Defined on Rectangular Grids

G. N. Cederquist
COOLEY ELECTRONICS LABORATORY
Department of Electrical and Computer Engineering
The University of Michigan
Ann Arbor, Michigan 48109

July 1976

Technical Memorandum No. 112

Prepared for
OFFICE OF NAVAL RESEARCH
Department of the Navy
Arlington, Virginia 22217



SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 013514-2-M	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): CONSYS: A Collection of FORTRAN Subroutines to Produce Contour Maps of Data Surfaces Defined on Rectangular Grids	5. TYPE OF REPORT & PERIOD COVERED
6. PERFORMING ORG. REPORT NUMBER: TM 112	
7. AUTHOR(s): Gerald N. Cederquist	8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0174
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Office of Naval Research, Department of the Navy, Arlington, Virginia 22217	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: July 1976
	13. NUMBER OF PAGES: 54
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Although care has been taken to ensure the correct functioning of the CONSYS system, neither the author nor The University of Michigan shall be liable for any direct or indirect incidential, consequential, or specific damages of any kind or

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Contour Maps
Computer Graphics
Device Independence
FORTRAN

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The CONSYS routines produce coordinate pairs for the straight-line vectors forming a contour map. The routines operate on user supplied samples of the x, y, and z coordinates that specify the data surface to be contoured. To reduce processing time, the routines assume that the data samples are available on a rectangular grid; surfaces represented by randomly scattered samples cannot be directly contoured by CONSYS. The routines

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

18.
from any cause whatsoever arising out of or in any way connected with the use or performance of the CONSYS system or its documentation.

20.
are device-independent in the sense that at execution time the user must supply the names of subroutines that CONSYS can call to dispose of contour line coordinate pairs. CONSYS works entirely in the user's coordinate system, and the user is responsible for any scaling and translation necessary to map the coordinate pairs into a viewable area on the plotting device or graphics terminal being used.

DISCLAIMER

Although care has been taken to ensure the correct functioning of the CONSYS system, neither the author nor The University of Michigan shall be liable for any direct or indirect, incidental, consequential, or specific damages of any kind or from any cause whatsoever arising out of or in any way connected with the use or performance of the CONSYS system or its documentation.

APPLICABILITY

This document describes version 1.2 of CONSYS, dated 15 May, 1976.

TABLE OF CONTENTS

Introduction 1

Type of Routine 1

Availability 2

Facilities Provided 2

How to Use CONSYS 3

How to Call CONSET 4

The Contour Labeling Scheme 5

How to Call CONLBL 5

How to Call CONTUR 7

How to Call CONEND 10

Logical I/O Units Referenced 11

Subprograms Loaded 11

Subprograms Required 11

Storage Requirements 12

Error Handling 12

Summary of Coordinate System Conventions 13

Example 1 14

Example 2 15

Appendix A: Getting the Data
Onto a Rectangular Grid 24

Appendix B: Method 25

Appendix C: The Structure of the Contour
Map Display File 27

Appendix D: Harding's Method for Dynamic Allocation
of Arrays Used in CONSYS 28

Appendix E: Listing of the Source Code 32

References 47

Bibliography 48

LIST OF ILLUSTRATIONS

Figure 1. Specification of the data grid by the X and Y arrays 8

Figure 2. Association of Z(I,J) values with the data grid 8

Figure 3. Example rectangular coordinate grid representation for a data surface with IX = 8 and IY = 5 13

Figure 4. Example Program 1 15

Figure 5. The computer terminal interaction with Example Program 2 which produced Figure 6 21

Figure 6. Example contour plot. Eleven contours at intervals of Ø.5 from -3 to +2 22

Figure 7. Example contour plot. Three contours at intervals of 5 from 5 to 15 23

Figure B1. Hartwig's numbering convention for the lines overlaying a grid cell 26

Figure B2. A contour line crossing line segments 1, 2, 4, and 5 26

CONTOUR MAPS OF DATA SURFACES

INTRODUCTION

Quite often in the course of analyzing a data surface:

$$z = f(x, y) \qquad (1)$$

(where x, y, and z are chosen from the real numbers), one wishes to produce a map in the x - y plane of the curve defined by letting z be a constant. More formally, one may choose to fix z at some value, say z', and solve for y as a function of x:

$$y = g(x, z') \qquad (2)$$

A contour line is defined as a set of connected points in the plane $z = z'$ which satisfy equation (2). Since g may be multiple-valued, a single value of z' may generate more than one contour line. The set of all contour lines generated by a value of z' is called the contour set of g at $z = z'$. A family of contour sets can be generated by a family of z' values. The projection of a family of contour sets onto the x-y plane is termed a contour map of the data surface f.

The routines in this package produce coordinate pairs for the straight-line vectors forming a contour map. The routines operate on user-supplied samples of the (presumably) continuously variable x, y, and z coordinates that constitute the data surface to be contoured. The routines place restrictions on the input values to reduce processing time. The routines are device independent in the sense that the user must supply the names of subroutines that CONSYS can call to dispose of contour line coordinate pairs.

TYPE OF ROUTINE

CONSYS is the collective name for the contouring routines; CONSYS is a collection of IBM FORTRAN IV SUBROUTINE and FUNCTION subprograms. The public entry points are CONTUR, CONSET, CONLBL, and CONEND. The routines are not in ANSI Standard FORTRAN (1966), and thus would need some modification before they could be transported to a non-IBM installation. Moreover, the routines do dynamic storage allocation of some work spaces and thus are somewhat operating-system-dependent; this latter objection to transportability could be removed by fixing the dimensions

of the work arrays and recompiling the source modules.

The object modules for CONSYS were produced by the H-level IBM FORTRAN Compiler.

AVAILABILITY

For use on the Michigan Terminal System (MTS) at The University of Michigan, the link-edited object module library containing these subroutines is in the file UNSP:DIGLIB. The user may use the routines by concatenating this file to the other object file(s) in the $RUN command which invokes the user's program.

For potential users outside The University of Michigan, CONSYS is available on magnetic tape; address inquiries to the author.

FACILITIES PROVIDED

CONSYS will produce coordinates for a contour map of a data surface which is defined on a rectangular grid. The rectangular grid representation of a data surface stores the z data values as might be expected, in a two-dimensional array Z(I, J). Separate X and Y one-dimensional arrays hold the x- and y-coordinates of the data surface. Recall that f is the function mapping an x-y pair into a z value; then the relationship between the X and Y vectors and the Z array inside the computer is:

$$Z(I,J) = f(X(I), Y(J)) \qquad (3)$$

Suppose for example that values of z have been collected at all possible combinations of x = 1, 2, and 4, and y = 1, 4, and 9. The data can then be fitted into a rectangular grid representation by letting X(1) = 1, X(2) = 2, X(3) = 4, Y(1) = 1, Y(2) = 4 AND Y(3) = 9. The Z array would be filled with values using equation (3) above.

Surfaces which are represented by scattered data points or nonrectangular grids may not be directly contoured by these routines (but see Appendix A - "Getting the Data Onto a Grid").

CONSYS contains a primitive contour labeling facility which the user may invoke by setting a switch in the subroutine calling parameters. For the applications envisaged for CONSYS (large data surfaces), the CPU time necessary to produce "nice" labels was considered an

extravagance. (The CPU time would increase by a factor of at least 1.5.) With a small amount of extra work, the user may collect the contour map coordinate pairs produced by CONSYS and reformat them before plotting, generating nice labels in the process.

CONSYS does not require that a user have all the data to be contoured present in memory at one time. Thus the user may produce very large maps by calling CONTUR repeatedly, once for each different region of the data surface. The user has control over the drawing of border lines for the map and can thus omit the border line when another map section is to be adjacent to the one currently being produced.

Since its work areas are dynamically allocated, CONSYS effectively imposes no limit on the number of data points in the data surface to be contoured.

To keep CONSYS device-independent, the user must write a labeling routine if he/she desires to invoke the labeling option of CONSYS. This routine is often only 10-20 statements long, but is usually dependent upon the graphical output device to which the user sends the map coordinates.

CONSYS works entirely in the user's coordinate space. In particular, contour map coordinate pairs produced by CONSYS are all in the user's coordinate space. The user is responsible for any scaling necessary to map the coordinate pairs into a viewable area on the plotting device or graphics terminal being used.

HOW TO USE CONSYS

The user must call subroutine CONSET once to initialize CONSYS before calling upon CONTUR to produce contour line coordinates. Once CONSET has been called, it need not be called again unless the user wants to change the value of one (or more) of the CONSET parameters.

If the user desires to use the labeling facility in CONSYS, he/she must call subroutine CONLBL to specify values for various labeling parameters. CONLBL must be called after CONSET has been called, but before the first call to CONTUR which specifies that labeling is to be done.

The user may then call on subroutine CONTUR to produce coordinates for a contour map. The coordinates produced are in the coordinate system and value range which the user specifies in the parameters to the call to CONTUR; thus the user may make repeated calls to CONTUR either to produce new

contours on an existing map or to map new data regions adjacent to those already mapped.

When the user is finished producing contours, he/she must call the routine CONEND to release the storage dynamically acquired during the contour labeling process.

HOW TO CALL CONSET

None of the calling parameters to CONSET is changed by CONSET in any fashion. The calling sequence for CONSET is

```
CALL CONSET(PNUP, PNDN, IOUNIT)
```

where

PNUP is the name of a subroutine which CONSYS can call to dispose of an (x, y) contour line coordinate pair. CONSYS calls PNUP via the statement

```
CALL PNUP(X, Y)
```

The subroutine should cause the beam or pen to be positioned invisibly to the point (x, y) in the user's coordinate space. X and Y are of type REAL. Examples of routines which perform such a function in MTS are CKMA (Computek and Tektronix routines), IGMA (Integrated Graphics (*IG) routines), and PENUP (Calcomp Plotter routines). PNUP must be declared EXTERNAL in the calling program.

PNDN is the name of a subroutine which CONSYS can call to dispose of an (x, y) contour line coordinate pair. CONSYS calls PNDN via the statement

```
CALL PNDN(X, Y)
```

The subroutine should cause a visible line to be drawn from wherever the beam or pen is positioned prior to the call to the point (x, y) in the user's coordinate space. X and Y are of type REAL. Examples of routines which perform such a function are CKVA (Computek and Tektronix routines), IGDA (*IG routines), and PENDN (Calcomp Plotter routines). PNDN must be declared EXTERNAL in the calling program.

IOUNIT is an integer constant or an INTEGER*4 expression which specifies the FORTRAN data set reference number (DSRN) on which error comments from CONSYS are to be written. If the value of IOUNIT is less

than zero or greater than nineteen, then error comments will not be written. Conversely, if IOUNIT is within the range Ø to 19, then in the event of errors, error comments will be produced via FORTRAN WRITE statements which look like

WRITE(IOUNIT, format)...

THE CONTOUR LABELING SCHEME

CONSYS' primitive contour labeling facility may prove adequate for a large number of applications. With this facility, the user specifies in his/her coordinate system a number of lines of constant x and a number of lines of constant y. Whenever a contour enters a cell in the rectangular grid which contains one of these constant-coordinate lines, the contour becomes a candidate for labeling in that cell. If the center of the grid cell is more than a user-specified distance (in user coordinate space) away from the centers of all other grid cells which have already been labeled, the contour is declared a successful candidate for labeling. In this event, CONSYS calls a user-specified subroutine with three parameters: the center coordinates of the grid cell (x and y), and the z-value of the contour which entered the grid cell; the user-specified routine is then responsible for producing a label for the contour and sending the label to the graphical output device.

The values of the constant coordinate lines, the name of the labeling routine, and the distance parameter for declaring a labeling candidate to be successful are passed to CONSYS by calling CONLBL.

HOW TO CALL CONLBL

If the user does not wish to use the labeling facility of CONSYS, he/she does not need to call CONLBL or to read this section. None of the calling parameters to CONLBL is altered in any fashion by CONLBL. The calling sequence for CONLBL is

CALL CONLBL(LABELR, DIST, NXL, XLOC, NYL, YLOC)

where

LABELR is the name of a subroutine which CONSYS will call to produce a label for a contour line. LABELR will be called with the FORTRAN statement

CALL LABELR(XC, YC, Z)

where

XC and YC are the type REAL x- and y-coordinates of the center of the grid cell containing the contour to be labeled, and

Z is the type REAL value of the z-coordinate of the contour line which is to be labeled.

LABELR must be declared EXTERNAL in the calling program.

DIST is a REAL constant or a REAL expression giving the minimum distance in the user's coordinate space which will allow a labeling candidate to be declared successful. If the distance from the center of a cell which is a candidate for labeling to any other cell which has already been labeled is greater than DIST units, then the candidate cell will be labeled by calling LABELR. If each label takes at most N characters with width S coordinate units per character, then a good value for DIST is S times (N+1). DIST must be > Ø.

NXL is an integer constant or an INTEGER*4 expression giving the number of constant-x lines to be used in the labeling process. NXL must be greater than or equal to zero.

XLOC is a REAL vector at least NXL entries long. (Even if NXL is zero, an argument must be supplied; however, in this case the argument will not be used.) Each entry in XLOC is the x coordinate in the user's coordinate space of a line of constant x value (i.e., a vertical line) to be used during the labeling process to determine candidate cells for labeling. (See "The Contour Labeling Scheme" above.)

NYL is an integer constant or an INTEGER*4 expression giving the number of constant-y lines to be used in the labeling process. NYL must be greater than or equal to zero.

YLOC is a REAL vector at least NYL entries long. (Even if NYL is zero, an argument must be supplied; however, in this case the argument will not be used.) Each entry in YLOC is the y coordinate in the user's coordinate space of a line of constant y

value (i.e., a horizontal line) to be used during the labeling process to determine candidate cells for labeling (see "The Contour Labeling Scheme" above).

Note: NXL and NYL may not both be zero at the same time.

HOW TO CALL CONTUR

Once CONSYS has been initialized by a call to CONSET (and, optionally, a call to CONLBL), the user may call CONTUR to produce coordinates for contour lines. None of the calling parameters to CONTUR is changed in any way by CONTUR. The calling sequence for CONTUR is

```
CALL CONTUR(X, IX, Y, IY, Z, IDX, CZ, NC, SWCHES)
```

where

X is a vector of x coordinates at least IX entries long. Each coordinate marks a vertical line which is the vertical boundary of a grid cell; see Fig. 1. The coordinates in X are in the user's coordinate space and must be in strictly ascending order with increasing subscript.

IX is an integer constant or an INTEGER*4 expression giving the number of x coordinate values in the X vector to use, i.e., the number of points along the first dimension of the Z array to be considered in the contouring process. IX must be within the range 2 through IDX inclusive.

Y is a vector of y coordinates at least IY entries long. Each coordinate marks a horizontal line which forms the horizontal boundary of a grid cell; see Fig. 1. The coordinates in Y are in the user's coordinate space and must be in strictly ascending order with increasing subscript.

IY is an integer constant or an INTEGER*4 expression giving the number of y coordinate values in the Y vector to use, i.e., the number of points along the second dimension of the Z array to be considered in the contouring process. IY must be greater than or equal to 2.

Note: The data grid upon which contours are to be produced need not be regular. The corners of the grid are located by the values in the X and Y arrays, as indicated by Fig. 1.

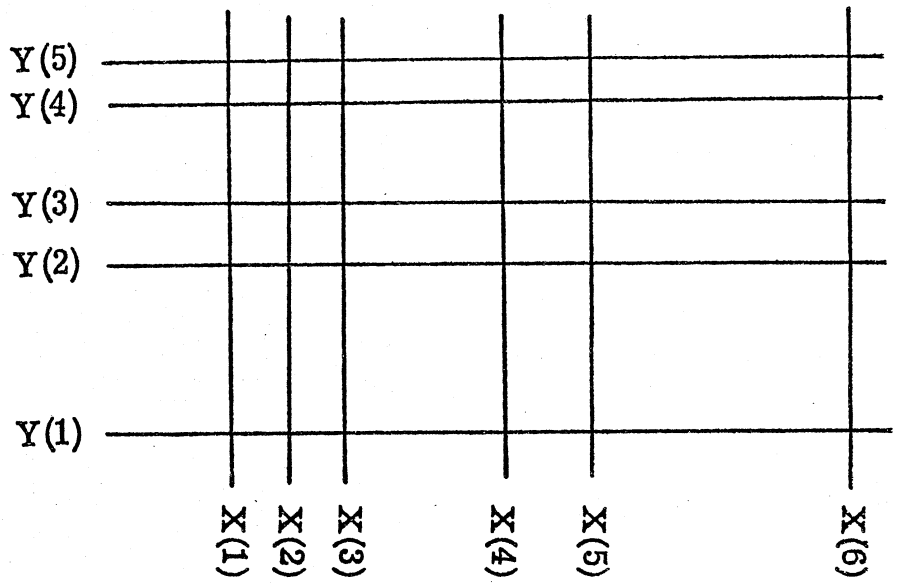


Fig. 1 Specification of the data grid by the X and Y arrays.

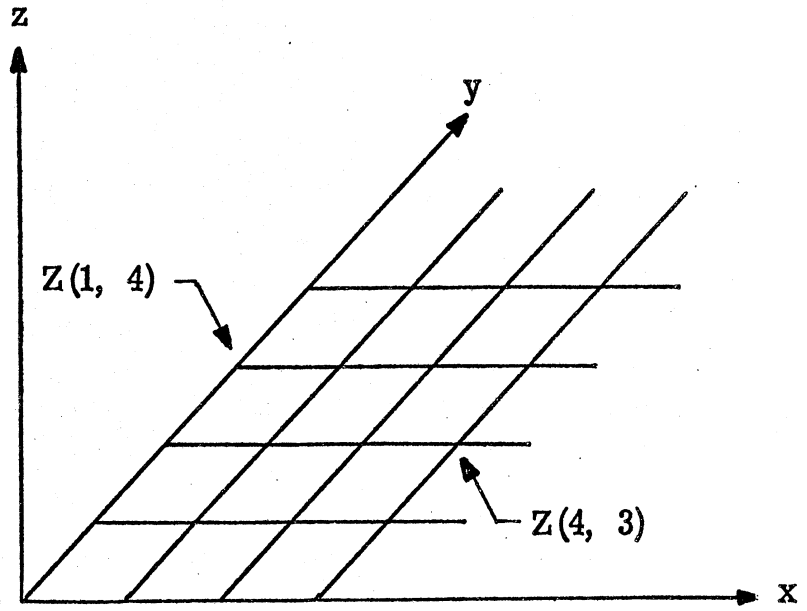


Fig. 2 Association of Z(I,J) values with the data grid.

Z is a two-dimensional array of REAL values which are the z coordinates of the data surface. The first dimension of Z is IDX while the second dimension of Z must be greater than or equal to IY. Each number in array Z gives a height associated with one position in the x-y plane as shown in Fig. 2. Z(I,J) gives a value for the surface at the Ith grid line in the x-direction and the Jth grid line in the y-direction; i.e., the correspondence between the Z array and the X and Y vectors is that

$$Z(I,J) = f(X(I), Y(J))$$

Note: The x-y-z coordinate system conventions are summarized in a later section for quick reference.

IDX an integer constant or an INTEGER*4 expression which must be equal to the value of the first subscript of the array Z as set in the DIMENSION statement. If Z is declared

```
DIMENSION Z(52, 3Ø)
```

then IDX should have the value 52. The data grid itself need not be the same size as array Z although it may not be larger (i.e., IDX must be greater than or equal to IX). The actual size of the grid is given by the arguments IX and IY. The restriction that IDX must be greater than or equal to 2 is imposed by CONSYS for error-checking purposes.

Note: Incorrectly specifying IDX is a major source of error. IDX need not, in general, be equal to the value of IX. If the map contains unusual jagged contours which don't belong, the cause is usually an incorrect IDX.

CZ is a vector of REAL values, at least NC entries long, which contains the z-coordinate values at which contours are to be produced. The coordinates in CZ are in the user's coordinate space.

NC is an integer constant or an INTEGER*4 expression whose value is the number of values in the CZ vector for which contours are to be produced. NC must be greater than or equal to 1; although CONSYS will work for NC = 1, it will function more efficiently on a per-contour basis when NC is greater than one since the overhead involved in contouring is not dependent upon the number of contours produced.

SWCHES is a type LOGICAL vector having five elements used to control various contouring options. The assignment of values is as follows:

SWCHES(1) T: Draw a border line between (X(1),Y(1)) and (X(IX),Y(1)).
F: Do not draw the above border line.

SWCHES(2) T: Draw a border line between (X(1),Y(1)) and (X(1),Y(IY)).
F: Do not draw the above border line.

SWCHES(3) T: Draw a border line between (X(1),Y(IY)) and (X(IX),Y(IY)).
F: Do not draw the above border line.

SWCHES(4) T: Draw a border line between (X(IX),Y(IY)) and (X(IX),Y(1)).
F: Do not draw the above border line.

SWCHES(5) T: Label contours using the labeling scheme described above.
F: Do not label contours.

HOW TO CALL CONEND

CONEND is called when all contouring of a specific data surface has been completed. It deallocates the dynamically acquired arrays which were used in the labeling process during calls to CONTUR to remember the center coordinates of cells which were labeled. (These arrays are not automatically deallocated when CONTUR returns, thus insuring that successive calls to CONTUR will not place labels too closely together.) If CONSYS was used with labeling turned off (i.e., SWCHES(5) is .FALSE.), then CONEND need not be called; if it is called, the call will be ignored and no diagnostic message will be generated.

All three parameters to CONEND are altered by CONEND to return information regarding the performance of the dynamic storage routines for labeling. Consequently, all three CONEND parameters must be the names of INTEGER variables; the parameters may not be integer constants or integer expressions. The user may choose to do nothing with the values returned in these variables, but if he/she calls CONEND, they must be supplied. The calling sequence is

```
CALL CONEND(NREGEN, MAXSIZ, NUSED)
```

where

NREGEN is the INTEGER number of regenerations of dynamic storage which took place.

MAXSIZ is the INTEGER number of bytes of storage allocated by the labeling routines.

NUSED is the INTEGER number of bytes of storage actually used by the labeling routines.

LOGICAL I/O UNITS REFERENCED

CONSYS references the I/O unit (Data Set Reference Number) IOUNIT passed into CONSET. If CONSYS must produce an error comment and CONSET has not been called (this is caused by calling the routines out of order), the unit 19 is used.

SUBPROGRAMS LOADED

CONSYS consists of three control sections, each of which has additional entry points. A COMMON region named CONCOM is also used. Thus CONSYS contributes the following symbols to the load map:

Control Section Name	Entry Points
CONTUR	CONSET CONLBL CONEND
DISTOK	DSINIT DSFINI
CTQQ	CTQQIN
CONCOM (COMMON section)	

SUBPROGRAMS REQUIRED

CONSYS uses the following MTS routines:

RCALL and ADROF in *LIBRARY.
IBCOM#, GETSPA and FREESP in the resident system.

RCALL, ADROF, GETSPA, and FREESP are used to perform dynamic allocation of temporary work arrays (see Appendix D). IBCOM# is part of the MTS FORTRAN run-time system and implements the WRITE statements used to output error messages.

STORAGE REQUIREMENTS

Statically allocated storage:

CONTUR	5Ø24 bytes
DISTOK	1228 bytes
CTQQ	2432 bytes
CONCOM	56 bytes
Total:	8740 bytes

Dynamically allocated storage:

2*NC bytes for sorting contour values
8 bytes per label

ERROR HANDLING

Extensive checks are made to see if parameters supplied by the user are within prescribed ranges. If an error is detected, then if IOUNIT is within the range Ø through 19 inclusive, an error comment is written on DSRN "IOUNIT". A nonzero RETURN statement is then executed and the call is ignored. A summary of the error handling is given below.

Module	Type of Return	Cause
CONLBL	RETURN 1	CONSET not yet called.
	RETURN 2	DIST, NXL, or NYL out of range.
CONTUR	RETURN 1	CONSET not yet called.
	RETURN 2	SWCHES(5) set, but CONLBL not called.
	RETURN 3	IX, IY, IDX, or NC out of range, or X and/or Y not sorted into ascending order.

In the event of an error, the user may send control to a different statement label in his/her calling program by using the "& statement label" construct at the end of the parameter list. For information on this, see "RETURN Statements in a SUBROUTINE Subprogram" on page 98 of the IBM FORTRAN IV Language manual, IBM form GC28-6515 (Ref. 2).

SUMMARY OF COORDINATE SYSTEM CONVENTIONS

To recapitulate, Fig. 3 shows how a grid is described by the X, Y and Z arrays. The z value in Z(I,J) corresponds to the point (X(I),Y(J)) in the x-y plane.

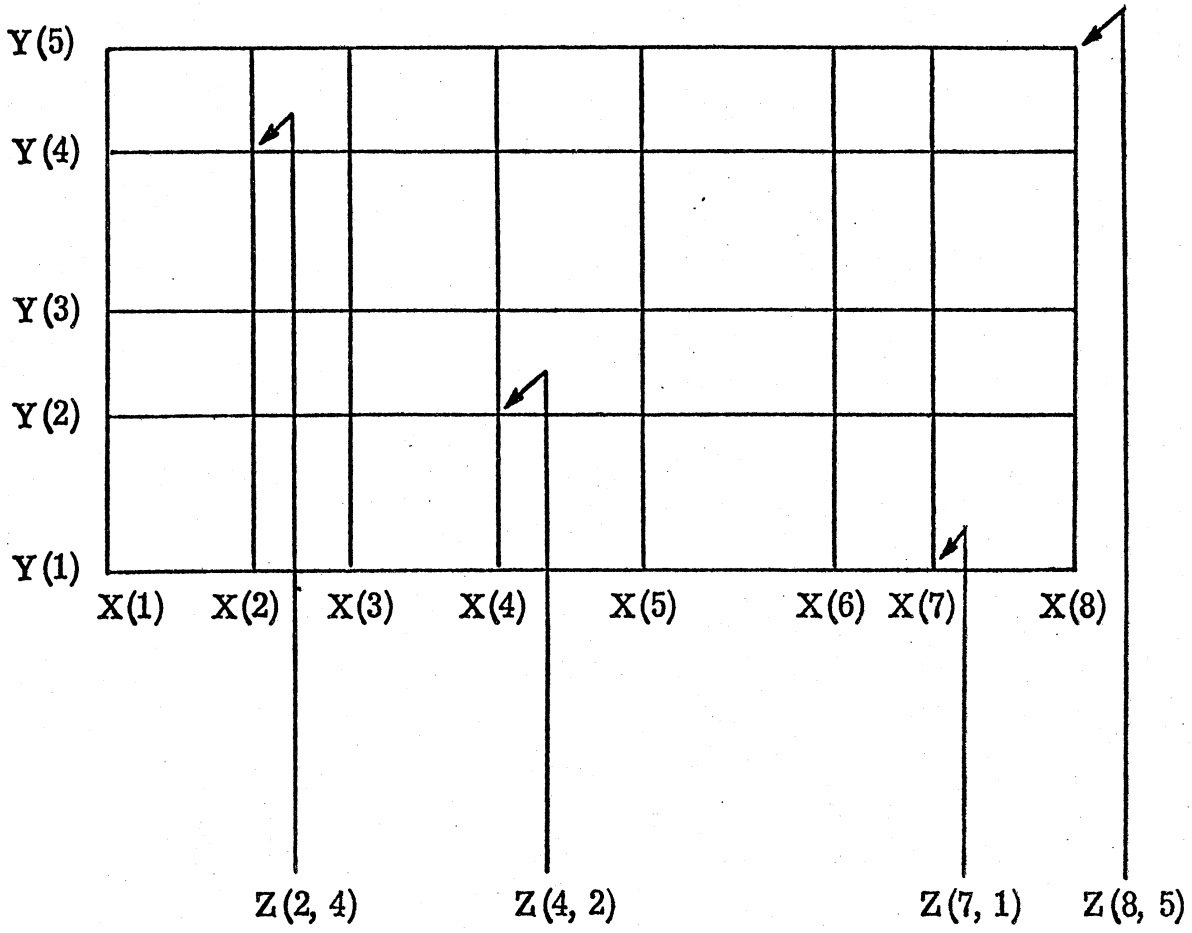


Fig. 3 Example rectangular coordinate grid representation for a data surface with IX = 8 and IY = 5.

The data grid in the x-y plane does not have to be regular. The positions of the corners are established by the values in the X and Y arrays. The user must be careful to store the data points in the Z array in the correct orientation, i.e., Z(I,J) corresponding to X(I),Y(J).

The grid may be of any size. In practice a 50 by 50 grid (only 2,500 points) plotted on a 10 by 10 inch graph produces satisfactorily smooth looking contour lines.

EXAMPLE 1

An architect has some elevation measurements taken on a 300' x 300' parcel of land, for which she would like to draw a contour map of constant elevations. The elevation measurements were taken on a square grid at 10 foot intervals.

She wishes to display the map on a Tektronix 4010 storage tube graphics terminal which has an x-axis range of 0-1023 and a y-axis range of 0-768. The map is to have all four border lines drawn and is to be both centered on the screen and as large as possible. No labeling of contours is needed. Contours are to be produced at elevations of 400, 410 and 420 feet.

Since there are 31 elevation measurements in each direction, the Z array should be dimensioned 31 by 31. The values in the X and Y arrays can be used to position the map on the screen, since CONSYS always works in the user's coordinate system. Thus the Y array should contain 31 entries, equally spaced between 0 and 768 (which results in an inter-entry spacing of 25.6). The X array similarly will contain 31 entries, equally spaced between 128 and 896. These entries will thus be centered about the x-axis, and the square shape of the parcel will be retained in the map.

The T4010 routines in AERO:TEKLIB supply two routines, CKMA and CKVA, which may be used to dispose of map coordinates.

The declarations for the program could be written as follows:

```
REAL X(31), Y(31), Z(31, 31)
REAL CZ(3)/400., 410., 420./
LOGICAL SWCHES(5)/4*.TRUE., .FALSE./
EXTERNAL CKMA, CKVA
```

Assuming error messages are to be written to I/O unit 6, the call to CONSET reads :

```
CALL CONSET(CKMA, CKVA, 6)
```

The X and Y arrays could be filled as follows:

```
DO 10 I = 1, 31
   VALUE = (I - 1)*25.6
   X(I) = VALUE + 128.
   Y(I) = VALUE
10 CONTINUE
```

Finally, the call to CONTUR reads:

```
      CALL CONTUR(X, 31, Y, 31, Z, 31, CZ, 3, SWCHES)
```

A stylized version of the completed program is shown in Fig. 4. Obviously one would want to make provision for more interactive capability then this simple example provides.

```
C Example Program 1 showing the use of CONSYS
C
      REAL X(31), Y(31), Z(31, 31)
      REAL CZ(3)/400., 410., 420./
      LOGICAL SWCHES(5)/4*.TRUE., .FALSE./
      EXTERNAL CKMA, CKVA
C
      CALL CONSET(CKMA, CKVA, 6)
      DO 10 I = 1, 31
         VALUE = (I - 1) *25.6
         X(I) = VALUE + 128.
         Y(I) = VALUE
10       CONTINUE
C
      .
      .
      .   Here, read the elevation data into the Z array
      .
      .
      CALL CKER
      CALL CONTUR (X, 31, Y, 31, Z, 31, CZ, 3, SWCHES)
      CALL CKTRAN
      READ 100, DUMMY
100   FORMAT (A1)
      STOP
      END
```

Fig. 4 Example Program 1.

EXAMPLE 2

On the following pages are a driver program and a labeling subprogram, an example printout showing the interaction with the driver program (Fig. 5), and two example contour maps produced with CONSYS (Figs. 6 and 7).

Subroutine ARRSET(N, X, A, B) simply stores N real

numbers evenly spaced between A and B inclusive into the vector X. Subroutine TC converts a timing measurement by the system routine TIME into minutes, seconds, and milliseconds for printing. All the rest of the subroutines called are available either in the MTS resident system, in the MTS system library *LIBRARY, or in the MTS Calcomp Plotter routines in the file *PLOTSYS.

The example contour maps were produced from two files of actual data taken in the field. They were originally plotted eleven inches wide and photo-reduced for inclusion here. For both plots, the following variables in the driver program had the values shown:

```
FIRSTL=1
LASTL=45
XSIZE=11
HGHT=Ø.Ø875
WIDTH=2
NXL=3,  YHOLD values=16, 31, 46
NYL=2,  YHOLD values=2Ø, 4Ø
```

Values which changed from one plot to another are noted in the contour map captions.

The terminal interaction used to produce Fig. 6 is shown in Fig. 5.

```
C  Example Program 2 showing the use of CONSYS
C
      REAL Z(61,274)
      REAL XP(61), YP(274), CZ(50), XLBL(20), YLBL(20)
      REAL*8 FMT(2)
      INTEGER*4 WIDTH, HOLD(20)
      INTEGER*4 T(2), NSPECT/274/, NLINES/61/
      INTEGER*2 INLEN
      INTEGER*4 RECLEN
      INTEGER*4 INUNIT/0/, MODS/16386/
      INTEGER*4 FIRSTL, LASTL, INC, NBRPLT
      EXTERNAL PENUPS, PENDNS, LABELR
      LOGICAL SWCHES(5)/5*.TRUE./
C
C
C
C  Global initialization:
C
      RECLEN = 4 * NLINES
      CALL CONSET(PENUPS, PENDNS, 8)
      YMAX = FLOAT(NSPECT-1) / 10.
      CALL ARRSET(NSPECT, YP, 0., YMAX)
      PRINT 100
C
10       CONTINUE
         PRINT 101
         READ(5, 201, END=99) FIRSTL, LASTL, XSIZE
         NBRPLT = IABS(FIRSTL - LASTL) + 1
         CALL ARRSET(NLINES, XP, 0., XSIZE)
C
C
C  Read the data into Z.
C
         INC = 1000
         IF( FIRSTL .GT. LASTL )  INC = -1000
         LINE = FIRSTL*1000
         DO 11 I = 1, NBRPLT
            CALL READ(Z(1, I), INLEN, MODS, LINE, INUNIT,
     +                &99)
            IF( INLEN .NE. RECLEN )  GO TO 99
            LINE = LINE + INC
11          CONTINUE
C
12          CONTINUE
            PRINT 102
            READ(5, 202, END=10) NC, CMIN, CMAX
            CALL ARRSET(NC, CZ, CMIN, CMAX)
C
15             CONTINUE
               CALL SETPFX(' ', 1)
               PRINT 103
               READ(5, 203, END=12)  HGHT, WIDTH, FMT
               CALL LABELI(HGHT, FMT)
```

```
17              CONTINUE
                CALL SETPFX(' ', 1)
                PRINT 104
                CALL SETPFX('?', 1)
                READ(5, 204, END=15)  NXL, (HOLD(I),
     +                                      I = 1, NXL)
                DO 19 I = 1, NXL
                   XLBL(I) = XP(HOLD(I))
19                 CONTINUE
21              CONTINUE
                CALL SETPFX(' ', 1)
                PRINT 105
                CALL SETPFX('?', 1)
                READ(5, 205, END=17)  NYL, (HOLD(I),
     +                                      I = 1, NYL)
                CALL SETPFX(' ', 1)
                DO 23 I = 1, NYL
                   YLBL(I) = YP(HOLD(I))
23                 CONTINUE
C
C  Call CONTUR to do the plot.
C
                CALL PGNHDR
                CALL PLTXMX(XSIZE + 7.5)
                CALL PXMARG(4.0)
                CALL PLTOFS(0.,1., 0.,1., 4., 1.)
                CALL CONLBL(LABELR, (WIDTH+2)*HGHT, NXL,
     +                      XLBL, NYL, YLBL, &99)
                PRINT 120
                CALL TIME(0)
                CALL CONTUR(XP, NLINES, YP, NBRPLT, Z, 61,
     +                      CZ, NC, SWCHES, &99)
                CALL TIME(3, 0, T)
                CALL CONEND(NREGEN, MAXSIZ, NUSED)
C
C  Put tick marks along the lower and right-hand edges.
C
                YL = YP(1)
                YLM = YL - 0.15
                DO 25 I = 1, NLINES, 5
                   XI = XP(I)
                   CALL PENUPS(XI, YL)
                   CALL PENDNS(XI, YLM)
25                 CONTINUE
                XR = XP(NLINES)
                XRM = XR + 0.15
                DO 27 I = 1, NBRPLT, 5
                   YI = YP(I)
                   CALL PENUPS(XR, YI)
                   CALL PENDNS(XRM, YI)
27                 CONTINUE
                CALL PLTEND
C
```

```
C  Compute how long it took and how much storage was used,
C  and print these numbers out.
C
                CALL TC(T(1), M, S, MS)
                PRINT 121, M, S, MS
                CALL TC(T(2), M, S, MS)
                PRINT 122, M, S, MS
                PRINT 123,  NREGEN, MAXSIZ, NUSED
                GO TO 21
C
99     CONTINUE
       CALL MTS
       GO TO 10
C
C
C
100    FORMAT('-SPECTRAL CONTOURING PROGRAM',/,' ')
101    FORMAT('&ENTER FIRST #, LAST #, & XSIZE IN INCHES:')
102    FORMAT('&ENTER # OF CONTOURS, MINVAL, & MAXVAL:')
103    FORMAT('&ENTER CHARACTER HEIGHT, # OF CHARS, & ',
     +        'FORMAT W/ *:')
104    FORMAT(' ENTER FOR X AXIS, # OF LABEL LINES & GRID ',
     +        'LOCN OF THOSE LINES.')
105    FORMAT(' ENTER FOR Y AXIS, # OF LABEL LINES & GRID ',
     +        'LOCN OF THOSE LINES.')
120    FORMAT(' CONTUR WILL NOW BE CALLED.')
121    FORMAT('0CPU TIME:',T20,I3,' MINS,',I3,'.',I3,
     +        ' SECS.')
122    FORMAT('0ELAPSED TIME:',T20,I3,' MINS,',I3,'.',I3,
     +        ' SECS.')
123    FORMAT('0', I5,' REGENERATIONS; ',I5,' BYTES ',
     +        'ALLOCATED, ',I5,' USED.')
C
203    FORMAT(F10.2, I10, 2A8)
204    FORMAT(21I10)
205    FORMAT(21I10)
201    FORMAT(2I10, F10.5)
202    FORMAT(I10, 2F10.5)
       END



       SUBROUTINE LABELR(XC, YC, Z0)
C
C  LABELR is called from CONTUR to output a label on a
C  contour.  XC and YC are the coordinates of the cell which
C  contains the contour, and Z0 is the contour value.
C
       REAL*8 FMT(1)
       REAL XC, YC, Z0
       REAL*4 PNUMBR, PSYMLN, DONT/-1.0/, RELATV/1.0/
       INTEGER*4 NCHAR
```

```
C
C
C
      NCHAR = PNUMBR(0., 0., HGHT, Z0, 0., FMT, DONT)
      X0 = XC - 0.5*PSYMLN(HGHT, NCHAR)
      Y0 = YC - 0.5*HGHT
      NCHAR = PNUMBR(X0, Y0, HGHT, Z0, 0., FMT, RELATV)
      RETURN
C
C
C
      ENTRY LABELI(HGHT, FMT)
C
C  LABELI is called from the display driver program in order
C  to pass in the values of the HGHT and FMT parameters
C  to LABELR.
C
      RETURN
      END
```

```
 SPECTRAL CONTOURING PROGRAM

 ENTER FIRST #, LAST #, & XSIZE IN INCHES: 1,45,11.,
 ENTER # OF CONTOURS, MINVAL, & MAXVAL: 11,-3.,2.,
 ENTER CHARACTER HEIGHT, # OF CHARS, & FORMAT W/ *: Ø.Ø875,2,WJF1.1.4*
 ENTER FOR X AXIS, # OF LABEL LINES & GRID LOCN OF THOSE LINES.
?3,16,31,46,
 ENTER FOR Y AXIS, # OF LABEL LINES & GRID LOCN OF THOSE LINES.
?2,2Ø,4Ø,
 CONTUR WILL NOW BE CALLED.

 CPU TIME:            Ø MINS,   3.771 SECS.

 ELAPSED TIME:        1 MINS, 15.726 SECS.

     1 REGENERATIONS;      576 BYTES ALLOCATED,   3Ø4 USED.
```

Fig. 5 The computer terminal interaction with Example Program 2 which produced Fig. 6. The CPU time given is for an IBM /360-67 computer.

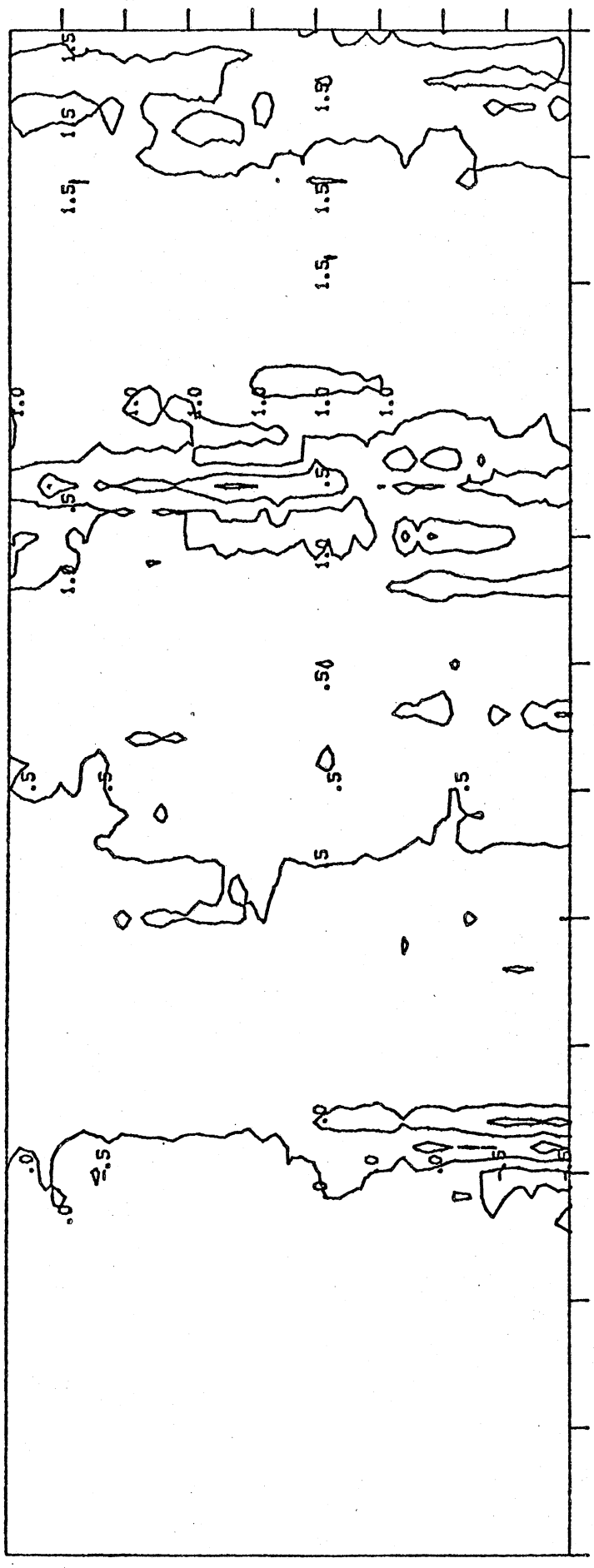


Fig. 6 Example contour plot. Eleven contours at intervals of Ø.5 from -3 to +2. 3.771 CPU seconds were used on an IBM /36Ø-67 computer. One labeling storage regeneration was done; 576 bytes were allocated, 3Ø4 were used.

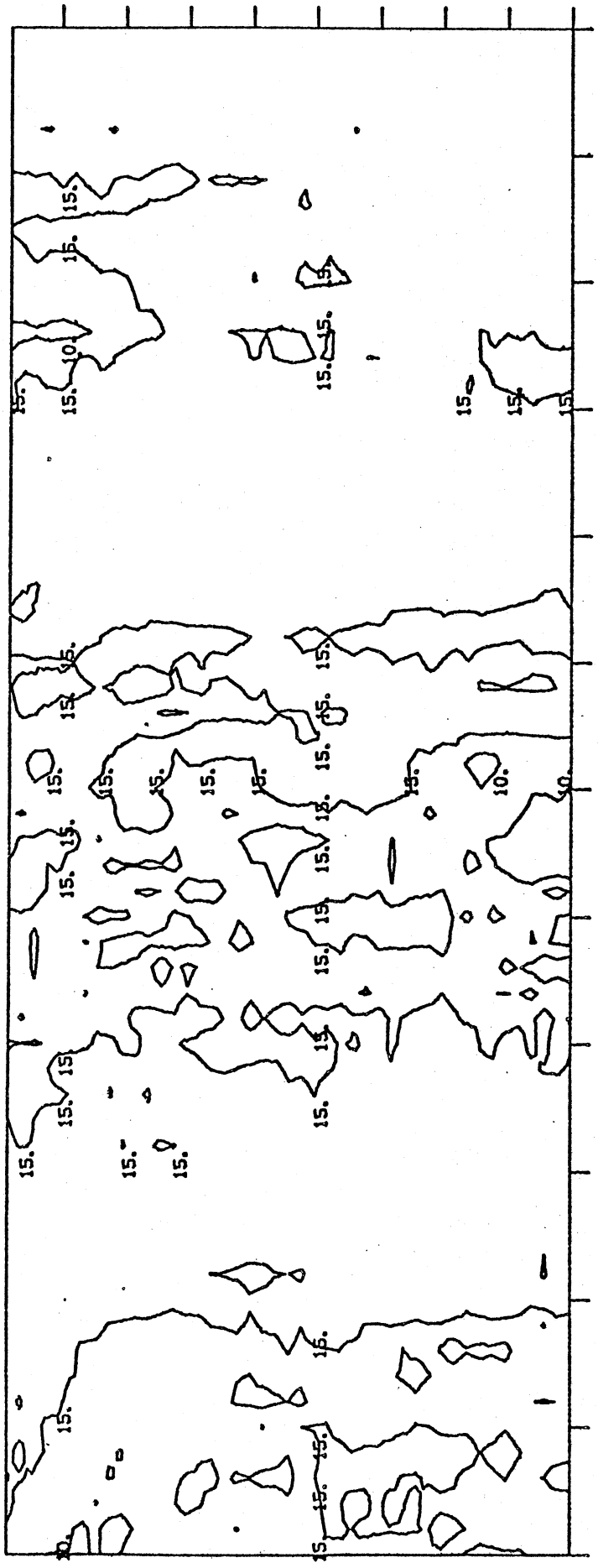


Fig. 7 Example contour plot. Three contours at intervals of 5 from 5 to 15. 3.662 CPU seconds were used on an IBM /360-67 computer. One labeling storage regeneration was done; 576 bytes were allocated, 344 were used.

Appendix A

GETTING THE DATA ONTO A GRID

Often it is impossible to gather data on a rectangular grid, let alone a regular grid. In these cases, before using CONSYS the user must find some way of generating equivalent gridded data from the data on hand. Some commonly used methods for this purpose are linear interpolation, Fourier series band-limited interpolation, polynomial interpolation, cubic spline interpolation, and membrane-with-point-loads interpolation. It would be impossible to do justice to any of these methods here, and they are still the subject of much ongoing research. Nonetheless, the user should be aware that the method he/she uses must be both mathematically and physically justifiable (i.e., it must be good science), and further that many interpolation methods place constraints on the data which the user must observe at the time the data are taken. In other words, an interpolation method must be scientifically chosen before the data are taken and not afterward.

Three potentially useful sources in the area of interpolation onto a grid are Refs. 3-5. In addition, workers in the fields of architecture, civil engineering, geology, and geography (among others) may be of some help in choosing an interpolation method. Further, journals in these fields often contain papers on interpolation.

Appendix B

METHOD

CONSYS uses an algorithm that is a modification of the algorithm of Hartwig (Ref. 1). The modifications appear to have cut the time required to contour large surfaces by a factor of two. The modified algorithm reads approximately as follows:

```
sort the values in CZ into ascending order;
for each cell in the grid do;
    for each contour value in CZ do;
        if the cell intersects the contour value at all
            then do;
              using the fact that CZ is sorted, find the
              smallest and largest values in CZ which do
              intersect the cell;
              call CTQQ (which utilizes Hartwig's
              algorithm) to produce the contour
              coordinates for the cell;
            end;
    end;
end;
```

The speedup is over Hartwig's original algorithm is attained in the step "if the cell intersects the contour value at all," wherein the cell is very quickly rejected if no contours pass through it; this is the case most of the time.

In the subroutine CTQQ, contour coordinates are produced for a rectangular cell according to Hartwig's method. Briefly, the center coordinates of the cell are computed as averages of the corner coordinates. The corners and the center are then imagined to be connected with line segments which are in turn numbered 1 through 8 in a clockwise direction; see Fig. B1.

Each segment is then tested in turn to see if the required contour intersects with it. If there is an intersection, then linear interpolation along the line segment is used to find the x and y coordinates of the intersection, and the coordinates are saved temporarily in an array. After all 8 line segments have been processed in this way, the coordinate values which were saved are sorted into proper order to form contour lines and are output by calling the user specified output routines. Figure B2 shows an example of a contour line crossing line segments 1, 2, 4 and 5. For more information, consult Ref. 1.

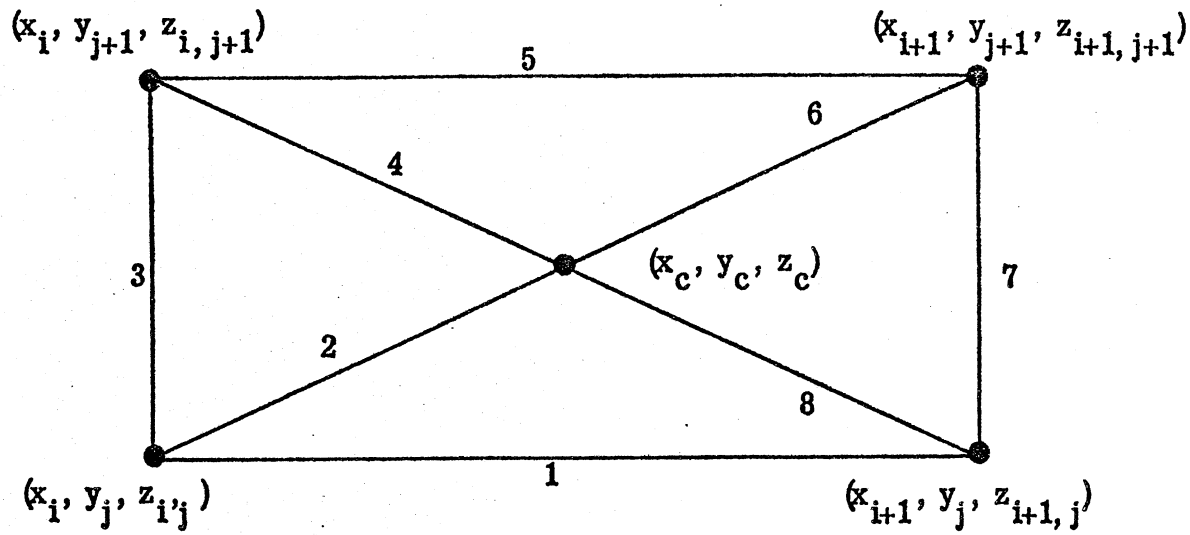


Fig. B1 Hartwig' numbering convention for the lines overlaying a grid cell.

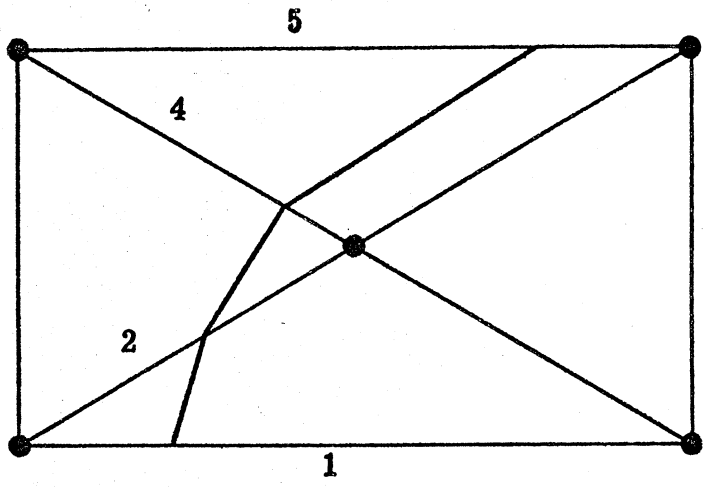


Fig. b2 A contour line crossing line segments 1, 2, 4, and 5.

Appendix C

THE STRUCTURE OF THE CONTOUR MAP DISPLAY FILE

Because of the method used (see Appendix B), the graphical output (called a display file) of CONSYS consists of many short visible lines interspersed with nearly as many invisible moves. If the graphical output device is capable of good resetability (and most are quite adequate), this sort of display file will produce a satisfactory map.

However, the display file will occupy more room in memory and take longer to draw than necessary. (The choppy character of the display file is a consequence of the fact that cheap display file storage was traded for expensive CPU time in the implementation of CONSYS.) As a remedy, it would be possible to construct a routine which would accept the CONSYS display file and quickly delete most of the invisible moves. Such a routine has not yet been written, however, and it is unclear that it would be cost effective especially when its development costs are considered. If someone wishes to construct such a routine, he/she is urged to contact the author for ideas on implementation.

Appendix D

HARDING'S METHOD FOR DYNAMIC ALLOCATION OF ARRAYS USED IN CONSYS

Leonard J. Harding of the University of Michigan Computing Center has related a method of allocating and using arrays whose length is determined at execution time. Harding's method is used within CONSYS to dynamically allocate temporary work arrays. The method is known to be suitable for FORTRAN programs compiled under the IBM G- and H-level FORTRAN compilers and may be suitable for other programming language/compiler combinations as well. The method depends critically upon the lack of run time bounds checking for arrays, and thus is not suitable for use with the WATFOR and WATFIV FORTRAN systems.

The basic trick is to recognize that if no run time checks are made on array bounds, it is possible to address any location in available storage by an appropriate choice of subscript. By taking advantage of this trick in a disciplined manner, we can build an algorithm for dynamically allocating, accessing, and deallocating arrays having any number of dimensions. The discussion below is restricted to one-dimensional arrays; the extension to multi-dimensional arrays is basically trivial.

The four steps in using the Harding method are 1) Allocation, 2) Offset Value Computation, 3) Array Referencing, and 4) Deallocation. Each of these steps is illustrated below using FORTRAN. In subroutine calling sequences, those parameters which are irrelevant to the problem of transporting the CONSYS code will be denoted by "xxx".

ALLOCATION

On the System/360 and /370, the addressable unit of storage is the 8-bit byte. FORTRAN data types which can be used in arrays can have 1, 2, 4, 8, or 16 bytes per array element. Here we assume that we are allocating an array of 4-byte floating point numbers, which are declared using the "REAL*4" keyword in IBM FORTRAN. Suppose for example that we wish to allocate an array whose name is to be "R" and whose length (in array elements) is found in the INTEGER variable "NR". The following statements will invoke the system routine "GETSPA" to allocate space for the array.

```
EXTERNAL GETSPA
CALL RCALL(GETSPA, xxx, xxx, SEGNO+NR*4, xxx, xxx, RADR)
```

Note that NR participates as part of the fourth parameter, which indicates the size of the storage region wanted; we multiply NR by 4 because storage is allocated in bytes rather than array elements. SEGNO is added because in the Michigan operating system, one can specify a storage segment number in which storage is to be allocated; in this case, because of an anomaly in the code produced by the FORTRAN compilers, we force the segment in which the storage is allocated to lie at a higher storage address than the program in which the so-called "base address" for R is situated (base address is explained below). This restriction on the position of the allocated storage may not be present in other machine/compiler designs; the problem is associated with the generation of negative numbers as storage addresses during the array accessing process.

GETSPA returns the storage address of the allocated region in the variable RADR, which must be declared to be of INTEGER type.

In addition to calling GETSPA to allocate space for the array, the user must also statically allocate one element for the array in his FORTRAN program. This can be done using the DIMENSION statement or one of its variants. Allocating this element does two things: 1) it tells the compiler that subscripts are legal constructs to follow the name of the array, and 2) it causes the compiler to assign a unique storage address, called the base address, to the first element in the array. For example, the following statement statically allocates one element for R:

```
REAL*4 R(1)
```

As a consequence of the static allocation of R, whenever the user references R with a subscript, the compiler produces code which adds the value of the subscript (suitably adjusted for the fact that R is composed of 4-byte elements) to the storage address of the first element of the statically allocated R (the base address).

ARRAY OFFSET COMPUTATION AND ARRAY REFERENCING

Once the value of RADR is known and R has been statically allocated, one can compute a so-called "array offset" which is needed to reference an individual array element.

The array offset for a dynamically allocated array is that numeric integer value which when added to an array subscript causes the computer to access the dynamically allocated area instead of the statically allocated array. For example, let the array offset for the array R be stored in the variable ROFS. Then the usual array reference R(1) to the first element of a statically allocated R is written as R(1+ROFS) to access the first element position in the dynamically allocated array area. Similarly, to refer to the Ith element of the dynamically allocated R array, we write R(I+ROFS).

To compute the value of the array offset, we must have access to an INTEGER-valued function subprogram which returns the storage address of its single argument. Borrowing from the Michigan operating system, we will call this function ADROF. The array offset for the R array can then be calculated by the following assignment statement:

```
ROFS = (RADR - ADROF(R)) / 4
```

In this statement, we divide the distance in storage address space between the first element of the dynamic R and the static R by the number of addressable storage locations per array element. If the method is used on a word-oriented machine such as the PDP-10 or the CDC 6600, the division need not be done since there is a one-to-one correspondence between array elements and storage addresses.

DEALLOCATION

In code using Harding's method and which is written to run on the Michigan system, storage can be deallocated by calling the system storage deallocation routine FREESP. The form of the call is

```
CALL RCALL(FREESP, xxx, xxx, RADR, xxx)
```

where RADR is the storage address of a region acquired by a call on GETSPA.

CONVERSION OF PROGRAMS

Programs which use Harding's method can be converted to run on other machines in two different ways: 1) change each dynamically allocated array into a statically allocated array by inserting the appropriate DIMENSION statement, or 2) if possible, write and install the subroutines which will allow the Harding method to be used on the new machine.

In the first conversion method, one removes all the calls to GETSPA and FREESP, and then dimensions all arrays to some acceptably large size. One may then use a text editor to change all array references of the form R(J+ROFS) to R(J). Alternatively, one may use a DATA statement to preset the value of ROFS to zero, leaving all array references unchanged; this will of course cause the compiler to produce an extra add instruction for every subscript calculation (the effect of which is to add in a zero ROFS value), but this conversion of the program text takes much less time. This last tack has much to recommend it if the storage burden for the extra add instructions can be borne, and if there is not a plethora of dynamic array references in the innermost loops of the program to be converted.

In the second conversion method, one must have a run-time environment in which it possible to allocate storage from within a running FORTRAN program. In addition, one must have an ADROF function and a compiler/machine architecture which can form the proper storage address for the dynamically allocated array. If the second conversion method is possible, it has much to recommend itself, since dynamically allocated arrays are quite useful in themselves, conversion of code which uses dynamically allocated arrays to use static arrays is quite simple, and the method is simple to learn, explain, and use. Moreover, it is easy to discipline oneself in the use of dynamically allocated arrays so that the errors which occur due to their use are acceptably few in number; i.e., they are not an especially error-prone construct.

Appendix E

LISTING OF THE SOURCE CODE

```
      SUBROUTINE CONTUR(X, IX, Y, IY, Z, IDX, CZ, NC, SWCHES)
C
C  Although care has been taken to ensure the correct
C  functioning of the CONSYS system, neither the author
C  nor The University of Michigan shall be liable for any
C  direct or indirect, incidental, consequential, or
C  specific damages of any kind or from any cause
C  whatsoever arising out of or in any way connected with
C  the use or performance of the CONSYS system or its
C  documentation.
C
C  CONTUR produces coordinate pairs for drawing a picture
C  which is a contour map of the data in the array Z.  Z is a
C  data surface, i.e., Z(I, J) = f(X(I), Y(J)).
C  The basic algorithm for this routine was suggested by:
C    G. W. Hartwig, "CONTUR - A Fortran IV Subroutine for Plot-
C    ing Contour Lines," Ballistic Research Laboratories Memo-
C    randum Report # 2282, Aberdeen Proving Ground, Maryland,
C    March, 1973.  (NTIS accession number AD-760 437).
C  This routine comprises the first half of Hartwig's algorithm;
C  it has been modified to reflect the fact that the most com-
C  putationally efficient procedure is to quickly reject surface
C  cells which contain no contours.  If a cell does contain one
C  or more contours, subroutine CTQQ is called to complete
C  Hartwig's procedure, i.e., actually form the coordinate pairs
C  for the contour lines.
C
C  The following code is for version 1.2 of CONSYS, produced
C  15 May, 1976.  GNC
C
      REAL Z(IDX, IY), X(1), Y(1), CZ(1)
      REAL MINDIS, XLOC(NXL), YLOC(NYL)
      LOGICAL NOWRIT/.FALSE./
      LOGICAL SORTED, SWCHES(5), SETUP1/.FALSE./,
     +        SETUP2/.FALSE./
      LOGICAL LBLLOC, DISTOK
      INTEGER*4 IOMAX/19/, ERRUNT/19/
      INTEGER*4 PARERR, CONCNT, CCP1, NTEMP
C  SEGA tells GETSPA to allocate storage in segment 10 - which
C  is a reasonable guarantee that the storage will be allocated
C  at a higher storage address than CONSYS.
      INTEGER*4 ADROF, GETGR0/3/, SEGA/Z0A000000/, PTRADR
      INTEGER*4 PL, PH
      INTEGER*2 PTR(1)
      EXTERNAL GETSPA, FREESP, LABELR, CKMA, CKVA
```

```
C
      COMMON /CONCOM/ LOWER, UPPER, OFS, XL, XR, XC, YL, YU, YC,
     +                ZLL, ZUL, ZLR, ZUR
      INTEGER*4 LOWER, UPPER, OFS
C
C
C
C
C  Check for obvious errors in the parameters.
C
      IF( .NOT. SETUP1 )  GO TO 980
      IF( SWCHES(5) .AND. .NOT. SETUP2 )  GO TO 983
C
      PARERR = 0
      IF( IX .GT. IDX )  PARERR = PARERR + 1
      IF( IX .LT. 2 )    PARERR = PARERR + 1
      IF( IY .LT. 2 )    PARERR = PARERR + 1
      IF( IDX .LT. 2 )   PARERR = PARERR + 1
      IF( NC .LT. 1 )    PARERR = PARERR + 1
      IF( PARERR .NE. 0 )  GO TO 986
C
C  Check X and Y to ensure they are in strictly ascending
C  order.  Note that the error message which is written if they
C  are not scares the user into checking both arrays, even
C  though the code doesn't check Y if X is bad.
C
      DO 880 I = 2, IX
         IF( X(I) .LE. X(I-1) )  GO TO 988
880      CONTINUE
      DO 881 I = 2, IY
         IF( Y(I) .LE. Y(I-1) )  GO TO 988
881      CONTINUE
C
C  Sort the array of contour values.
C
      IF( NC .EQ. 1 )  GO TO 5
         CALL RCALL(GETSPA, 2, GETGR0, SEGA+2*NC,
     +                      2, TRASH, PTRADR)
         OFS = (PTRADR - ADROF(PTR))/2
         DO 1 M = 1, NC
            PTR(M+OFS) = M
1           CONTINUE
         M = NC-1
2           CONTINUE
            SORTED = .TRUE.
            DO 4 K = 1, M
               PL = PTR(K+OFS)
               PH = PTR(1+K+OFS)
               IF( CZ(PH) .GT. CZ(PL) )  GO TO 3
                  PTR(K+OFS) = PH
                  PTR(1+K+OFS) = PL
                  SORTED = .FALSE.
3              CONTINUE
```

```
4              CONTINUE
               IF( SORTED )  GO TO 6
               M = M - 1
               IF( M .GE. 1 )  GO TO 2
            GO TO 6
5        CONTINUE
            OFS = Ø
            PTR(1) = 1
            GO TO 6
6        CONTINUE
         CZMAX = CZ(PTR(NC+OFS))
         CZMIN = CZ(PTR(1 +OFS))
         CALL CTQQIN(PTR, CZ, CKMA, CKVA)
C
C  Begin the contouring process by looking at each cell in the
C  surface in turn.  For each cell, we ask the question, "Does
C  this cell contain any contour lines at the user-specified
C  values in the CZ array?"  If the answer is no, we immediately
C  proceed to the next cell.  If the answer is yes, we find
C  the lower and upper limits in the sorted CZ array of contour
C  values which intersect this cell, and pass this information
C  and the cell coordinates to subroutines CTQQ (via CONCOM)
C  where the coordinates for the contours are produced.
C
         IYM1 = IY - 1
         YU = Y(1)
         DO 38 J = 1, IYM1
            YL = YU
            YU = Y(J+1)
            YC = Ø.5*(YL + YU)
            XR = X(1)
            ZLR = Z(1, J)
            ZUR = Z(1, J+1)
            DO 37 I = 2, IX
               XL = XR
               XR = X(I)
               ZLL = ZLR
               ZUL = ZUR
               ZLR = Z(I, J)
               ZUR = Z(I, J+1)
               ZMIN = AMIN1(ZLL, ZUL, ZLR, ZUR)
               IF( ZMIN .GT. CZMAX )  GO TO 37
               ZMAX = AMAX1(ZLL, ZUL, ZLR, ZUR)
               IF( ZMAX .LT. CZMIN )  GO TO 37
               IF( ZMAX .EQ. ZMIN )  GO TO 37
                  DO 12 CONCNT = 1, NC
                     ZØ = CZ(PTR(CONCNT+OFS))
                     IF( ZØ .LT. ZMIN )  GO TO 12
                        IF( ZØ .GT. ZMAX )  GO TO 37
                           LOWER = CONCNT
                           UPPER = LOWER
                           IF( UPPER .EQ. NC )  GO TO 14
                              CCP1 = CONCNT + 1
```

```
                        DO 11 II = CCP1, NC
                           IF( ZMAX .LT. CZ(PTR(II+OFS)) )
     +                                              GO TO 14
                           UPPER = II
11                         CONTINUE
                        GO TO 14
12                   CONTINUE
14                CALL CTQQ
C
C  If the user specified via SWCHES(5) that labels are to be
C  drawn, find out here if this cell is a candidate for
C  labeling, and if it is, call LABELR to draw the label.
C
                  IF( .NOT. SWCHES(5) )  GO TO 29
                     LBLLOC = .FALSE.
                     IF( NXL .EQ. 0 )  GO TO 23
                        DO 22 M = 1, NXL
                           IF( .NOT.(XL .LE. XLOC(M) .AND.
     +                                 XLOC(M) .LT. XR) )  GO TO 22
                              LBLLOC = .TRUE.
                              GO TO 27
22                         CONTINUE
23                   CONTINUE
                     IF( NYL .EQ. 0 )  GO TO 27
                        DO 26 M = 1, NYL
                           IF( .NOT.(YL .LE. YLOC(M) .AND.
     +                                 YLOC(M) .LT. YU) )  GO TO 26
                              LBLLOC = .TRUE.
                              GO TO 27
26                         CONTINUE
27                   CONTINUE
                     IF( .NOT. LBLLOC )  GO TO 28
                        IF( DISTOK(XC, YC) )
     +                                    CALL LABELR(XC, YC, Z0)
28                   CONTINUE
29                CONTINUE
37             CONTINUE
38          CONTINUE
C
C  Now give back the space we acquired in order to sort the CZ
C  array.
C
         CALL RCALL(FREESP, 2, 0, PTRADR, 0)
C
C  Draw border lines, if the user indicated via the SWCHES
C  vector that they are wanted.
C
         XL = X(1)
         XR = X(IX)
         YL = Y(1)
         YU = Y(IY)
         IF( .NOT. SWCHES(1) )  GO TO 42
            CALL CKMA(XR, YL)
```

```
         CALL CKVA(XL, YL)
42     CONTINUE
       IF( .NOT. SWCHES(2) )  GO TO 44
          CALL CKMA(XL, YL)
          CALL CKVA(XL, YU)
44     CONTINUE
       IF( .NOT. SWCHES(3) )  GO TO 46
          CALL CKMA(XL, YU)
          CALL CKVA(XR, YU)
46     CONTINUE
       IF( .NOT. SWCHES(4) )  GO TO 48
          CALL CKMA(XR, YU)
          CALL CKVA(XR, YL)
48     CONTINUE
       RETURN
C
C
C  Handle bad parameters in the CONTUR call here.
C
980    CONTINUE
       IF( NOWRIT )  GO TO 981
          WRITE(ERRUNT, 995)
          WRITE(ERRUNT, 999)
981    CONTINUE
       RETURN 1
C
983    CONTINUE
       IF( NOWRIT )  GO TO 984
          WRITE(ERRUNT, 996)
          WRITE(ERRUNT, 999)
984    CONTINUE
       RETURN 2
C
986    CONTINUE
       IF( NOWRIT )  GO TO 987
          WRITE(ERRUNT, 997)  PARERR, IX, IY, IDX, NC
          WRITE(ERRUNT, 999)
987    CONTINUE
       RETURN 3
C
988    CONTINUE
       IF( NOWRIT )  GO TO 989
          WRITE(ERRUNT, 994)
          WRITE(ERRUNT, 999)
989    CONTINUE
       RETURN 3
C
C  Format statements for CONTUR error comments.
C
994    FORMAT(' **** ERROR: SUBROUTINE "CONTUR" HAS BEEN CALLED',
     +        'WITH EITHER THE X OR',/,6X,'THE Y VECTOR (OR ',
     +        'BOTH) NOT IN STRICTLY ASCENDING ORDER.')
995    FORMAT(' **** ERROR: SUBROUTINE "CONTUR" HAS BEEN CALLED',
```

```
     +       ' WITH NO',/,6X,'PRECEDING INITIALIZATION CALL TO',
     +       ' SUBROUTINE "CONSET".')
996   FORMAT(' **** ERROR: SUBROUTINE "CONTUR" HAS BEEN CALLED',
     +       ' WITH SWCHES(5)',/,6X,'SPECIFYING CONTOUR ',
     +       'LABELING TO BE DONE, BUT AN INITIALIZATION',/,6X,
     +       'CALL TO SUBROUTINE "CONLBL" HAS NOT YET BEEN ',
     +       'MADE.')
997   FORMAT(' **** THERE ARE ',I1,' ERROR(S) IN THE PARA',
     +       'METERS IN A CALL',/,6X,'TO SUBROUTINE "CONTUR".',
     +       '  DIAGNOSTIC INFORMATION FOLLOWS:',/,6X,'IX = ',
     +       I15,'   IY = ',I15,/,6X,'IDX = ',I15,'   NC = ',
     +       I15)
999   FORMAT(' **** DUE TO THE ABOVE ERROR, CONTUR WILL NOT ',
     +       'DRAW A CONTOUR MAP,',/,6X,'BUT WILL INSTEAD ',
     +       'IGNORE THE CALL.')
C
C
C
C
C
      ENTRY CONSET(CKMA, CKVA, IOUNIT)
C
C  CONSET is called to set up the names of the functions to
C  dispose of picture coordinates, and to pass in the logical
C  I/O unit upon which error comments (if any) will be written.
C
C
C
C
      ERRUNT = IOUNIT
      NOWRIT = ERRUNT .LT. 0  .OR.  ERRUNT .GT. IOMAX
      SETUP1 = .TRUE.
      RETURN
C
C
C
C
C
      ENTRY CONLBL(LABELR, MINDIS, NXL, XLOC, NYL, YLOC)
C
C  CONLBL is called to set up CONTUR for outputting labels of
C  contour line values somewhere near the contour line itself.
C
C
C
C
      IF( .NOT. SETUP1 )  GO TO 1990
      PARERR = 0
      IF( MINDIS .LT. 0. )  PARERR = PARERR + 1
      IF( NXL .LT. 0 )      PARERR = PARERR + 1
      IF( NYL .LT. 0 )      PARERR = PARERR + 1
      IF( NXL+NYL .LT. 1 )  PARERR = PARERR + 1
      IF( PARERR .NE. 0 )  GO TO 1993
```

```
         SETUP2 = .TRUE.
         NTEMP = MAXØ(NXL, NYL, NXL*NYL)
         CALL DSINIT(NTEMP, MINDIS)
         RETURN
C
C  Handle errors in parameters to CONLBL here.
C
1990  CONTINUE
      IF( NOWRIT )  GO TO 1991
         WRITE(ERRUNT, 1997)
         WRITE(ERRUNT, 1999)
1991  CONTINUE
      RETURN 1
C
1993  CONTINUE
      IF( NOWRIT )  GO TO 1994
         WRITE(ERRUNT, 1998)  PARERR, MINDIS, NXL, NYL
         WRITE(ERRUNT, 1999)
1994  CONTINUE
      RETURN 2
C
C  Format statements for CONLBL error comments.
C
1997  FORMAT(' **** ERROR: SUBROUTINE "CONLBL" HAS BEEN CALLED',
     +       ' WITH NO',/,6X,'PRECEDING INITIALIZATION CALL ',
     +       'TO SUBROUTINE "CONSET".')
1998  FORMAT(' **** THERE ARE ',I1,' ERROR(S) IN THE ',
     +       'PARAMETERS IN A CALL TO',/,6X,'SUBROUTINE ',
     +       '"CONLBL".  DIAGNOSTIC INFORMATION FOLLOWS:',/,6X,
     +       'MINDIS = ',G13.6,/,6X,'NXL = ',I15,'  NYL = ',I15)
1999  FORMAT(' **** DUE TO THE ABOVE ERROR, CONLBL WILL NOT ',
     +       'INITIALIZE THE',/,6X,'CONTOUR LABELING ROUTINES,',
     +       ' BUT WILL INSTEAD IGNORE THE CALL.')
C
C
C
C
C
      ENTRY CONEND(/NREGEN/, /MAXSIZ/, /NUSED/)
C
C  CONEND is called after a complete contour map has been
C  produced in order to release dynamically acquired storage.
C
      CALL DSFINI(NREGEN, MAXSIZ, NUSED)
      RETURN
      END



      LOGICAL FUNCTION DISTOK(X, Y)
C
C  Although care has been  taken  to  ensure  the  correct
```

```
C  functioning  of  the  CONSYS system, neither the author
C  nor The University of Michigan shall be liable for  any
C  direct   or  indirect,  incidental,  consequential,  or
C  specific  damages  of  any  kind  or  from  any   cause
C  whatsoever  arising out of or in any way connected with
C  the use or performance of  the  CONSYS  system  or  its
C  documentation.
C
C  CONTUR calls DISTOK to see whether a contour label can
C  be placed at (X, Y) and be more than MINDIS units away
C  from any other label previously placed on the contour map.
C  If a label can be safely placed on the map, DISTOK returns
C  the value .TRUE. after saving the values of X and Y in a
C  dynamically allocated local array.  If the label would
C  fall within MINDIS units from a previous label, DISTOK
C  simply returns the value .FALSE.
C
C  The following code is for version 1.2 of CONSYS, produced
C  15 May, 1976.  GNC
C
      REAL X, Y, MINDIS
      REAL COORD(1)
      INTEGER*4 ADDR/0/, NEWADR, OFS, NEWOFS, ADROF
      INTEGER*4 CURLEN, MAXLEN, NEWLEN, REGEN
C  SEGA tells GETSPA to allocate storage in segment 10 - which
C  is a reasonable guarantee that the storage will be allocated
C  at a higher storage address than CONSYS.
      INTEGER*4 GETGR0/3/, SEGA/Z0A000000/
      LOGICAL DSINIT, DSFINI
      EXTERNAL GETSPA, FREESP
C
C
C
      DISTOK = .FALSE.
      IF( CURLEN .LT. 1 )  GO TO 9
         DO 8 I = 1, CURLEN, 2
            XI = COORD(I+OFS)
            IF( ABS(X-XI) .GT. MINDIS )  GO TO 5
               YI = COORD(1+I+OFS)
               IF( SQRT((X-XI)**2+(Y-YI)**2) .LE. MINDIS )
     +                                                    RETURN
5           CONTINUE
8           CONTINUE
9     CONTINUE
C
      DISTOK = .TRUE.
      IF( CURLEN+2 .LE. MAXLEN )  GO TO 15
         REGEN = REGEN + 1
         NEWLEN = 2*MAXLEN
         CALL RCALL(GETSPA, 2, GETGR0, SEGA+4*NEWLEN,
     +                     2, TRASH, NEWADR)
         NEWOFS = (NEWADR-ADROF(COORD)) / 4
         DO 12 I = 1, MAXLEN
```

```
            COORD(I+NEWOFS) = COORD(I+OFS)
12          CONTINUE
         CALL RCALL(FREESP, 2, Ø, ADDR, Ø)
         ADDR = NEWADR
         OFS = NEWOFS
         MAXLEN = NEWLEN
15    CONTINUE
      COORD(1+CURLEN+OFS) = X
      COORD(2+CURLEN+OFS) = Y
      CURLEN = CURLEN + 2
      RETURN
C
C
C
C
C
      ENTRY DSINIT(NUM, MINDIS)
C
C  DSINIT is called from CONTUR to make an initial allocation of
C  space for the array to save label coordinates, and to
C  initialize variables which describe the state of the
C  coordinate-saving buffer.
C
      MAXLEN = 12*NUM
      CALL RCALL(GETSPA, 2, GETGRØ, SEGA+4*MAXLEN,
     +                   2, TRASH, ADDR)
      OFS = (ADDR - ADROF(COORD)) / 4
      REGEN = Ø
      CURLEN = Ø
      DSINIT = .FALSE.
      RETURN
C
C
C
C
C
      ENTRY DSFINI(/NREGEN/, /MAXSIZ/, /NUSED/)
C
C  DSFINI is called from CONTUR to deallocate the space which
C  was dynamically acquired to hold the label coordinate
C  values.  If the user did not specify labeling, then there
C  will be no space to deallocate; therefore, we must be a bit
C  careful:
C
      IF( ADDR .NE. Ø )  CALL RCALL(FREESP, 2, Ø, ADDR, Ø)
      ADDR = Ø
      NREGEN = REGEN
      MAXSIZ = 4*MAXLEN
      NUSED  = 4*CURLEN
      DSFINI = .FALSE.
      RETURN
      END
```

```
      SUBROUTINE CTQQ
C
C  Although care has been  taken  to  ensure  the  correct
C  functioning  of  the  CONSYS system, neither the author
C  nor The University of Michigan shall be liable for  any
C  direct   or   indirect,  incidental,  consequential,  or
C  specific  damages  of  any  kind  or  from  any   cause
C  whatsoever  arising out of or in any way connected with
C  the use or performance of  the  CONSYS  system  or  its
C  documentation.
C
C  CTQQ is called from CONTUR to produce contour coordinate
C  values for the grid cell bounded by XL and XR, YL and YU, and
C  ZLL, ZUL, ZUR, and ZLR.  Coordinate pairs thus produced are
C  disposed of via calls to the user-specified routines CKMA
C  and CKVA.  The Z values to be contoured are stored in
C  CZ(PTR(LOWER+OFS)),...,CZ(PTR(UPPER+OFS)).
C
C  The algorithm for this routine was taken from:
C    G. W. Hartwig, "CONTUR - A Fortran IV Subroutine for
C    Plotting Contour Lines," Ballistic Research Laboratories
C    Memorandum Report # 2282, Aberdeen Proving Ground,
C    Maryland, March, 1973. (NTIS accession number AD-760 437).
C
C  The following code is for version 1.2 of CONSYS, produced
C  15 May, 1976.  GNC
C
      REAL CZ(1)
      LOGICAL KCHK(8), CENTER
      REAL PX(8), PY(8), PTEMP
      EQUIVALENCE (PX(1), PX1), (PX(2), PX2), (PX(3), PX3),
     +            (PX(4), PX4), (PX(5), PX5), (PX(6), PX6),
     +            (PX(7), PX7), (PX(8), PX8)
      EQUIVALENCE (PY(1), PY1), (PY(2), PY2), (PY(3), PY3),
     +            (PY(4), PY4), (PY(5), PY5), (PY(6), PY6),
     +            (PY(7), PY7), (PY(8), PY8)
      INTEGER*2 PTR(1)
C
      COMMON /CONCOM/ LOWER, UPPER, OFS, XL, XR, XC, YL, YU, YC,
     +                ZLL, ZUL, ZLR, ZUR
      INTEGER*4 LOWER, UPPER, OFS
C
C
C
C
      XC = 0.5*(XL + XR)
      XLMXR = XL - XR
      XLMXC = XL - XC
      XRMXC = XR - XC
      YLMYC = YL - YC
```

```
      YLMYU = YL - YU
      YUMYC = YU - YC
      ZC = Ø.25*(ZLL + ZUL + ZLR + ZUR)
      DO 12Ø LEVEL = LOWER, UPPER
         ZØ = CZ(PTR(LEVEL+OFS))
         TLL = ZLL - ZØ
         TUL = ZUL - ZØ
         TLR = ZLR - ZØ
         TUR = ZUR - ZØ
         TC  = ZC  - ZØ
C
         IC = Ø
         CENTER = .FALSE.
         DO 11 M = 1, 8
            KCHK(M) = .FALSE.
11           CONTINUE
C
C  Segment 1:
C
         IF( TLL*TLR .GT. Ø. )  GO TO 19
            KCHK(1) = .TRUE.
            IF( TLL*TLR .EQ. Ø. )  GO TO 12
               IC = IC + 1
               PX(IC) = TLL * XLMXR/(ZLR-ZLL) + XL
               PY(IC) = YL
               GO TO 18
12          CONTINUE
               IF( TLL .EQ. Ø. )  GO TO 15
                  IC = IC + 1
                  PX(IC) = XR
                  PY(IC) = YL
                  GO TO 17
15             CONTINUE
                  IC = IC + 1
                  PX(IC) = XL
                  PY(IC) = YL
                  IF( TLR .NE. Ø. )  GO TO 16
                     IC = IC + 1
                     PX(IC) = XR
                     PY(IC) = YL
16                CONTINUE
                  GO TO 17
17             CONTINUE
               GO TO 18
18          CONTINUE
19       CONTINUE
C
C  Segment 2:
C
         IF( TLL*TC .GT. Ø. )  GO TO 29
            KCHK(2) = .TRUE.
            IF( TLL*TC .EQ. Ø. )  GO TO 22
               IC = IC + 1
```

```
                FAC = TLL/(ZC - ZLL)
                PX(IC) = XLMXC*FAC + XL
                PY(IC) = YLMYC*FAC + YL
                GO TO 28
22           CONTINUE
                IF( TC .NE. Ø. )  GO TO 25
                   CENTER = .TRUE.
                   IC = IC + 1
                   PX(IC) = XC
                   PY(IC) = YC
25              CONTINUE
                GO TO 28
28           CONTINUE
29        CONTINUE
C
C  Segment 3:
C
          IF( TUL*TLL .GT. Ø. )  GO TO 39
             KCHK(3) = .TRUE.
             IF( TUL*TLL .EQ. Ø. )  GO TO 32
                IC = IC + 1
                PX(IC) = XL
                PY(IC) = TLL * YLMYU/(ZUL-ZLL) + YL
                GO TO 38
32           CONTINUE
                IF( TUL .NE. Ø. )  GO TO 35
                   IC = IC + 1
                   PX(IC) = XL
                   PY(IC) = YU
35              CONTINUE
                GO TO 38
38           CONTINUE
39        CONTINUE
C
C  Segment 4:
C
          IF( TUL*TC .GE. Ø. )  GO TO 49
             KCHK(4) = .TRUE.
             IC = IC + 1
             FAC = TUL/(ZC - ZUL)
             PX(IC) = XLMXC*FAC + XL
             PY(IC) = YUMYC*FAC + YU
49        CONTINUE
C
C  Segment 5:
C
          IF( TUL*TUR .GT. Ø. )  GO TO 59
             KCHK(5) = .TRUE.
             IF( TUL*TUR .EQ. Ø. )  GO TO 52
                IC = IC + 1
                PX(IC) = TUL * XLMXR/(ZUR-ZUL) + XL
                PY(IC) = YU
                GO TO 58
```

```
52          CONTINUE
               IF( TUR .NE. Ø. )  GO TO 55
                  IC = IC + 1
                  PX(IC) = XR
                  PY(IC) = YU
55             CONTINUE
               GO TO 58
58          CONTINUE
59       CONTINUE
C
C  Segment 6:
C
         IF( TUR*TC .GE. Ø. )  GO TO 69
            KCHK(6) = .TRUE.
            IC = IC + 1
            FAC = TUR/(ZC - ZUR)
            PX(IC) = XRMXC*FAC + XR
            PY(IC) = YUMYC*FAC + YU
69       CONTINUE
C
C  Segment 7:
C
         IF( TLR*TUR .GE. Ø. )  GO TO 79
            KCHK(7) = .TRUE.
            IC = IC + 1
            PX(IC) = XR
            PY(IC) = TUR * YLMYU/(ZUR-ZLR) + YU
79       CONTINUE
C
C  Segment 8:
C
         IF( TLR*TC .GE. Ø. )  GO TO 89
            KCHK(8) = .TRUE.
            IC = IC + 1
            FAC = TC/(ZC - ZLR)
            PX(IC) = XRMXC*FAC + XC
            PY(IC) = YLMYC*FAC + YC
89       CONTINUE
C
C  Now derive the line segments to be drawn from the contents
C  of the PX and PY arrays.
C
         IF( IC .LE. 1 )  GO TO 117
            IF( IC .GE. 6 .OR.
     +           (IC .EQ. 5 .AND. CENTER) )  GO TO 1ØØ
               IF( .NOT. KCHK(8) )  GO TO 95
                  DO 94 L = 1, 7
                     IF( .NOT. KCHK(L) )  GO TO 93
                        PX(IC+1) = PX(1)
                        PY(IC+1) = PY(1)
                        DO 92 M = 1, IC
                           PX(M) = PX(M+1)
                           PY(M) = PY(M+1)
```

```
92                          CONTINUE
                          IF( MOD(L, 2) .EQ. 1 )  GO TO 95
93                      CONTINUE
94                      CONTINUE
                     GO TO 97
95                IF( .NOT. CENTER .OR. KCHK(1) )  GO TO 97
                     PTEMP = PX1
                     PX1 = PX2
                     PX2 = PTEMP
                     PTEMP = PY1
                     PY1 = PY2
                     PY2 = PTEMP
                     GO TO 97
97                CONTINUE
                  CALL CKMA(PX1, PY1)
                  DO 98 M = 2, IC
                     PXM = PX(M)
                     PYM = PY(M)
                     CALL CKVA(PXM, PYM)
98                   CONTINUE
                  GO TO 109
100            CONTINUE
                  IF( IC .GE. 6 )  GO TO 104
                     PX6 = PX5
                     PY6 = PY5
                     PX5 = PX2
                     PY5 = PY2
104               CONTINUE
                  IF( KCHK(2) )  GO TO 105
                     CALL CKMA(PX5, PY5)
                     CALL CKVA(PX6, PY6)
                     CALL CKVA(PX1, PY1)
                     CALL CKMA(PX2, PY2)
                     CALL CKVA(PX3, PY3)
                     CALL CKVA(PX4, PY4)
                     GO TO 108
105               CONTINUE
                     CALL CKMA(PX1, PY1)
                     CALL CKVA(PX2, PY2)
                     CALL CKVA(PX3, PY3)
                     CALL CKMA(PX4, PY4)
                     CALL CKVA(PX5, PY5)
                     CALL CKVA(PX6, PY6)
                     GO TO 108
108               CONTINUE
                  GO TO 109
109            CONTINUE
117         CONTINUE
120         CONTINUE
         RETURN
C
C
C
```

```
C
      ENTRY CTQQIN(PTR, CZ, CKMA, CKVA)
C
C  CTQQIN is called from CONTUR to initialize the addresses of
C  certain very commonly used variables and subroutine entry
C  points so that the prologue of the routine CTQQ will be as
C  short as possible.  (CTQQ is effectively in the innermost
C  loop of the contour computation.)
C
      RETURN
      END
```

REFERENCES

1. Hartwig, G. W., CONTUR - A FORTRAN IV Subroutine for Plotting Contour Lines, Ballistic Research Laboratories Memorandum Report #2282, Aberdeen Proving Ground, Maryland, March 1973. (NTIS accession number AD-760 437)

2. IBM System/360 and System/370 FORTRAN IV Language, Form number GC28-6515, IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, New York 10020, January 1971.

3. Akima, H., "Algorithm 474 - Bivariate Interpolation and Smooth Surface Fitting Based on Local Procedures," Communications of the ACM, January, 1974.

4. _____, "A Method of Bivariate Interpolation and Smooth Surface Fitting for Values Given at Irregularly Distributed Points," OT Report 75-70, U. S. Department of Commerce/Office of Telecommunications, Boulder, Colorado, 1975.

5. Tobler, W., "Tuning an Interpolated Lattice," Department of Geography, The University of Michigan, Ann Arbor, 1976.

BIBLIOGRAPHY

Herzog, Bertram, DRAWL in MTS, Compgraph Publishers, Ann Arbor, October 1971.

Conklin, James W., and Mark Barnett, A Basic Software Package for the Computek Terminal on MTS, Center for Research on Learning and Teaching, The University of Michigan, Ann Arbor, 1971.

Phillips, Richard L., "Extensions to the Computek Routines for Storage Tube Terminals," Department of Aerospace Engineering, The University of Michigan, Ann Arbor, 1972.

_____, MTS Interactive Graphics Subroutine Libraries AERO:CKLIB, AERO:TEKLIB, and AERO:T4002, Department of Aerospace Engineering, The University of Michigan, Ann Arbor, 1972.

_____, "Software Tools for Computer Graphics," Department of Aerospace Engineering, The University of Michigan, Ann Arbor, 1976.

Van Roekel, John, MGI User's Guide, Department of Aerospace Engineering, The University of Michigan, Ann Arbor, May 1972.

Blinn, James F., The Integrated Graphics System, The University of Michigan Computing Center, Ann Arbor, May 1972.

Newman, W. M., and R. F. Sproull, Principles of Interactive Computer Graphics, McGraw-Hill Book Co., New York, 1973.

Walters, R. F., "Contouring by Machine: A User's Guide," The American Association of Petroleum Geologists Bulletin, Vol. 53, No. 11, November, 1969, pp. 2324 - 2340.