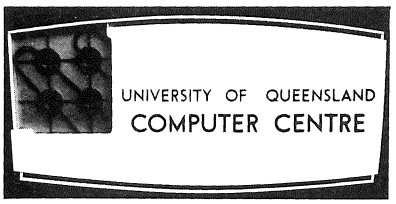


COMPUTER CENTRE BULLETIN

Volume 2, Number 7.
7th July, 1969.

Editor:
H. L. Smythe.

THIS EDITION

This issue of the Bulletin continues to inform clients of the Computer Centre about the discovery of further PDP-10 FORTRAN IV errors. Clients are asked to report any suspected errors to the Administrative Officer so that Digital Equipment Australia can be immediately notified.

Readers should find our main article, *Some Thoughts on the Construction of Scientific Programs*, particularly interesting and instructive. The Bulletin also lists the library accessions for March.

Readers and external users are encouraged to submit articles on some aspect of computing relating to their work, undertaken either as an individual or as a member of a group project. Articles up to 2,000 words may be published. If you are using the computer in a specialised field, be it commerce, administration, medicine, engineering, or even education, please write to the Editor about your work.

Helping the Bulletin *will* help you!

STAFF OF THE COMPUTER CENTRE

RETURNED.....

After an absence of two months, Systems Programmer *John Row* is once more immersed in work at the Computer Centre. As assistant navigator on the yacht "Kaleena", John took part in the Auckland to Suva race, sailing 1,140 miles in 8½ days. The "Kaleena" had the distinction of being the only Australian yacht in the race. Now, it's work again as normal, enlivened only by the occasional sea shanty emitting from the Programmers' Room.

LIBRARY ACCESSIONS

For the benefit of book enthusiasts, the Bulletin continues to publish the monthly accessions of books and periodicals on computing of the University Libraries. This month's list details the acquisitions of March 1969.

Hays, David G. *Introduction to Computational Linguistics.* (410.18 HAY, Engin.Lib.)

Allan, D.M. *Computer Programmes for Distribution Analysis and Assignment in Transportation Studies.* 1968. (Q519.92 ALL, Engin.Lib.)

Cutler, Donald I. *Introduction to Computer Programming.* 1964. (519.92 CUT, Architecture Lib.)

Introduction to ALGOL. 1964. (519.92 BAU, Engin.Lib.)

Stuart, Fredric. *FORTRAN Programming.* 1969. (519.92 STU, Pharmacy Lib.)

Gschwind, Hans W. *Design of Digital Computers.* 1967. (621.381958 GSC, Engin.Lib.)

Jensen, Randall W. *IBM Electronic Circuit Analysis Program.* 1968. (621.381 JEN, Engin.Lib.)

Harper, Goin Neil, ed. *Computer Applications in Architecture and Engineering.* 1968. (651.8 HAR, Architecture Lib.)

Smith, John U. *Computer Simulation Models.* 1968. (658.505 SMI, Agriculture Lib.)

Geck, Elisabeth. *Johannes Gutenberg: From Lead Letter to the Computer.* 1968. (655.1 GEC, Main Lib.)

Leed, Jacob, ed. *The Computer and Literary Style.* 1966. (808 LEE, Main Lib.)

Engeler, Erwin. *Formal Languages; Automata and Structures.* 1968. (510 ENG, Main Lib.)

PDP 10 FORTRAN IV ERRORS

With any relatively new computer system, it is likely that software errors undetected by the manufacturer will occur in released programs. Unfortunately, this is the situation with the PDP 10, and we ask for your co-operation in detecting these errors. From problems discovered by users, several errors in the compiler and the FORTRAN Operating System have been isolated. These "bugs" have been referred to Digital Equipment Australia for correction, and, in addition, the Bulletin is publishing them each month to help Computer Centre clients avoid these trouble areas. In some cases, it is possible for corrections to be made locally, but very often, these errors involve basic functions of the compiler, and it is not feasible for local changes to be made. Please accept our apologies, and in the meantime, here are some more traps for unwary players.

1. If a subroutine uses variable dimensioning of arrays in its calling sequence and also includes arithmetic statement functions, the code produced by the compiler *will not work.*

2. It has been found that a program with a large number of subroutines, greater than ten in most cases, may not load properly, and will probably lead to the message:

   ILLEGAL UUO AT LOCATION XXX

   This is a LOADER error which the Computer Centre has reported to DEA.

3. Real constants that have more significant digits than can be expressed in machine word length, will produce incorrect values when they are converted by the compiler. No error message is produced, and clients are recommended to use only nine significant digits for real constants.

4. Many users have reported that the use of a slash within a FORMAT statement, sometimes produces incorrect results. This has been traced to an error in BATCH, the program which runs the batch processing jobs, and is not a fault of the FORTRAN compiler.

One main cause of this error has been detected and corrected. Users, however, are asked to report any further occurrences of this problem.

5. No check is made for overflow and underflow in complex arithmetic. Thus $(1 + j\ \delta) * (1 + j\ \varepsilon)$ where δ and $\varepsilon \rightarrow 0$ will lead to an improper value of the real part of the product because of underflow in the evaluation of $\delta * \varepsilon$ as part of the real component.

It is recommended that, at this stage, a check should be made of the significance of both parts of complex numbers involved in complex arithmetic so that this situation may be avoided.

The staff are planning to implement a general procedure for the overall handling of overflow and underflow conditions. However, progress is presently halted by lack of information on a parallel scheme currently proposed by DEA.

6. Complex or Double Precision quantities raised to real powers, generate calls to CEXP.3 or DEXP.3 which pass the real exponent without a low order word.
For example:

```
DOUBLE PRECISION   C,CC
COMPLEX      A
CC = C**5.1
AA = A**7.9
```

Change the exponent type explicitly, thus:

```
**DBLE(5.1) or 5.1DØ
**CMPLX(7.9,Ø) or (7.9,Ø)
```

7. In last month's Bulletin, we reported that the function ATAN returns an incorrect value (0.0) for an argument of $\frac{\Pi}{2}$. It has since been found that the error lies not in ATAN but in SINGL. The problem is being further investigated.

Users are urged to report any errors they have discovered to the Administrative Officer (Mr. John Jauncey, extension 8471), including evidence such as listings and card decks. This will greatly assist staff members in their investigation of the problem.

SOME THOUGHTS ON THE CONSTRUCTION OF SCIENTIFIC PROGRAMS

Dr. J.L. Meek

This article describes some of the more common characteristics of scientific programs, and discusses the ways by which program organization can become more efficient.

The author, Dr. J.L. Meek, B.E., B.Sc., M.S. (Cal.), Ph.D., was one of the original "Gap" programmers on the GE 225. He worked on Structural Dynamics and Finite Element Theory with Professor Clough at the University of California, Berkeley. At present, Dr. Meek holds the position of Reader in Civil Engineering at the University of Queensland. His research interests are in Finite Element Applications in Structural Mechanics.

Scientific programs, at least those that we have encountered in the analysis of structures and the elasto-plastic continuum, have certain common characteristics. These are summarized as follows:

(1) The input data can be arranged in a well-organized, tabular form.

(2) The basic core of the program consists of the solution of a large number of simultaneous equations. If the problem is non-linear in nature, the solution may involve many passes through the equation complex.

(3) Result output may be copious in extent, and yet, in the main, may be of little use except for a few salient values.

(4) The complete program may be so long and complex that it consumes core storage and defies complete visualization as a coded whole.

These features can lead to programs that are very demanding in core storage and difficult to code in an optimum way. They have led not only to a study of the nature of the equations involved, but also to an examination of the structure of the program. Thus, our first exercise was concerned with the nature of the simultaneous equations to be solved. A study of population density in the coefficient matrix showed that the iteration methods such as Gauss-Seidel could be useful to save storage, while a study of the structure of the equations has led to band width minimization and the use of efficient band solvers. A banded structure of equations is shown in Figure 1 in which non-zero terms are arranged along the forward diagonal of the matrix. In this case, each term shown is a (6x6) submatrix, this dimension (6) corresponding to the six equilibrium equations at each joint of a 3-dimensional framed structure.

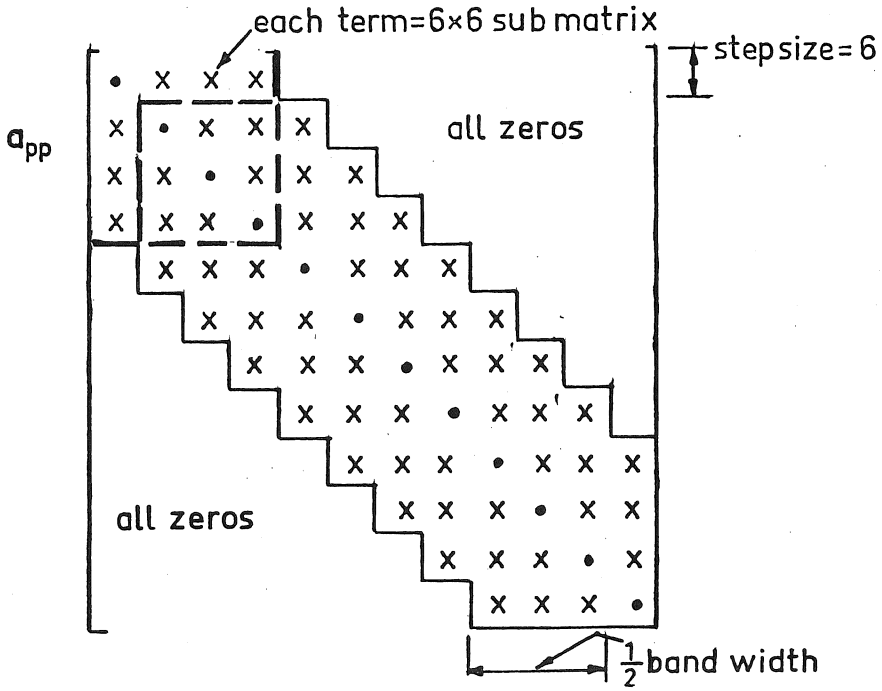


FIGURE 1

To further reduce the use of the core store, it is necessary to segment the program into its basic functions, and to describe the desired outcome of each concisely. Each segment will then become a self-contained unit (overlay) that may be checked and coded independently of the remaining portions, except, of course, through connecting labelled common store.

One such suitable breakdown is as follows:

(1) Input Function

Basic data for a particular problem is read into store, and checked as far as possible for accuracy. Error messages are printed, together with an echo print-out of data. The detection of an error will activate a flag that terminates the program at the end of this segment. Such an input program may be quite complex, and include checking of data for incorrect character and format punching in addition to simple numeric accuracy.

(2) Setup Function

At this stage, the coefficient matrix for the linear equations is generated, and further checks are applied to the data. Again, an error will cause a flag to be set that terminates the program at the end of the segment.

(3) Decomposition Function

This includes the formation of the right-hand sides to the equations (load vectors), and the modification of the whole system for boundary conditions, such as prescribed displacements. Finally, the decomposition of the band matrix ready for back-substitution, is performed.

(4) Solution Function

Here, the back-substitution is carried out. For large scale systems, it is important that iteration on residuals be included as an option, since many problems are not well-conditioned and need this

refinement. The iteration on residuals requires some double precision arithmetic in the calculation of the residuals. Our experience shows that 2 or 3 iterations are generally sufficient.

If the problem is non-linear, this segment may contain many other features. The analysis may then be contained here, or may have to return to (2) after passing this overlay, depending on the method of analysis employed, (i.e. incremental or iterative, or both).

(5) Result Interpretation Function

A primitive scheme will produce a blanket print-out of all results. A refinement may include a search for maxima or minima. The most suitable form for result presentation will generally be graphical; for example, contour plots for stress values in the analysis of the stressed continuum, or bending moment and shear force envelopes for members of framed structures.

If several load cases are incorporated, this feature may select prescribed load combinations and also plot influence functions.

In the programs we have produced, not all these features have been written in, but we have sufficient experience to know that most of them will be of great benefit to the ultimate program user. A feature of such a scheme is that once input-output overlays have been written for one class of problem, they may be easily modified for similar problems.

In the program organization, efficient use should be made of function subprograms, subroutines and data statements. In fact, a study should be made as to the coding efficiency (in machine language) of commonly occurring operations (particularly in matrix algebra work).

For example, clearing the array A(50,50) requires the coding:

```
      DO 100 I=1,50
      DO 100 J=1,50
100   A(I,J)=0.0
```

whereas the subroutine for the same operation may be:

```
      SUBROUTINE CLEAR(A,N)
      DIMENSION A(1)
      DO 100 I=1,N
100   A(I)=0.0
      RETURN
      END
```

Note that the use of the one-dimensional array A(1) in the subroutine allows its use for arrays of one or more dimensions. The call then would be:

```
      CALL CLEAR (A,2500)
```

When such an operation is repeated many times, it is significant to note that not only is the absolute coding shortened, but also the FORTRAN source program. Similar use can be made of matrix operations using subroutines with variable dimension statements. Subroutines common to all segments can be then stored permanently along with the main calling program.

Preset data should be organized in data statements, (written in the main program if used in more than one segment).

For example, if A(2,2) is used as $\begin{bmatrix} 2.0 & 1.0 \\ 1.0 & 2.0 \end{bmatrix}$, it is generated as:

```
      DATA A/2.0, 1.0, 1.0, 2.0/
```

The data statement may be used also in the presetting of a sequence of operations on the rows and columns of a matrix. For example, suppose that A(12,12) is to have rows interchanged and columns interchanged to give the rearranged sequence (1,7,2,8,3,9,4,10,5,11,6,12). The coding is as follows:

```
      DIMENSION A(12,12), B(12,12),ICH(12)
      DATA ICH/1,7,2,8,3,9,4,10,5,11,6,12/
      DO 100 J=1,12
      L=ICH(J)
      DO 100 I=1,12
100   B(J,I)=A(L,I)
      DO 110 J=1,12
      L=ICH(J)
      DO 110 I=1,12
110   A(I,J)=B(I,L)
```

Finally, care should be taken to avoid elaborate recurrence relationships when generating locations in arrays when a simple I=I+1, type of statement in a loop will suffice.

It must also be mentioned that these programs require efficient organization of backing stores (tape and/or disc). For example, in the program setup, data may be generated for members in sequential form and stored on disc. If the coefficient matrix is generated in block form (rather than all being in core at once), this data will have to be obtained in random access fashion. Again, partial information calculated and stored at (2) will generally be needed for recall in the result function (5).

EXAMPLES

(1) In this example, the effect of ill-conditioning of equations is demonstrated and the result given of the iteration on the residuals. The cantilever beam of uniform EI=½, shown in Figure 2, is loaded with a force at the free end. For a length of beam 100 units, the theoretical value of the deflection (v) at the free end for unit load is:

$$v = 6.\dot{6} \times 10^5 \qquad (1)$$

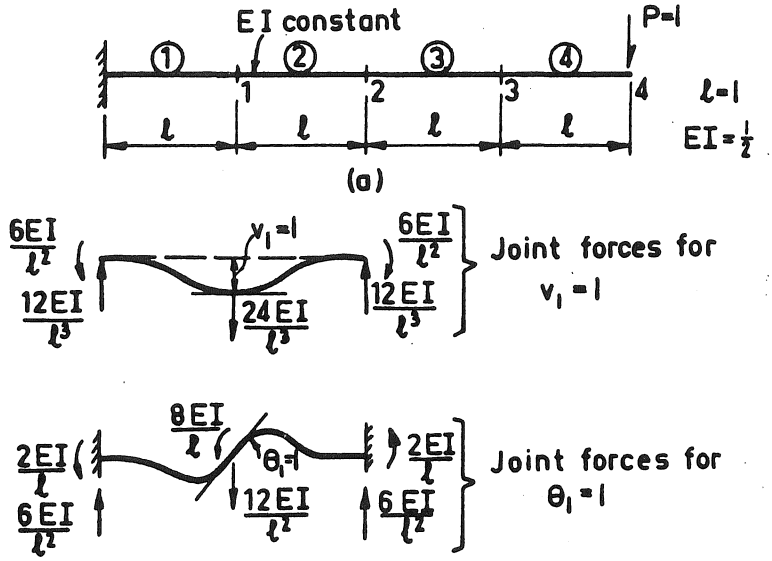


FIGURE 2

Development of Stiffness Matrix for Cantilever Beam

If the beam is subdivided (as shown in Figure 2) into the segments (1) to (4) with nodes 1 to 4, the calculation in finite form requires the deflection and rotation at each node so that 8 equations must be solved. In this case, the deflection at the end will be still substantially that given in Example 1. However, when a large number of subdivisions is made (100 for example), the value for the end deflection is in considerable error. This is because a rotation term near the left hand end has a big (or magnified) influence on the deflection at the free end. That is, a round-off error in a rotation term there will cause a much greater error in the end deflection. The analysis of the beam was carried out using 50 and 100 intervals of subdivision. The successive values of the end deflection for the 100 intervals are given below as the iteration is carried out on residuals.

6.433	x 10^5	Initial Solution
6.55877	x 10^5	1
6.66629	x 10^5	2
6.666557	x 10^5	3
6.66656700	x 10^5	4
6.66656731	x 10^5	5
6.66656732	x 10^5	6

It is seen that all significant improvement has been reached on the 3rd iteration. This is shown pictorially in Figure 3 in which plots have been given for the sum of squaresof residuals against number of cycles of iteration for both 50 and 100 segments.

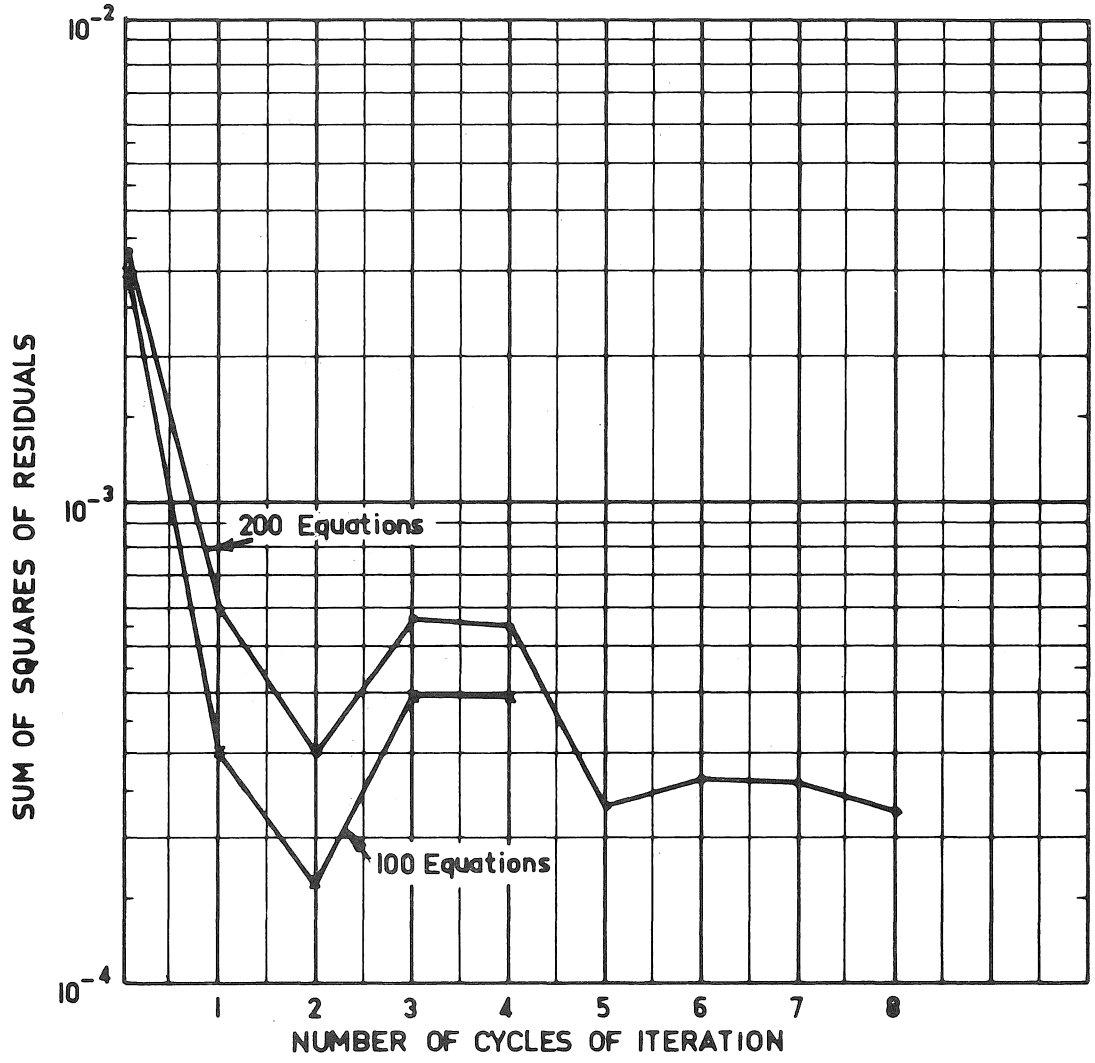
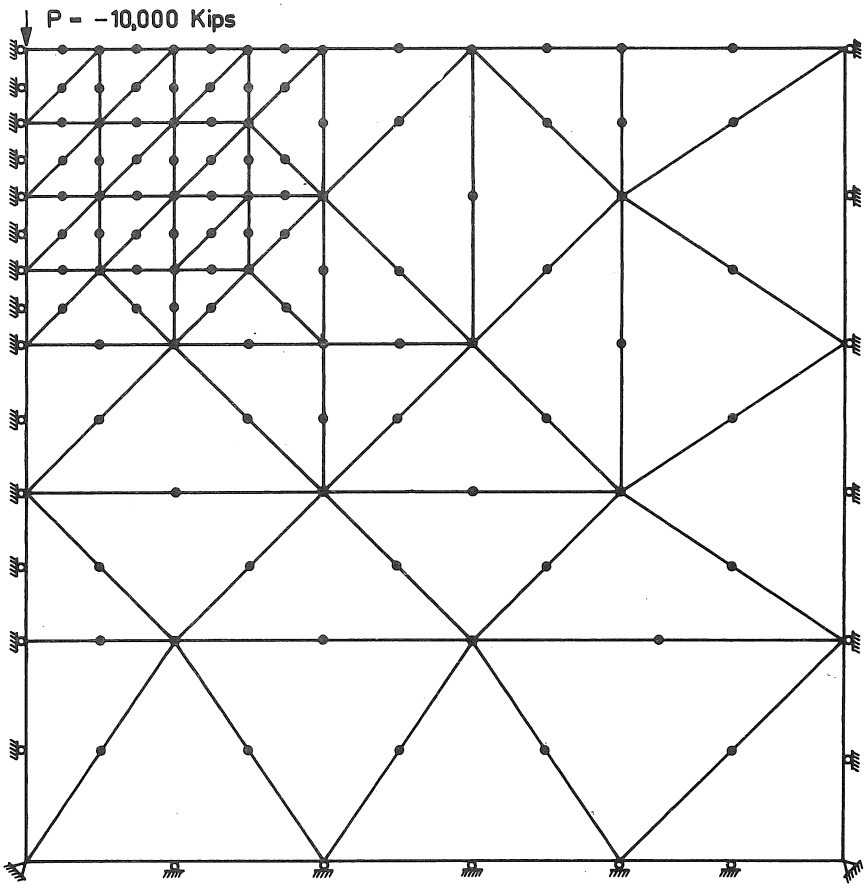


FIGURE. 3.

(2) In this example, the solution is given to an elasticity problem using the finite element method of analysis. The problem is to calculate the stresses in, and displacement of, a semi-infinite continuum subjected to a free P=10000 kips perpendicular to the free surface (see Figure 4).



ELEMENT LAYOUT
BOUSSINESQ PROBLEM. LINEAR STRAIN TRIANGLE.

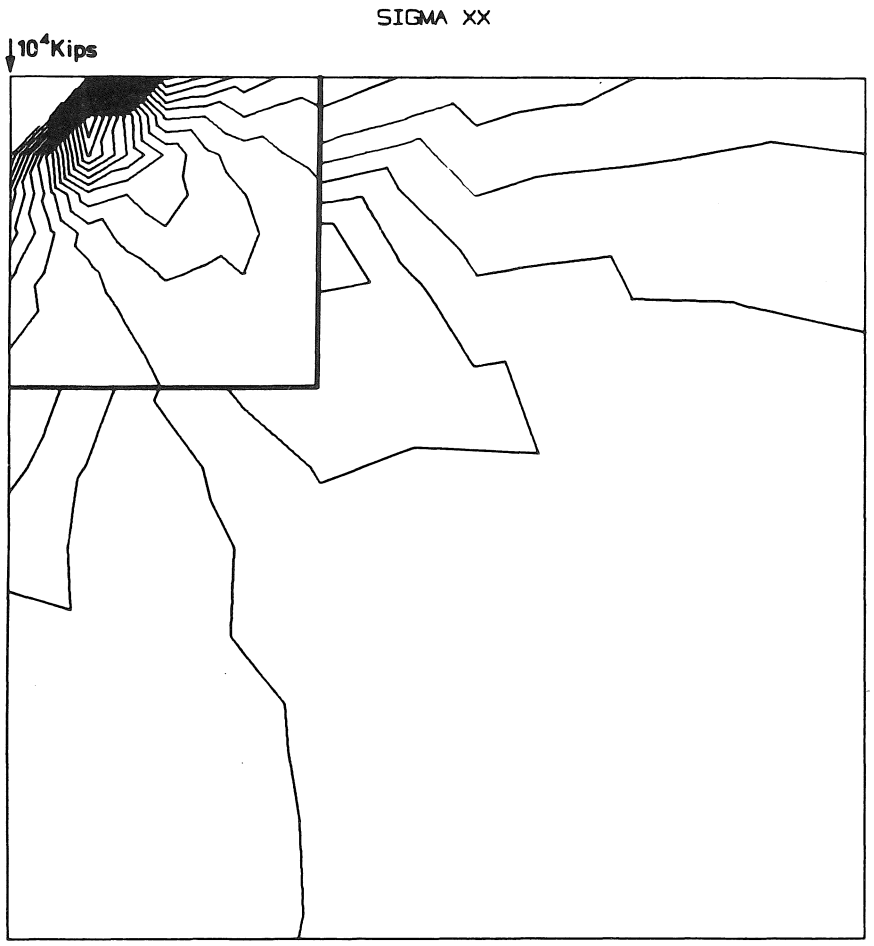
FIGURE 4

In Figure 4, the setup for the preparation of data is shown. Each node represented by a "dot" at the apices and mid-points of sides of the triangles, will have displacements (u,v), and hence to solve the system, the number of linear simultaneous equations will be twice the number of nodes. It should be evident that the problem will require a coordinate

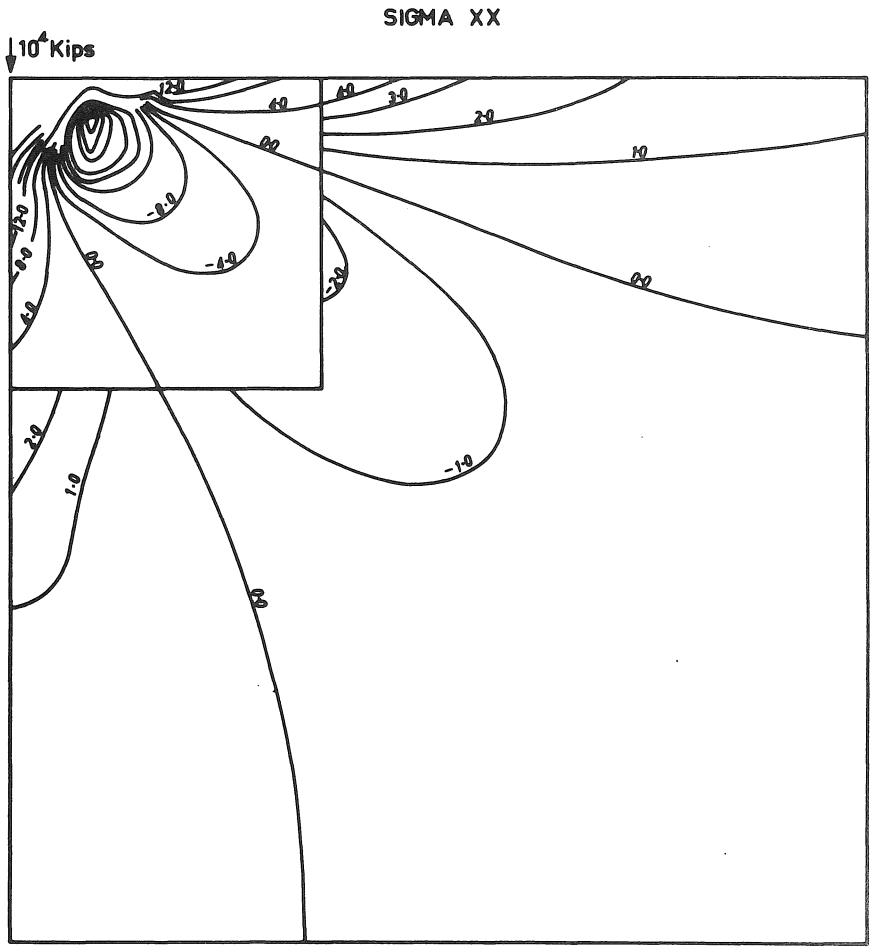
array (only values at apices need be input since the remainder may be calculated), and an element array giving the node numbers of each element. If (NUM) is the total number of nodes, and the node numbers for the triangles are contained in the array NODES(200,6), it is clear that a check of data should be made such that no node number is less than zero nor greater than (NUM). That is, if (NUMEL) is the number of elements, the check is as follows:

```
      IFLAG=0
      DO 100 I=1,NUMEL
      DO 100 J=1,6
      IF(NODES(I,J).GT.0.AND.LE.NUMEL) GO TO 100
      PRINT 20,I
20    FORMAT(17H ERROR ELEMENT NO/I5)
      IFLAG=1
100   CONTINUE
```

A graphical presentation of output is shown in Figure 5. In this figure, the left-hand drawing is the output produced by the computer plotter, whereas the right-hand drawing is the result of smoothing these curves. The contour lines have been drawn from the stress values at the apices and mid-points of sides of the triangles in Figure 4, assuming linear variation between nodes. To do this, each main triangle in Figure 4 is subdivided into 4 subtriangles, and each of these treated separately. The blocked-out portions in the figures show how contour spacing may be altered in zones of high stress gradient.



RADIAL STRESS (σ_R)
BOUSSINESQ PROBLEM L.S.T.



RADIAL STRESS (σ_R)
BOUSSINESQ PROBLEM L.S.T.

FIGURE 5