ELECTROSTATIC PLOTTING

PLOTTING ON AN ELECTROSTATIC PRINTER/PLOTTER

by

CHRISTOPHER A. BRYCE, B.Sc.

A Project

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

April 1975

MASTER OF SCIENCE (1974) (Computation)

McMASTER UNIVERSITY Hamilton, Ontario

TITLE: Plotting on an Electrostatic Printer/Plotter.

AUTHOR: Christopher A. Pryce, B.Sc. (McMaster)

SUPERVISOR: Professor D. Kenworthy

NUMBER OF PAGES: viii, 199

ABSTRACT

A survey of printers and plotters is given, and in particular the operation and capabilities of electrostatic printer/plotters is discussed. An implementation of a plotting system which plots on an electrostatic printer/plotter is presented. This plotting system is designed to be compatible with the Benson-Lehner plotting system. The standard "PLOT" routine is replaced by a two pass system, which generates plots on the printer/plotter. In addition to the plotting system, an implementation of a graph utility is presented. This utility provides a single one pass system that plots one or more functions (where the function has one value for each value of x.)

ACKNOWLEDGEMENTS

I wish to thank my supervisor Dr. D. Kenworthy for his assistance during the work on this project. I would especially like to thank Dr. D. Kenworthy and Professor K. Redish for their invaluable assistance and time during the writing of this project. Acknowledgements are also due for assistance at various times to Dr. D. Wood, Dr. N. Solntseff, G. Hicks, and M. Belec.

Finally, I would like to thank Mrs. Margaret Harris for her excellent typing.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 General Discussion of Printers and Plotters (Survey)

Computers communicate with users (humans) in human readable form primarily by means of the following three methods: line printer output, plotter output and teletype or CRT output. Of these methods, hardcopy communication is provided by the line printer, the plotter, and the teletype. What computers communicate comprises the following: the errors that the computer detects in the user's instructions (in Job Control Language, in the program, or in the job organization) and the results that the computer produces as a result of those instructions (the program listings and the results of the user's program). As a rule, the computer is required to produce large quantities of output, relative to the size of the input. Either by explicit user instructions, or by installation defaults, the computer usually produces program listings, quite often with cross-references, loader maps, results of user's programs, and a listing of the job control language. Unfortunately, most of this output gets wasted, as only a small portion actually will be used. It is all potentially useful, but the use of any given portion will depend upon the success or lack of success of the program. As a result, the only communication device capable of keeping up with the volumes of output normally produced is the line printer. Plotters and teletypes provide special purpose output abilities. With the plotter, pictorial representation, when properly used, can say more than thousands of pages filled with relevant numbers. Plotters can be used in drafting and providing the reliability and precision necessary in today's technological society, and can also be used to display results from scientific investigation. Teletypes provide an interactive ability that allows the user to dynamically communicate with, and interact with a computer. Generally speaking, plotters and teletypes have a limited speed, which would choke or stifle the potential output of most computers if used for the same purposes as the line printer.

The line printers available today vary greatly. There are small line printers designed to take output from a CRT and to print the screen buffer onto paper at about 100 or so lines per minute. Then there are large line printers, typically used on large computers, which print at a rate of 1000 to 1200 lines per minute. Within this range lie many printers of different sizes, with different print speeds, and to date these printers have mainly been characterized by being of mechanical type. A mechanical printer (or an impact printer) usually consists of a typewriter like ribbon together with a rotating drum, or a moving chain, and prints by striking the paper when the desired print character comes into position. Most large computers support 2 or 3 large 1000 lines/minute printers and sometimes several additional smaller printers (usually remote from the computer). These mechanical (impact) printers usually have 64 or 96 characters in their character sets, and in some cases this could be a restriction.

Other methods of printing include spitting chemicals onto special paper, burning special paper (with heat), a photographical type of printing, and electrostatic printing. Generally speaking the paper in each case is more expensive, but compared to impact printers, breakdown is usually much more infrequent. Print quality varies generally from worse than to much better than an impact printer. Printing speed varies from less than 1000 lines/minute to 5000 lines/minute, and the character sets on non-impact printers can include up to 256 characters. Special type-sets such as Roman or Helvetian are sometimes available, and often non-standard character sets, such as French, German, Greek or Japanese, can be software generated. Non-standard character sets will add to computer processing time.

Plotting too has usually been a mechanical process. Mechanical plotters divide basically into two types, flatbed plotters, and drum type plotters. Flatbed plotters may be small but most of them are large, of the order of 2 feet by 3 feet. These are mainly used in drafting systems to provide very accurate plots, with the usual step size or precision being about 0.001 inches. Flatbed plotters plot by moving a pen on a fixed piece of paper. Drum type plotters can also be either small or large, with the physical size of the plotter usually related to the precision. These plotters plot by moving the paper lengthwise, back and forth, and moving the pen from side to side. The paper for these plotters comes, usually, in rolls of widths 11, 30, or 36 inches, with a virtually infinite length (usually 125 - 150 feet, with some up to 367 feet). In general these plotters provide very accurate plots with a precision of 0.0025 to 0.005 inches, although some only have a precision of about 0.01 inches.

As in printing, plotting can be done by non-mechanical means also, by using the same techniques as used in the non-impact printing. In fact many devices are both printers and plotters. In such a printer/plotter, printing is either a special case of plotting, with software generated characters, or in addition to the plotting capability, a line buffer is provided together with a character generator (which relieves the computer from generating characters). Plotting is accomplished by setting points into an output line of densely packed points (with densities varying from 72.5 points per inch to 200 points per inch). Then successive lines of points, just as tightly packed (as in the line), are plotted. Thus in a manner similar to a television scan, an image or plot can be made.

1.2. Specific Discussion of Electrostatic Printer/Plotters.

To obtain the data and specifications of both the printer/plotters in this section and the plotters in the previous section a partial survey was made using available sales brochures and information. The survey was not very exhaustive, and obviously claims and specifications given in a sales brochure must be weighed accordingly.

The Electrostatic Printer/Plotter (EPP) is a versatile device. Most often the EPP comes as a printer/plotter, but occasionally just a printer or just a plotter, is provided. It varies in width from 8 inches (80 characters) to about 22 inches (264 characters), but most often the width is either 8 inches (80 characters) or 11 inches (132 characters).

As seen in figure 1.1., the EPP works in an efficient, simple manner. Paper, either roll paper or fan-fold paper, is passed between the writing head and the rear electrode. The writing head extends across the page and consists of densely packed writing nibs. These nibs individually, upon command, create minute electrostatic dots on the paper. As the paper advances, a liquid toner is applied to produce permanently visible points. Then, by passing air over the paper as it advances, the paper is dried and is ready to handle upon being stacked. Print characters (hardware)

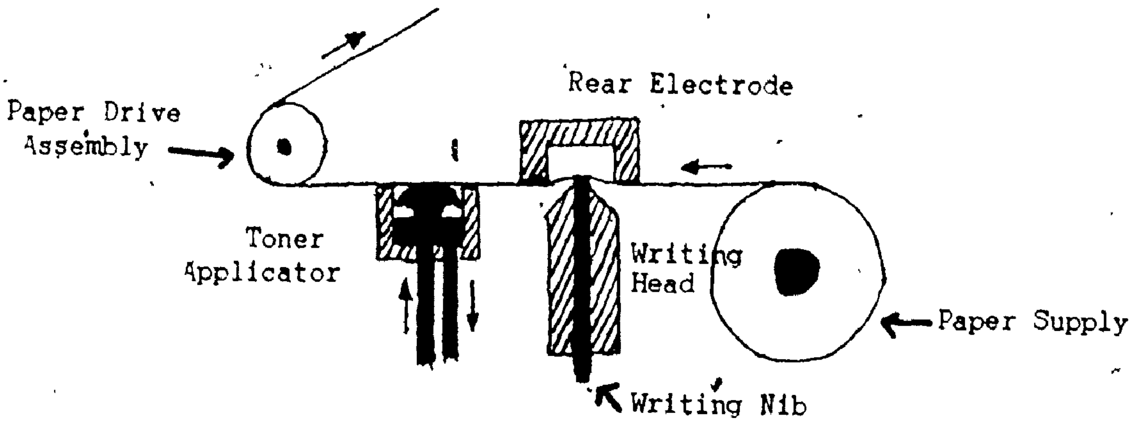


figure 1.1. Schematic of an EPP

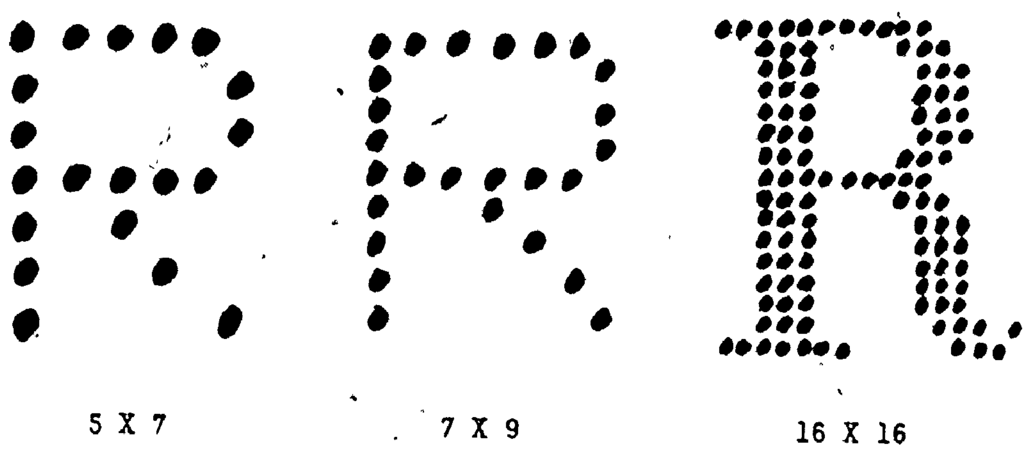


figure 1.2. Character Fonts

are produced in a matrix, (also called a grid or font), varying from 5 points x 7 points to 16 points x 20 points as illustrated in figure 1.2. Sometimes special type-fonts are provided (for example Roman, Helvetian, French, German . . .). Printing speed varies from 190 lines/minute to 5000 lines/minute depending on the page width and nib density. In the table (figure 1.3.) are included several different machines, but it is evident that the more character positions per line, and the more dense the nibs, the slower the print speed. The restrictions here appear to be electronic. Since one printer is able physically to print at 5000 lines/minute, in theory the rest should be capable of it. The problem would seem to be one of electronic buffering or high speed communication lines. It would not be surprising to see a 14 inch printer with 200 dots per inch printing in the 2000 to 5000 lines/minute range in the near future. The biggest problem at this point would be paper folding and developing high speed operators to remove the output as it is available.

Plotting is theoretically as fast as printing. Sometimes the computer will not be able to supply the information fast enough for maximum plot speed, but if the information were available, the plotter could take it. Comparing an EPP with other plotters, as far as plotting speed is concerned, is not fair. These plotters can produce an 8-1/2 x 11 page of plot, in theory, from 1 second for the fastest to about 24 seconds for the slowest. On the average this theoretical plot would take about 7 seconds. This time would be completely independent of what is to be plotted. In practice on a small mini computer this time will run up to 20 to 30 seconds. These times would compare with conventional plotter speeds running from 3 to 20 minutes. Also where, conventional plotters require

Paper Width	Character Width	Point Density (pts/inch)	Print Speed (lpm)	Plot Speed (ips)
8 inch (7.75)	80	72.5	600	1.6
	80	80	1000(a)/	2.1(a)/
			5000(s)	10.4(s)
11 inch (10.5)	132	100	500	1.2
		160	300	0.75
		200	500	0.92
		100	1200	2.8
		200	200	1.65
20 inch (18)	180	160	190	0.45
	232	100	300	0.75
21 inch	232	100	1200	3.0

(a) - asynchronous (s) - synchronous

figure 1.3. Table of Different EPP Specifications

an operator to reset the pen and paper for the next user, on the EPP the output gets stacked just like output from a line printer. The precision or accuracy of the EPP ranges from 0.014 inches to 0.005 inches, which compares unfavorably with the precision of most conventional plotters (0.005 inches to 0.0025 inches). As far as precision or step size is concerned, to date, the EPP is just starting to approach that of conventional plotters. There are other things to consider though. Most conventional plotters are incremental and if the pen gets nudged out of line (for example, someone walks too close to it) all successive plot movements are out by a constant amount, and this type of error is cumulative. Secondly if the user, by mistake, takes the pen off the edge of the paper the rest of the plot is lost, which could have been informative as to what went wrong. On an EPP there is no pen to knock off line and if the user goes off the page, the software is usually set up to ignore these points and simply to show the pen when it comes back onto the page, if it does.

All in all the conventional plotter, for the moment, has its place in plotting just for the better possible resolution, but where regular resolution (0.005 inches) is good enough the electrostatic plotter is by far the best.

1.3. Description of the Implementation Environment

This project is being implemented using a Versatec Matrix Printer/ Plotter 200A. The 200A can use 8.5 x 11 inch fan fold paper or 8.5 inches x 500 feet rolls. Primarily fan fold paper is used. The resolution is 72.5 dots per inch both vertically and horizontally. There are 560 writing nibs with a possible plot width of 7.75 inches. In printing mode, The printing speed is 600 lines per minute, with a corresponding plotting speed of 1.6 inches of plot per second.

As the plotting width is only 7.75 inches compared to 10.95 inches for the Benson-Lehner on the CDC6400, we could either leave everything in inches and let the user make his own allowances or scale everything down so that the user can ignore the difference. At present, everything is left in inches, but eventually a switch will be built into the plot package to allow either inches or scaled (10.95 to 7.75).

The printer/plotter is attached to a Hewlett-Packard 2100A which is operating under the DOSM operating system. The project software was written primarily in FORTRAN, and partially in HP ASSEMBLER, but an all FORTRAN version is possible. The standard HP FORTRAN IV compiler and the standard HP ASSEMBLER were used.

1.4. Overview of the Project

The objectives of the project are to:

1) develop a plot package starting at routine PLOT which is capable of plotting on an electrostatic printer/plotter, this PLOT routine is written to be compatible with the Benson-Lehner plotting system and capable of supporting the parts of the Benson-Lehner system which call PLOT.

2) develop a graph routine capable of taking several functions and creating a graph or plot, as simply as possible, over a desired range, and with a desired scale.

1.4.1. Plot Package

The objectives were chosen for several reasons. First of all PLOT is a key routine in any plotting system, and is the routine through which all plotter instructions must pass. Typically the user writes his program, and through either direct or indirect calls to PLOT, instructs the plotter on how to create the plot. Routines such as LINE, NUMBER, SYMBOL, etc. ..., allow the user easily to produce lines, write numbers, write characters, etc. ..., and these routines all perform their functions by making calls to PLOT. In the diagram below this typical program organization is illustrated.

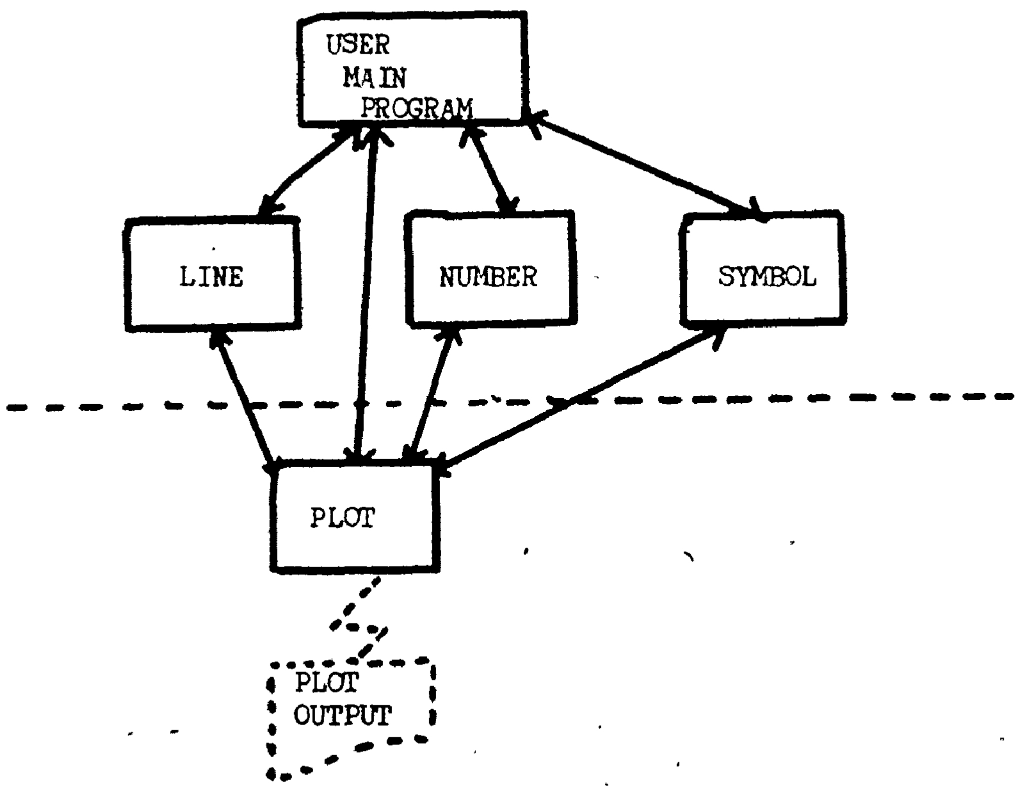


figure 1.4. Typical Plotting Program Organization

The PLOT routine was chosen as the place to break in because this allows the standard plot utilities to be used with the EPP. Therefore the user need not change any of his program in order to plot on the electrostatic plotter, as long as the electrostatic PLOT routine is substituted for the standard PLOT package. Another reason for starting at routine PLOT, is because the differences between plotting systems are minimal in routine PLOT and to switch the package to another plotting system is easiest at this routine. These are the basic reasons why the plotting system starts at routine PLOT.

The reasons for choosing the Benson-Lehner system for a model, is that this is the plotting system on the CDC6400 at McMaster. This means there are practical reasons to choose this system such as:

1) The availability of information, both user information and software information, on this plotting system;
2) plotter library routines could be easily converted to a HP2100 (after the PLOT package is implemented);
3) users could run on one or the other interchangeably;
4) during development of the plotter package it could be tested, on the CDC6400, for compatibility with special plot routines (LINE, GREEK, NUMBER, ...)

For these overwhelming reasons this project was designed to be compatible with the Benson-Lehner plotting system.

1.4.2. Graph Utility

This utility virtually created itself, during the experimental period, where simple plotter output was being generated. This output was an attempt to determine just how the printer/plotter actually plots, and just how best to do the plotting. Upon mastering the ability to generate simple plots, and how best to buffer the plotting process, it became evident that a function with one and only one Y-value for each X-value could be easily plotted as soon as it became available. Thus functions with this restriction can be easily plotted in the XxY coordinate plane. Note that this is all that the utility is intended to do. With little difficulty these functions were plotted, and gradually extras such as borders, reporting limits, and plotting axes were added to the graph utility.

What evolved was a routine, GRAPH, with the ability to plot 1 or more (up to 10) functions over a specified range of x and y, and with scales determined by the user. The author believes that this could be useful to obtain, with as little work as possible, graphs of 1 or more functions with 0.014 inch accuracy. By varying ranges and scales very good approximations could be obtained for the determination of the roots of equations, etc.

CHAPTER 2

THEORY

This chapter elaborates the theoretical base, upon which the Plotter Package and the Graph Utility is based.

2.1. General Description of the Method Used

Basically the plotter instructions come as three arguments to the subroutine, "PLOT" (X, Y, PEN). Excluding the special frills that are usually present, the first two arguments specify the point to which the pen should be moved, with PEN indicating whether the pen should be up (not plotting), or down (plotting). By convention the X-direction is chosen to be down the page, with an almost unlimited range, or, at worst, a very large range relative to the Y-direction. The Y-direction is chosen to be across the page, with a limited range usually of the order of 8 inches, 11 inches, or 30 inches.

The Electrostatic Printer/Plotter (EPP) plots one row or line of dots across the page at a time. Thus, as the printer/plotter proceeds down the page (in the positive X-direction), the plots or tracings can be made. But --- the EPP can only move in the positive X-direction (down the page). As a result, only as long as the calls to "PLOT" come in ascending order of the X-coordinate, can the plotter calls be handled as they occur. This describes a special case, and in fact, is the case dealt with in the Graph Utility. The Graph Utility is designed to plot the function f(x), where f(x) is unique for all values of x. This restriction allows for a one pass system, in which the function is plotted as soon as it is evaluated.

In a conventional plotting system the pen is allowed to move in any direction (right, left, up the page, down the page). Of importance to this project is that it can move both up the page and down the page, where an electrostatic printer/plotter cannot. Note that freedom of movement in the Y-direction presents no problem to an EPP. Therefore in order to do conventional plotting on an EPP, the problem of pen movements up the page (in the negative X-direction) must be solved.

There is no simple solution to this problem involving a one pass system, where plotting is done as the plotter instructions come in since the very last call might involve plotting a line at the very top of the plot (involving a small X-coordinate). Therefore a two pass system is needed where, in some way, the movements of the pen are recorded until the user indicates that he has finished plotting. Then a second pass does the plotting.

The most straightforward solution is that in which a core image, or, more realistically, a disk image of the final plot is constructed. This image initially would be empty (no points). Then as the plotter instructions come in, they are processed and plotted on to the image by inserting the necessary bits. Then when the user is finished the image is copied on to the printer/plotter. There are drawbacks to this solution, however. The storage required, whether it is core storage or disk storage, is considerable. Just considering the case at McMaster University, the printer/plotter has a resolution of 72.5 points per inch and the computer involved has 16-bit words. Now consider a frame 7.75" x 7.75" involving 560 points x 560 points. The storage required for this frame would be 313,600 bits (39,200 bytes or 19,600 words). This would not fit into the core (12,000 word) and would have to go on to the disk, taking 5.5 tracks (or 2.75 per cent of the 200 tracks available). The real catch here is that only a small plot is being considered. It is not unreasonable for a user to want a 50 or 100 inch plot (in the X-direction). On some machines this solution might be reasonable but on the HP2100A at McMaster, it is not a practical solution.

The solution chosen is to take the calls to "PLOT" as they come in and to reorganize these calls into acceptable sub-plots (that is plots ascending in the X-direction only). This is done during the first pass, and for all but extreme cases, the storage is less as the (X,Y) coordinates to be stored are quite compact. The storage requirements are completely independent of the coordinate values, depending only on the number of plotting instructions. The set of all the sub-plots created is the user's plot. In the second pass, these sub-plots are plotted in parallel. This means that for a particular X-value many sub-plots may be adding points to the plot. When a sub-plot is exhausted, it is deleted, and when all the sub-plots are done, the plot is complete. That in a nutshell is the method used in the plotting package. In the next section, definitions and terminology relating to this method are formalized. To help tie things together this is the correspondence: -

The term <u>sub-plot</u> used above corresponds to the term <u>string</u> which is a collection of contiguous line segments.

The term <u>plot</u> used above corresponds to the term <u>list</u> which is a

of all of

2.2 Definitions and Terminology

For the purposes of this project the concepts of a point, of a line, of a straight line, a straight line segment, of a string and of a list must be formalized.

These first few definitions are intentionally intuitive rather than being rigorous or complete. All definitions are in terms of the XxY coordinate plane.

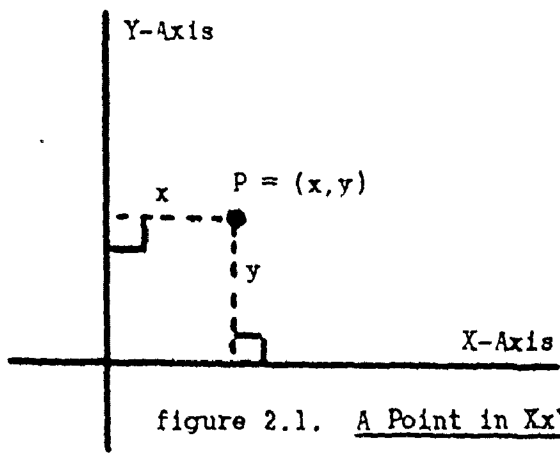


figure 2.1. A Point in XxY Plane

a point P is defined as an ordered pair (x,y) where (x,y) represents the position in the XxY plane, and where x is the displacement from the Y-Axis, and y is the displacement from the X-Axis.

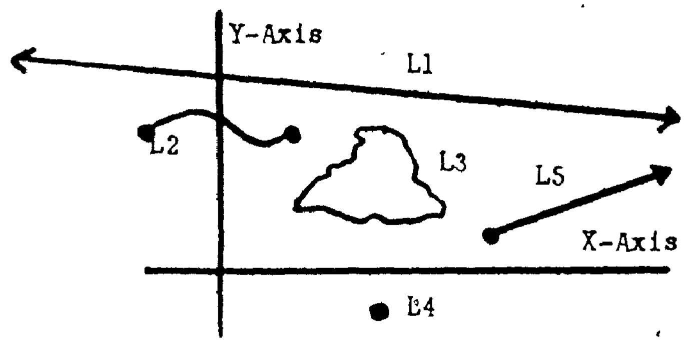


figure 2.2 Lines in XxY Plane

- a line is a set of contiguous points.

A line can have finite length (for example L2, L3, L4) or can have infinite length (for example L1, L5). A line may have no end points (L1, L3), one end point (L5) or two end points (L2, L4).

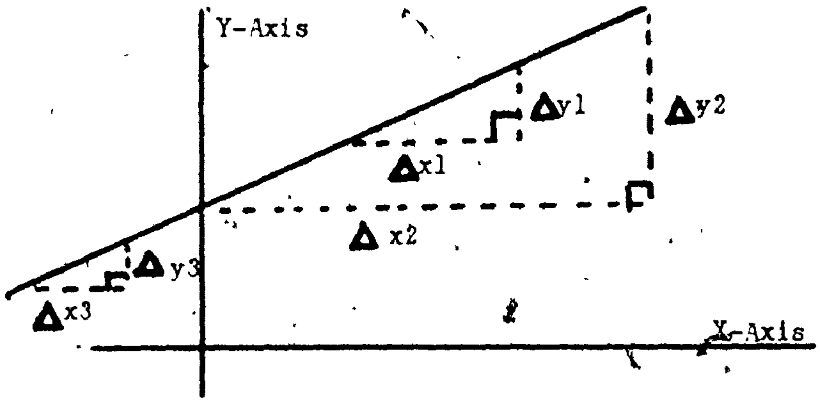


figure 2.3. A Straight Line

- a straight line is a line for which the ratio of the change in the y displacement to the change in the x displacement is constant.

i.e. in figure 2.3.

$$\frac{\Delta y1}{\Delta x1} = \frac{\Delta y2}{\Delta x2} = \frac{\Delta y3}{\Delta x3} \text{ and } \frac{\Delta yi}{\Delta xi} = K \text{ for all } i$$

This constant K is called the slope.

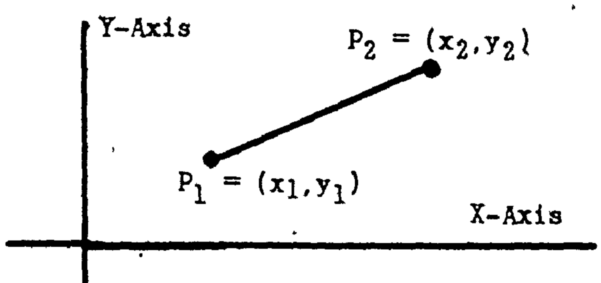


figure 2.4. A Straight Line Segment

- given two points P_1 and P_2 on a straight line, then the straight line segment of P_1 and P_2 is the set of points lying between P_1 and P_2, and including P_1 and P_2.

Henceforth, the line segment of P_1 and P_2 will be used.

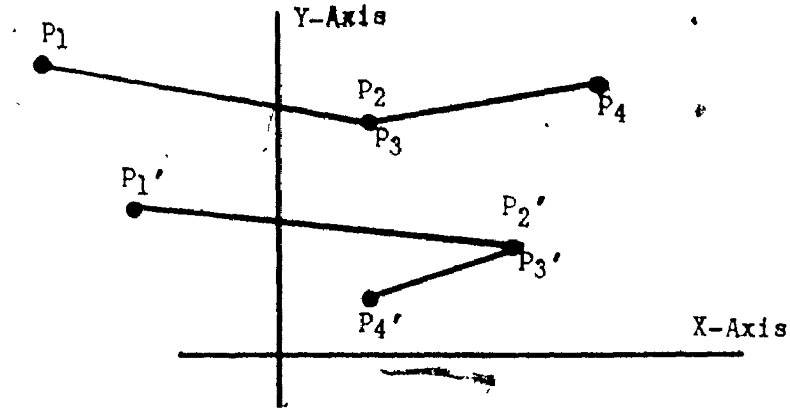


figure 2.5. Contiguous Line Segments

- two line segments (P_1,P_2) and (P_3,P_4) are said to be contiguous if $P_2 = P_3$.

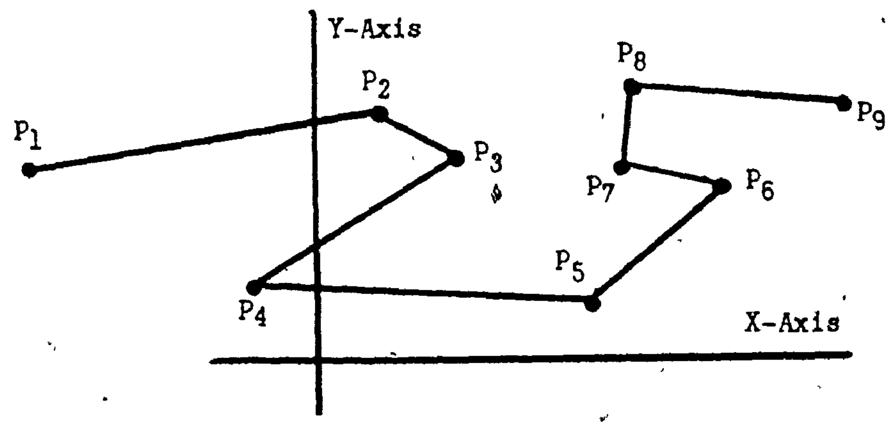


figure 2.6. N-Tuple of Line Segments (String)

- for $n>0$ an n-tuple of line segments

$((P_1,P_2), (P_3,P_4), \ldots, (P_{2n-1}, P_{2n}))$ is a string S if (P_{2i-1}, P_{2i}) and (P_{2i+1}, P_{2i+2}) are contiguous for all i, where $1 \leq i < n$

This is usually written as an (n+1)— tuple of points $(P_1,P_2,P_3, \ldots, P_{n+1})$.

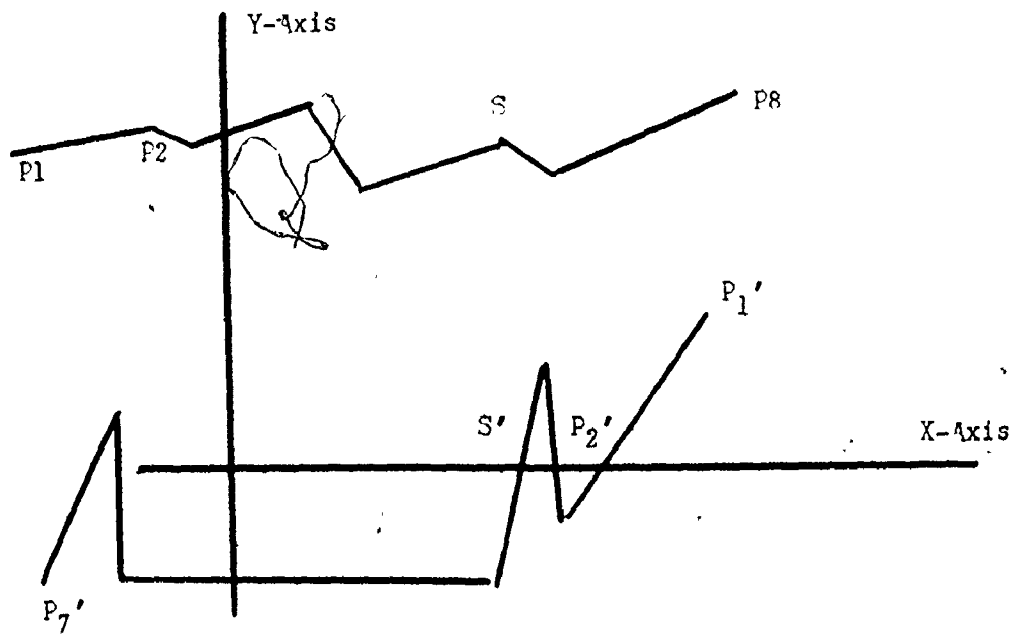


figure 2.7. <u>Monotonic Strings</u>

In this project only <u>monotonic strings</u> are used

Either a string $(P_1, P_2, \ldots, P_{n+1})$ is an <u>up string</u>

in which case $x_i \geq x_{i+1}$, $1 \leq i \leq n$

or a string $(P_1, P_2, \ldots, P_{n+1})$ is a <u>down string</u>

in which case $x_i \leq x_{i+1}$, $1 \leq i \leq n$

In figure 2.7. S' is an up string and S is a down string.

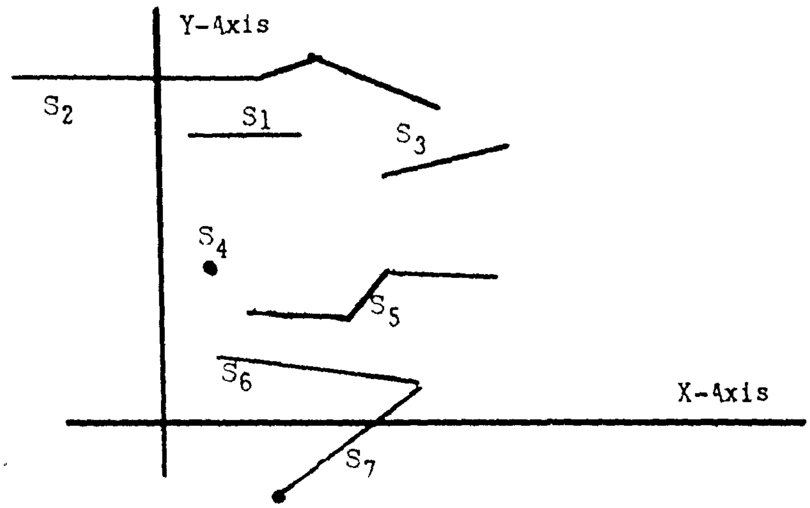


figure 2.8. List (Of Strings)

- In figure 2.8. is a set of down strings $\{S_1, S_2, \ldots, S_7\}$ An ordering is defined on this set of strings descending on the first x coordinate in each string.

The set of down strings $\{S_1, S_2, \ldots, S_n\}$ is said to be ordered if $x_i \leq x_{i+1}$, for all i, $1 \leq i \leq n$ where these x_i's are the first x value in the string S_i.

A set of ordered down strings is called a list and is written $(S_1, S_2, S_3, \ldots, S_7)$.

i.e. an ordering of $\{S_1, S_2, \ldots, S_7\}$ in figure 2.8. would be $(S_2, S_1, S_4, S_6, S_5, S_7, S_3)$.

2.3. Detailed Description of Method Used in Plot Package

As it is the convention in plotting that across the page is the Y-direction and down the page is the X-direction, subsequent diagrams and illustrations will be oriented in this way so as to better relate these illustrations to the actual plots.

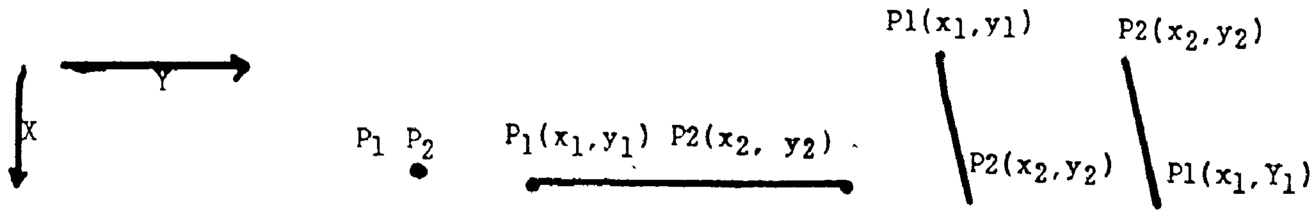
The method used in the plotting process breaks conveniently into two parts. The first part incorporates the accumulation of the plotter instructions and the reorganization of these instructions into a set of strings (or sub-plots) which can be handled by an electrostatic printer/plotter. This first part acts as a buffering process between the user's plotter instructions and the printer/plotter. This process is embodied in the "PLOT" subroutine which does the accumulation and reorganization.

The second part performs the actual plotting of the set of strings on to the printer/plotter. In this second part the set of strings is ordered to create a list. Then an interpretive process plots the information in the list. This second part is incorporated by the subroutine "PRINT".

2.3.1. Description of PASS I (PLOT) (SAVING PROCESS)

Before describing the high level method used, the low level method will first be discussed. First, the construction of the strings will be considered (i.e. the pen movements to be noted). Plotting, or moving the pen, with the pen up can be practically ignored. Only the pen position at the end of this pen movement, and the fact that the previous string, if any, has ended, is of any importance.

Plotting with the pen down is a different story. This can take place in 4 different ways as illustrated in figure 2.9.



(a) $(x_1,y_1) = (x_2,y_2)$ (b) $x_1 = x_2$ (c) $x_1 < x_2$ (d) $x_1 > x_2$

figure 2.9. <u>Possible Line Segments</u>

There are 4 cases:

case (a) - the two points P_1 and P_2 are one and the same point.

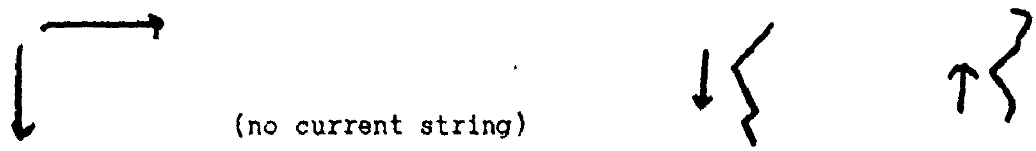
case (b) - the two points P_1 and P_2 have the same X-coordinate.

case (c) - the X-coordinate for P_1 is less that that of P_2

case (d) - the X-coordinate for P_1 is greater that that of P_2.

Notice that differences in the Y-coordinate have no bearing on classifying cases. In case (a) there is a zero length line segment which in plotting does have a physical meaning as that is how a single point is plotted. To be more precise case (a) is simply a special case of case (b) and they will be considered together since they are logically the same.

What is to be done in the various cases in figure 2.9. depends entirely on whether a string is currently under construction or not and, if a string is under construction, whether the string is an up string or a down string. Thus there are three possible situations to be considered for the occurrences of the different types of line segments. (See figure 2.10.)



(I) Pen (up) II Pen (down) III Pen (down)

figure 2.10. <u>3 Pen Situations</u>

There are 3 cases:

Case I - pen is up and no string in progress.

Case II - pen is down creating a down string (+ X-direction)

Case III - pen is down creating an up string (- X-direction)

It is temporarily feasible to construct an up string which is converted into a down string at its completion.

In Case I, a line segment as described above, is encountered with no string in progress. It is a simple case of starting a new string. When a string is started it is declared to be going up (- X-direction) or going down (+ X-direction). In the line segment cases (a,b,c) where x_1 is less than or equal to x_2 the string is declared to be going down, and when the next line segment is encountered the situation is as described in Case II. The equals case ($x_1 = x_2$) could be left undeclared until a declaration was necessary, but the saving in core would be small as the occurrence of this, where it leads into an up string, is statistically small.

In line segment case (d) where x_1 is greater than x_2 the string is declared to be an up string, and on subsequent line segments a Case III situation exists. In both cases above, the string is started by setting the current string $S = ((x_1, y_1), (x_2,y_2))$.

In case II, where a down string was already in progress, in cases (a,b,c) the new line segment is simply added to the string. Therefore for the ith line segment $S = ((x_1,y_1), (x_2,y_2), \ldots, (x_i,y_i), (x_{i+1}, y_{i+1}))$. But in case (d) where the line segment is starting to go up, the current string is terminated and an up string is started.

In case III, where an up string is already in progress, the situation is opposite to case II. In cases (a,b,d), a string is continued and to add the ith line segment to the string $S = ((x_1,y_1), (x_2,y_2), \ldots, (x_i,y_i), (x_{i+1}, y_{i+1}))$. In case (c) the current up string must be terminated and a down string started. The final strings must all be down strings, and so the up string must be converted. To convert the up string $S = ((x_1,y_1), (x_2,y_2), \ldots, (x_n,y_n)$ it is inverted, becoming the corresponding down string $S = ((x_n,y_n), (x_{n-1}, y_{n-1}), \ldots , (x_1,y_1))$. (Remark - this reversing process must also be done when in case III the pen is picked up and moved, or the terminal plot call made).

With these techniques and procedures it is possible to take any sequence of pen movements (or plotter instructions), and to organize these movements into strings (or sub-plots) and in fact into down strings. The plotting information at the end of this first pass becomes a set of strings (or sub-plots). Thus the basic philosophy fragments the unmanageable plot into sub-plots all of which are manageable (Divide and conquer).

2.3.2. Description of PASS 2 (PRINT) (PLOTTING PROCESS)

As in 2.3.1. the low level method is described first. First the method that the printer/plotter uses to plot must be considered. The printer/plotter plots by putting points onto the output line as required. Successive lines form a plot or picture, in the same manner as a television picture is formed. Consider the plotting of a line segment over 5 or 6 of these lines: as the slope of the line changes a problem becomes evident. (See figure 2.11.)

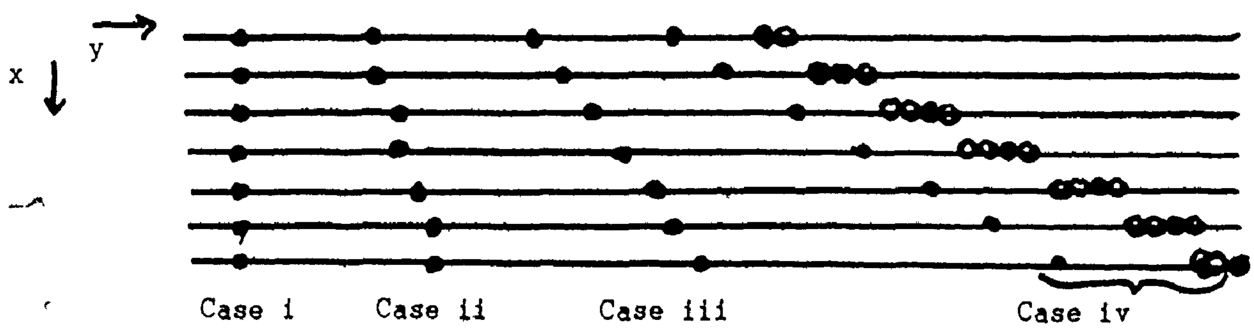


figure 2.11. Line Segments Over 7 Plot-Lines

Starting in case (i) and case (ii) where the slope is small there is no problem as a continuous line results. This holds true until the slope becomes greater than 1. Then in case (iv) where the slope is very large there are large gaps in the line: it is smooth but the lack of continuity is undesirable. Looking at case (ii) there is another slight problem but this problem is actually a solution to the first problem. In case (ii) the line is staggered with a few rough edges, but at least it is continuous. There is no solution to this problem and it is preferable to the disjointed line as in case (iv). Therefore changing case (iv) to be like case (ii) as indicated by the empty circles in figure 2.11. is the answer. A few rough (or staggered) lines will occur

but continuity takes precedence. The solution is to interpolate backwards half way and forward half way. Then a range is plotted for each line depending on the slope, and, in fact, is equal in length to the value of the slope. (This is because lines are 1 unit apart ($\Delta X=1$).
In this way each string creates a range on the line.

PASS 2 (or PRINT) is entered when the plot has been completed and is in the form of a set of down strings (or sub-plots). This pass faces the challenge of decoding (or interpreting) this set of strings and producing the desired plot.

It is quite possible to handle this set of strings as they are. However, this would be very inefficient since each string would be checked on each line although few strings extend over the whole range of x values. So in the interest of efficiency an ordering of the set of down strings is made to produce the corresponding <u>list</u>. (Note that until this ordering is done the set is not necessarily a list). At this point the starting x values for each string are in ascending order. This means that when a string S_j occurs where the starting x-value in S_j, x_j is greater than the current x-value, x, we know that this string and all strings from S_j to S_n can be ignored for this x. When this S_j is reached or after handling S_n, the output line image is plotted. Excessive work is also avoided by removing the string when it is finally exhausted. Both these techniques are illustrated in figure 2.12.

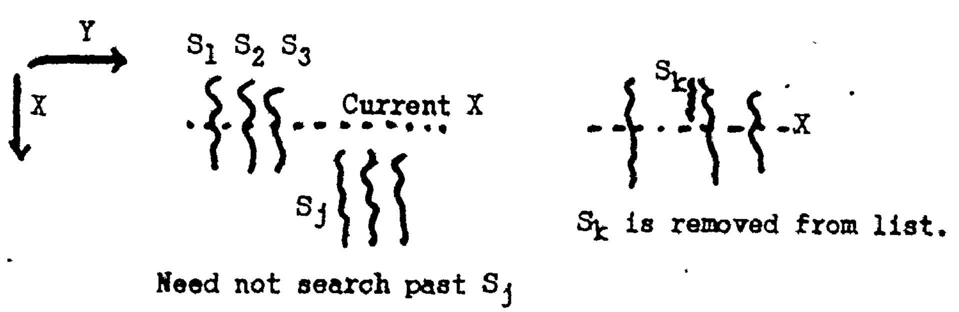


figure 2.12. <u>Logical organization of strings</u>

In the diagram in figure 2.12. the strings are shown not as they physically occur in the plot, but where they logically occur in the list. These two techniques help to minimize the work to be done. This is a destructive process as strings are destroyed when they are used up. Also note that when the last string is exhausted the plot is finished and out on the paper.

How does one handle the individual strings? Because of the careful construction in PASS I, it is known that these are all down strings. A down string is simply a contiguous n-tuple of monotonic non-decreasing (x value) line segments, and the only problem is how line segments are handled. The correspondence between the x value and the line segment involved must be considered. This can occur in five logically different ways as illustrated in figure 2.13.

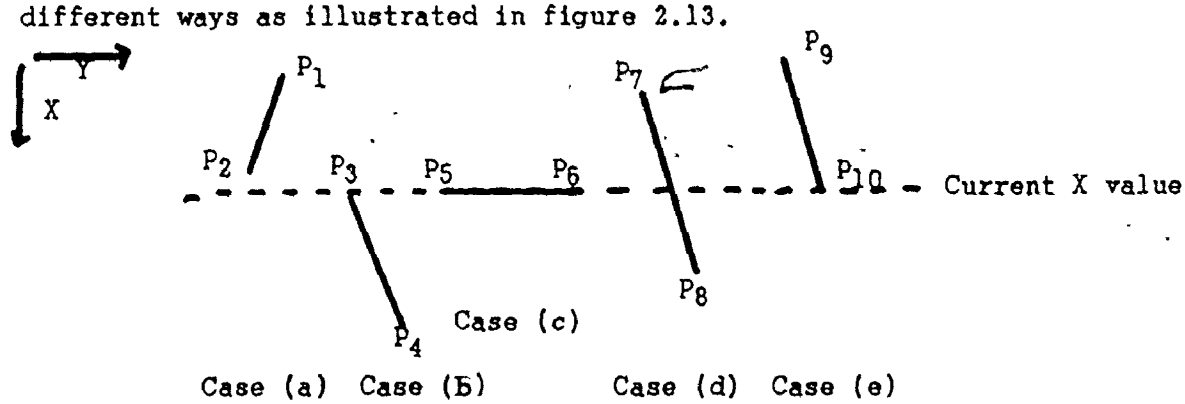


figure 2.13. X Value and Line Segments

There are 5 cases:

Case (a) the line segment (P_1, P_2) lies below current x value.

i.e. $x_1 < x_2 < x$

Case (b) the line segment (P_3, P_4) starts at the current x value. i.e. $x_3 = x < x_4$

Case (c) the line segment (P_5, P_6) lies on the line $y=x$

i.e. $x_5 = x = x_6$

Case (d) x is inside the line segment (P_7, P_8)

i.e. $x_7 < x < x_8$

Case (e) x is at the end of the line segment (P_9, P_{10})

i.e. $x_9 < x = x_{10}$.

Case (a): the line segment lies completely above the x value. This line segment is skipped, and it is deleted from the string. The next line segment is checked, again as one of the above cases. If no line segments are left, the string is deleted from the list. Case (a) can occur only at the initial x value:

i.e.- if $S = (P_1, P_2, \ldots, P_7)$ then S becomes $(P_2, P_3, \ldots, P_7)$ and the line segment (P_2, P_3) is immediately considered;

- in the special case $S = (P_1, P_2)$ S would no longer be a string so the entire string is deleted from List.

Case (b): the line segment starts at the x value. From figure 2.13. it is clear that y_3 must be included in the plot, but also we must interpolate towards y_4. As the x's change by 1 unit for each line, it turns out that the next y value is $\underline{y_3 + \text{Slope}}$ and to interpolate half way we go to $\underline{y_3 + 1/2 \text{ Slope}}$. So, in order to get a continuous line for this line segment the points between and including $\underline{y_3}$ and $\underline{y_3 + 1/2 \text{ slope}}$, are put into the line buffer.

Case (c): (P_5, P_6) lies on the line $y = x$. The range of y values from $\underline{y_5}$ to $\underline{y_6}$ are put into the line buffer.

Case (d) x lies inside the line segment and not at an end point. This is the general case where y values in the line must be interpolated and the range (y - 1/2 slope, y + 1/2 slope) put into the line buffer.

Case (e): x lies at the end of the line segment. This is similar to case (b) and is handled in a similar fashion, as the y values from (y_{10} - 1/2 slope, y_{10}) are put into the line image.

In addition, in cases (c) and (e), an extra step is necessary. The line segment is skipped as in case (a), and the next line segment checked immediately, if one exists. If no line segments are left the string is deleted from List.

Note that both the old line segments and the old strings are destroyed as soon as they are finished. It is also noteworthy that this iterative process allows the processing of strings as well as line segments, since the end of one line segment triggers the consideration of the next line segment. When no more line segments exist in the string, the string is automatically destroyed.

So in brief, the set of down strings is converted into the ordered set of strings, LIST. Then by processing only the relevant strings for each x, and by interpolating as necessary for each line segment, the line buffer is formed and plotted. This continues until all the strings have been processed.

2.4. Description of Theory Used in Graph Utility

As stated earlier the Graph utility is that special case of a plot in which only increasing x values occur. This particular utility is designed to plot functions and, to be more precise, to plot functions with unique values for all x values. Although not theoretically important the package can handle several functions at once.

The theory upon which the graph utility is based is similar to that theory in the plot package. In order to plot continuous lines interpolation is necessary. In this case instead of using line segments and slope, the actual x-values are used. Y-values are calculated for each value of x, and in order to interpolate we must look back to the last y-value and ahead to the next y-value. Thus three y-values are used to determine the range, the last y, the present y, and the next y. The interpolation simply involves halving the two intervals. Then these two interpolated values, together with the present y-value, are used to determine the range of y-values to be put into the line image. As seen in figure 2.14 below, the present y must be included to guard against the case on the right. In this case the present y does not lie in between the interpolated values but must be in range to be put into the line image.

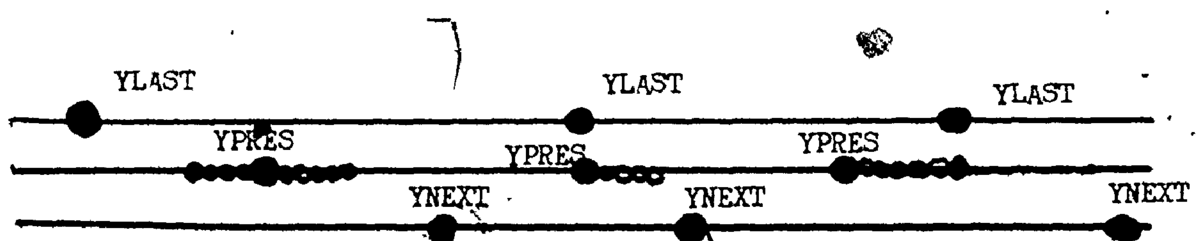


FIGURE 2.14. Graphing Using 3 Consecutive Y-Values

The X-Axis and the Y-Axis are included automatically in the graph. In addition there are borders on each side and most important the limits of the graph and the scales used are reported. The Axes give the user reference points and the reporting of the limits and scales guards against keypunching and programing mistakes which could cause a communications breakdown (i.e. the user thinks he asked for one thing, but the computer is actually told something else).

CHAPTER 3

IMPLEMENTATION DESCRIPTION

This chapter explains how the theory developed in Chapter 2 is converted into a plotting system.

3.1. General Description and Breakdown of the Plot Package

As explained in Chapter 1, plotting is accomplished by describing sequences of pen movements. The pen is instructed to move from its current location to a new location, either with the pen up (and not plotting) or with the pen down (and plotting). As few plots are made up of one continuous line, the ability to move to a location without plotting is necessary. The plotter instruction is issued as a call to the subroutine PLOT. The user may call PLOT directly or may call special purpose plot utilities which in turn call PLOT. However, it is necessary that PLOT be called in order to perform the actual plotting. It is for this reason this Plotting Package begins with routine PLOT.

Figure 3.1. is a pictorial representation of the PLOT Package. In brief, the user's program either directly or indirectly calls PLOT with the plotter instructions. PLOT takes these calls, and reorganizes them into strings. The strings are stored away and upon a completion call, PLOT calls PRINT which does the actual plotting. PLOT has a utility routine called INVRT which converts an up string into a down string. PLOT and INVRT make up the first pass of the plotting package.

PRINT takes the strings created in PLOT, uses the utility ORDER to create the ordered set of strings (or list), and then by using two more utilities INSRT (to put points into the line buffer) and BFOUT (to plot the contents of the line buffer), plots the line segments contained in the list. Then control is passed back to PLOT and thence back to the user. PRINT, ORDER, INSRT, and BFOUT make up the second pass of the plotting package.

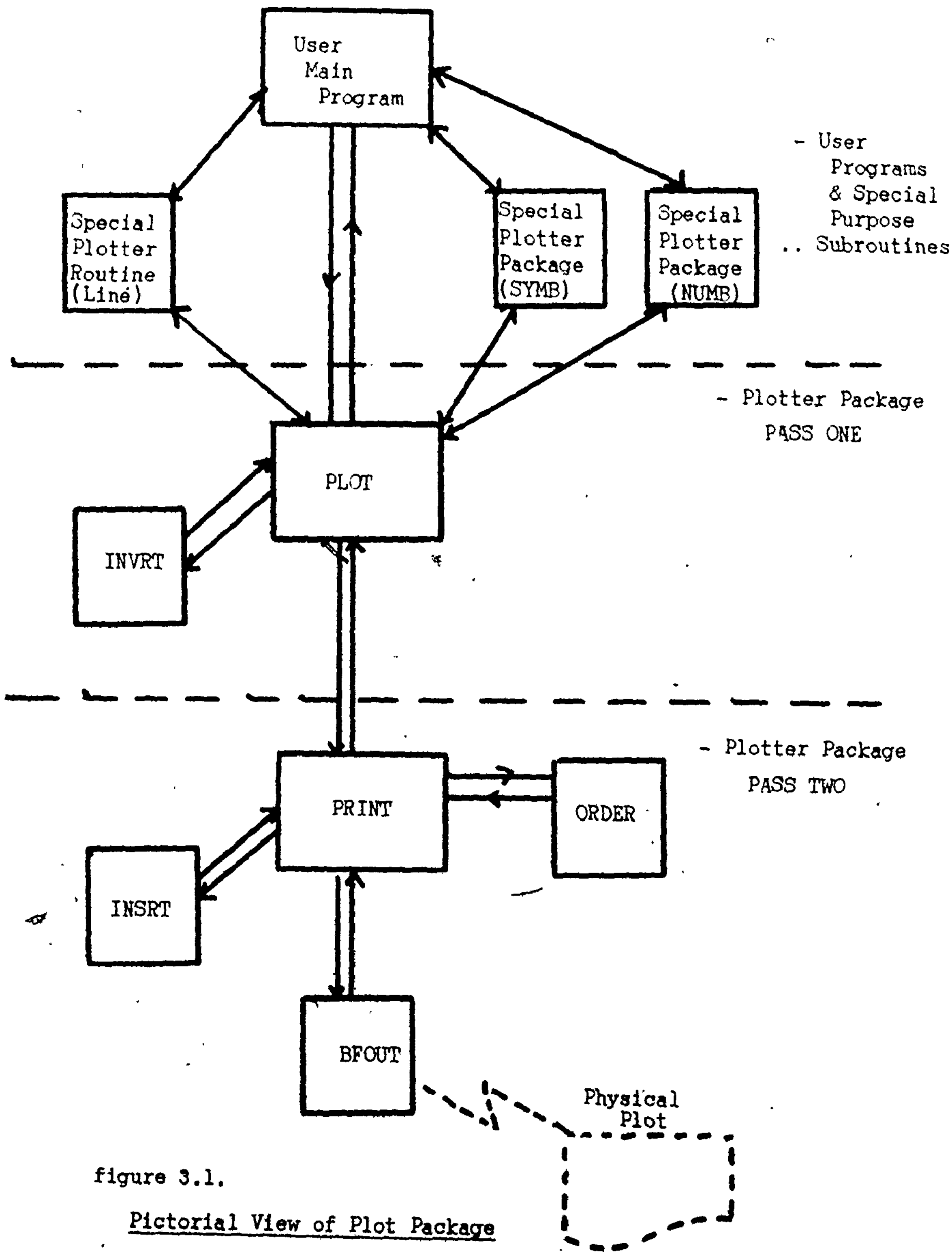


figure 3.1.

Pictorial View of Plot Package

3.2. "PLOT" Routine

This routine is the basic routine in the plotter package, and was written to be compatible with the Benson-Lehner Plotting System. Both the user and the high level plotter routines (e.g. SYMBOL) use PLOT to do their plotting. The calls to PLOT contain three arguments (X,Y, and PEN). X and Y are reals which usually contain the target location to which the pen will be moved, and PEN is encoded with either the pen position (up, off the paper, or down, on the paper) or special plot calls. These special calls allow for the setting or resetting of the origin, for the establishment of translation, and for returning the current pen position in the XY-plane. On the very first call to PLOT the scale factor can be adjusted to expect calls in centimetres rather than in inches (which is the default scale factor). Figure 3.2. is a table of plot calls with the corresponding action to be taken.

On the first call the needed initialisation is done, including resetting the scale for centimetres if required. Then from this point, on the first call, and on all subsequent calls, we proceed to convert X, and Y to integer plotter units, and to use current origin and translations to obtain an (X,Y) position in absolute plotter units (1 plotter unit = 1 step or 0.0138 inches).

Now we concern ourselves with PEN, according to the actions indicated in the table in figure 3.2. We shall ignore the special calls to PLOT as these are mainly bookkeeping problems. The main concern is when the pen is down (PEN=2) and plotting. Both up strings and down strings are constructed and stored sequentially in array LIST. There is a tag array, TAGS, which is set to the index of the first member of the string, when

PEN value	Action
-20	- only on 1st call, set scale for centimetres
-3, -2, -1	- set origin to arguments (X,Y)
	- change sign of pen and decode pen again.
0	- return current pen position to user
1	- plot with pen unchanged
2	- put pen down and plot to (X,Y)
3	- pick pen up and move to (X,Y)
40	- reset translation
999	- end of plot, put plot out.

figure 3.2. TABLE OF PEN CALLS

that member is put into LIST. Two words are left in front of each string, the first is set to zero and used later by PRINT, and the second is a pointer to the start of the next sequential string, in array LIST. To better see this structure, see figure 3.3., where the organization is illustrated.

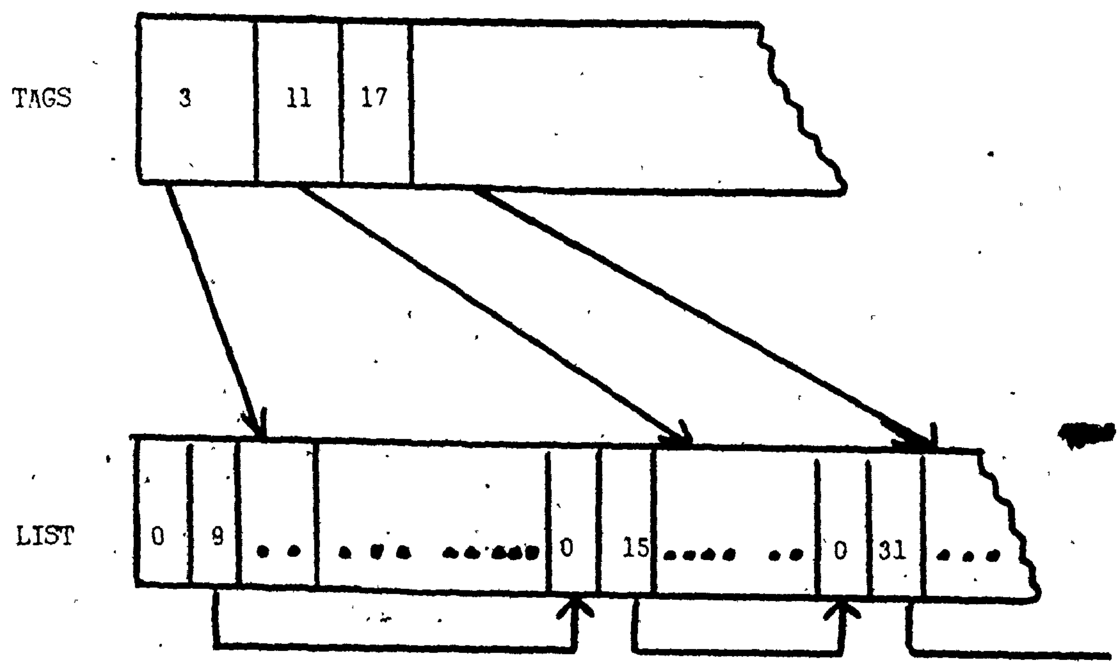


figure 3.3. TAGS ( ) and LIST ( ) STRUCTURE

An up string is stored with coordinate pairs backwards in LIST. Therefore instead of the sequence x_1, y_1, x_2, y_2, ... as in a down string, the coordinates are stored y_1, x_1, y_2, x_2, ... instead. Then when an up string has been completed the INVRT utility is called to reverse the up string to form the corresponding down string. The coordinates are stored in this strange manner because it simplifies the INVRT routine.

When a terminal call to plot is made (PEN=999), plot calls PRINT to put the plot onto paper.

3.3. "INVRT" Routine

This is a service routine for subroutine PLOT. It takes an up string and converts it into a down string. This conversion has been facilitated by the storing of the up strings with the coordinates backwards ((y,x) instead of (x,y)). To convert the string (y_1, x_1, y_2, x_2, ..., y_n, x_n) into a down string simply interchange the n pairs (y_1, x_n), (x_1, y_n), (y_2, x_{n-1}), ...
to get (x_n, y_n, x_{n-1}, y_{n-1}, ... , x_1, y_1).

3.4. "PRINT" Routine

PRINT is called from PLOT upon a terminal plot call (PEN = 999). PRINT is called with three arguments:

LIST - which contains the set of down strings,

TAGS - which contains the pointers to the start of the strings in LIST

and ITAG - which is the number of tags in TAGS (and also the number of strings in LIST). Note that at this point all strings are down strings.

PRINT takes this information and creates a plot in the following manner:

First ORDER is called to perform a tag sort on the strings in LIST, by reorganizing the tags in TAGS so that the initial x values of the strings indicated in consecutive TAGS are in ascending order. This process converts the set of strings in LIST into a list of strings as defined in Chapter 2.

Then we start at x = 0 and for increasing x values the following procedure is followed. Starting at the first string, the range of y values is determined for each string for this x value, and with necessary interpolations performed. These values are inserted into the line buffer using utility INSRT. We continue until we find either the last string or a string where the initial x value is greater than the current x value. Then the line buffer is plotted onto paper by routine BFOUT. Each string is deleted when it is completely plotted, and this whole process continues until no more strings exist. At this point we have finished and PRINT returns control to PLOT.

3.5. "ORDER" Routine

This routine is called from PRINT in order to convert the set of strings into a list. As Fortran does not allow string or pointer types, strings are stored sequentially in an array, LIST, with a corresponding tag array, TAGS. In this way the string data structure is achieved. In order to sort the strings, ORDER performs a tag sort on the tag array TAGS, and the array LIST.

The logic in this routine was obtained from a previous fourth year project at McMaster on sorting, and this was a normal sort routine. It was considered in their report to be in the top 2 or 3 of the routines that they considered but, more important to this project, only 3 or 4 lines had to be changed in order to make it into a tag sort. The sort logic seems to be very efficient and, both small (in core taken) and simple.

As this was written by someone else no attempt will be made to describe the method. The source is well anotated to indicate which changes were made.

3.6. "INSRT" Routine

PRINT calls INSRT to put a point into the line buffer. The formal parameter POINT is an integer number from 1 to the number of points in the line buffer, indicating the bit to be placed in the line buffer (LINE). As this machine is not bit addressable, the word address is determined, and a bit inserted into this word at the proper position. This routine could have been written in Fortran, and in fact a Fortran version was written and tested. The Assembler version is about one half as large (34 words versus 67 words) and much more efficient (i.e. in number of instructions to be executed). Since this routine is used so often, the Assembler version is used in the implementation, and the Fortran version inserted only into the documentation.

3.7. "BFOUT" Routine

PRINT calls BFOUT to plot the line buffer, LINE, on the printer/plotter. The plot is accomplished by an executive call. This call does a binary write - without - wait to the printer/plotter. This means that the output operation is initiated and allowed to go on automatically while program control is returned to PRINT. This allows PRINT to be concurrently processing the strings for the next value of x.

Through careful testing it was found that three speeds could be obtained on the plotter through different methods of plotting, 10000 lines were plotted and the plotting time determined by subtracting the looping time from the elapsed time. The cases and times are as follows:

1) writing-with-wait (normal) took 108.1 seconds.

2) writing-without-wait and checking for status before starting successive write took 117.9 seconds.

and 3) writing-without-wait with no status checking took 89.7 seconds.

In each of the above cases, including case (3), there was no problem with overwriting of the buffer. It was also found that plot speed could be increased by writing large buffers (3500 words instead of 35 words). It was only here that overwriting of the buffer could happen, if no status check were made. This is true on our machine but to help to protect the users of other machines buffer copying is done (though it is not necessary). The line buffer LINE is copied into BUFF and simultaneously LINE set to zero for the next x value. Then BUFF is written instead of LINE.

3.8. Reason for Version II

The plot package as described so far is quite satisfactory. In fact, this is probably the best version from a theoretical point of view. But unfortunately on the HP2100A there is a severe restriction. Only about one thousand calls to plot can be made, and although in some cases this is sufficient, this restriction could seriously hamper some plot users. The reason that only 1000 calls to plot are allowed is that the calls are stored in an (x,y) coordinate pair in the array LIST. Since the HP2100A only has

12K decimal words of core, and only 8K is available to the user, after his program is added only 2000 words are available for LIST without overflowing the available space. The word length is 16 bits and so packing is not feasible (as it would be on the CDC6400 with 60 bit words). Therefore there is no easy solution that will allow larger plots.

Obviously any expansion of the plotting ability is going to have to involve the disk. So what we have to do is, to use the disk and to take advantage of the fact that we basically have 2 passes, PASS 1 (PLOT, INVRT) which accumulates the plotter information and PASS 2 (PRINT, ORDER, INSRT, and BFOUT) which plots the information on the paper.

The solution arrived at incorporates both these features.

1) only one string at a time is kept in core. As soon as a string is finished it is put into a sector buffer and written to disk (Routine CLEAR). Instead of having an array LIST in core we gradually construct an image of LIST on disk. The assignment (LIST (I) = ...) and the accesses (. = LIST (I)) of array LIST are simulated with a subroutine PUT and a function LIST respectively. To tidy up the last sector buffer a call to CLSEF writes the last sector to disk. Also the calls to INVRT are handled through routine IVRTA. All these routines share common storage and are written in Assembler. This set of routines is called DSKIO. In the diagram in figure 3.4. we are at step 1.

2) Next we start to swap out core with a call to SWAP1 (Step 2 in figure 3.4.). SWAP1 saves core from 24000B to 30000B onto disk and replaces it from a binary data file called "SWAP2"

(Step 3 in figure 3.4.). Then control is passed to "SWAP2". Upon return the core swapped out in Step 2 is swapped back in (Step 9).

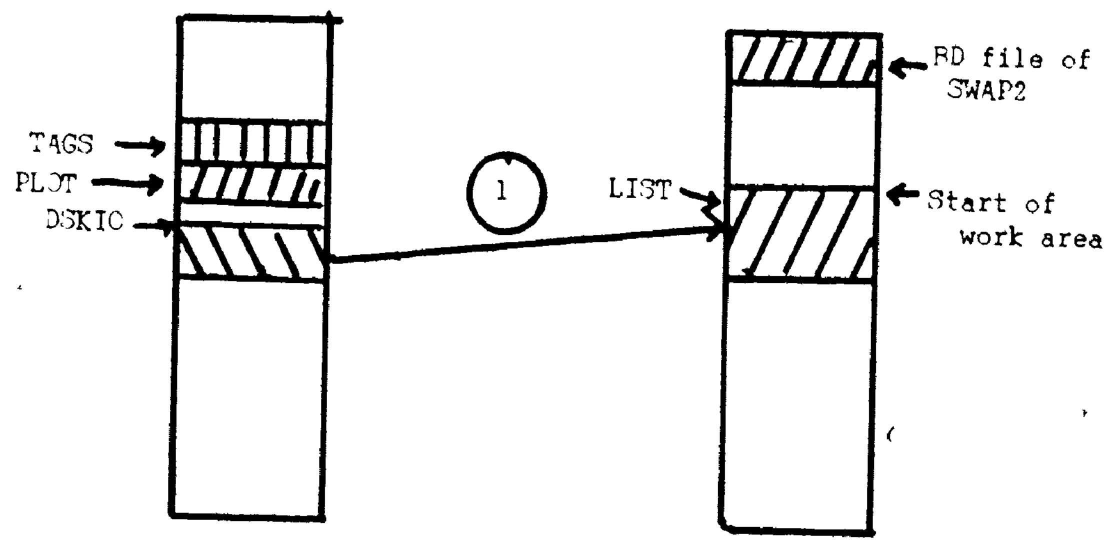
3) SWAP2 swaps out the core from 10000B to 24000B (which will be called LO-CORE in this section). This is Step 4 in figure 3.4. Next, the tag array TAGS is moved down to the start of user core (10000B) (Step 5 in figure 3.4.). Then the LIST array is read in from disk and placed right after the TAGS array. (Step 6 in figure 3.4.) Next the normal call to PRINT is made (Step 7) and then LO-CORE is reset (Step 8). Finally we return to SWAP1.

The above is the basic process needed to swap core. Unfortunately all of this is complicated because the HP2100A is a page-addressable machine and about 100B words of base page links must also be carefully swapped when entering and leaving SWAP2. The binary date file "SWAP2", which contains the object code for SWAP2 (in a loaded form), is configured by the program SETHI which only has to be run once.

All this work gains us the core from 10000B to 24000B for use by TAGS and LIST. This represents about 6000 decimal words which leaves 5500 words for LIST, representing an additional 3500 words for LIST over Version I (or about 1750 additional plot calls). Note that this is the true measure of the expansion of Version II over Version I as more core will have no effect on the difference between the two versions (it will still be 1750 additional plot calls). In fact, at about 20K, Version I would frequently [illegible] the better version for simple plots as there would be little if any restriction left.

- During PASS 1

in PLOT



- During SWAP1

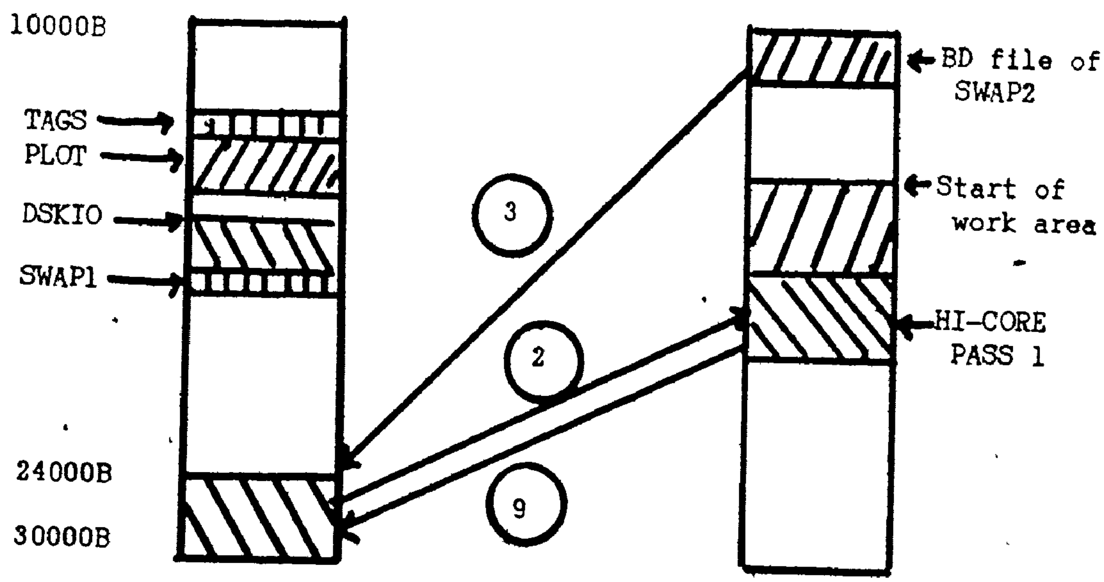
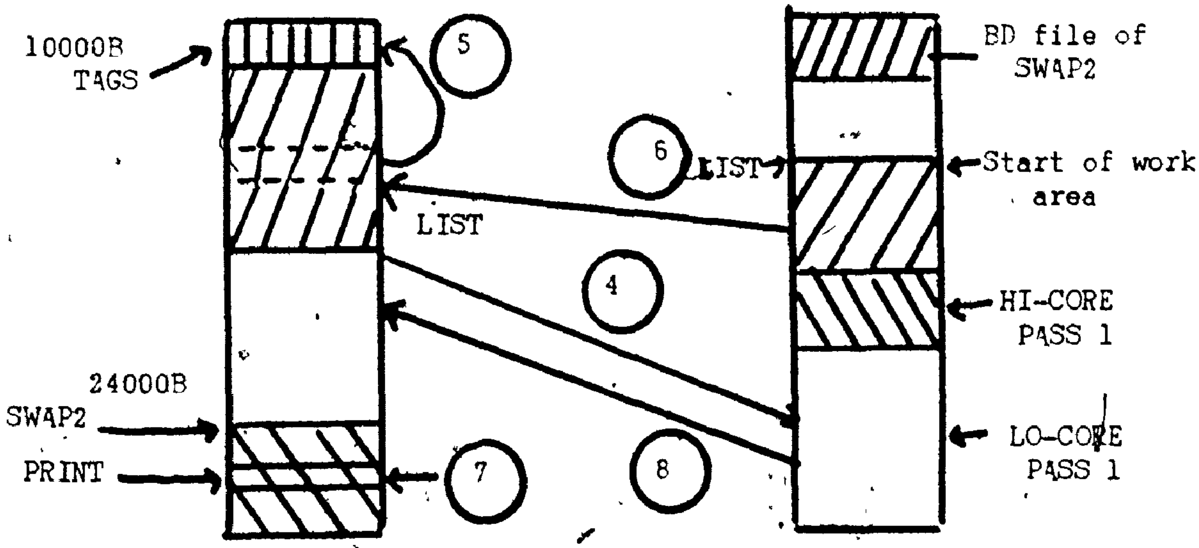


figure 3.4 (page 1 of 2)

- In SWAP 2



1 put LIST out to work area

2 copy HI-CORE PASS 1 to work area

3 copy "SWAP2" into HI-CORE, call SWAP2

4 copy LO-CORE PASS 1 to work area

5 move TAGS to 10000B

6 read LIST into core

7 call PRINT & plot

8 reset LO-CORE PASS 1 into LO-CORE, return to SWAP1

9 reset HI-CORE PASS 1 into HI-CORE, return to PLOT

figure 3.4. Version II of PLOT Package

3.9. "PLOT" Routine (Version II)

There are no logic changes whatsoever between Version I PLOT and Version II PLOT. The only changes necessary are in accesses of the array, LIST. An assignment to the array is changed into a CALL PUT (value, index, ...). Unfortunately several flags from PLOT have to be passed in case we overflow the string buffer in the Disk I/O package. A reference to the array LIST doesn't change in the code, but uses a function with the same name rather than an array access. Because the string record is in the Disk I/O package, the call to INVRT becomes a call to IVRTA which calls INVRT. Finally there are two subroutines CLEAR, called to put the string out when it is terminated, and CLSEF, which puts the last sector onto the desk if necessary.

Then instead of calling PRINT we call SWAP1 which begins the core swapping process that ultimately calls PRINT.

3.10. "DSKIO" Package

This is the assembler package that includes:

1) PUT - to simulate an assignment (LIST (I) =)

2) LIST - to simulate a reference (= LIST (I))

3) CLEAR - take strings when they are done and route them to disk.

4) CLSEF - if last sector not written to the disk write it and

5) SWAPI - to start core swapping process that ends with PASS 2 in core with LIST array (from disk).

This package was written in Assembler because this allows for common buffers, pointers and flags. In addition, it was much easier to write in Assembler than in Fortran. Since this is a very machine dependent feature writing it in HP Assembler is quite reasonable.

3.10.1. "PUT" Routine

This routine simulates an assignment statement involving the array LIST. A string record is local to this routine, and after checking that the index is either in the string or after it, the value is put into the string record (i.e. no assignments are allowed before the starting index of the string, and in fact it should be impossible for PLOT to make such a call but the error checking is left in anyway). The only limit on these insertions is the size of the string record, 128 words. If this limit is reached, overflow occurs. The string is ended and a call to INVRT made if necessary. Then a new string is started (i.e. we break a large string into parts)

3.10.2. "LIST" Function

This routine simulates a reference to the array LIST. Again, as in PUT, the index is checked to see that it is referring to an element in the current string. Then the value is returned. (i.e. ...= LIST (I) — returns the Ith element of LIST)

3.10.3. "CLEAR" Routine

This routine takes the current string and starts putting it into the current sector buffer. When the buffer becomes full, it is written onto the disk in the work area. We continue until we have all of the string. This is essentially "flushing" the string record.

3.10.4. "CLSEF" Routine

This routine takes care of the last sector buffer mentioned in 3.10.3. If the buffer is empty we don't do anything, otherwise we write it out. This is essentially closing the file.

3.10.5. "SWAP1" Routine

This is called from the PLOT routine and initiates the swapping sequence. First HI-CORE is written to the work area on disk. Then HI-CORE is loaded from the binary data file names "SWAP2". Finally we transfer control to SWAP2 which continues the swapping sequence. Upon return SWAP1 resets HI-CORE as it was initially.

3.11. "SETHI" Package

This package includes the "SETHI" program and the "SWAP2" routine. SETHI is a routine which configures SWAP2 and writes the binary version onto data file called SWAP2. SWAP2 is the second part of the swapping sequence which saves LO-CORE, moves TAGS down to the first word available in user memory, reads in the LIST array from disk and then calls PRINT to plot the information. When PRINT is done SWAP2 resets LO-CORE and returns to SWAP1.

3.11.1. "SETHI" Program

This is a main program which configures the SWAP2 binary code in HI-CORE and then writes out the code to the binary data file called SWAP2. SWAP2 is configured by forcing it to load at address 24000B and copying the link area near the end of core, so that it can be restored when SWAP2 is executed. Then we have enough information that it can be saved on "SWAP2".

3.11.2 "SWAP2" Routine

This routine completes the swapping procedure. First the link area, which had been saved near the end of core during the execution of SETHI, is swapped with the link area (from the user's program). Then the user core below the partition point (10000B to 24000B) is written to the work area on disk. Next the tag array TAGS is moved down to the start of the user area of core (10000B). Then immediately after the last tag, we read in the LIST array from disk (which was saved in PASS 1).

Now we call PRINT to do the plotting, and when the plotting is completed, all these steps must be undone so that we can return to the user normally. So the core where TAGS and LIST is now (10000B to 24000B) is reset. Then the link area is reset to that of the user. Now we return.

3.12. "GRAPH"Utility

This utility is provided to allow the plotting of functions in a simple way. It is designed to handle functions which have a unique value for each x. Instead of a two pass system, GRAPH is a one pass system where the functions are evaluated, and the function values plotted as soon as they are available. GRAPH also allows the user to plot several functions simultaneously on the same graph. There exists no real limit on this number, except an artificial one of 10 which is imposed because three arrays had to be declared with dimensions equal to the number of functions allowed.

The GRAPH utility was designed to be as easy to use as possible, while still allowing enough features to make it useful. The author hopes that a happy medium was achieved.

The output is bordered on all sides, and should the X or Y axis occur in the frame requested, it is automatically plotted. Then complete reporting is done for the minima, the maxima, and the scales used for both X and Y.

3.12.1. "GRAPH" Routine

This routine is called directly from the users program. The user specifies a frame by:

1) X minimum and X maximum

2) Y minimum

and 3) X scale and Y scale

Then the user indicates how many functions are to be plotted on the graph, and the name of the function. The user supplies these functions by supplying a function of the form F(I,X) where X is the x value to be used in the evaluation and I chooses which function to be used. Note I could also limit the number of terms in a series to be used in evaluation. See figure 3.5.

```
FUNCTION F (I,X)

F = 0.493
IF (I.EQ.1) RETURN
F = F + 0.32 * X
IF (I.EQ.2) RETURN
F = F + 1.23 * X * X
IF (I.EQ.3) RETURN
.
.
.
END
```

figure 3.5. Example of Possible Function Routine

Graph works fundamentally in the same way as plot except that the y values are evaluated each time instead of using line segments and their slopes. In fact the utilities INSRT and BFOUT from the PRINT program are also used in the GRAPH routine. Just as PRINT interpolates values, GRAPH interpolates, only using actual values in each case. The plots produced from GRAPH are a little more continuous, as no round off error can occur (since the function value is calculated for each plot line). Examples of plots from GRAPH are included in Chapter 4.

CHAPTER 4

RESULTS

This chapter illustrates and comments on the output from the plot package, and from the graph utility. Plots were made with both the Benson-Lehner on the CDC6400 and the Versatec on the HP2100A. These plots are compared, discussed, and criticized. The case study approach is used, and there are basically two cases undertaken for the PLOT Package and two cases undertaken for the Graph utility. No attempt was made to try any special purpose plotter routines (e.g. LETTER) as these are not yet available on the HP2100A. Admittedly, routines such as LETTER, generate numerous calls to PLOT, and these might tend to push the PLOT Package to the limits of its capacity. (It is easy to overload the PLOT Package in any event.)

4.1. Results of the Plot Package

Two case studies were chosen to provide some non-trivial examples of plotting, and also to show the strengths and weaknesses of the Electrostatic printer/plotter. The Benson-Lehner on the CDC6400 (at McMaster) was used to plot the counterparts of the EPP plots, so that comparison could be made. When comparing these plots, keep in mind that the Benson-Lehner has a stepsize or precision of 0.005 inches compared to the Versatec at 0.0138 inches. It is not possible for this printer/plotter to compete evenly with the precision of the Benson-Lehner.

4.1.1. Plotter: Case Study I (SNOOPY)

This plot program draws a picture (of Snoopy) by reading in plotter instructions from a card deck. Each card contains one set of plot parameters. The program for this is very simple (see figure 4.1.) for very good reasons. This same program with a different data deck could plot a different picture (Charlie Brown). Therefore the plot from this program is dependent solely on the data supplied and not on the program.

The plotter output from the Versatec is shown in figure 4.2.1. and the plot from the Benson-Lehner is figure 4.2.2.

Comments on Versatec "SNOOPY" and Benson-Lehner "SNOOPY".

1. The two plots are virtually identical in size, in shape, and in general appearance.
2. The most striking difference is the darkness of the Benson-Lehner "SNOOPY". This is due to two differences in the machines. The Benson-Lehner uses an ink that is much darker than the Versatec plotting, and the Benson-Lehner pen draws a line that is thicker than the dot size on the Versatec. (See under "SNOOPY'S" mouth).
3. Next the Versatec plot looks rougher and disjointed in spots, where the Benson-Lehner is smooth and continuous. There are several reasons for these differences. The precision difference accounts for some of the roughness: also, the thicker pen will tend to cover up some small bumps in the output (they will not be quite as noticeable). Because a pen is physically being used on Benson-Lehner, continuity is no problem compared to the Versatec where with this precision even contiguous points look slightly disjointed. Finally, the word size on the HP2100A (16 bits) results in some round off error, which shows up on the plot as a line that doesn't

TEST PROGRAM #.1 [SNOOPY] PAGE # 1.

```
FTN4,L
      PROGRAM SNUPY
C
C
C
      CALL PLOT( 2.0, 2.5, -3 )

 50   READ(5,1) X, Y, N
      IF(X.GE.99.0)  GO TO 100
      CALL PLOT( X, Y, N )
      GO TO 50
100   CONTINUE
      CALL PLOT( 2.0, 0.0, -3)
      CALL PLOT(0.0, 0.0, 999)
  1   FORMAT(2F9.4, I3)
      STOP
      END
      END$
```

figure 4.1. Test Program "SNUPY"

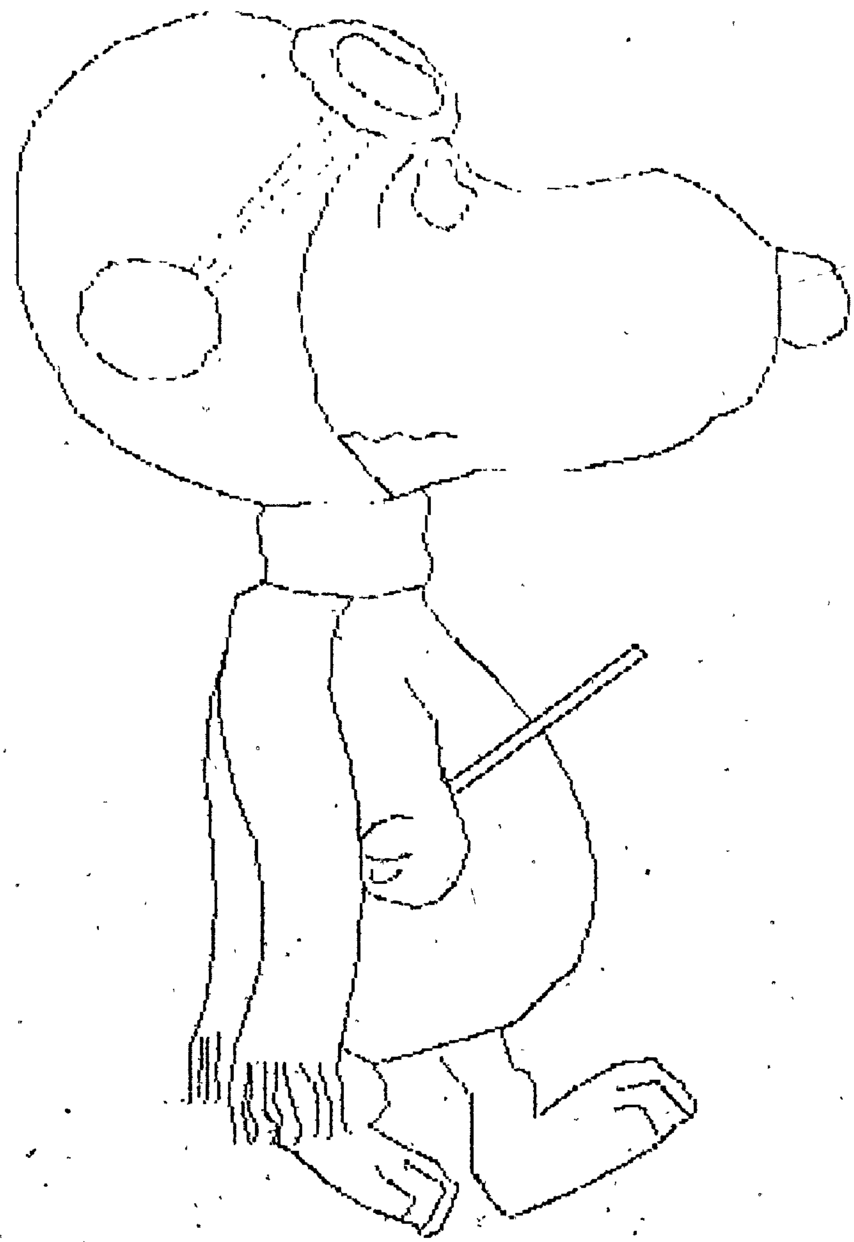


figure 4.2.1. "SNOOPY" on EPP

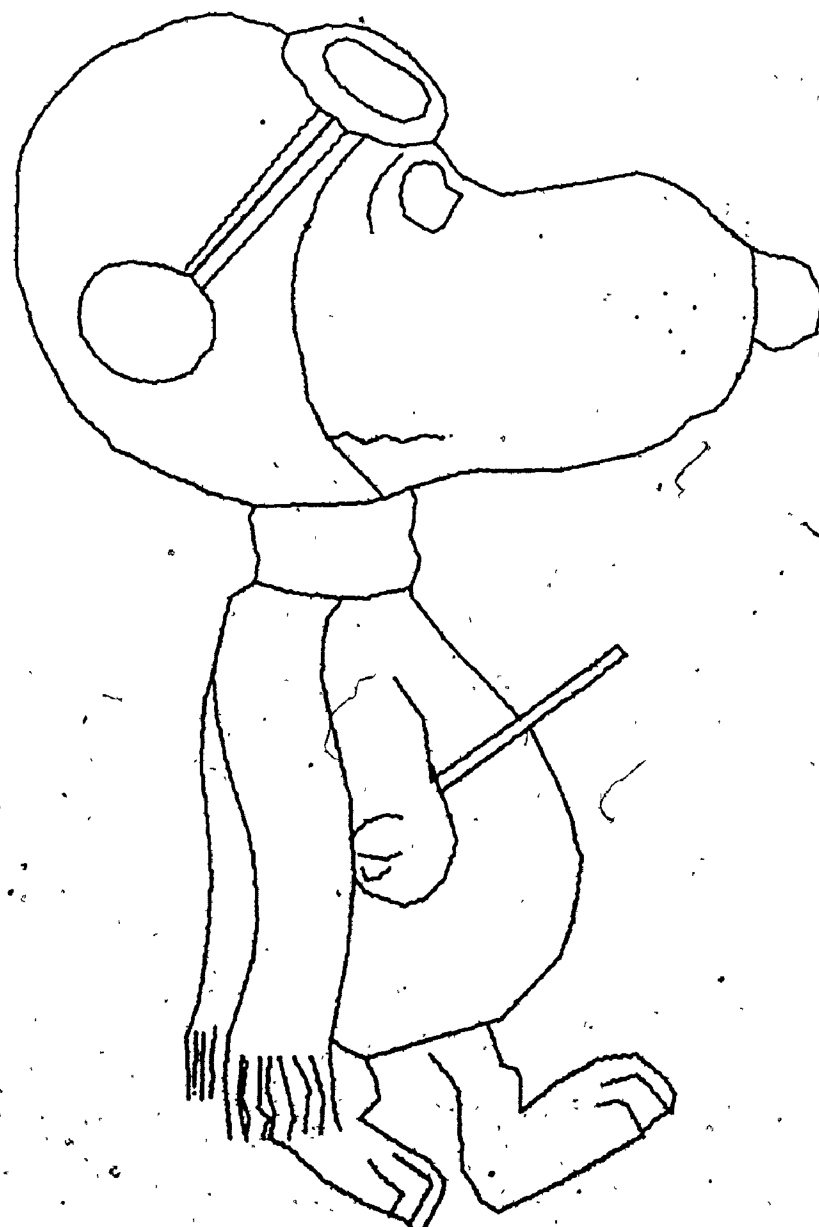


figure 4.2.2. "SNOOPY" on Benson-Lehner

quite meet a second line. (See "SNOOPY'S" Left heel).

In summary, the Versatec "SNOOPY" had a few rough edges, mainly due to the precision, but is quite a reasonable facsimile of the same plot on the Benson-Lehner. This is especially true when you consider that "SNOOPY" only takes about 15 seconds to be plotted.

4.1.2. Plotter: Case Study II (HARMONOGRAPHS)

This example is the plotting of the path of a double conical pendulum, and was taken from an article in Software-Practice and Experience (Volume 2, Pages 293-301 (1972)). In figures 4.3. is the program listing. Twelve parameters, A1, ... , F1, and A2, ... , F2, control the resulting plot. Then for each value of T from 0.0 to 200.0 on X and Y are determined and plotted. In the article a stepsize of 0.05 was suggested. But, this creates too many calls to plot, for even the disk version to handle. This version could handle a stepsize of 0.10, so in all these examples this stepsize is used. In some of the plots the plotter output became a little ragged as a result of this, but these plots proved to be a useful test for the plot package.

Three of these plots are chosen for test cases here. They are chosen to illustrate different problems and also to demonstrate different abilities of the printer/plotter. In figures 4.4.1. through 4.4.6. are the three examples plotted on both the Versatec and the Benson-Lehner.

- In figures 4.4.1. and 4.4.2.
  A1, ..., F1 = 1.00, 2.00, 2.00, -2.00, -2.00, 1.00
  and A2, ..., F2 = 0.30, 0.65, 0.65, 0.65, 0.65, 3.01
- In figures 4.4.3. and 4.4.4.
  A1, ..., F1 = 1.00, 3.00, 0.00, 0.00, 3.00, 0.20
  and A2, ..., F2 = 1.00, 0.00, 0.30, -3.00, 0.00, 5.00
- In figures 4.4.5. and 4.4.6.
  A1, ..., F1 = 0.25, 4.00, 0.00, 0.00, 0.00, 1.00
  and A2, ..., F2 = 2.00, 0.00, 0.00, 4.00, 0.00, 2.00

```
FTN4,L
      PROGRAM HRMNC
      REAL  A1, B1, C1, D1, E1
      REAL  A2, B2, C2, D2, E2
      REAL DELTA, T, TMAX
      INTEGER  PEN
C
      DATA  TMAX  / 200.0 /
C
C
C
C   READ STEP FACTOR
C
      READ(5,1) DELTA
  1   FORMAT(F10.0)
C
C   READ IN 12 PARAMETERS
C
C
      READ(5,2)  A1, B1, C1, D1, E1, F1
      READ(5,2) A2, B2, C2, D2, E2, F2
  2   FORMAT( 6F10.0 )
C
C
C   REPORT STEP SIZE
C
C   AND PARAMETERS
C
C
      WRITE(6,3) DELTA, A1, B1, C1, D1, E1, F1,
     1                  A2, B2, C2, D2, E2, F2
C
  3   FORMAT(//////,1X,19HPLOTTING HARMONICS  ,
     1        //,1X,16HWITH STEPSIZE =  , F9.3,
     2        //,1X,15HAND PARAMETERS  ,
     3        //,1X,20HA1 B1 C1 D1 E1 F1 =  ,
     4        6F9.3,
     5        //,1X,20HA2 B2 C2 D2 E2 F2 =  ,
     6        6F9.3,///)
C
C
C   MOVE PEN TO  ( 4, 3 )
C
      CALL PLOT ( 4.0, 3.0, 3)
C
C
```

Figure 4.3: (Page 1 of 2)

```
TEST PROGRAM  # 2  [ HARMONOGRAPHS ]          PAGE #  2

C    INITIALIZE  T, PEN
C
       T = 0.0
       PEN = 3
C
C
C    BACK UP ONE STEP
C
       T = T - DELTA
C
 20    T = T + DELTA
C
       IF( T.GT.TMAX ) GO TO 40
C
C    CALCULATE ALL OUR FACTORS
C
       R1 = 100.0 * EXP( -A1*0.01*T )
       R2 = 100.0 * EXP( -A2*0.01*T )
       S1 = R1 * SIN( F1*T )
       S2 = R2 * SIN( F2*T )
       T1 = R1 * COS( F1*T )
       T2 = R2 * COS( F2*T )
C
       X = 511.0 + B1*S1 + B2*S2 + C1*T1 + C2*T2
       Y = 511.0 + D1*S1 + D2*S2 + E1*T1 + E2*T2
C
C
       X = X / 140.0
       Y = Y / 140.0
       CALL PLOT ( X, Y, PEN )
C
C
       PEN = 2
       GO TO 20
C
C    SEND IN TERMINATOR PLOT CALL
C
 40    CALL PLOT ( 0.0, 0.0, 999 )
       STOP
       END
       END$
```

Figure 4.3. Test Program "HRMNC"

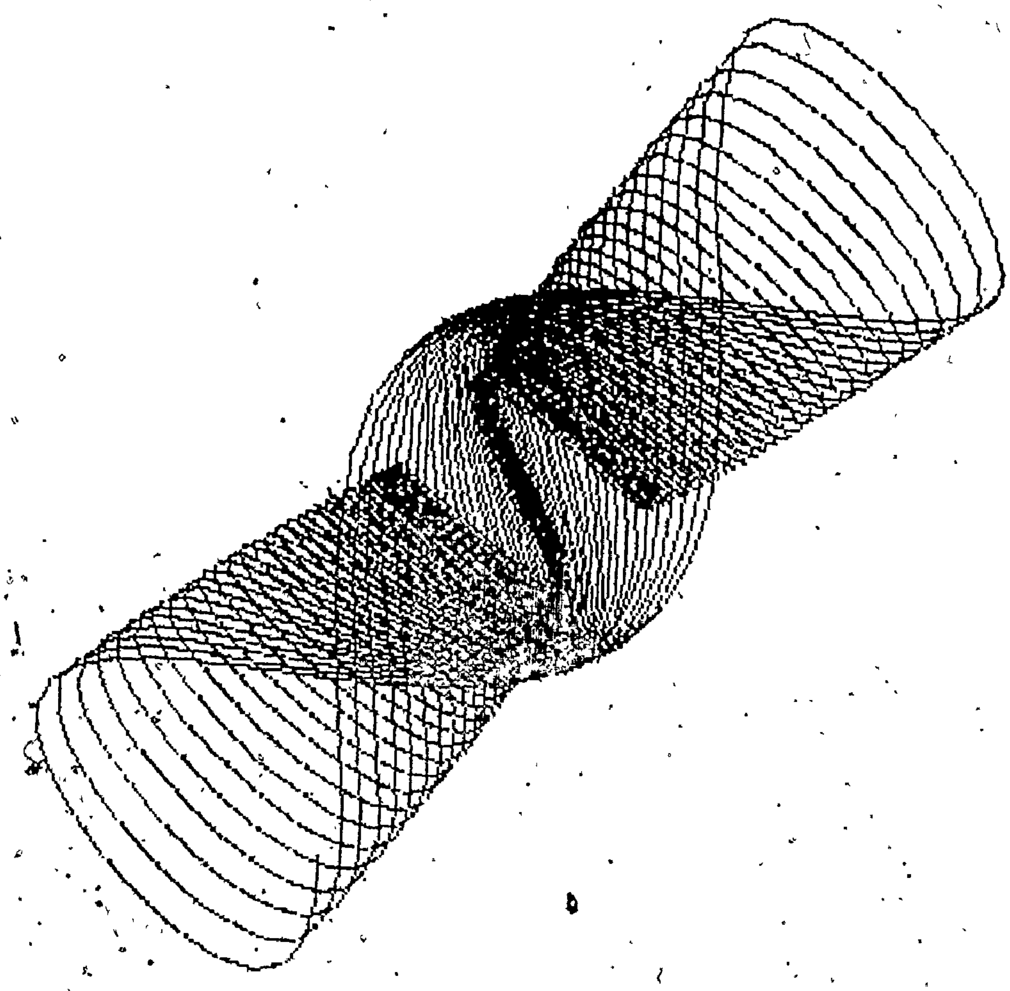


figure 4.4.1. HARMONOGRAPH (1) on EPP

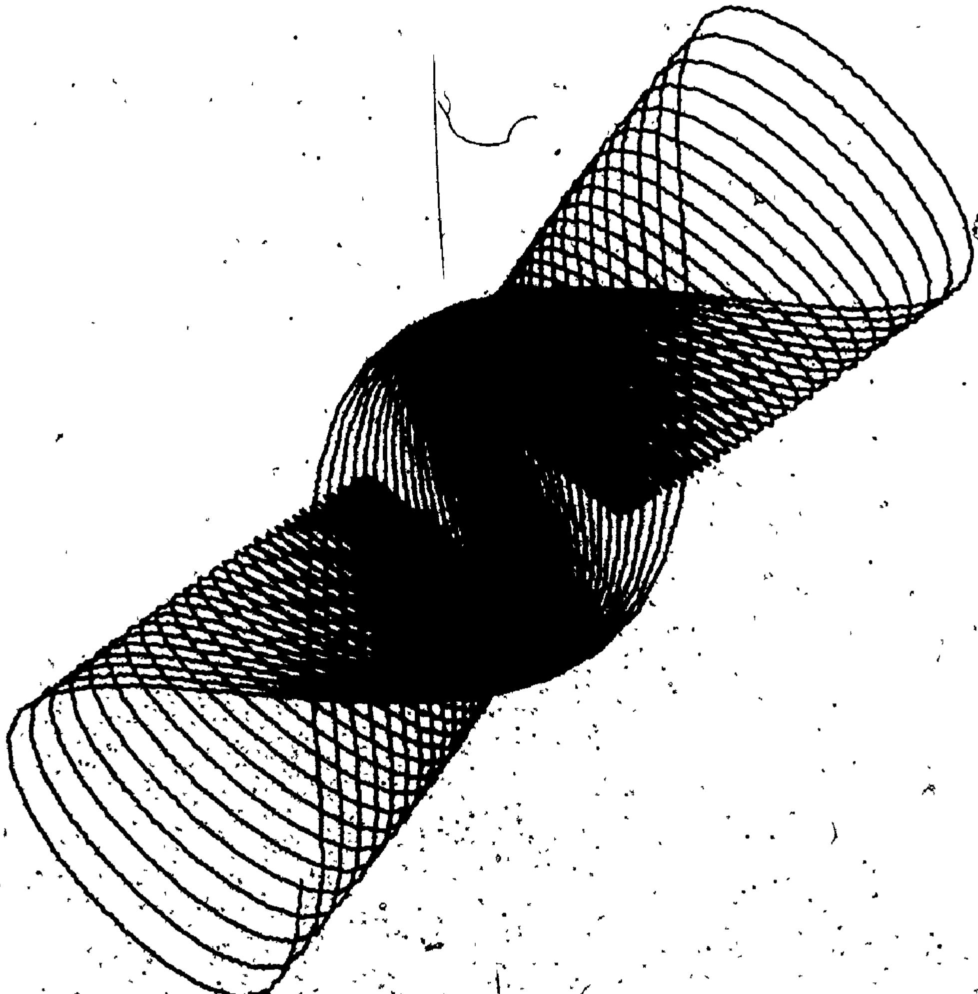


figure 4.4.2. HARMONOGRAPH (1) on Benson-Lehner

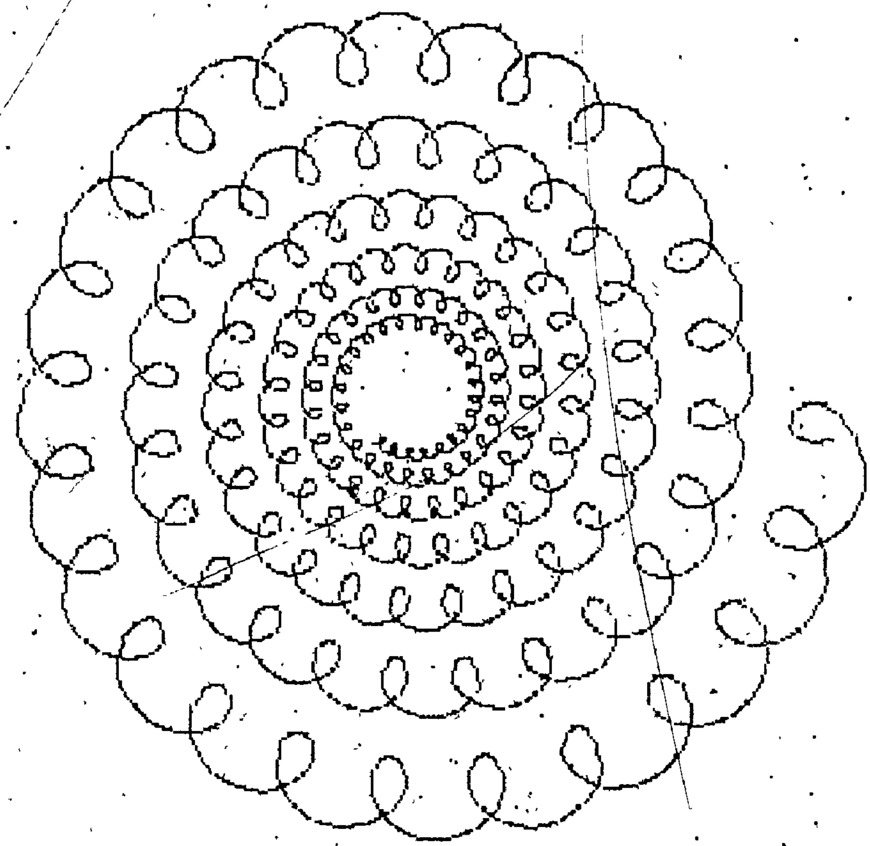


figure 4.4.3. HARMONOGRAPH (2) on EPP

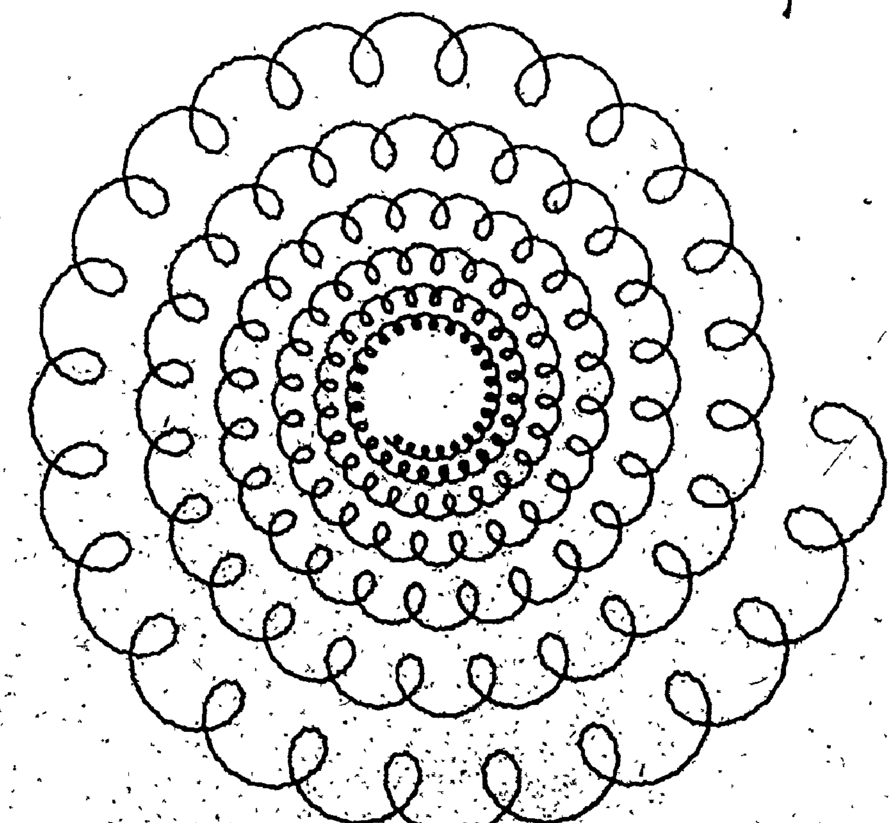


figure 4.4.4. HARMONOGRAPH (2) on Benson-Lehner

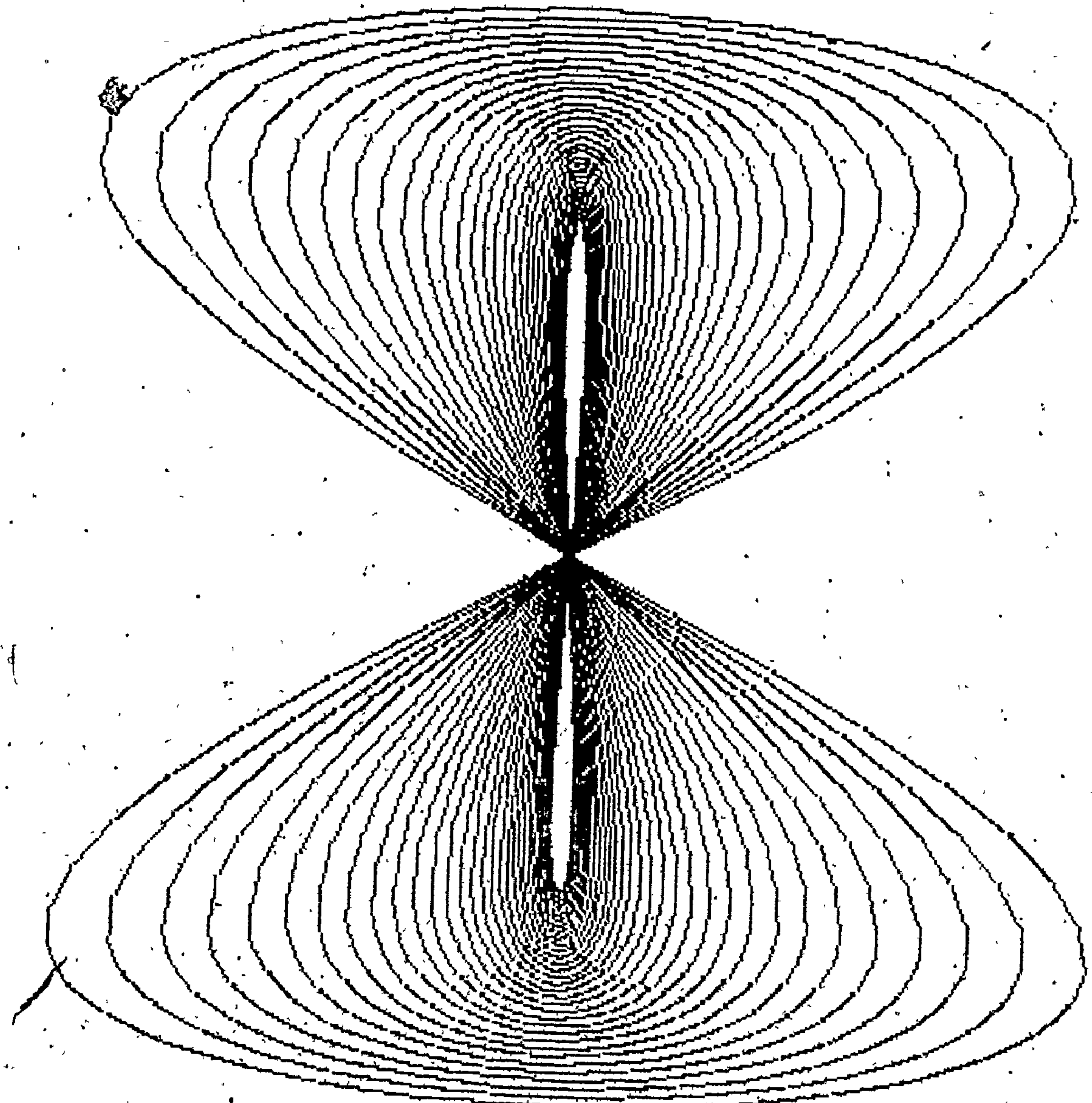


figure 4.4.5. HARMONOGRAPH (3) on EPP

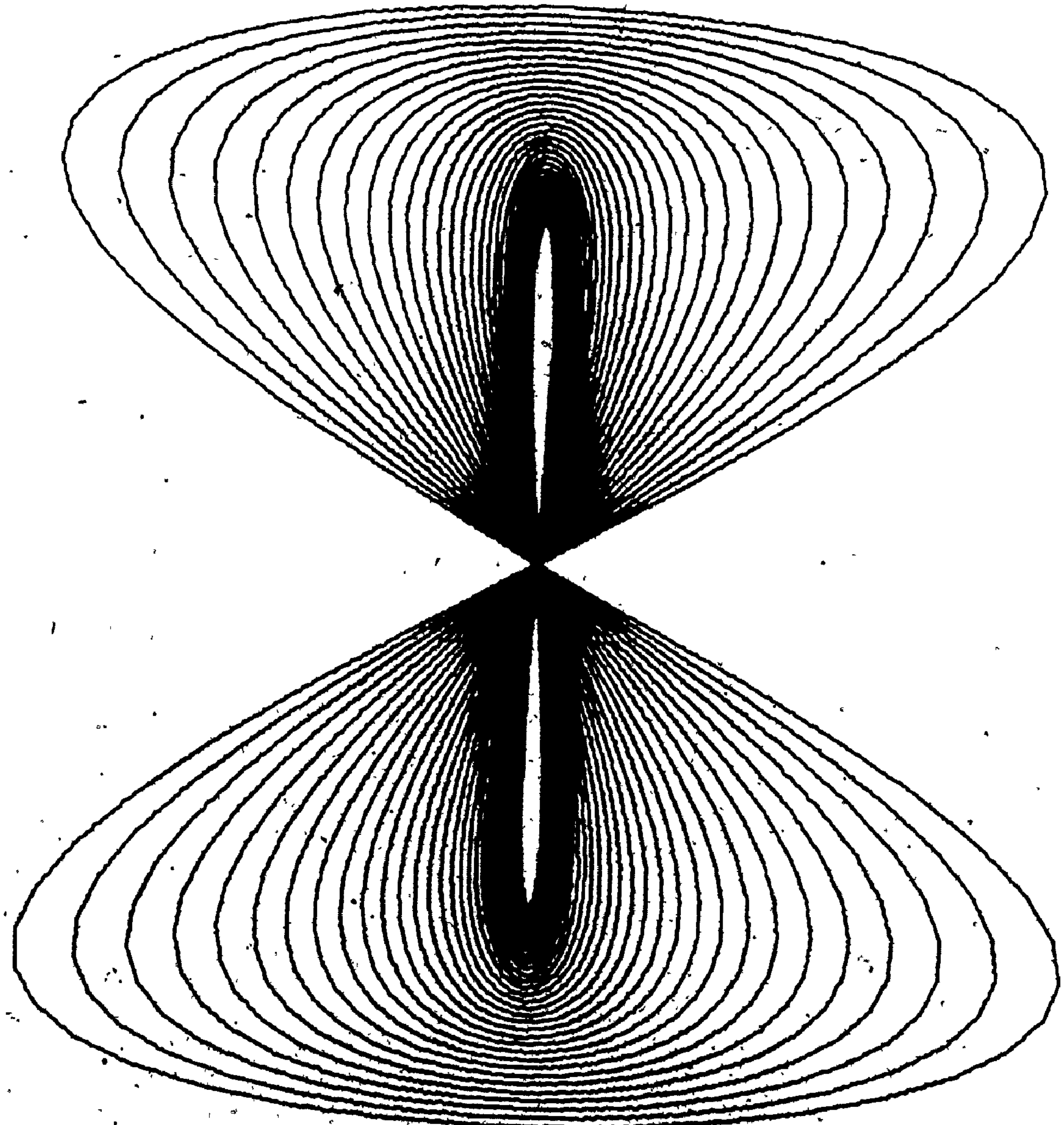


figure 4.4.6. HARMONOGRAPH (3) on Benson-Lehner

Comments on the three pairs of plots.

1) All three of these plots are art forms more than anything else, especially in the first case (4.4.1. and 4.4.2.) where a three-dimensional effect is obtained.

2) As in the "SNOOPY" case, the precision and continuity of the Benson-Lehner plots create a more pleasing result.

3) Of the three plots the second, with the shrinking spiral (4.4.3. and 4.4.4.), is the Versatec plot that compares best with its Benson-Lehner counterpart.

The first plot (4.4.1. and 4.4.2.) was chosen to illustrate the problem of over-plotting. On the Benson-Lehner, when the pen crosses another line or falls on top of a previous line, the resultant plot is darker at this point. This can be easily seen in the centre of the Benson-Lehner plot (4.4.2.). Contrast this with the Versatec plot (4.4.1.) where no overplotting is possible. Since a point can only be plotted once, it can't get any darker. The result is an almost grey plot in the same region. This is primarily because overwriting cannot take place and also, the allowable density (72.5 points per inch) is not high enough to create a black area. (i.e. if the entire page were plotted it would look grey. If a more precise EPP were used the result would be black.)

The second plot was chosen (4.4.3. and 4.4.4.) to consider small movements of the pen. As the pen swings inward the movements become smaller and smaller creating smaller and smaller loops. The Versatec plot is a little rough in spots (again because of precision), but on the whole quite reasonable. The Benson-Lehner does very well. This type of plotter would run into a problem near the centre if a ball-point were used instead of

The third plot was chosen because, in addition to showing some overwriting problems as in the first case, the round off error problem is easily seen as lines of breaks appear to radiate out at several places in the loops. This round off error is due to the word length problem on the HP2100A (only 16 bits).

4.2. Results of the Graph Utility

Two cases were chosen to illustrate what the graph utility can do.

- In the first case a family of ten sine functions is plotted, illustrating how the index can choose one member of a family. (See figure 4.5. and figure 4.6.) This can be done in several ways, depending on the ingenuity of the user. This case also tested the speed of this package, as 10 sine calculations per dot line amount to a considerable amount of work. The speed was greatly affected (taking about 20 seconds for the page instead of about 10 seconds in a single case). These plots are continuous, with no breaks, as no round off error can accumulate, since real arithmetic is used. The lines are still a little rough owing to the precision.

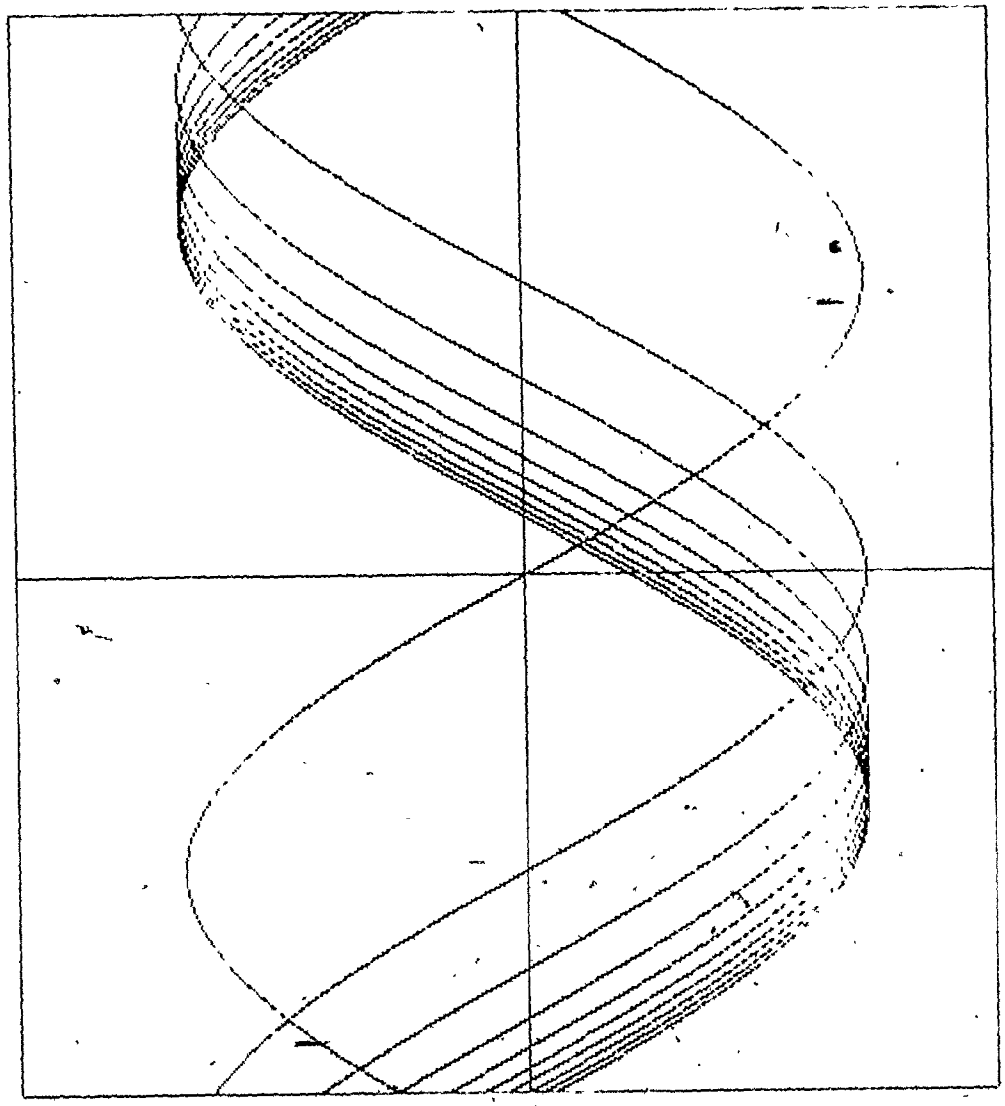
- In the second case, two straight lines, and a circle are graphed. Note that a circle cannot normally be plotted but by treating it as two functions, one positive and one negative function, these problems can sometimes be overcome. (See figure 4.7. and figure 4.8.)
- The outputs in figures 4.6. and 4.8. are reduced from the normal size for purposes of this project.

GRAPH TEST PROGRAM # 1 [CBFN2] PAGE # 1

```
FTN4,L
      PROGRAM CBFN2
      EXTERNAL F
      CALL GRAPH(-2.0,2.75,1.0,-2.0,1.0,10,F)
      STOP
      END
      FUNCTION F(I,X)
      F = 2.0 + SIN(X + 3.1415*FLOAT(I))
      RETURN
      END
      END$
```

figure 4.5. GRAPH Test Program "CBFN2"

XMIN = -3.00



YMIN = -3.00 XMAX = 2.75 YMAX = 2.74

SCALE USED IN X-DIRECTION(DOWN) 1 UNIT = 1.00 INCHES.
IN Y-DIRECTION(ACROSS)(1 UNIT = 1.00 INCHES.

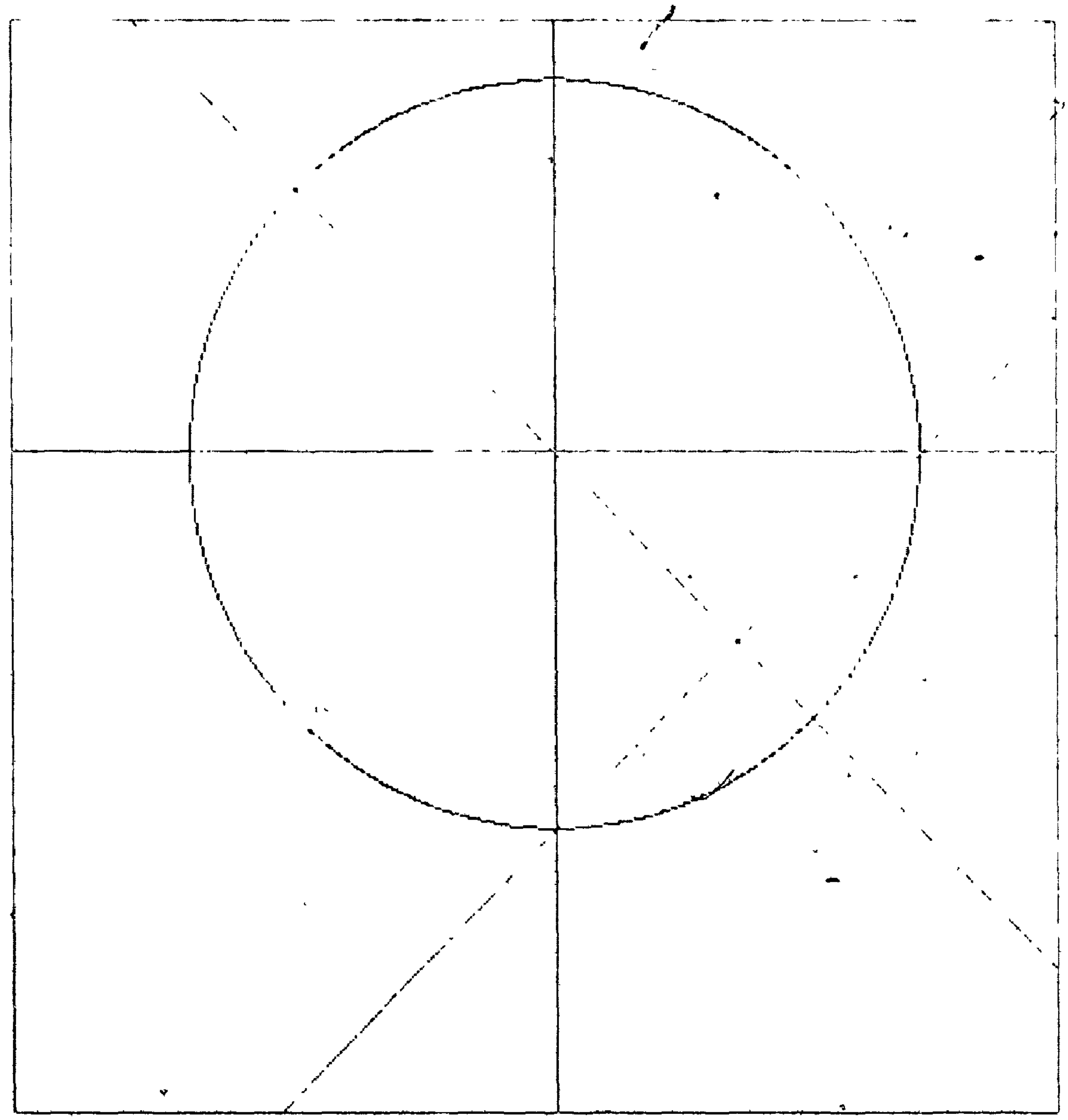
figure 4.6. GRAPH Output from "CBFN2"

GRAPH TEST PROGRAM # 2 [FNSP] PAGE # 1

```
FTN4,L
      PROGRAM FNSP
      EXTERNAL FSP
      CALL GRAPH(-3.3,3.5,1.0,-3.0,1.0,4,FSP)
      STOP
      END
      FUNCTION FSP ( I, X )
      FSP = 0.0
      GO TO ( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ), I
 1    CONTINUE
      Y = 4.0 - X*X
      IF(Y.GT.0.0) FSP = SQRT(Y)
      RETURN
 2    CONTINUE
      Y = 4.0 - X*X
      IF(Y.GT.0.0) FSP = - SQRT(Y)
      RETURN
 3    CONTINUE
      FSP = X
      RETURN
 4    CONTINUE
      FSP = 2.0 - X
      RETURN
 5    CONTINUE
      RETURN
 6    CONTINUE
      RETURN
 7    CONTINUE
      RETURN
 8    CONTINUE
      RETURN
 9    CONTINUE
      RETURN
10    CONTINUE
      RETURN
      END
      END$
```

figure 4.7. GRAPH Test Program "FNSP"

XMIN = -2 30



YMIN = -3.00 XMAX = 3 50 YMAX = 2 74

SCALE USED IN X-DIRECTION(DOWN) 1 UNIT = 1 00 INCHES.
IN Y-DIRECTION(ACROSS) 1 UNIT = 1 00 INCHES

figure 4.8. GRAPH Output from "FNSP"

CHAPTER 5

CONCLUSIONS

5.1. Evaluation of the Plot Package

In Chapter I the following objectives were stated:

> to develop a plot package, starting at routine PLOT, which is capable of plotting on an electrostatic printer/plotter, this PLOT routine to be compatible with the Benson-Lehner plotting system and capable of supporting the parts of the Benson-Lehner plotting system which call PLOT.

I believe that I have achieved those objectives. First of all, in Version I of the plotting system, we have a working, and tested plotting system. This is shown by the examples in Chapter 4 and in Appendix E. These plots were all produced by Version II, but the two versions are logically identical as far as the plotter logic is concerned. (To ensure this is so, when the final version of PLOT (in Version II) was available, it was converted back to Version I by substituting for the disk storage references, by core storage references. So, as far as humanly possible these routines are logically the same). The basic system involved in the plotter logic, (storing strings of line segments in a first pass and plotting these strings in a second pass), makes effective use of the available core, and attempts to maximize plotting speed in the second pass. Two versions came about as a result of the severe restriction that core size placed on the number of calls to plot (about 1000). Thus the disk storage version was created, adding about 1700 potential calls to plot (i.e. Version II can accommodate 2750 calls to plot). In speed, Version I with core storage is superior, as no disk accesses are needed during Pass 1. Pass 2 will be handled at the same speed in both versions. With only 12K available on the HP2100A, Version I is not too useful whereas Version II, though it is still restricted, is a useful plotting aid. With more core available the restriction on Version I would become tolerable and at about 20K (providing about 5000 calls to PLOT) Version I would become superior because of its faster processing speed. At present, a few of the newer features in the Benson-Lehner PLOT routine are not available, but as the plot-library from the Benson-Lehner, is converted to the HP2100A the necessary additions will be made.

In summary, I have implemented and tested a plotting system which can plot on an electrostatic printer/plotter. It is almost completely compatible with the Benson-Lehner plotting system. Also, additional tests on the CDC6400, producing plots on the printer, have shown that routines such as LETTER are compatible with my plotting system. I believe that the basic design of the package is sound, and well structured, so that improvements and modification should involve no major rewriting.

5.2. Evaluation of the Graph Utility

The objectives of this utility were as follows:

- develop a graph utility capable of taking several functions and creating a graph or plot, as simply as possible, over a desired range, and with a desired scale.

As seen in the examples in Chapter 4, this has been done very successfully. I believe that the most appealing feature is its ease of use. The programs can be very simple, and quickly written. (i.e. within a half hour, and without writing complicated Fortran programs, a user could get a desired function plotted, and then re-plotted to graph the function using the most useful scales and limits)

5.3. Future Improvements and Additions

1. The plot package will first need a few simple modifications in order to make it as compatible as possible with the current Benson-Lehner plotting system on the CDC6400.
2. The Benson-Lehner plotter library, will have to be converted to the HP2100A. A few HP library routines are available, and may be feasible alternatives to some of the Benson-Lehner routines.
3. Documentation on the use of the plot package should be prepared and published (locally), for users of the package. Only a page or two will be needed, highlighting the minor differences between plotting on the CDC6400, and plotting on the HP2100A. For the most part the CDC documentation can be used on the HP (saving duplication of a 50 page document).
4. A major improvement, involving major changes, would be to try to utilize the Simultaneous-Print-Plot facility of the Versatec. At present the Plot Package does not allow this.
5. The graph utility could be improved by putting out scale

indicators on the margins and on the axes. Some experimentation would be needed to arrive at the best possible combination.

6. Also a facility to put out headings with a special call to either GRAPH or, maybe, an independent utility, could be added to the graph utility.

APPENDIX A

Instructions for Implementing and Using the Plot Package

1) Implementation (Version II)

- first the implementation at McMaster will be given
- then the likely procedures necessary elsewhere.

a) Plot routines are compiled or assembled and the relocatable binary generated is added to the system library. (to actually add these routines a system generation is needed)

b) During system generation the routine SETHI can either be included in the system as a system utility or left as a user program. This does not make any significant difference. Before running any plot jobs a binary file of suitable length must be created and called SWAP2. Then the routine SETHI is run. This initializes SWAP2 for all subsequent uses of the plotter package. Only if someone purges SWAP2, or writes onto SWAP2, will the file need to be reinitialized.

- to implement this package on another HP with different core size would mean a few small changes in DSKIO and SETHI to adjust for the extra core. If much more core is available Version I of the package with expanded arrays would be much more attractive.
- if a different electrostatic printer/plotter were used with different resolution or with a different type of interface changes in constants and buffer sizes in PRINT

and BFOUT would be needed.

- On different machines an all Fortran form of Version I would be easiest to use. Changes would probably be needed in BFOUT, for I/O on the particular machine.

2) Use of the Plot Package

a) The user writes a program calling any special plotter routines, and possibly calling PLOT directly. The only restriction is that all calls to PLOT be compatible with the Benson-Lehner plotting system, and the reduced width of the printer/plotter.

b) Then the user compiles his program.

c) Next load the program. This loading will automatically load plot routines from library.

d) Execute the program.

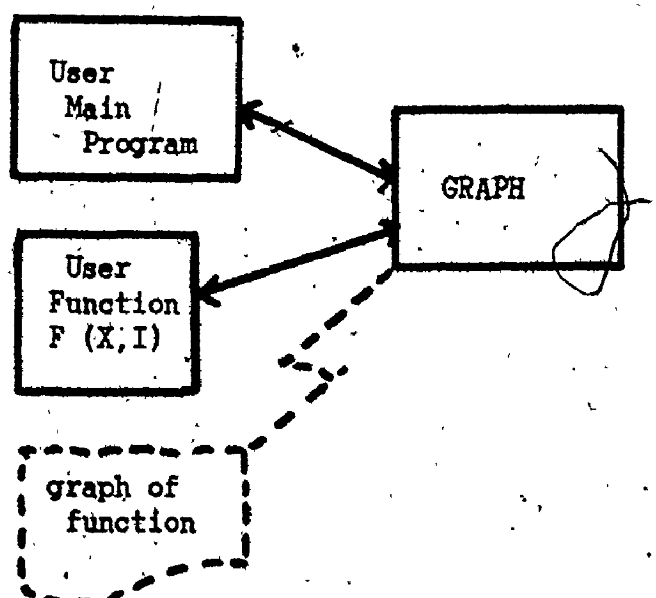
e) CDC documentation on the Benson-Lehner plotting system would be almost completely compatible with the HP plotting system as far as user is concerned.

APPENDIX B

Instructions for Using GRAPH.

- at McMaster the GRAPH routine will be added to the library, along with the plot package. On a different machine this might be handled differently.

a) the organization of the package is shown in the diagram below. The user writes a main program and makes calls to GRAPH. In addition the user writes a Function subprogram which is called from GRAPH, and is an actual parameter of GRAPH. The format of the function is given below.



The function the user supplies is exemplified by:

```
      FUNCTION   F  (X,I)

      GOTO  (1,2,3,4, ..., 10), I
 1     F = SIN (X)
      Return
 2     F = COS (X)
      Return
```

```
3    F = X*X
     Return
     ..
     ..
     ..
     End
```

This allows multiple function definitions, so that several functions can be plotted at once.

The fortran call is exemplified by:

CALL GRAPH (XMIN, XMAX, XSCAL, YMIN, YSCAL, NOFN, FNM)

where XMIN - minimum X value for function (in absolute units)

XMAX - maximum X value (in absolute units)

YSCAL - desired scale in X direction (absolute units 1 inch)

YMIN - minimum Y value (in absolute units)

YSCAL - scale in Y direction (absolute units/inch)

NOFN - number of functions to be plotted

and FNM - is an external name of the function to be called.

b) The program and function are compiled and loaded normally (assuming GRAPH is in the system library). Then the program is executed, producing a graph.

APPENDIX C

Program Algorithms of Plot Package and Graph Utility

All algorithms are written in pseudo-PASCAL. This is done to take advantage of the structured programs, and data structures available in PASCAL. As much as possible legal PASCAL is used. However, some of the logic in the HP ASSEMBLER programs are beyond the ability of legal PASCAL to express. So in these cases the author took liberties with the language.

The algorithms appear in the following order:

```
PROCEDURE PLOT( VAR XR,YR : REAL;
                        PEN : INT16 ) ;

     "  IN THIS ROUTINE THERE ARE 2 MAIN
        DATA STRUCTURES

        LIST     -- WHICH IS A LINKED LIST OF
                    STRINGTYPE ( STRING )
                    THIS VALUE IS PASSED TO
                    "PRINT" & POINTS TO THE
                    1ST STRING IN THE LIST

        STRING   -- WHICH IS ALSO A LINKED LIST
                    OF COORDINATES ( "COORD" )
                    OR LINE-SEGMENTS.  IN THIS
                    ROUTINE "STRING" POINTS TO THE
                    CURRENT STRING IN "LIST"
                                                  "

LABEL  92,120,141;

TYPE
      INT16 = 0..177777B;

      COORD = RECORD
                NXTPOINT : ^COORD;
                X,Y      : INT16;
              END;

      STRINGTYPE = RECORD
                     NXTSTRING : ^STRINGTYPE;
                     FIRSTPOINT : ^COORD;
                   END;

      STRINGSTATE = ( NEW,UP,DOWN );

VAR
   STATEOFSTRING : STRINGSTATE;

   X,Y,XOLD,YOLD : INT16;

   FIRSTTIME     : BOOLEAN;

   STRING, LIST  : ^STRINGTYPE;
```

```
BEGIN " OF PLOT "

  IF FIRSTTIME
  THEN

   BEGIN
    INITIALISATION,
    IF PEN = -20
    THEN
     BEGIN
      CALLSINCENTIMETRES;
      GOTO 92,
     END;
   END;

  CONVERTTOPLOTTERUNITS(XR, YR, X, Y);

  IF LEGAL(PEN)
  THEN

 120 :

   CASE PEN OF

   -3, -2, -1 : BEGIN
                 RESETORIGIN;
                 PEN := -PEN;
                 GOTO 120;
                END;

            0 : SETCURRENTPOSITION(XR, YR);

            1 : PLOTWITHPENUNCHANGED: "UP/DOWN"

            2 : " PLOT WITH PEN DOWN "

                CASE STATEOFSTRING OF

                  NEW : BEGIN
```

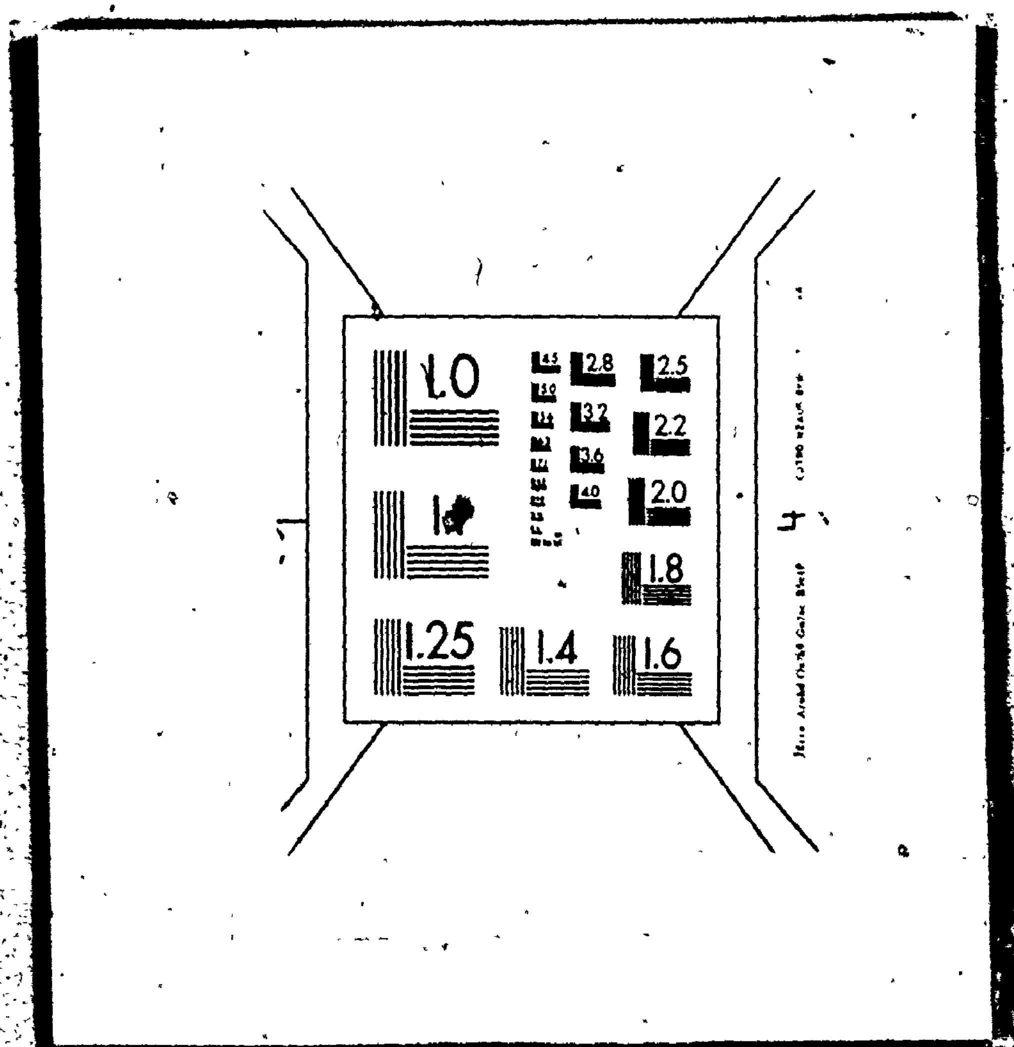
```
               ENDOLDSTRING;
               CASE (XOLD X) OF
                TRUE   STARTNEWSTRING(DOWN);
               FALSE   STARTNEWSTRING(UP);
               END;
              END;

        DOWN  BEGIN
               IF X>XOLD THEN GOTO 141;
               ADDLINESEGMENT(X,Y);
              END;

          UP  BEGIN
               IF XOLD<X THEN
                      BEGIN
                       INVRT(STRING);
                       GOTO 141;
                      END;
               ADDLINESEGMENT(X,Y);
              END;

        END;  " CASE  STATEOFSTRING "

    3 : " PLOT WITH PEN UP "
        BEGIN
         IF STATEOFSTRING = UP
          THEN INVRT(STRING);
         STATEOFSTRING := NEW;
        END;

   40 : SETNEWTRANSLATIONS;

  999 : " END OF THIS PLOT
         PUT PLOT OUT TO
         PRINTER/PLOTTER   "
        BEGIN
         IF STATEOFSTRING = UP
          THEN INVRT(STRING);

         PRINT(LIST);
         FIRSTTIME := TRUE;
        END;

END  " CASE  PEN "
```

```
ELSE   " PEN NOT LEGAL "
  [illegible]

  [illegible]

END   " OF PLOT "
```

```
PROCEDURE INVRT(VAR FIRSTPOINT : ↑COORD);

VAR
   NEXTPT, LASTPT : ↑COORD;

BEGIN   " SUBROUTINE INVRT "

  LASTPT := NIL; " END-OF-STRING "

  " GET POINTER TO NEXT POINT IN STRING
    THEN POINT THIS CURRENT POINT TO
    PREVIOUS POINT

    NOW SET LAST POINT TO CURRENT POINT
    AND CURRENT POINT TO NEXT POINT

    KEEP GOING TILL END-OF-STRING    "

  REPEAT

   NEXTPT := FIRSTPOINT↑.NXTPOINT;
   FIRSTPOINT↑.NXTPOINT := LASTPT;
   LASTPT := FIRSTPOINT;
   FIRSTPOINT := NEXTPT;

  UNTIL
   FIRSTPOINT = NIL;

END; " OF INVRT "
```

```
PROCEDURE PRINT ( VAR LIST : ↑STRINGTYPE );

     " IN THIS ROUTINE THERE ARE 2 MAIN
       DATA STRUCTURES

       LIST    -- WHICH IS A LINKED LIST OF
                  STRINGTYPE ( STRING )
                  THE FORMAL ARGUMENT PASSED
                  TO THIS ROUTINE POINTS TO
                  THE 1ST STRING IN THE LIST

       STRING  -- WHICH IS ALSO A LINKED LIST
                  OF COORDINATES ( "COORD" )
                  OF LINE-SEGMENTS   IN THIS
                  ROUTINE "STRING" ACTUALLY
                  POINTS TO THE CURRENT STRING
                  IN THE LIST                "

LABEL  35;

TYPE   INT16 = 0..177777B;

       COORD = RECORD
                 NXTPOINT   : ↑COORD;
                 X,Y        : INT16;
               END;

    STRINGTYPE = RECORD
                  NXTSTRING  : ↑STRINGTYPE;
                  FIRSTPOINT : ↑COORD;
                 END;

VAR
       X,X1,X2,Y1,Y2,YMIN,YMAX,YLAST,YNEXT

                  : INT16;

      LINE  : ARRAY [1..35] OF INT16;

      STRING : ↑STRINGTYPE;

BEGIN  " OF PRINT "

       " IN THIS ROUTINE (X1,Y1) AND (X2,Y2)
         ARE USED TO SIMPLIFY THE ALGORITHM
         THEY ARE USED AS ABREVIATIONS FOR
```

2



```
        X1 - FIRSTPOINT↑ X
        Y1 - FIRSTPOINT↑ Y

        X2 - FIRSTPOINT↑.NXTPOINT↑ X
        Y2 - FIRSTPOINT↑ NXTPOINT↑ Y

ORDER(LIST);
PAGESKIP;
INITIALISATION;

WHILE SOME(LIST) DO  " IE - LIST NOT NIL "

BEGIN

 NEXT(X);   " X CURRENT LINE POSITION "
 FIRST(LIST);   " GET POINTER TO 1ST STRING "


 WHILE XIN(STRING) DO   " X >,= X1 "
 BEGIN

  INITIALIZE(YMIN,YMAX), "RANGE INDICATORS IN LINE[ ]"

35 :
  IF X < X2   " X2=END-OF-1ST LINE-SEGMENT "
  THEN

   BEGIN
    INTERPOLATE(YLAST,YNEXT);
    IF X = X1 THEN YLAST := Y1;
    MINMAX(YMIN,YMAX,YLAST);
    MINMAX(YMIN,YMAX,YNEXT);
   END

  ELSE

   BEGIN
    IF X = X2
    THEN
     BEGIN
      INTERPOLATE(YLAST);
      IF X = X1 THEN YLAST := Y1;
      MINMAX(YMIN,YMAX,YLAST);
```

```
      MINMAX(YMIN,YMAX,Y2);
      END; " X=X2 "

     SKIPLINESEGMENT;

     IF DONE(STRING)  " FIRSTPOINT↑ NXTPOINT=NIL "
     THEN
      DELETE(STRING)
     ELSE
      GOTO 35;

    END;  " IF X < X2 ".

   INSERTBITS(YMIN,YMAX);   " INTO LINE "
   NEXT(LIST);

  END;  "WHILE XIN(STRING)"

  BFOUT(LINE);

 END;  "WHILE SOME(LIST)"

 PAGESKIP;

END;  " OF PRINT "
```

```
PROCEDURE INSRT( VAR LINE : ARRAY [1..35] OF INT16,
                     POINT : 1..560; );

TYPE  INT16 = 0 ,177777B;  " 16 BIT INTEGERS "

VAR
  I,J : INTEGER;

BEGIN  " SUBROUTINE INSRT "

  I  = (POINT+15)   16;
  J  = POINT - 16(I-1);

  LINE[I]  = INCLUSIVEOR(LINE[I],BIT(J));

END.  " OF SUBROUTINE INSRT "
```

```
PROCEDURE BFOUT(VAR LINE : ARRAY [1..35] OF INT16; );

TYPE   INT16 = 0..177777B;   " 16 BIT INTEGERS "

VAR
  BUFF : ARRAY [1..35] OF INT16;
     J : INTEGER;

BEGIN  " SUBROUTINE BFOUT "

  BUFF  = LINE;   " COPY ARRAY LINE[] "

  CLEAR(LINE);    " LINE[] := 0 ";

  WRITEBINARYWITHOUTWAIT(BUFF);

END;  " OF SUBROUTINE BFOUT ";
```

```
        " GLOBAL TYPES & VARIABLES "

TYPE
        INT16 = 0..177777B;

VAR
        BUFF : ARRAY [ 1..128 ] OF INT16;

PROCEDURE PUT(VAR VALUE, INDEX : INT16;);

BEGIN " PUT "

  IF  TOOSMALL(INDEX)
  THEN

    REPORTERROR  " RN GE "

  ELSE

   BEGIN

    IF  TOOLARGE(INDEX)
    THEN
     BEGIN,
      ENDOLDSTRING;   " INVRT & CLEAR "
      STARTNEWSTRING; " AT OLD ( X, Y ) "
     END;  " IF TOOLARGE "

    ADD(VALUE,BUFF,INDEX);

  END;  " IF TOOSMALL "

END;  " OF PUT "
```

```
FUNCTION LIST(VAR INDEX : INT16;) : INT16;

BEGIN " LIST "

  IF TOOSMALL(INDEX) OR TOOLARGE(INDEX)
  THEN

    REPORTERROR " RN GE "

  ELSE

     SET(LIST,BUFF,INDEX);

END; " OF LIST "


PROCEDURE CLEAR;

BEGIN " CLEAR "

  TRANSFERSTRING(BUFF,SCTR);

  "  WRITE 'SCTR' TO DISC AS IT IS
     FILLED                         "

END; " OF CLEAR "


PROCEDURE CLSEF;

BEGIN " CLSEF "

  CLEAR; " CURRENT STRING "
  EMPTYLASTSECTOR;
  REINITIALIZEPOINTERS;

END; " OF CLSEF "
```

```
PROCEDURE SWAP1(VAR  TAGS,
                     ITAG,
                     IMAX : INT16);

VAR
   WORKFILE,
   SWAP2     : FILE OF INT16;

   TRK       : 0..128;

BEGIN " SWAP1 "

  GETFREETRACKINWORKAREA(TRK);

  WRITE(WORKFILE,HICORE,TRK,0);

  READ(SWAP2,HICORE);

  SWAP2;

  READ(WORKFILE,HICORE,TRK,0);

  INITIALIZE;

END; " OF SWAP2 "
```

```
PROGRAM SETHI(SWAP2).

CONST  CORESIZE = 37600B.
       PARTITIONPOINT = 24000B.

VAR
   BUFFERSIZE : 0..CORESIZE.
   SWAP2 : FILE OF INT16.

BEGIN  "SETHI"

  BUFFERSIZE := CORESIZE - PARTITIONPOINT;

  SAVE(BASEPAGELINKS,CORESIZE-200B,100B);

  WRITE(SWAP2,PARTITIONPOINT,BUFFERSIZE);

END.   "SETHI"
```

```
PROCEDURE SWAP2(VAR TRK,TRACK : 0..199;
                TAGS : ARRAY[1..500] OF INT16;
                ITAG,IMAX : INT16 );

CONST  FIRSTWORDAVAILABLEMEMORY = 10000B;

TYPE
   INT16 = 0..177777B;

VAR
   WORKFILE : FILE OF INT16;

BEGIN  "SWAP2"

  GETSPECIALARGUMENTS;

  SAVE(BASEPAGELINKS,CORESIZE-100B,100B);

  RESET(BASEPAGELINKS,CORESIZE-200B,100B);

  "SAVE CURRENT LO-CORE ( 10000B - 23777B )"

  WRITE(WORKFILE,LOCORE);

  "MOVE TAGS ARRAY DOWN TO 10000B"

  SHIFT(TAGS,FIRSTWORDAVAILABLEMEMORY);

  "READ IN 'LIST()' FROM DISK TO FOLLOW DIRECTLY
   AFTER 'TAGS()' "

  READ(WORKFILE,FIRSTWORDAVAILABLEMEMORY+ITAG);

  " NOW CALL 'PRINT' AS PER NORMAL "

  PRINT(LIST,TAGS,ITAG);
```

```
"PUT LO-CORE BACK INTO CORE JUST AS IT WAS";
READ(WORKFILE, LOCORE);

[illegible]

END "SW[illegible]";
```

```
ALGORITHM  GRAPH   --  MARCH  1, 1975          PAGE #  1


PROCEDURE GRAPH( VAR XMIN, XMAX, XSCAL,
                     YMIN,       YSCAL    REAL,
                     NUM                  INTEGER, ).

TYPE  INT16 = 0  1[illegible].

VAR
   LINE    ARRAY [1  35] OF INT16,

   YLAST, YPRES, YNEXT, YMIN, YMAX, XAXIS
        INT16,


   DELTX, DELTY, YMAX, X,
        REAL;


BEGIN   " OF GRAPH "

  CALCULATESTEPSIZE(DELTX, DELTY);
  CALCULATE(XAXIS);
  PAGESKIP;

  REPORT(XMIN),
  TOPORBOTTOMBORDER,


  FIRST(X);

  WHILE X < XMAX DO

  BEGIN

   IF X = 0

   THEN  " SPECIAL - YAXIS "

    PLOT(YAXIS)

   ELSE
```

```
   BEGIN " NORMAL CASE "

    FOR EACH FUNCTION DO
     BEGIN
      INTERPOLATE(X,YLAST,YPRES,YNEXT);
      GETRANGE(YMIN,YMAX,YLAST,YPRES,YNEXT);
      INSRT(YMIN,YMAX);
     END; " FOR EACH FUNCTION "

    ADDBORDERSANDXAXIS;
    BFOUT(LINE);

   END; " NORMAL CASE "

  END; " WHILE X < XMAX "

  TOPORBOTTOMBORDER;
  CALCULATE(YMAX);
  REPORT(YMIN,XMAX,YMAX,XSCAL,YSCAL);

  PAGESKIP;

END. " OF GRAPH "
```

APPENDIX D

Program Listings of Plot Package

and Graph Utility

The program listings appear in the following order:

```
FTN4,L,T
      SUBROUTINE PLOT (XR,YR,PEN)
C
C   SUBROUTINE "PLOT"
C
C
C   WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975
C
C   BY C. A  BRYCE, FOR A MSC  PROJECT
C
C
C   "PLOT" IS THE BASE ROUTINE WRITTEN FOR
C              THIS "PLOT" PACKAGE.  THIS
C              ROUTINE IS DESIGNED TO BE A
C              REPLACEMENT FOR THE NORMAL
C              PLOT ROUTINE IN A PLOTTING
C              SYSTEM.  IN PARTICULAR THIS
C              PLOT ROUTINE WAS WRITTEN TO BE
C              COMPLETELY COMPATIBLE WITH THE
C              "BENSON-LEHNER PLOTTING SYSTEM"
C              AS IMPLEMENTED ON THE CDC6400
C              AT MCMASTER UNIVERSITY.  ALL THE
C              ROUTINES THAT NORMALLY CALL
C              "PLOT" CAN SUCCESSFULLY CALL
C              THIS "PLOT" ROUTINE.
C              THIS ROUTINE TAKES THE CALLS
C              TO PLOT AS THEY COME IN, AND
C              REORGANIZES AND RECODES THESE
C              CALLS SO THAT EVENTUALLY THE
C              ROUTINE "PRINT" CAN DECODE THEM
C              AND PUT OUT THE DESIRED PLOT.
C
C
C   THERE ARE 2 VERSIONS OF "PLOT".  ONE
C   VERSION USES ARRAY STORAGE, WHICH LIMITS
C   THE POTENTIAL SIZE OF PLOT THAT "PLOT"
C   CAN HANDLE, AND THE OTHER USES DISK
C   STORAGE, WHICH INCREASES THE POTENTIAL
C   SIZE BY A FACTOR OF ABOUT 3
C   THIS INCREASE FACTOR IS ONLY DUE TO
C   THE LIMITED CORE SIZE OF 12K ( DECIMAL ),
C   AND WITH A LARGER CORE OF 24K, IT WOULD
C   BE MUCH PREFERABLE TO USE THE ARRAY VERSION.
C
C
C   VERSION  I  OF "PLOT"
```

```
C   ( ARRAY STORAGE )
C
C   THIS WAS THE 1ST VERSION CREATED AND
C   BASICALLY THE STRINGS CREATED IN PLOT
C   ARE STORED IN AN INTEGER ARRAY "LIST( )"
C   THIS WAS BY FAR THE PREFERABLE AND
C   SIMPLEST WAY TO HANDLE THINGS. HOWEVER,
C   "LIST( )" COULD ONLY BE 2000 WORDS, AND
C   THIS ALLOWS JUST UNDER 1000 CALLS TO PLOT
C
C   VERSION  I  OF "PLOT" USES 2 SUBROUTINES
C
C   1) INVRT - THIS ROUTINE ALLOWS THE SIMPLE
C              INVERSION OF A STRING.  THIS IS
C              USED AS A SUBROUTINE AS IT IS
C              NEEDED IN 3 PLACES
C
C   2) PRINT - THIS ROUTINE IS CALLED ON
C              COMPLETION OF A PLOT ( OR
C              UPON OVERFLOW OF AVAILABLE
C              STORAGE ).  "PRINT" DOES THE
C              PHYSICAL PLOTTING ONTO THE
C              PRINTER/PLOTTER
C
C
C   VERSION II  OF "PLOT"
C   ( DISK STORAGE )
C
C   BY USING DISK STORAGE FOR STRINGS
C   CREATED IN PLOT, THE EFFECTIVE SIZE WAS
C   EXTENDED BY ABOUT 3500 WORDS. ( OR
C   ABOUT 1700 CALLS TO PLOT ).
C
C   VERSION  II  OF "PLOT" USES 6 SUBROUTINES
C
C   1) IVRTA - THIS IS AN ASSEMBLER ROUTINE
C              THAT FORMS A CALL TO "INVRT"
C              USING ARGUMENTS LOCAL TO THE
C              DISK I/O ROUTINE
C
C   2) PUT   - THIS EFFECTIVELY ACCOMPLISHES AN
C              ASSIGNMENT:
C              [ LIST ( I )  = ...... ]
C
C   3) LIST  - THIS FUNCTION ACCOMPLISHES AN
C              ARRAY ACCESS WITHOUT ANY NEEDED
```

```
C                CHANGE IN THE CODING EXCEPT TO
C                UNDIMENSION "LIST".
C                EQUIVALENT TO
C                [            =  LIST ( I )        ]
C
C   4) CLEAR  - THIS ROUTINE EMPTIES A COMPLETED
C                STRING FROM THE RECORD BUFFER IN
C                THE DISK I.O PACKAGE
C
C   5) CLSEF  - THIS "CLEAR"'S LAST STRING OUT,
C                AND PUTS LAST BUFFER OUT TO THE
C                DISK
C
C   6) SWAP1  - THIS ACCOMPLISHES THE 1ST PART
C                OF THE CORE SWAPPING PROCESS.
C                AND EVENTUALLY LINKS TO "PRINT"
C
C
C   IN BOTH VERSIONS THE EFFECTIVE LOGIC IS
C   THE SAME, WITH THE EXECPTION OF REFERENCES
C   TO ITEMS IN "LIST()", AND OF THE NEEDED
C   CLEARING & CLOSING OPERATIONS IN VERSION II.
C
C
C   METHOD
C
C   IN THE 1ST CALL TO "PLOT" INITIALISATION IS
C   DONE TO FLAGS, ORIGIN, TRANSPOSITION, ETC.,
C   AND CHANGING "FACTOR" TO CENTIMETRES IS
C   ALLOWED.
C   THEN (XR,YR) IS CONVERTED FROM REAL TO
C   INTEGER PLOTTER UNITS (X,Y).  THEN ORIGIN,
C   AND TRANSLATION CHANGES ARE MADE TO GET
C   AN (X,Y) IN ABSOLUTE PLOTTER UNITS ( FROM (0,0) )
C   NEXT "PEN" IS DECODED AND NECESSARY CODE IS
C   EXECUTED IN ORDER TO ---
C       - ADD TO CURRENT STRING
C       - START A NEW STRING,
C       - RESET ORIGIN OR TRANSLATION
C       - RETURN CURRENT CO-ORDINATES
C       - OR TERMINAL CALL TO "PLOT"
C         INITIATING A CALL TO "PRINT".
C
C
C   KEY VARIABLES :
C
```

```
C    XR      - REAL CALLING VALUE OF X CO-ORDINATE
C
C    YR      - REAL CORRESPONDING VALUE OF Y
C
C    PEN     - INTEGER CALLING VALUE OF "PEN"
C              "PEN" IS CODED TO TELL "PLOT"
C              WHAT TO DO WITH (XR,YR)
C
C
      REAL      XR, YR
      INTEGER   PEN
C
C
C    LOCAL VARIABLES
C
C    X,Y     - INTEGER WORKING VALUE OF CO-ORDINATES
C              CORRESPONDING TO (XR,YR)
C
C    XOLD, YOLD
C            - INTEGER CO-ORDINATES OF CURRENT
C              POSITION ( OR CO-ORDINATES FROM
C                        PREVIOUS CALL TO "PLOT" )
C
C    XORG, YORG
C            - INTEGER CO-ORDINATES OF CURRENT
C              ORIGIN ( RELATIVE TO (0,0) )
C
C    XTR, YTR
C            - INTEGER CO-ORDINATES OF CURRENT
C              TRANSLATION ( TO BE TAKEN AWAY
C                            FROM (X,Y)       )
C
C    FLAG    - INTEGER FLAG TO INDICATE A STRING
C              GOING DOWN ( FLAG=0 )
C              GOING UP   ( FLAG=1 )
C              OR NEITHER ( FLAG=-1)
C
C    PENP    - INTEGER FLAG INDICATING CURRENT
C              PEN POSITION.
C              ( =2, PEN DOWN / =3, PEN UP )
C
C    FACTOR  - REAL CONVERSION FACTOR USED TO
C              CONVERT (XR,YR) TO PLOTTER UNITS.
C              ( BY DEFAULT SET FOR CALLS IN
C                INCHES, BUT CAN BE SET TO
C                CENTIMETRES ON 1ST CALL (ONLY) )
```

```
C
C   LIST() - INTEGER ARRAY (VERSION I) TO HOLD
C            STRINGS  ( IN VERSION II - A FUNCTION )
C
C   I, IX  - INDICES USED WITH "LIST()"
C
C   TAGS() - INTEGER ARRAY HOLDING POINTERS
C            (INDICES) TO HEAD OF STRINGS IN
C            "LIST()"
C
C   ITAG   - CURRENT INDEX IN "TAGS()"
C
C   MAXL, MAXT
C          - MAXIMUM SIZE FOR "LIST()" & "TAGS()"
C            RESPECTIVELY
C            ( IN VERSION  I -- (2000,500)
C            & IN VERSION II -- (5500,500) )
C
C   KOUNT  - INTEGER FLAG TO INDICATE 1ST CALL
C            TO "PLOT" (=0) OR NOT (=1).
C
C   IW     - LOGICAL UNIT OF PRINTER ( 8 )
C
C
      INTEGER  X, Y, XOLD, YOLD, XORG, YORG
      INTEGER  XTR, YTR
      INTEGER  FLAG, PENP, KOUNT, IW
      INTEGER  LIST, I, IX, TAGS, ITAG
      INTEGER  MAXL, MAXT
      REAL     FACTOR
      DIMENSION  TAGS ( 500 )
C
C     DIMENSION  LIST ( 2000 )  -- VERSION I
C
C
C   INITIALIZE DATA
C
      DATA   KOUNT  /     0  /
      DATA   IW     /     8  /
      DATA   MAXL   /  5500  /
C$    DATA   MAXL   /  2000  /  -- VERSION I
      DATA   MAXT   /   500  /
C
C
C*******************************************
C
```

```
C   START OF SUBROUTINE "PLOT"
C
C   ON THE 1ST CALL TO "PLOT" (KOUNT=0)
C
C    INITIALIZE
C      LAST (X,Y) -- (XOLD,YOLD)
C      ORIGIN     -- (XORG,YORG)
C      TRANSLATION-- (XTR,YTR)
C      PEN POSITION- PENP
C      UP/DOWN FLAG- FLAG
C      INDEX IN "LIST()" & "LIST(1)
C      INDEX IN "TAGS()" - ITAG
C      SCALE FACTOR - FACTOR
C  &   SAY NOT 1ST CALL TO "PLOT" (KOUNT=1)
C
C
       IF(KOUNT)  92, 90, 100
 90     KOUNT = 1
        XOLD = 0
        YOLD = 0
        XORG = 0
        YORG = 0
        XTR = 0
        YTR = 0
        PENP = 3
        I = 1
        CALL PUT( 1,1,I,FLAG,TAGS,ITAG,MAXT )
        ITAG = 0
        FLAG = - 1
        FACTOR = 72.5
C
C
C   ON 1ST CALL IF PEN = -20 SET
C   FACTOR TO CENTIMETRES INSTEAD
C   OF INCHES AND RETURN
C
C
        IF(PEN+20)  2000, 91, 100
 91      FACTOR = 0.3937 * FACTOR
 92      RETURN
C
C
C*************************************
C
C   AFTER 1ST CALL THIS IS THE EFFECTIVE
C   STARTING POINT IN "PLOT".
```

```
C
C   - CONVERT (XR,YR) FROM REAL IN INCHES (CM )
C       INTO INTEGER (X,Y) IN PLOTTER UNITS.
C
C   - THEN APPLY ORIGIN (XORG,YORG) AND TRANS-
C     LATION (XTR,YTR) TO GET AN (X,Y) IN
C     ABSOLUTE PLOTTER UNITS ( RELATIVE TO (0,0) )
C
C
 100  X = FACTOR * XR
      Y = FACTOR * YR
      X = X + XORG - XTR
      Y = Y + YORG - YTR
C
C
C   NOW INTERROGATE "PEN" IN ORDER
C   TO DETERMINE WISHES OF USER
C
C   IF "PEN" HAS THE VALUE
C
C   -3,-2,-1 - SET NEW ORIGIN TO (X,Y)
C              CHANGE SIGN OF "PEN"
C              & PROCEED NORMALLY (=> 120)
C
C     0      - RETURN CURRENT POSITION IN
C              (XR,YR) IN PLOTTER UNITS
C
C     1      - (=>121) & TAKE LAST PEN
C              POSITION AS CURRENT ONE
C
C     2      - (=>140) & PLOT WITH PEN DOWN
C
C     3      - (=>127) & PLOT WITH PEN UP
C
C    40      - (=>126) & RESET TRANSLATION
C              TO (X,Y)
C
C   999      - (=>999) & PUT OUT PLOT
C
C  OTHER     - (=>2000) & REPORT ERROR IN
C              CALL TO "PLOT"
C
C
      IF(PEN)   130, 128, 120
 120  IF(PEN-1)  121, 121, 122
 121  IF(PENP-2)  127, 140, 127
```

```
 122   IF(PEN-3)  140, 127, 123
 123   IF(PEN-40)  125, 126, 124
 124   IF(PEN-999)   125, 999, 125
 125  GO TO 2000
C
C
C*************************************
C
C   SET TRANSLATION  (XTR,YTR)
C
 126  XTR = X - XORG + XTR
      YTR = Y - YORG + YTR
      RETURN
C
C*************************************
C
C   PLOTTING WITH PEN UP ("PEN"=3)
C
C   IF THE LAST STRING JUST FINISHED
C   AND WAS GOING "UP" (FLAG=1), THEN
C   CALL "INVRT" AND INVERT STRING
C
C   THEN  RESET (XOLD,YOLD) & SET FLAG
C   TO INDICATE NO ACTIVE STRING EXISTS
C
C
C$127 IF ( FLAG.EQ.1 )  CALL INVRT( LIST(I), I )
 127  IF ( FLAG.EQ.1 )  CALL IVRTA
      XOLD = X
      YOLD = Y
      FLAG = - 1
      RETURN
C
C*************************************
C
C   RETURN CURRENT CO-ORDINATE POSITION
C   ( IN PLOTTER UNITS ) RELATIVE TO PLOTTER
C   ORIGIN (0,0)
C
 128  XR = FLOAT ( XOLD )
      YR = FLOAT ( YOLD )
      RETURN
C
C*************************************
C
C   "PEN" = -3, -2, -1
```

```
C
C    THEN RESET ORIGIN (XORG,YORG) TO
C    (X,Y). SWITCH SIGN ON PEN, & (=>120)
C    DECODE PEN AGAIN.
C
C    IF "PEN" < -3, => 3000 & REPORT ERROR
C
 130  IF(PEN+3) 2000, 131, 131
 131   XORG = X
       YORG = Y
       PEN = - PEN
      GO TO 120
C
C***********************************************
C
C    PLOTTING WITH PEN DOWN
C
C    - SET PEN POSITION TO DOWN (PENP=2)
C
C    - WILL THERE BE OVERFLOW IN "LIST()"?
C      IF SO, => 3000 & REPORT IT
C
C    - WORKING ON A STRING, IF SO WHAT KIND ??
C      =-1 , =>141, NEW STRING
C      = 0 , =>145, DESCENDING STRING
C    & = 1 , =>160, ASCENDING STRING.
C
 140  PENP = 2
      IX = LIST ( I )
      IF ( (IX+1).GT.MAXL )  GO TO 3000
      IF ( FLAG )  141, 145, 160
C
C
C    START NEW STRING
C
C    - RESET "I"
C
C    - ( CLEAR OUT LAST RECORD - VERSION II )
C
C    - PUT A WORD OF ZEROES INTO "LIST()"
C
C    - IS THERE ROOM FOR NEXT LINE SEGMENT
C      ( (X1,Y1), (X2,Y2) ) OR WILL THERE
C      BE OVERFLOW IN "LIST()" ?
C      IF SO, => 3000 & REPORT IT.
C
```

```
C    - OVERFLOW IN "TAGS( )" ?
C      IF SO, => 3000 & REPORT IT
C
C    - PUT POINTER TO HEAD OF STRING INTO
C      "TAGS( )"
C
C    - DESCENDING OR ASCENDING STRING ?
C      (  X    XOLD  )
C
C
 141  I = LIST ( I )
      CALL CLEAR
      IF ( I.GT MAXL )  GO TO 3000
      CALL PUT( 0, I, I, FLAG, TAGS, ITAG, MAXT )
C$    LIST ( I ) = 0
      I = I + 1
      IF ( (I+4).GT MAXL )  GO TO 3000
      CALL PUT( I+5, I, I, FLAG, TAGS, ITAG, MAXT )
C$    LIST ( I ) = I + 5
      ITAG = ITAG + 1
      IF ( ITAG.GT.MAXT )  GO TO 3000
      TAGS ( ITAG ) = I + 1
      IF ( X-XOLD )  143, 142, 142
C
C
C   PUT A DESCENDING LINE SEGMENT INTO
C   "LIST()", IN NORMAL ORDER, SET FLAG=0
C
 142   FLAG = 0
       CALL PUT( XOLD, I+1, I, FLAG, TAGS, ITAG, MAXT )
       CALL PUT( YOLD, I+2, I, FLAG, TAGS, ITAG, MAXT )
       CALL PUT( X, I+3, I, FLAG, TAGS, ITAG, MAXT )
       CALL PUT( Y, I+4, I, FLAG, TAGS, ITAG, MAXT )
C$     LIST ( I+1 ) = XOLD
C$     LIST ( I+2 ) = YOLD
C$     LIST ( I+3 ) = X
C$     LIST ( I+4 ) = Y
       GO TO 180
C
C
C   PUT AN ASCENDING LINE SEGMENT INTO
C   "LIST()" IN REVERSE ORDER.
C
C   IE - ( Y1, X1, Y2, X2 )
C
C   SET FLAG=1
```

```
C
 143   FLAG = 1
       CALL PUT( XOLD,I+2,I,FLAG,TAGS,ITAG,MAXT )
       CALL PUT( YOLD,I+1,I,FLAG,TAGS,ITAG,MAXT )
       CALL PUT( X,I+4,I,FLAG,TAGS,ITAG,MAXT )
       CALL PUT( Y,I+3,I,FLAG,TAGS,ITAG,MAXT )
C$     LIST ( I+2 ) = XOLD
C$     LIST ( I+1 ) = YOLD
C$     LIST ( I+4 ) = X
C$     LIST ( I+3 ) = Y
       GO TO 180
C
C**************************************************
C
C   CURRENTLY IN A DESCENDING STRING
C
C   - CHECK FOR REVERSAL ( X < XOLD )
C     IF SO, THIS MUST BE THE START OF
C     AN ASCENDING STRING.
C     SO => 141 & START A NEW STRING
C
C   - IF NO PROBLEM PUT (X,Y) INTO
C     STRING, THEN => 179
C
C
 145   IF ( X-XOLD )  141, 150, 150
 150   CALL PUT( X,IX,I,FLAG,TAGS,ITAG,MAXT )
       CALL PUT( Y,IX+1,I,FLAG,TAGS,ITAG,MAXT )
C$150  LIST ( IX ) = X
C$     LIST ( IX+1 ) = Y
       GO TO 179
C
C**************************************************
C
C   CURRENTLY IN AN ASCENDING STRING
C
C   - CHECK FOR REVERSAL ( X > XOLD )
C     IF SO, THIS MUST BE THE START OF
C     A DESCENDING STRING,
C     SO INVERT CURRENT STRING BEFORE
C     STARTING A NEW STRING ( => 141 )
C
C   - IF NO PROBLEM PUT (X,Y) INTO
C     STRING IN REVERSE ORDER
C     ( IE -  Y, THEN X )
C
```

```
C
 160    IF ( XOLD-X )  161, 170, 170
 161     CALL IVRTA
C#161    CALL INVRT( LIST(I), I )
         GO TO 141
 170     CALL PUT( Y,IX,I,FLAG,TAGS,ITAG,MAXT )
         CALL PUT( X,IX+1,I,FLAG,TAGS,ITAG,MAXT )
C#170    LIST ( IX ) = Y
C#       LIST ( IX+1 ) = X
C
C    UPDATE "LIST(I)" TO "LIST(I)"+2
C    SO THAT THIS POINTS TO NEXT AVAILABLE
C    LOCATION IN "LIST()"
C
C    THEN SET (XOLD,YOLD) <= (X,Y)
C
 179  CALL PUT( LIST(I)+2,I,I,FLAG,TAGS,ITAG,MAXT )
C#179 LIST ( I ) = LIST ( I ) + 2
 180  XOLD = X
      YOLD = Y
      RETURN
C
C**********************************************************
C
C    PLOT IS FINISHED, OR AT LEAST THE CALLS
C    TO PLOT ARE.  INVERT THE LAST STRING IF
C    IT IS NECESSARY, AND SET I TO "LIST(I)"
C
C    IN VERSION I -- SIMPLY CALL "PRINT"
C
C    WHILE IN VERSION II --
C                  CALL "CLSEF" TO PUT OUT LAST
C                             BIT OF STORAGE
C                  THEN CALL "SWAP1" TO PERFORM
C                   NEEDED SWAPPING BEFORE GET-
C                   TING TO "PRINT"
C
C    IN BOTH VERSIONS, UPON RETURNING SET
C     "KOUNT" TO INDICATE 1ST TIME SO THAT
C     ANOTHER PLOT IS POSSIBLE FROM
C     SCRATCH.
C
 999  IF ( FLAG.EQ.1 )  CALL IVRTA
C#999 IF ( FLAG.EQ.1 )  CALL INVRT ( LIST(I), I )
      I = LIST(I)
      CALL CLSEF
```

```
      CALL SWAP1 ( TAGS, ITAG, I )
C$    CALL PRINT ( LIST, TAGS, ITAG )
      KOUNT = 0
      RETURN
C
C****************************************
C
C   ERROR MESSAGE TO INDICATE ILLEGAL
C
C    PEN ENTRY
C
 2000 WRITE ( IW, 1 )  X, Y, PEN
 1    FORMAT ( 10X, 20H**ERROR -  THERE IS    ,
     1         27HAN ERROR IN PEN IN ENTRY   ,
     2         //, 10X, 3HX =, I4, 5X, 3HY =,
     3         I4, 5X, 5HPEN =, I4, //      )
      RETURN
C
C****************************************
C
C   ERROR MESSAGE FOR OVERFLOW IN ONE
C
C   OF "LIST()", OR "TAGS()".
C
C   AFTER ERROR MESSAGE ( => 999 )
C    & ATTEMPT TO PLOT WHAT HAS
C    BEEN ACCUMULATED TO DATE
C
 3000 ILIST = LIST(I)
      WRITE ( IW, 2 )  I, ILIST, ITAG
 2    FORMAT ( 10X, 20H**ERROR - OVERFLOW L ,
     1         7HIST, (   , I4, 5H ) = , I5, /,
     2         30X, 7HITAG = , I5 )
      GO TO 999
      END
      END$
```

```
FTN4,L,T
      SUBROUTINE INVRT ( LIST, I )
C
C   SUBROUTINE "INVRT"
C
C
C   WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975
C
C   BY C  A  BRYCE, FOR A MSC  PROJECT
C
C
C   "INVRT"  IS A SIMPLE UTILITY ROUTINE USED
C                 ONLY BY "PLOT".  IT IS USED TO
C                 REVERSE OR INVERT THE STRING
C                 IN ARRAY "LIST()".  AS FAR
C                 AS THIS ROUTINE IS CONCERNED
C                 THE STRING STARTS IN "LIST(1)".
C                 THIS IS A POINTER TO START OF
C                 NEXT STRING, AND FROM THIS AND
C                 "I" THE LENGTH OF THE STRING
C                 CAN BE DETERMINED.  THEN BY
C                 GETTING THE INDEX OF THE START
C                 OF THE NEXT STRING ( "KLAST" )
C                 SIMPLY INTERCHANGE --
C                 LIST(2)    <==>   LIST(KLAST-1)
C                 LIST(3)    <==>   LIST(KLAST-2)
C
C
C
C        TILL  LIST(N)    <==>   LIST(M)
C        WHERE       M = N+1
C
C
C   KEY VARIABLES
C
C   LIST() - INTEGER ARRAY CONTAINING STRINGS
C
C   I      - INTEGER VALUE OF CURRENT POSITION
C            IN "LIST()"
C
C
      INTEGER      LIST, I
      DIMENSION   LIST ( 0 )
C
C
C
```

```
C   LOCAL VARIABLES
C
C   KLAST, K, KK
C         - INDICES AND COUNTERS USED IN LOOP
C
C   I1, I2 - FINAL INDICES USED WITH "LIST()".
C
C   ISTORE - USED FOR INTERCHANGING
C
C
      INTEGER   KLAST, K, KK, I1, I2, ISTORE
C
C*************************************************
C
C
      KLAST = LIST(1) - I + 1
      K = (KLAST-1) / 2
      DO 210  KK = 1, K
          I1 = KK + 1
          I2 = KLAST - KK
          ISTORE = LIST(I1)
          LIST(I1) = LIST(I2)
210       LIST(I2) = ISTORE
      RETURN
      END
      END$
```

```
FTN4,L
      SUBROUTINE PRINT ( LIST, TAGS, ITAG )
C
C     SUBROUTINE "PRINT"
C
C
C     WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975
C
C     BY C. A  BRYCE, FOR A MSC  PROJECT
C
C
C     "PRINT" IS CALLED FROM SUBROUTINE "PLOT"
C                  EITHER DIRECTLY IN VERSION I,
C                  OR INDIRECTLY IN VERSION II
C                  IT'S PURPOSE IS TO DO THE ACTUAL
C                  PHYSICAL PLOTTING TO THE
C                  ELECTRO-STATIC PRINTER/PLOTTER.
C                  IT PLOTS THE STRINGS IN "LIST()"
C                  USING THE TAG ARRAY "TAGS()",
C                  AND THE NUMBER OF TAGS "ITAG".
C
C
C     "PRINT" USES 4 SUBROUTINES.
C
C     1) EXEC  - THE SYSTEM EXECUTIVE TO GENERATE
C                PAGE THROWS BEFORE & AFTER PLOTTING
C
C     2) ORDER - A ROUTINE THAT DOES A TAG-SORT
C                ON "LIST()", WITH TAGS "TAGS()"
C                AND NUMBER OF TAGS "ITAG"
C
C     3) INSRT - A ROUTINE THAT ADDS NECESSARY BITS
C                INTO OUTPUT BUFFER "LINE()"
C
C     4) BFOUT - THE ROUTINE THAT PUTS "LINE()" OUT
C                TO THE PRINTER/PLOTTER
C
C
C     THIS ROUTINE ACCOMPLISHES ITS PURPOSE BY
C     FIRST DOING A TAG-SORT ON "LIST()", "TAGS()",
C     AND "ITAG".  THIS PUTS STRINGS INTO
C     ASCENDING ORDER SO THAT THE REST OF THIS
C     ROUTINE CAN WORK PROPERLY.
C     THEN LINE BY LINE, (X=0,1,2,3,...), PASS
C     DOWN THE PAGE, FIRST PUTTING OUT BLANK
C     LINES, ( "LINE()"=0 ) UNTIL X REACHES FIRST
```

```
C    STRING ( X=LIST(TAGS(1)) )   THEN START
C    PLOTTING BY PUTTING THE Y POINTS AS THEY
C    OCCUR IN THE STRINGS AND ARE IN THE RIGHT
C    RANGE   THE 1ST, 2ND, 3RD,    STRING
C    IS CHECKED, AND THE NEEDED POINTS PUT
C    INTO "LINE()" UNTIL A STRING IS MET THAT
C    STARTS BELOW CURRENT X VALUE ( X=LIST(TAGS()) )
C    ALSO TO SAVE EXCESSIVE WORK IF THE FIRST
C    STRING IS EXHAUSTED, IT IS NOT CHECKED
C    ANYMORE, AND THIS GOES FOR THE NEXT 'FIRST'
C    STRING.  THUS ONLY THE STRINGS STARTING AT
C    THE FIRST STRING WITH SOMETHING TO PLOT FOR
C    PREVIOUS X ( X-1 ) TO THE FIRST STRING PAST
C    WHERE PLOTTING OCCURS FOR CURRENT X IS
C    CHECKED   WHEN NO MORE STRINGS EXIST TO
C    PLOT WE ARE DONE
C    TO IMPROVE THE QUALITY OF THE PLOT, THE
C    PRINTER IS PAGED BOTH BEFORE AND AFTER
C    PLOTTING.
C
C
C    KEY VARIABLES
C
C    LIST() - INTEGER ARRAY CONTAINING STRINGS
C             OR RECORDS TO BE PLOTTED
C
C    TAGS() - INTEGER ARRAY CONTAINING TAGS OR
C             POINTERS TO START OF STRINGS IN
C             "LIST()"
C
C    ITAG   - NUMBER OF STRINGS IN "LIST()"
C
      INTEGER  LIST, TAGS, ITAG
      DIMENSION  LIST ( 0 ), TAGS ( 0 )
C
C
C    LOCAL VARIABLES
C
C    LINE() - INTEGER BUFFER USED TO STORE
C             PLOTTED POINTS ( INITIALLY=0 )
C
C    MAX    - INTEGER CONSTANT OF NUMBER OF
C             POINTS IN "LINE()"+1 ( 561 )
C
C    BIAS   - INTEGER CONSTANT OF BIAS USED
C             IN STORING SLOPE
```

```
C
C    SLPMX  - REAL CONSTANT OF MAXIMUM ALLOWABLE
C             SLOPE X BIAS ( EFFECTIVELY 561 0 )
C
C    X      - INTEGER VALUE IN PLOTTER UNITS
C             OF CURRENT X CO-ORDINATE
C             ( INITIALLY = -1 )
C
C    IMIN   - MINIMUM INDEX-1 IN "TAGS()"
C             TO BE CHECKED ( INITIALLY=0 )
C
C    I      - INDEX USED FOR "TAGS()"
C
C    IX1    - WHEN CONSIDERING A LINE SEGMENT
C             ( X1, Y1) TO ( X2, Y2) IN
C             ORDER TO GET THE SLOPE ,
C             IX1 IS THE INDEX IN "LIST()" OF
C             X1
C
C    IY1    - INDEX OF Y1, ( IX1+1 )
C
C    IX2    - INDEX OF X2, ( IX1+2 )
C
C    IY2    - INDEX OF Y2, ( IX1+3 )
C
C    ISL    - INDEX WHERE BIASED SLOPE FOR
C             CURRENT LINE SEGMENT IS IN
C             "LIST()", ( IX1-2 )
C
C    LAST   - INDEX OF START-OF-NEXT
C             STRING
C
C    SLOPE  - REAL VALUE OF SLOPE
C
C    YMIN   - MINIMUM Y VALUE FOR CURRENT
C             STRING IN "LINE()"
C
C    YMAX   - MAXIMUM CORRESPONDING TO YMIN
C
C    YLAST  - LAST Y VALUE IN LINE SEGMENT
C
C    YPRES  - PRESENT Y VALUE IN LINE SEGMENT
C
C    YNEXT  - NEXT Y VALUE IN LINE SEGMENT
C
C    Y      - INDEX USED IN FILLING "LINE()"
```

```
C
C
      INTEGER  LINE, MAX, BIAS, X, IMIN, I
      DIMENSION  LINE ( 35 )
      INTEGER  IX1, IY1, IX2, IY2, ISL, LAST
      REAL     SLPMX, SLOPE
      INTEGER  Y, YMIN, YMAX
      INTEGER  YLAST, YPRES, YNEXT
C
C
C   INITIALIZE DATA
C
      DATA  LINE  / 35*0 /
C
C    IN ABOVE DATA STATEMENT 35*0 IS NON-ANSI
C
C    THIS SETS 35 WORDS IN LINE() TO 0
C
      DATA   MAX, BIAS / 561, 58 /
C
C
C***************************************
C
C   START OF SUBROUTINE "PRINT"
C
C
C   PUT "STRINGS" IN ARRAY LIST() INTO
C       ASCENDING ORDER BY DOING A TAG
C       SORT ON [ LIST(), TAGS(), & ITAG ].
C
      CALL ORDER ( LIST, TAGS, ITAG )
C
C
C   PERFORM A PAGE EJECT SO THAT PLOT
C   STARTS ON TOP OF PAGE.
C
      CALL EXEC ( 3, 1110B, 63 )
C
C   INITIALIZE  -- X, & IMIN TO -1
C               -- CALCULATE "SLPMX", SLOPE-MAX.
C
C   THEN SKIP THE BUFFERING ( BY GOING TO 11 )
C
C
      SLPMX = FLOAT(MAX*BIAS)
      X = - 1
```

```
      IMIN = 0
      GO TO 11
C
C*****************************************************
C
C   BUFFER OUT CONTENTS OF LINE()
C
 10   CALL BFOUT ( LINE )
C
C   INCREMENT X & START I AT IMIN
C
 11   X = X + 1
      I = IMIN
C
C   IN THE NEXT BLOCK OF CODE, BY STARTING
C        WITH I AT IMIN+1 AND WHILE I IS LESS
C        THAN ITAG, THE STRINGS ARE SEARCHED
C        FOR EXISTING STRINGS ( TAGS(I) > 0 )
C
C        IF A STRING IS FOUND, PROCEED TO 30
C           TO DO NECESSARY CALCULATIONS,
C        AND IF NOT CONTINUE SEARCHING
C        ( I <= I + 1 ), BUT REMEMBERING
C        TO ELIMINATE DEAD STRINGS AT
C        BEGINNING. ( TAGS(I)=0, & I=IMIN+1 )
C
C        ALSO WHEN IMIN=ITAG, THERE ARE NO MORE
C        STRINGS TO PLOT, SO RETURN
C
C
C
 20   I = I + 1
       IF ( I.GT.ITAG )  GO TO 10
       IX1 = TAGS ( I )
       IF ( IX1.GT.0 )  GO TO 30
       IF ( IMIN.LT.(I-1) )  GO TO 20
       IMIN = IMIN + 1
       IF ( IMIN.LT.ITAG )  GO TO 20
C
C   SKIP TO TOP OF NEXT PAGE
C
C
      CALL EXEC ( 3, 1110B, 63 )
      RETURN
C
C*****************************************
```

```
C
C
C    INITIALIZE YMIN & YMAX FOR THIS STRING
C
C
 30    YMIN = MAX
       YMAX = 0
C
C    CALCULATE INDEXES IN LIST() FOR
C    ISL, LAST, IY1, IX2, & IY2.
C
C
 32    ISL = IX1 - 2
       LAST = IX1 - 1
       LAST = LIST ( LAST )
       IY1 = IX1 + 1
       IX2 = IY1 + 1
       IY2 = IX2 + 1
C
C    IF "X1" < 0   => 35 AND CALCULATE SLOPE
C                       WHEN "X1" < "X2"
C
C    OTHERWISE ONLY CALCULATE SLOPE WHEN
C    X = "X1" < "X2".
C
C    IN THE CASE WHEN X < "X1" , THIS IS 1ST
C       STRING PAST THE STRINGS TO BE PUT
C       OUT SO BUFFER-OUT LINE() ( => 10 )
C
C
        IF ( LIST(IX1).LT.0 ) GO TO 35
        IF ( X-LIST(IX1) )  10, 35, 40
 35     IF ( LIST(IX1).EQ.LIST(IX2) ) GO TO 60
        SLOPE = FLOAT( BIAS*( LIST(IY1)-LIST(IY2) ) ) /
      1          FLOAT( LIST(IX1)-LIST(IX2) )
        SLOPE = SIGN(AMIN1(ABS(SLOPE),SLPMX),SLOPE)
        LIST ( ISL ) = IFIX ( SLOPE )
C
C
C    NOW CONSIDER   X , "X2"
C
C     - NORMAL CASE  X < "X2" --  => 70
C       ( "X1" =< X < "X2" )
C
C     - END-OF-LINE-SEGMENT WHEN X = "X2"
C       --- => 60 ( HANDLE SPECIAL CASES )
```

```
C
C      - AT START WHEN X=0, IT IS POSSIBLE
C        FOR - "X1" =, "X2" < X = 0
C        -- SKIP THIS LINE SEGMENT
C
C
 40    IF ( X-LIST(IX2) ) 70, 60, 50
 50     LIST ( IY1 ) = LAST
        IX1 = IX2
        TAGS ( I ) = IX1
        IF ( IX2 LT LAST )  GO TO 32
C
C
C    STRING IS FINISHED ( IX2=LAST )
C
C       KILL TAG FOR THIS STRING,
C       ( TAGS(I) = 0 ),
C
C       => 80 AND PUT NECESSARY POINTS
C            INTO  LINE().
C
C
        TAGS ( I ) = 0
        GO TO 80
C
C*********************************************
C
C
C    GET TO 60 WHEN X = "X2".
C
C       CALCULATE A YLAST & A YNEXT FOR CASES
C
C       IF  X = "X1",    YLAST = "Y1"
C                      & YNEXT = "Y2",
C
C       IF  X > "X1",    YLAST = "Y2" - SLOPE/2
C                      & YNEXT = "Y2".
C
C       IN ANY EVENT ADJUST YMIN & YMAX AS
C          REQUIRED FOR THESE VALUES OF
C          YLAST AND YNEXT.
C
C
 60     IF ( X GT LIST(IX1) )  GO TO 61
         YLAST = LIST ( IY1 )
         GO TO 62
```

```
C
 61      YLAST = LIST(IY2) - ( LIST(ISL)/(2*BIAS) )
C
 62     IF ( YLAST.LT YMIN )  YMIN = YLAST
        IF ( YLAST.GT YMAX )  YMAX = YLAST
C
 64     YNEXT = LIST ( IY2 )
        IF ( YNEXT LT.YMIN )  YMIN = YNEXT
        IF ( YNEXT GT YMAX )  YMAX = YNEXT
C
C
C    SKIP LINE SEGMENT BY ADJUSTING POINTERS
C    ISL, IX1,  & IX2
C
C    BUT BEFORE COMPLETELY SKIPPING SEGMENT
C    CHECK TO SEE IF THERE IS AN [ X2, Y2 ]
C    ( IX2 < LAST ).
C
C
        ISL = IX1
        IX1 = IX2
        IX2 = IX1 + 2
        IF ( IX2 LT.LAST )  GO TO 68
C
C    AT END-OF-STRING
C
C    -- KILL TAG FOR STRING ( TAGS(I)=0 )
C
C    -- => 80 AND INSERT NECESSARY POINTS.
C
         TAGS ( I ) = 0
         GO TO 80
C
C
C    COMPLETE LINE SEGMENT SKIP BY ADJUSTING
C    TAG, LIST(IY1), IY1, IY2
C
C    CHECK IF INFINITE SLOPE ( X="X2" ) ?
C    IF SO -- => 64
C    IF NOT-- => 35 AND CALCULATE NEW SLOPE.
C
C
 68     TAGS ( I ) = IX1
        LIST ( IY1 ) = LAST
        IY1 = IX1 + 1
        IY2 = IX2 + 1
```

```
      IF ( X.EQ LIST(IX2) )  GO TO 64
      GO TO 35
C
C
C*******************************************
C
C
C   GET HERE UNDER NORMAL CONDITIONS
C
C   THAT IS - "X1" =,< X < "X2".
C
C   INTERPOLATE TO GET VALUE OF
C      Y IN RANGE [ "Y1", "Y2" ]
C      & THEN BY GOING BACK & FORWARD
C      0.5*SLOPE GETTING VALUES
C      FOR  YLAST & YNEXT.
C      IF X="X1" THEN YLAST="Y1".
C
C
 70   YLAST = LIST ( ISL ) * ( X-LIST(IX1) )
     1            -  LIST ( ISL ) / 2
      YLAST = ( YLAST / BIAS ) + LIST ( IY1 )
      YNEXT = YLAST + ( LIST ( ISL ) / BIAS )
      IF ( X EQ LIST(IX1) )  YLAST = LIST ( IY1 )
C
C
C   NEXT ADJUST  YMIN & YMAX AS NECESSARY TO
C      ACCOMODATE  YLAST & YNEXT SO THAT
C      -- YMIN < ALL PTS IN LINE() < YMAX
C
C
 75   IF ( YLAST.LT.YMIN )  YMIN = YLAST
      IF ( YLAST.GT.YMAX )  YMAX = YLAST
      IF ( YNEXT.LT.YMIN )  YMIN = YNEXT
      IF ( YNEXT.GT.YMAX )  YMAX = YNEXT
C
C
C   CHECK IF ANYTHING TO PUT INTO  LINE()
C
C   THAT IS -- 0 < YMIN =,< YMAX < MAX.
C
C   IF SO -- PUT THEM INTO  LINE()
C
C
 80   IF ( YMIN.GE.MAX.OR.YMAX.LE.0 )  GO TO 20
C
```

```
      IF ( YMIN.LE.0 )     YMIN = 1
      IF ( YMAX.GE.MAX )  YMAX = MAX - 1
C
      DO 90  Y = YMIN, YMAX
 90   CALL INSRT ( LINE, Y )
C
C
C   NEXT STRING  ( =  20 )
C
C
      GO TO 20
C
C++++++++++++++++++++++++++++++++++++
C
C   END OF "PRINT"
C
      END
      END$
```

```
FTN4,L
      SUBROUTINE ORDER ( LIST, TAGS, ITAG )
C
C     SUBROUTINE "ORDER"
C
C
C     ADAPTED AT MCMASTER UNIVERSITY, FEBRUARY 1975
C
C     BY C. A. BRYCE, FOR A MSC. PROJECT
C
C
C     THIS SUBROUTINE HAS BEEN OBTAINED
C     TO PROVIDE AN EFFICIENT SORTING PROCESS.
C     THE SORT IN ADDITION HAD TO BE A
C     TAG-SORT.  THIS WAS NOT ORIGINALLY A
C     TAG-SORT, BUT WAS ADAPTED INTO ONE.
C
C     IT WAS FOUND IN A 4TH YEAR PROJECT AT
C     THE APPLIED MATH DEPARTMENT AT
C     MCMASTER UNIVERSITY.  THE PROJECT WAS
C
C     TITLED --
C       "INVESTIGATIONS OF AVAILABLE SORTING
C       ALGORITHMS AND THEIR BEHAVIOUR FOR
C       DIFFERENT ORDERINGS", AND WAS
C
C     AUTHORED BY --
C         MILKHA SINGH, AND BETTY M. RYDE
C       ( 1969-70 )
C
C
C     THIS SORT WAS CHOSEN BECAUSE IT RATED
C     HIGHLY ( IN THE TOP 2 ) AND BECAUSE
C     ONLY 4 CHANGES WERE NEEDED TO MAKE
C     IT INTO A TAG-SORT.
C
C
C     IN THIS DOCUMENTATION NO ATTEMPT WILL
C     BE MADE TO EXPLAIN HOW THE SORT WORKS
C     OTHER THAN TO SHOW WHERE THE NEEDED
C     CHANGES WERE MADE.
C
C
C     SUBROUTINE ORDER IS CALLED FROM "PRINT"
C               IN ORDER TO SORT THE TAGS IN
C               TAGS() TO THE STRINGS IN
```

```
C                 LIST().  EACH TAG POINTS TO
C                 TOP OF A STRING, AND TAGS()
C                 IS REARRANGED SO THAT INDEXING
C                 THROUGH TAGS() WILL ALSO BE
C                 GOING THROUGH ASCENDING VALUES
C                 FOR THE TOP OF EACH STRING
C
C
C     KEY VARIABLES
C
C     LIST() - INTEGER ARRAY CONTAINING STRINGS
C              OR RECORDS TO BE PLOTTED
C
C     TAGS() - INTEGER ARRAY CONTAINING TAGS OR
C              POINTERS TO START OF STRINGS IN
C              "LIST()"
C
C     ITAG   - NUMBER OF STRINGS IN "LIST()"
C
      INTEGER LIST, TAGS, ITAG
      DIMENSION  TAGS ( 0 ), LIST ( 0 )
C
C     LOCAL VARIABLES
C
C     I, J, K, L, M - INTEGER VARIABLES USED
C                     IN SORTING PROCESS
C
C     II, IL        - INTEGER VARIABLES ADDED ( CAB )
C                     REPLACING I, & L IN PART
C                     OF SORT
C
C
      INTEGER  I, J, K, L, M
      INTEGER  II, IL
C
C
C     M <= # OF ELEMENTS TO BE SORTED
C
      M = ITAG
C
C     DIVIDE INTO 2 SUBSETS
C
C     WHEN M = 0 THE PROCESS IS COMPLETE //
C
 1    M = M / 2
      IF ( M.EQ.0 )  RETURN
```

```
C
C     SET K TO # ELEMENTS - M
C
        K = ITAG - M
C
C     THE FOLLOWING  " DO 3 " DO-LOOP
C     ACCOMPLISHES MIRACLES AND DOES
C     MOST OF THE WORK
C
        DO 3  J = 1, K
        I = J
 2      L = I + M
C
C
C     THE FOLLOWING 2 LINES SETTING
C     II, & IL WERE ADDED TO GET
C     NEEDED INDICES IN "LIST()"
C
        II = TAGS ( I )
        IL = TAGS ( L )
C
C     COMPARE  LIST(II) : LIST(IL)
C
        IF ( LIST(II).LE.LIST(IL) )  GO TO 3
C
C     THE FOLLOWING 2 LINES WERE ADDED
C     NOTICE THAT THE TAGS ARE
C     INTERCHANGED INSTEAD OF THE
C     ARRAY ELEMENTS ( LIST() )
C
        TAGS ( L ) = II
        TAGS ( I ) = IL
        I = I - M
        IF ( I GT 0 )  GO TO 2
 3      CONTINUE
C
C     REDIVIDE WORK TO BE DONE
C
        GO TO 1
        END
        END$
```

```
PAGE 0001

0001                    ASMB,R,L,T,C
INSRT R 000002
 ENTR X 000001
LINE  R 000000
POINT R 000001
MASK  R 000021
TBL   R 000022
A       000000
B       000001
** NO ERRORS*
```

```
PAGE 0002 #01  << INSRT -- INCL OR OF BIT INTO "LINE" -- FEB/75 >>

                    ASMB,R,L,T,C
      00000                NAM INSRT,7
*
*    SUBROUTINE "INSRT"
*
*
*    WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975
*
*    BY C. A. BRYCE, FOR A MSC. PROJECT
*
*
*    "INSRT" IS CALLED FROM SUBROUTINE "PRINT"
*              AND IS DESIGNED TO DO A
*              LOGICAL INCLUSIVE OR OF A BIT
*              AT THE "POINT"'TH BIT POSITION
*              IN THE ARRAY "LINE".  ALL CALLS
*              FROM "PRINT" ARE SET UP SO THAT
*              "POINT" IS IN THE PROPER RANGE.
*
*
*    AN ASSEMBLER ROUTINE WAS USED RATHER THAN
*    A FORTRAN ROUTINE AS IT WAS SMALLER
*    ( 42B COMPARED TO 103B FOR FORTRAN )
*    AND SUBSTANTIALLY FASTER.  AS THIS IS
*    USED FREQUENTLY, THIS SPEED FACTOR BECAME
*    THE DECIDING FACTOR.
*
*
*    KEY VARIABLES
*
*    LINE() - INTEGER ARRAY CONTAINING PLOTTING
*             INFORMATION.
*
*    POINT  - THE BIT POSITION IN "LINE" WHERE
*             BIT IS TO BE ADDED.
*
*    TBL()  - LOOKUP TABLE CONTAINING THE
*             DIFFERENT REQUIRED BITS.
*
*    MASK   - 4-BIT LOWER END MASK.
*
*    A & B  - EQUIVALENCES TO A & B REGISTERS.
```

```
  FORTRAN EQUIVALENT ( TESTED FEB/13/75 )

        SUBROUTINE INSRT ( LINE, POINT )
C
C       FORTRAN VERSION OF "INSRT"
C
        INTEGER  LINE, POINT, TBL
        DIMENSION  LINE ( 0 ), TBL ( 16 )
C
        DATA  TBL / 100000B, 40000B, 20000B,
     1               10000B,  4000B,  2000B,
     2                1000B,   400B,   200B,
     3                 100B,    40B,    20B,
     4                  10B,     4B,     2B,
     5                   1B
C
        I = (POINT+15) / 16
        J = POINT - 16*(I-1)
C
C       ON HP2100A "IOR" IS THE INCLUSIVE-OR
C                        FUNCTION
C
        LINE ( I ) = IOR ( LINE(I), TBL(J) )
C
        RETURN
        END
```

```
*
*   DECLARE "INSRT" TO BE THE ENTRY-POINT
*   AND ".ENTR" TO BE AN EXTERNAL.
*
                            ENT INSRT
                            EXT .ENTR
*

*
*   THE ADDRESSES FOR "LINE" AND FOR "POINT"
*   WILL BE STORED HERE BY ".ENTR".
*
00000 000000  LINE  BSS 1
00001 000000  POINT BSS 1
*
*
**************************************************************
*
*     WE SET UP THE ENTRY/EXIT LINE - INSRT
*
*     WE START HERE
*
00002 000000  INSRT BSS 1
*
*   CALL ".ENTR" TO SET UP FORTRAN LINKAGE.
*
00003 016001X       JSB .ENTR
00004 000000R       DEF LINE
*

*
*   GET THE VALUE OF "POINT" - 1
*   AND PUT IT IN BOTH A & B
*
*
00005 003400        CCA          A <= -1
00006 142001R       ADA POINT,I  A <= POINT-1
00007 064000        LDB A        B <= POINT-1
```

```
0116*
0117*   TAKE THE COPY IN B AND RIGHT SHIFT IT 4 BITS
0118*   THUS EFFECTIVELY DIVIDING BY 16, AND
0119*   BY ADDING THE ADDRESS OF LINE(1) OBTAINING
0120*   THE ADDRESS OF THE PROPER ELEMENT OF LINE()
0121*
0122*
0123  00010 005121         BRS,BRS
0124  00011 005121         BRS,BRS         RIGHT SHIFT B 4 PLACES
0125  00012 046000R        ADB LINE        B <= ADDRESS IN ARRAY LINE
0126*
0127*   MASK OFF LAST 4 BITS OF A AND USE
0128*   THE RESULT AS THE INDEX FOR LOOKUP
0129*   TABLE "TBL"
0130*
0131*   THEN USING THE ADDRESS IN B, OR IN
0132*   VALUE FROM LINE() AND THEN REPLACE
0133*   IT BACK INTO LINE().
0134*
0135*
0136  00013 012021R        AND MASK        JUST LAST 4 BITS OF A
0137  00014 042022R        ADA TBL         GET ADDRESS IN "TBL"
0138  00015 160000         LDA A,I         A <= ENTRY FROM "TBL"

0140  00016 130001         IOR B,I         A <= IOR ( A-REG, LINE() )
0141  00017 170001         STA B,I         LINE() <= RESULT IN A
0142*

0144*
0145*   RETURN
0146*
0147  00020 126002R        JMP INSRT,I
0148*
```

```
0150*
0151*      DATA NEEDED
0152*
0153  00021 000017  MASK  OCT 17
0154  00022 000023R TBL   DEF *+1
0155*
0156*      TBL IS THE ADDRESS OF THE NEXT WORD
0157*
0158  00023 100000        OCT 100000
0159  00024 040000        OCT  40000
0160  00025 020000        OCT  20000
0161  00026 010000        OCT  10000
0162  00027 004000        OCT   4000
0163  00030 002000        OCT   2000
0164  00031 001000        OCT   1000
0165  00032 000400        OCT    400
0166  00033 000200        OCT    200
0167  00034 000100        OCT    100
0168  00035 000040        OCT     40
0169  00036 000020        OCT     20
0170  00037 000010        OCT     10
0171  00040 000004        OCT      4
0172  00041 000002        OCT      2
0173  00042 000001        OCT      1
0174*

0176*
0177  00000         A     EQU 0
0178  00001         B     EQU 1
0179*
0180                      END
** NO ERRORS*
```

.ENTR	00082	00103	
A	00177	00114	00138
B	00178	00140	00141
INSFT	00099	00081	00147
LINE	00089	00104	00125
MASK	00153	00136	
POINT	00090	00113	
TBL	00154	00137	

```
FTN4,L
      SUBROUTINE BFOUT ( LINE )
C
C   SUBROUTINE "BFOUT"
C
C
C   WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975
C
C   BY C. A. BRYCE, FOR A MSC. PROJECT.
C
C
C   "BFOUT" IS CALLED FROM SUBROUTINE "PRINT",
C              AND IT'S PURPOSE IS TO OUTPUT
C              THE CONTENTS OF "LINE" ONTO THE
C              PRINTER/PLOTTER.
C
C
C   METHOD
C
C   IN PUTTING OUT THE CONTENTS OF "LINE()" TO THE
C   PRINTER/PLOTTER, DOUBLE BUFFERING IS USED
C   IN ORDER TO TAKE ADVANTAGE OF THE CAPABILITY
C   OF WRITING WITHOUT WAITING.  "LINE()" IS
C   COPIED TO "BUFF()", AND SIMULTANEOUSLY
C   "LINE()" IS BEING CLEARED ( <= 0 ).  WITH
C   THE INFORMATION NOW IN "BUFF()", A BINARY
C   WRITE WITHOUT WAIT IS INITIATED ON "BUFF()".
C   THEN CONTROL IS RETURNED BACK TO "PRINT" TO
C   ALLOW "PRINT" TO PROCEED WITH CALCULATIONS
C   FOR THE NEXT LINE.
C
C   ALTHOUGH THROUGH TESTING IT WAS FOUND THAT
C   ON THE HP, THIS DOUBLE BUFFERING WASN'T
C   ABSOLUTELY NECESSARY, IT WAS INCLUDED IN
C   ORDER TO PROTECT AGAINST POSSIBLE PROBLEMS
C   WITH DIFFERENT MACHINES.  IN THIS TESTING
C   IT WAS ALSO FOUND THAT A STATUS CALL WASN'T
C   NECESSARY IN THE TYPE OF WRITING DONE IN
C   "BFOUT".  WITH BUFFERS LARGER THAN 35 WORDS
C   STATUS CHECKING WOULD HAVE BEEN NECESSARY.
C
C
C   KEY VARIABLES
C
C   LINE () - INTEGER ARRAY CONTAINING PLOTTING
C              INFORMATION.
```

```
C
C    BUFF () - INTEGER ARRAY USED FOR COPYING
C              INFORMATION FROM "LINE".
C
C    J         - INDEX USED WITH "LINE()" & "BUFF()"
C
C
C
      INTEGER  LINE, BUFF, J
      DIMENSION  LINE ( 35 ), BUFF ( 35 )
C
C
C    FIRST - COPY  LINE() => BUFF()
C
C            & SIMULTANEOUSLY  0 => LINE()
C
C
      DO 100  J = 1, 35
      BUFF ( J ) = LINE ( J )
  100 LINE ( J ) = 0
C
C
C
C
C    NEXT - DO AN "EXEC" CALL TO DO A
C           WRITE ( AN RCODE OF 2) TO
C
C           LOGICAL UNIT 8 ( CNWD = 10B )
C           IN BINARY MODE ( CNWD = CNWD + 100B )
C           AND DON'T WAIT ( CNWD = CNWD + 20000B )
C
C           FROM BUFFER ( NAMED "BUFF" )
C
C           AND THE BUFFER LENGTH ( 35 ).
C
C
      CALL EXEC ( 2, 20110B, BUFF, 35 )
C
C    THE PRINTER HAS BEEN STARTED, BUT
C    RETURN TO "PRINT" AND SAVE THE NORMAL
C    WAITING PERIOD.
C
C
      RETURN
      END
      END$
```

```
PAGE 0001

0001                              ASMB,R,L,T,C
PUT     R 000007
LIST    R 000315
CLEAR   R 000332
CLSEF   R 000631
IVPTA   R 000101
SWAP1   R 000654
 ENTR   X 000001
ERR0    X 000002
EXEC    X 000003
INVRT   X 000004
 PAUS   X 000005
VAL     R 000000
IND     R 000001
I       R 000002
FLAG    R 000003
TAGS    R 000004
ITAG    R 000005
MANT    R 000006
RANGE   R 000021
ERRR    R 000035
OVFLW   R 000043
O 2     R 000050
BASI    R 000110
BASAD   R 000111
BUFF    R 000114
IL      R 000314
LOOP    R 000351
CNT     R 000363
ADD     R 000364
SCTR    R 000365
SCTRL   R 000565
SCTRI   R 000566
EMPTY   R 000567
SCTR0   R 000601
E.LP    R 000607
GTRK    R 000617
P128    R 000627
SECT    R 000630
FOUR    R 000645
M201    R 000646
TRK     R 000647
TRACK   R 000650
TGS     R 000651
ITG     R 000652
```

PAGE 0002

```
IMAX   R 000653
RWHI   R 000705
GTTRK  R 000717
TCONS  R 000730
LUN    R 000731
S/TRK  R 000732
RESET  R 000733
M19    R 000746
RTPK   R 000747
RWRCD  R 000750
RSECT  R 000751
R14    R 000752
BFSZ   R 000753
FNAME  R 000754
SWAP2  R 000757
PP     R 000757
A        000000
B        000001
.        000053
LWAM     000100
MSECT    000511
**  NO ERRORS*
```

```
                 ASMB,R,L,T,C
      00000             NAM DSKIO,7

*
*
*    "DSKIO" -- DISK I/O PACKAGE ( VERSION II )
*
*
*    WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975.
*
*    BY C. A. BRYCE, FOR A MSC. PROJECT
*
*
*    "DSKIO" IS A SET OF 5 SUBROUTINES & 1 FUNCTION
*              THAT PROVIDES DISK STORAGE TO PLOT
*              INSTEAD OF ARRAY STORAGE.  ALSO,
*              THE BEGINNING OF THE SWAPPING "SWAP1"
*              IS INITIATED IN THIS PACKAGE.
*
```

```

   THE FOLLOWING ROUTINES MAKE UP "DSKIO":

   PUT        - THIS ROUTINE ALLOWS PLOT TO SIMULATE
                AN ASSIGNMENT STATEMENT OF THE
                FORM -- LIST(I) =

   LIST       - THIS FUNCTION RETURNS THE VALUE IN
                THE RECORD CONSTRUCTED IN PUT &
                SIMULATES AN ACCESS OF THE PSEUDO-
                ARRAY [ LIST ] --      = LIST(I)

   CLEAR      - THIS ROUTINE TRANSFERS THE CURRENT
                STRING FROM THE RECORD ( OR STRING )
                BUFFER INTO A SECTOR BUFFER   THE
                SECTOR BUFFER IS AUTOMATICALLY FLUSHED
                WHEN IT BECOMES FULL

   CLSEF      - THIS ROUTINE ALLOWS THE FLUSHING OF
                THE LAST SECTOR BUFFER TO DISK, AND
                THE RE-INITIALIZATION OF SOME VARIABLES
                USED IN "DSKIO" SO THAT WHEN WE RETURN
                WE CAN START OVER

   IVRTA      - THIS ROUTINE SIMPLY CALLS "INVRT" WHICH
                CONVERTS AN UP-STRING INTO A DOWN-STRING
                THE KEY POINT IS THAT IT DOES IT WITH
                THE STRING RECORD ( BUFF )

   SWAP1      - THIS ROUTINE BEGINS THE SWAPPING PACKAGE
                HI-CORE IS SAVED ON DISK. AND THEN "SWAP2"
                IS READ INTO THIS SAME CORE, AND A CALL
                IS MADE OF SWAP2 WHICH COMPLETES THE
                SWAPPING PROCESS

```

```
PAGE 0005 #01  << DSKIO -- DISK I/O PACKAGE < VERS  II > >>

*
*
*   ENTRY POINTS
*
*
                  ENT PUT
                  ENT LIST
                  ENT CLEAR
                  ENT CLSEF
                  ENT IVPTA
                  ENT SWAP1

*
*
*   EXTERNALS USED
*
*
                  EXT .ENTR
                  EXT ERR0
                  EXT EXEC
                  EXT INVPT
                  EXT .PAUS
```

PAGE 0006 #01 << SUBROUTINE PUT(VAL,I,) >>

```
*
*   SUBROUTINE PUT  -- IN FORTRAN -- CALL PUT(VAL,I,
*
*   PUT  - INSERTS VAL INTO INDEX I IN PSEUDO-
*          ARRAY LIST   THIS SIMULATES AN ASSIGNMENT
*          STATEMENT ON LIST  ( LIST(IND)=VAL )
*
*   PUT REQUIRES SEVERAL ARGUMENTS SIMPLY BECAUSE
*       THE CURRENT VALUES OF I, FLAG, TAGS, ITAG,
*       AND MAXT ARE NEEDED WHEN THE RECORD-
*       BUFFER OVERFLOWS AND A STRING IS BROKEN
*       INTO TWO SUB-STRINGS
*
*

 00000 000000  VAL   BSS 1
 00001 000000  IND   BSS 1
 00002 000000  I     BSS 1
 00003 000000  FLAG  BSS 1
 00004 000000  TAGS  BSS 1
 00005 000000  ITAG  BSS 1
 00006 000000  MAXT  BSS 1

 00007 000000  PUT   NOP

*
*   CLEAR UP THE LINKAGE
*
 00010 016001X       JSB  ENTR
 00011 000000R       DEF VAL

*
*
*   CHECK THE RANGE OF THE INDEX
*
*   ON RETURN FROM RANGE---
*
*      B   -TIVE   --- NO PROBLEM
*      B   +TIVE   --- OVERFLOW
*
 00012 016021R       JSB RANGE
 00013 006021        SSB,RSS
```

```
*
*
*   IF OVERFLOW GO TO "OVFLW"
*
00014 026043R         JMP OVFLW

*
*
*   IF IND IN RANGE
*
*   A <= ADDRESS IN THE BUFFER
*
*   B  = VALUE
*
*   VALUE PUT INTO BUFFER
*
*
00015 042111R         ADA BASAD
00016 166000R         LDB VAL.I
00017 174000          STB A.I


*
*   RETURN
*
00020 126007R         JMP PUT.I
*
```

```
*
*
*  RANGE.- CHECKS TO SEE IF "IND" IS IN
*          RANGE BY COMPARING IT TO THE
*          BASE INDEX
*
*
00021 000000  RANGE NOP
*
*
*  A := IND + 2 - BASI
*
*
00022 062110R        LDA BASI
00023 003004         CMA,INA
00024 002004         INA
00025 002004         INA
00026 142001R        ADA IND,I
*
*
*  IF  A-REG . 0
*     THINGS ARE OK
*
*  IF NOT REPORT ERROR (ERRP)
*
*
00027 002002         SZA
00030 002020         SSA

00031 026035R        JMP ERRP

*
*
*  FOR EXITTING WE SET B TO
*
*  INDEX IN RECORD - 201B
*
00032 066646R        LDB M201
00033 044000         ADB A

*
*  RETURN
*
00034 126021R        JMP RANGE,I
```

```
*
*
*   ERRR -  AT THIS POINT WE REPORT ERROR THRU
*
*             "ERR0"    --- "RN GE"
*
*
 00035 062760R ERRR  LDA =ARN
 00036 066761R       LDB =AGE
 00037 016003X       JSB ERR0
 00040 016005X       JSB  PAUS
 00041 002400        CLA

*
*   RETURN
*
 00042 126007R       JMP PUT,I


*
*
*  OVFLW - THIS HANDLES THE CASE WHEN SOMEONE
*          TRIES TO PUT SOMETHING INTO AN INDEX
*          THAT IS NOT IN THE RECORD   IN THIS CASE
*          WE PUT OUT CURRENT RECORD AND START A
*          NEW STRING, AND RESTART THE STRING
*          STARTING AT LAST (X,Y)
*

*
*   1ST CHECK FLAG TO SEE IF WE MUST INVRT
*
 00043 162003R OVFLW LDA FLAG,I
 00044 002003        SZA,RSS
*
*   NO NEED TO INVRT STRING
*
 00045 026050R       JMP O.2

*
*   CALL INVRT INDIRECTLY THRU "IVRTA"
*
 00046 016101R       JSB IVRTA
 00047 000050R       DEF *+1
```

```
*
*   CLEAR OUT RECORD BUFFER
*
*   GET NEXT ADDRESS OF 'ITAG' & CHECK IF
*   'ITAG' > 'MAXT'
*
*   SET 'ITAG' IF OK
*
 00050 016332R 0 2    JSB CLEAR
 00051 000052R        DEF *+1

 00052 162005R        LDA ITAG,I
 00053 136005R        ISZ ITAG,I
 00054 166006R        LDB MAXT,I
 00055 007004         CMB,INB
 00056 044000         ADB A
 00057 006021         SSB,RSS
*
*   ITAG IS TOO BIG ( > MAXT )
*
*   REPORT OVERFLOW
*
 00060 026035R        JMP ERRP

*
*
*   SET  TAGS(ITAG) = I
*
*
 00061 042004R        ADA TAGS
 00062 066110R        LDB BASI
 00063 176002R        STB I,I

 00064 006004         INB
 00065 174000         STB A,I

*
*
*   NOW RESTART STRING BY MOVING LAST TWO IN BUFF
*
*   TO THE START OF THE BUFFER
*
*   BUT 1ST SET   -- LIST(I)
*
 00066 006004         INB
```

PAGE 0011 #01 , RANGE , ERRR , OVFLW >>

```
0288  00067 006004         INB
0289  00070 076113R        STB BUFF-1

0291  00071 062310R        LDA BUFF+124
0292  00072 072114R        STA BUFF
0293  00073 062311R        LDA BUFF+125
0294  00074 072115R        STA BUFF+1

0296*
0297*
0298*   'IND' = 'IND' + 4
0299*
0300  00075 162001R        LDA IND,I
0301  00076 042645R        ADA FOUR
0302  00077 172001R        STA IND,I


0304*
0305*    TRY AGAIN
0306*
0307  00100 026012R        JMP PUT+3
```

PAGE 0012 #01 << RANGE / ERRP / OVFLW >>

```
*
*
*   IVRTA   - MAKES A CALL TO "INVRT" BUT USES A LOCAL
*             BUFFER 'BUFF' TO CALL WITH
*
 00101 000000  IVRTA NOP
 00102 036101R       ISZ IVRTA

 00103 016004X       JSB INVRT
 00104 000107R       DEF *+3
 00105 000113R       DEF BUFF-1
 00106 000110R       DEF BASI

*
*   RETURN
*
 00107 126101R       JMP IVRTA,I
*



*
*
*   STORAGE USED FOR RECORD STRUCTURE USED
*
*   IN 'PUT' & 'LIST'
*
*
 00110 000002  BASI  OCT 2
 00111 000111R BASAD DEF *
 00112 000000        OCT 0
 00113 000000        OCT 0
 00114 000000  BUFF  BSS 128
```

PAGE 0013 #01 << FUNCTION LIST(I) >>

```
*
*
*   FUNCTION LIST  -- THIS ROUTINE SIMULATES AN
*                     ARRAY ACCESS OF THE FORM
*
*            IX = LIST(I)
*
*     LIST RETURNS VALUE OF "ITH" ELEMENT OF ARRAY
*

00314 000000  IL    BSS 1
00315 000000  LIST  NOP

00316 016001X       JSB  ENTR
00317 000314R       DEF IL

*
*   SET UP COMMON EXIT IN 'PUT'
*
00320 062315R       LDA LIST
00321 072007R       STA PUT

*
*   SET INDEX TO IL
*
00322 162314R       LDA IL,I
00323 172001R       STA IND,I

*
*   CHECK THE RANGE OF INDEX
*
00324 016021R       JSB RANGE

00325 006021        SSB,RSS

*
*   OUT-OF-RANGE
*
00326 026035R       JMP ERRP
```

```
*
*   A <= VALUE OF LIST(I)
*
00327 042111R        ADA BASAD
00330 160000         LDA A,I

*
*   RETURN
*
00331 126007R        JMP PUT,I
```

```
*
*   SUBROUTINE CLEAR -- TRANSFERS STRING FROM BUFF
*                       INTO SECTOR BUFFER 'SCTR'
*
*

      00332 000000  CLEAR NOP
      00333 036332R       ISZ CLEAR

*
*
*   CALCULATE WORD COUNT IN STRING-BUFFER ( BUFF )
*
      00334 062113R       LDA BUFF-1
      00335 070001        STA B
      00336 006004        INB
      00337 003000        CMA
      00340 042110R       ADA BASI
*
*
*   IF A < 0 SOMETHING TO GO INTO SECTOR
*
      00341 002021        SSA,RSS
*
*   QUICK RETURN
*
      00342 136332R       JMP CLEAR,I
*
*   UPDATE NEW BASE INDEX
*
*   SAVE -TIVE WORD COUNT
*
      00343 076110R       STB BASI
      00344 066111R       LDB BASAD
      00345 006004        INB
      00346 072363R       STA CNT
      00347 076364R       STB ADD

*
*   B <= CURRENT SECTOR ADDRESS
*
      00350 066566R       LDB SCTRI
```

```
PAGE 0016 #01  <<  SUBROUTINE CLEAR  -- TRANSFER STRING TO SECTOR >>

*
*  TRANSFER RECORD OVER TO SECTOR
*
*  BUFFER CHECKING EACH TIME FOR
*
*  A FULL SECTOR ( B = SCTRL )
*
 00351 056565R LOOP  CPB SCTRL
 00352 016567R       JSB EMPTY
 00353 162364R       LDA ADD,I
 00354 170001        STA B,I

 00355 006004        INB
 00356 036364R       ISZ ADD

 00357 036363R       ISZ CNT

 00360 026351R       JMP LOOP

*
*
*  SAVE CURRENT SECTOR ADDRESS
*
 00361 076566R       STB SCTRI


*
*  RETURN
*
 00362 126332R       JMP CLEAR,I


*
*  DISC SECTOR STORAGE
*
 00363 000000  CNT   NOP
 00364 000000  ADD   NOP
 00365 000000  SCTR  BSS 128
 00565 000565R SCTRL DEF *
 00566 000365R SCTRI DEF SCTR
```

```
      WRITE 'SCTR' TO DISK -- CHECKING FOR

      END OF TRACK

00567 000000  EMPTY NOP

00570 036630R       ISZ SECT
00571 062630R       LDA SECT
00572 040511        ADA MSECT

      NEED ANOTHER TRACK?

00573 002021        SSA,RSS

      REQUEST ANOTHER WORKAREA TRACK

00574 016617R       JSB GTRK

      WRITE OUT 'SCTR'.

00575 016003X       JSB EXEC
00576 000605R       DEF *+7
00577 000055        DEF  +2
00600 000055        DEF  +2
00601 000365R SCTR0 DEF SCTR
00602 000627R       DEF P128
00603 000647R       DEF TRK
00604 000630R       DEF SECT


      SET 'SECT()' TO 0

00605 002400        CLA
00606 066601R       LDB SCTR0

00607 170001  E.LP  STA B,I
00610 006004        INB
00611 056565R       CPB SCTRL
00612 002001        RSS
00613 026607R       JMP E.LP
```

```
*
*   SET 'SCTRI' TO START OF 'SCTR' (SCTR0)
*
 00614 066601R         LDB SCTR0
 00615 076566R         STB SCTRI
*
*   RETURN
*
 00616 126567R         JMP EMPTY,I


*
*   REQUEST A NEW TRACK FROM
*
*   THE WORKAREA
*
 00617 000000  GTRK   NOP
*
*   THIS CALLS GTTRK WHICH DOES
*
*   THE WORK
*
*   GTRK HAS TO TIDY UP SECT & TRK
*
 00620 016717R         JSB GTTRK
 00621 002400          CLA
 00622 072630R         STA SECT
 00623 066647R         LDB TRK
*
*   IF 1ST CALL SET TRACK TO TRK
*
 00624 052650R         CPA TRACK
 00625 076650R         STB TRACK
*
*   RETURN
*
 00626 126617R         JMP GTRK,I


*
*   CONSTANTS FOR DISK ACCESS
*
 00627 000200  P128   OCT 200
 00630 000027  SECT   DEC 23
```

```
0565*
0566*   SUBROUTINE CLSEF  -- CLOSE THE WORKFILE
0567*
0568*     TRANSFER LAST STRING TO 'SLTR'
0569*     & WRITE WHAT IS STILL IN 'SLTR'
0570*     (OF THE LAST PORTION OF THE
0571*       BUFFER 'SLTR' TO THE DISK)
0572*

0574  00631 000000  CLSEF NOP
0575  00632 036631R       ISZ CLSEF

0577  00633 016372R       JSB CLEAR
0578  00634 000635R       DEF *+1
0579  00635 016567R       JSB EMPTY

0581*
0582*   INITIALIZE 'SECT' & 'BUFF-1' TO 0
0583*
0584*   & BASI TO 2
0585*
0586  00636 002400        CLA
0587  00637 072630R       STA SECT
0588  00640 072113R       STA BUFF-1
0589  00641 002004        INA
0590  00642 002004        INA
0591  00643 072110R       STA BASI

0593*
0594*   RETURN
0595*
0596  00644 126631R       JMP CLSEF,I

0598*
0599*   CONSTANTS
0600*
0601  00645 000004  FOUR  OCT    4
0602  00646 177577  M201  OCT -201
```

```
*
*   SUBROUTINE SWAP1 -- SWAP HI-CORE FOR "SWAP2"
*                        CALL "SWAP2"
*
*
      00647 000000  TRK    BSS 1
      00650 000000  TRACK  OCT 0
      00651 000000  TGS    BSS 1
      00652 000000  ITG    BSS 1
      00653 000000  IMAX   BSS 1

      00654 000000  SWAP1  NOP

      00655 016001X        JSB  ENTR
      00656 000651R        DEF TGS

*
*   CALCULATE CORE SIZE ABOVE 'PP'
*
*   FROM PARTITION POINT (PP) & LWAM
*
*   (LAST-WORD-AVAILABLE(USER)-MEMORY)
*
      00657 062757R        LDA PP
      00660 003004         CMA,INA
      00661 040100         ADA LWAM
      00662 072753R        STA BFSZ

*
*   CONFIGURE 'RWHI' ROUTINE FOR A WRITE
*   RWRCD = 2
*
*   GET A TRACK ON DISK
*
      00663 036750R        ISZ RWRCD
      00664 016717R        JSB GTTRK
```

```
*
*   WRITE HI-CORE TO WORK AREA
*
 00665 016705R        JSB RWHI

*
*   READ OVERLAY ("SWAP2") INTO HI-CORE
*
*   FROM BD-FILE CALLED "SWAP2"
*
 00666 016003X        JSB EXEC
 00667 000676R        DEF *+7
 00670 000752R        DEF P14
 00671 000055         DEF  +2
 00672 100757R        DEF SWAP2,I
 00673 000753R        DEF BFSZ
 00674 000754R        DEF FNAME
 00675 000751R        DEF RSECT
*

*
*   CALL SWAP2
*
 00676 116757R        JSB SWAP2,I
 00677 000647R        DEF TRY
*

*
*   UPON RETURN FROM "SWAP2"
*
*   - RESET HI-CORE (FROM WORK AREA OF DISK)
*
*   - DEALLOCATE WORK AREA ON DISK
*
 00700 002404         CLA,INA
 00701 072750R        STA RWPCD

 00702 016705R        JSB RWHI

 00703 016733R        JSB RESET

*
*   RETURN
*
 00704 126654R        JMP SWAP1,I
```

```
*
*   RWHI - WRITES/READS HI-CORE
*            TO    /FROM WORK AREA ON DISK
*
*            DEPENDING ON 'RWRCD'
*
 00705 000000  RWHI  NOP
 00706 016003X       JSB EXEC
 00707 000716R       DEF *+7
 00710 000750R       DEF RWRCD
 00711 000055        DEF  +2
 00712 100757R       DEF SWAP2,I
 00713 000753R       DEF BFST
 00714 000647R       DEF TRK
 00715 000751R       DEF RSECT
*
 00716 126705R       JMP RWHI,I
*
*   REQUEST A TRACK FROM WORK AREA
*
 00717 000000  GTTRK NOP
 00720 016003X       JSB EXEC
 00721 000727R       DEF *+6
 00722 000057        DEF  +4
 00723 000730R       DEF TCONS
 00724 000647R       DEF TRK
 00725 000731R       DEF LUN
 00726 000732R       DEF S/TRK

*
*   RETURN
*
 00727 126717R       JMP GTTRK,I
*
*   CONSTANTS FOR REQUESTING TRACKS
*
 00730 100001  TCONS OCT 100001
 00731 000000  LUN   NOP
 00732 000000  S/TRK NOP
```

```
PAGE 0023 #01  << SUBROUTINE SWAP1 -- SWAP HI-CORE FOR "SWAP2" >>

*
*  RESET --- SECT TO #(SECTORS)-1
*            - DEALLOCATE WORK-AREA
*
 00733 000000  RESET NOP
 00734 060511        LDA MSECT
 00735 003000        CMA
 00736 072630R       STA SECT

 00737 002400        CLA
 00740 072650R       STA TRACK

*
*   SET (PTRK) TO 0
*
 00741 066747R       LDB PTRK
 00742 016003X       JSB EXEC
 00743 000745R       DEF *+2
 00744 000746R       DEF M19

*
*  RETURN
*
 00745 126733R       JMP RESET,I
*
 00746 177755  M19   DEC -19
 00747 000267  RTRK  OCT 267
```

```
0756*
0757*   STORAGE REQUIRED IN SWAP1
0758*
0759  00750 000001   PWRCD OCT 1
0760  00751 000000   RSECT OCT 0
0761  00752 000016   P14   DEC 14
0762  00753 000000   BFSZ  NOP
0763  00754 051527   FNAME ASC 3,SWAP2
      00755 040520
      00756 031040
0764  00757 024000   SWAP2 OCT 24000
0765  00757          PP    EQU SWAP2
0766*
0767*
0768*   EQUIVALENCES USED
0769*
0770*

0772  00000          A     EQU 0
0773  00001          B     EQU 1
0774  00053                EQU 53B
0775  00100          LWAM  EQU 100B
0776  00511          MSECT EQU 511B
      00760 051116
      00761 043505
0777                       END
**  NO ERRORS*
```

	00774	00503	00504	00655	00698	00711	
ENTR	00076	00111	00355	00618			
PAUS	00080	00210					
=AGE		00208					
=ARN		00207					
A	00772	00145	00193	00258	00277	00386	
ADD	00471	00430	00446	00450			
B	00773	00408	00447	00516			
BASAD	00236	00143	00385	00427			
BASI	00335	00168	00273	00320	00411	00426	00591
BFSZ	00762	00631	00657	00700			
BUFF	00339	00289	00291	00292	00293	00294	00319
	00407	00588					
CLEAR	00400	00066	00251	00401	00420	00465	00577
CLSEF	00574	00067	00575	00596			
CNT	00470	00429	00452				
E.LP	00516	00520					
EMPTY	00483	00445	00529	00579			
ERR0	00077	00209					
ERRP	00207	00184	00265	00380			
EXEC	00078	00501	00652	00695	00709	00744	
FLAG	00101	00231					
FNAME	00763	00658					
FOUR	00601	00301					
GTRK	00536	00496	00556				
GTTRK	00708	00544	00640	00720			
I	00100	00274					

@IMAX	00614						
IND	00099	00172	00300	00302	00368		
INVRT	00079	00317					
ITAG	00103	00254	00255				
@ITG	00613						
IVRTA	00314	00068	00241	00315	00325		
LIST	00353	00065	00361				
LOOP	00444	00454					
LUN	00725	00714					
LWAM	00775	00630					
M19	00753	00746					
M201	00602	00192					
MAXT	00104	00256					
MSECT	00776	00487	00733				
O.2	00251	00236					
OVFLW	00231	00130					
P128	00561	00506					
PP	00765	00628					
PUT	00106	00064	00150	00216	00307	00362	00391
R14	00761	00654					
RANGE	00161	00123	00198	00373			
RESET	00732	00681	00751				
RSECT	00760	00659	00702				
RTRK	00754	00743					
RWHI	00694	00645	00679	00704			
RWRCD	00759	00639	00677	00697			
S/TRK	00726	00715					
SCTR	00472	00474	00505				

SCTR0	00505	00514	00524			
SCTRI	00474	00435	00460	00525		
SCTPL	00473	00444	00518			
SECT	00562	00485	00486	00508	00546	00587	00735
SWAP1	00616	00069	00686				
SWAP2	00764	00656	00665	00699	00765		
THGS	00103	00272					
TCONS	00724	00712					
TGS	00612	00619					
TRACK	00611	00551	00552	00738			
TRK	00610	00507	00547	00666	00701	00713	
VAL	00098	00112	00144				

PAGE 0001

```
0001                        ASMB,L,R,T,C
 SETHI  R 000000
 SWAP2  R 014000
 EXEC   X 000001
 PRINT  X 000002
 PFSZ   R 000025
 R15    R 000026
 LWAM     000100
 FNAME  R 000027
 LOOP   R 014045
 RESET  R 014101
 LPRST  R 014105
 SAVE   R 014115
 LPSV   R 014122
 ADDR   R 014131
 COUNT  R 014132
 LIST   R 014133
 IMAX   R 014134
 ITAG   R 014135
 N19    R 014136
 RWRCD  R 014137
 PP     R 014140
 SIZE   R 014141
 TAGS   R 014142
 TRACK  R 014143
 TRK    R 014144
 RWLO   R 014145
 A        000000
 B        000001
 FWAM     000254
 UBFWA    000255
 N100   R 014157
 RSECT    000053
          000053
**  NO ERRORS*
```

```
PAGE 0002 #01  << SETHI -- CONFIGURE & STORE HI-CORE -> SWAP2 >>

                      ASMB,L,R,T,C
      00000                 NAM SETHI,3

*
*
*    MAIN PROGRAM "SETHI" & SUBROUTINE "SWAP2"
*
*
*    WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975
*
*    BY C  A. BRYCE, FOR A MSC  PROJECT
*
*
*    "SETHI" IS A MAIN PROGRAM WHICH ALLOWS THE
*               SAVING OF A CONFIGURED VERSION
*               OF "SWAP2" INTO A DATA FILE
*               CALLED "SWAP2"   THIS IS PART
*               OF THE SWAPPING PROCESS IN THE
*               PLOTTER PACKAGE ( VERSION II )
*
*
*    METHOD
*
*    "SETHI" IS A SIMPLE PROGRAM WHICH CONFIGURES
*    HI-CORE, SO THAT IT CAN BE COPIED TO A DATA
*    FILE
*    THE CORE TO BE SAVED IS THAT CORE PAST THE
*    POINT AT WHICH CORE IS TO BE PARTITIONED
*    ( CURRENTLY THE PARTITION POINT IS 24000B )
*    TO CONFIGURE THIS CORE, A COPY OF THE BASE-
*    PAGE LINKS MUST BE MADE AT THE EXTREME END
*    OF HI-CORE.  THEN AND ONLY THEN CAN AN "EXEC"
*    CALL BE MADE TO WRITE HI-CORE TO DISK
*
*
```

```
*
*   KEY VARIABLES
*
*   PP       - PARTITION POINT WHERE HI-CORE
*              BEGINS ( ADDRESS OF "SWAP2" )
*
*   LWAM     - A SYSTEM CONSTANT - ADDRESS OF
*              THE LAST WORD AVAILABLE IN
*              MEMORY FOR THE USER
*
*   BFSZ     - BUFFER SIZE. THIS IS CALCULATED
*              BY DETERMINING THE NUMBER OF WORDS
*              ABOVE THE PARTITION POINT   IT
*              IS THEN USED IN WRITING HI-CORE TO
*              DISK
*
*   N100     - CONSTANT  - 100B
*
*


*
*
*   DECLARE ENTRY POINTS "SETHI" & "SWAP2"
*
                          ENT SETHI
                          ENT SWAP2
*
*   DECLARE EXTERNALS "EXEC" & "PRINT"
*
                          EXT EXEC
                          EXT PRINT
```

```
PAGE 0004 #01  << SETHI -- CONFIGURE & STORE HI-CORE -> SWAP2 >>

0070*
0071*
0072*   ENTRY POINT "SETHI"
0073*
0074*   MAIN PROGRAM
0075*
0076*

0078  00 00 000000  SETHI NOP


0080*
0081*
0082*   BFSZ = LWAM - PP
0083*
0084*   ( BUFFER-SIZE = LWAM - PARTITIONPOINT )
0085*
0086*
0087  00001 062140R        LDA PP
0088  00002 003004         CMA,INA
0089  00003 040100         ADA LWAM

0091  00004 072025R        STA BFSZ
0092*


0094*
0095*     WE SAVE LINK AREA AT LWAM - 200B
0096*
0097  00005 064100         LDB LWAM
0098  00006 046157R        ADB N100
0099  00007 046157R        ADB N100
0100*
0101*   'SAVE' IS A SUBROUTINE LOCATED IN "SWAP2"
0102*           THAT COPIES  100B WORDS FROM LINK AREA
0103*           TO AREA STARTING WITH ADDRESS [B-REG]
0104*
0105  00010 016115R        JSB SAVE
0106*
```

```
*
*   NOW WRITE HI-CORE TO DISK ONTO A FILE
*
*   CALLED "SWAP2"
*
*
 00011 016001X        JSB EXEC
 00012 000021R        DEF *+7
 00013 000026R        DEF R15
 00014 000055         DEF  +3
 00015 014000R        DEF SWAP2
 00016 000025R        DEF BFR2
 00017 000027R        DEF FNAME
 00020 000053         DEF RSECT
*

*
*     STOP
*
 00021 016001X        JSB EXEC
 00022 000024R        DEF *+2
 00023 000061         DEF  +6
 00024 026021R        JMP *-3
*
```

```
*
*
*   STORAGE REQUIRED FOR "SETHI" AND
*   ALSO NOT NEEDED FOR "SWAP2"
*
*   NOTE   SINCE "SETHI" WILL NOT EXIST
*          WHEN "SWAP2" IS EXECUTED  ANYTHING
*          NEEDED BY "SWAP2" MUST BE LOCAL TO
*          "SWAP2". (IE - ABOVE PP )
*
*          THE REVERSE IS NOT TRUE. AND INFACT
*          THE ROUTINE 'SAVE' IS USED BY "SETHI"
*          WHEN IT LIES ABOVE THE PARTITION
*          POINT
*
 00025 000000  BFSZ  NOP
 00026 000017  P15   DEC 15
*
*   LWAM IS A BASE PAGE CONSTANT THAT IS
*            AT ABSOLUTE ADDRESS 100B
*
 00100         LWAM  EQU 100B
 00027 051527  FNAME ASC 3,SWAP2
 00030 040520
 00031 031040
*
```

PAGE 0007 #01 << SWAP2 -- RESTORE LINKS / CALL PRINT / RETURN >>

```
*
*
*
*   "SWAP2" IS A ROUTINE THAT IS CALLED FROM "SWAP1"
*             IN THE PLOTTER PACKAGE  "PLOT" CALLS
*             "SWAP1" WHICH IN TURNS SAVES THE
*             EXISTING HI-CORE   THEN IT COPIES IN
*             "SWAP2", AND TRANSFERS EXECUTION
*             TO "SWAP2".  "SWAP2" IS EXPECTED TO
*             RESTORE IT S BASE PAGE LINK AREA AFTER
*             SAVING THE BASE PAGE LINKS OF "PLOT" &
*             "SWAP1" .
*             THEN WHEN "SWAP2" HAS RECONFIGURED ITSELF
*             IT MUST SAVE LO-CORE OUT ONTO DISK, SO THAT
*             THIS AREA CAN BE USED   AFTER IT DOES THIS
*             IT SHIFTS THE "TAGS()" ARRAY TO 1ST WORD
*             AVAILABLE IN USER MEMORY ( FWAM ), AND
*             READS "LIST()" ARRAY RIGHT IN AFTER THIS
*             IN CORE   THEN AFTER SETTING THINGS UP
*             IT CALLS "PRINT".  THEN IT HAS TO RESTORE
*             LO-CORE AND BASE PAGE LINKS SO IT CAN
*             SUCCESSFULLY RETURN
*
*
*   KEY VARIABLES
*
*   TRK     - NEXT AVAILABLE TRACK ON DISK
*
*   TRACK   - TRACK ADDRESS OF START OF
*             "LIST()" ARRAY
*
*   TAGS    - ADDRESS OF TAG ARRAY, THE TAGS
*             THAT POINT INTO "LIST()"
*
*   ITAG    - INTEGER VALUE OF # OF STRINGS
*             IN "LIST()" OR SIZE OF "TAGS()"
*
*   IMAX    - INTEGER VALUE OF LAST VALUE OF
*             "I" IN PLOT ROUTINE, OR SIZE
*             IN WORDS OF "LIST()"
*
*   FWAM    - SYSTEM CONSTANT - ADDRESS OF
*             1ST WORD AVAILABLE TO USER IN
*             MEMORY ( AT ADDRESS 254B ).
*
```

```
*   SIZE    - VALUE CALCULATED IN "SWAP2" OF
              SIZE OF LO-CORE [FWAM,PP]
*
*   UBFWA   - SYSTEM CONSTANT - ADDRESS OF 1ST
*             WORD IN LINK AREA
*
*   LIST    - ADDRESS OF WHERE "LIST(.)" AREA
*             IS LOADED INTO LO-CORE FROM DISK
*
*   COUNT, ADDR
*                - COUNTERS AND INDEX USED IN "SWAP2"
*
*

*
*
*   FORCE "SWAP2" TO BE LOADED AT 'FWAM'+14000B
*
*   SINCE 'FWAM' = 10000B
*
*     "SWAP2" IS LOADED AT 24000B
*
    14000                  ORG SETHI+14000B

*
*     THIS IS WHERE SWAP2 IS LOADED
*
*

    14000 000000  SWAP2 NOP
*
*
*   GET ADDRESS OF ARGUMENT LIST [B-REG]
*
*   THEN INCREMENT FOR PROPER RETURN
*
*
    14001 166000R         LDB SWAP2,I
    14002 036000R         ISZ SWAP2
```

```
*
*
*  GET ARGUMENT LIST AND GET REQUIRED VALUES
*     & ADDRESSES
*
*  TRK     - VALUE   - NEXT AVAILABLE TRACK ON DISK
*
*  TRACK   - VALUE   - TRACK ADDRESS OF START OF
*                      LIST()
*
*  TAGS    - ADDRESS - OF START OF TAGS() ARRAY
*
*  ITAG    - VALUE   - SIZE OF TAGS()
*
*  IMAX    - VALUE   - SIZE OF LIST()
*
*
14003 160001          LDA B,I
14004 002004          INA
14005 072144R         STA TRK

14006 006004          INB
14007 160001          LDA B,I
14010 072143R         STA TRACK

14011 006004          INB
14012 160001          LDA B,I
14013 072142R         STA TAGS

14014 006004          INB
14015 160001          LDA B,I
14016 160000          LDA A,I
14017 072135R         STA ITAG

14020 006004          INB
14021 160001          LDA B,I
14022 160000          LDA A,I
14023 072134R         STA IMAX
*
```

```
*
*
*   SAVE 1ST 100B WORDS IN LINK AREA
*
*   AT 'LWAM'-100B
*
*

      14024 064100         LDB LWAM
      14025 046157R        ADB N100
*
*   USE 'SAVE' AGAIN TO SAVE LINK AREA
*
      14026 016115R        JSB SAVE
*

*
*   NOW USE 'RESET' TO SET LINK AREA
*   TO CONTENTS OF 'LWAM'-200B
*   SAVE STARTING ADDRESS IN "ADDR"
*
      14027 064100         LDB LWAM
      14030 046157R        ADB N100
      14031 046157R        ADB N100
      14032 076131R        STB ADDR
*
      14033 016101R        JSB RESET
*

*
*
*   NOW CALCULATE SIZE OF LQ-CORE
*
*   SIZE = PP - FWAM.
*
*
      14034 060254         LDA FWAM
      14035 003004         CMA,INA
      14036 042140R        ADA PP
      14037 072141R        STA SIZE
*
```

```
*
*   SAVE LO-CORE ON DISK WITH A CALL
*
*   TO "RWLO"
*
*
14040 016145R        JSB RWLO
*

*
*   NOW MOVE "TAGS()" ARRAY DOWN TO 'FWAM'
*
14041 062135R        LDA ITAG
14042 003004         CMA,INA
14043 072132R        STA COUNT
14044 064254         LDB FWHM

*
14045 162142R LOOP   LDA TAGS,I
14046 170001         STA B,I
*
14047 006004         INB
14050 036142R        ISZ TAGS
14051 036132R        ISZ COUNT
14052 026045R        JMP LOOP
*

*
*   JUST FOR EXTRA PROTECTION PUT 1 WORD
*
*   OF ZERO'S BETWEEN "TAGS()" & "LIST()"
*
*
14053 002400         CLA
14054 170001         STA B,I
*
*
*   B-REG CONTAINS STARTING ADDRESS OF "LIST()"
*
*   SO SAVE IT IN "LIST"
*
*
14055 006004         INB
14056 076133R        STB LIST
```

```
PAGE 0012 #01  << SWAP2 -- RESTORE LINKS / CALL PRINT / RETURN >>

0371*
0372*   READ IN "LIST()" ARRAY FROM DISK
0373*
0374*
0375*


0377  14057 016001X         JSB EXEC
0378  14060 014067R         DEF *+7
0379  14061 000054          DEF  +1
0380  14062 000055          DEF  +2
0381  14063 114133R         DEF LIST,I
0382  14064 014134R         DEF IMAX
0383  14065 014143R         DEF TRACK
0384  14066 000053          DEF RSECT

0386*
0387*
0388*   CALL PRINT(LIST,TAGS,ITAG)
0389*
0390*   NOTE - 'LIST' IS ADDRESS - SO INDIRECT
0391*          'FWAM' IS ADDRESS WHERE TAGS() IS NOW
0392*
0393*
0394*
0395  14067 016002X         JSB PRINT
0396  14070 014074R         DEF *+4
0397  14071 114133R         DEF LIST,I
0398  14072 100254          DEF FWAM,I
0399  14073 014135R         DEF ITAG
0400*


0402*
0403*     NOW WE RESET LO-CORE
0404*
0405*   1ST SET 'RWRCD' TO 1 FOR
0406*   A READ REQUEST
0407*
0408*
0409  14074 002404          CLA,INA
0410  14075 072137R         STA RWRCD
```

```
*
*   READ LO-CORE BACK IN
*
 14076 016145R        JSB RWLO
*

*
*
*   PUT LINK AREA BACK AS IT WAS
*   WHEN "SWAP2" WAS CALLED
*
*
 14077 016101R        JSB RESET
*

*
*   NOW -- RETURN
*
 14100 126000R        JMP SWAP2,I
*
```

```
*
*   RESET -- COPIES A 100B WORD BUFFER
*             FROM 'ADDR' TO LINK AREA
*

14101 000000  RESET NOP
*
*   SET COUNT TO -100B
*   SET B-REG WITH 'UBFWA'
*
14102 062157R       LDA N100
14103 072132R       STA COUNT
*
14104 064255        LDB UBFWA
*

*
*   A-REG <= [ADDR]
*
14105 162131R LPRST LDA ADDR.I

*
*
*   DO A PRIVELEGED WRITE INTO PROTECTED
*   CORE ( WITHOUT ENCOUNTERING MEMORY
*            PROTECT.)
*
*   A-REG => [ B-REG ] & INB
*
14106 016001X       JSB EXEC
14107 014111R       DEF *+2
14110 014136R       DEF N19

*
*
*   NEXT 'ADDR'
*
*   FINISHED ?? [COUNT]
*
14111 036131R       ISZ ADDR
14112 036132R       ISZ COUNT
14113 026105R       JMP LPRST
*
14114 126101R       JMP RESET,I
*
```

```
0479*
0480*   SAVE -- DOES THE REVERSE OF 'RESET'
0481*           BUT HAS NO PROBLEM READING FROM
0482*           PROTECTED CODE
0483*

0485  14115 000000  SAVE  NOP

0487*
0488*
0489*   SET COUNT TO -100B &
0490*    'ADDR' TO 'UBFWA'
0491*
0492  14116 062157R        LDA N100
0493  14117 072132R        STA COUNT
0494*
0495  14120 060255         LDA UBFWA
0496  14121 072131R        STA ADDR
0497*
0498*     B SET BEFORE CALL TO THIS ROUTINE
0499*
0500*
0501*   A-REG <= [ADDR]
0502*
0503  14122 162131R LPSV   LDA ADDR,I
0504*
0505*   A-REG => [ B-REG ]
0506*
0507  14123 170001         STA B,I

0509*
0510*   NEXT 'ADDR', 'B-REG'
0511*
0512*   DONE ?? (COUNT)
0513*
0514  14124 036131R        ISZ ADDR
0515  14125 034001         ISZ B
0516  14126 036132R        ISZ COUNT
0517  14127 026122R        JMP LPSV

0519*
0520*   RETURN
0521*
0522  14130 126115R        JMP SAVE,I
0523*
```

```
PAGE 0016 #01  << SWAP2 -- RESTORE LINKS / CALL PRINT / RETURN >>

*
*
*   STORAGE FOR RESET & SAVE
*
14131 000000  ADDR  NOP
14132 000000  COUNT NOP
14133 000000  LIST  NOP
14134 000000  IMAX  NOP
14135 000000  ITAG  NOP
14136 177755  M19   DEC -19
14137 000002  RWRCD OCT 2
14140 014000R PP    DEF SWAP2
14141 000000  SIZE  NOP
14142 000000  TAGS  NOP
14143 000000  TRACK NOP
14144 000000  TRK   NOP

*
*
*   'RWLO'.WRITES OR READS LO-CORE
*
*   DEPENDING ON 'RWRCD'
*
*   INITIALLY 'RWRCD'=2 (WRITE)
*
14145 000000  RWLO  NOP
14146 016001X       JSB EXEC
14147 014156R       DEF *+7
14150 014137R       DEF RWRCD
14151 000055        DEF .+2
14152 100254        DEF FWAM,I
14153 014141R       DEF SIZE
14154 014144R       DEF TRK
14155 000053        DEF RSECT

14156 126145R       JMP RWLO,I
```

```
*
*
*   EQUIVALENCES
*
*   A & B REGISTERS
*
*   FWAM, UBFWA, RSECT, ,
*
00000               A      EQU 0
00001               B      EQU 1
*
00254               FWAM   EQU 254B
00255               UBFWA  EQU 255B
*
14157 177700        N100   OCT -100
00053               RSECT  EQU 53B
00053                      EQU 53B
                           END SETHI
** NO ERRORS*
```

**ORG	*****	00226					
	00578	00117	00129	00379	00380	00554	
A	00570	00275	00280				
ADDR	00529	00308	00452	00472	00496	00503	00514
B	00571	00261	00266	00270	00274	00279	00345
	00360	00507	00515				
BFSZ	00148	00091	00119				
COUNT	00530	00340	00349	00444	00473	00493	00516
EXEC	00067	00114	00127	00377	00462	00551	
FNAME	00155	00120					
FWAM	00573	00320	00341	00398	00555		
IMAX	00532	00281	00382				
ITAG	00533	00276	00338	00399			
LIST	00531	00369	00381	00397			
LOOP	00344	00350					
LPRST	00452	00474					
LPSV	00503	00517					
LWAM	00154	00089	00097	00292	00305		
N100	00576	00098	00099	00293	00306	00307	00443
	00492						
N19	00534	00464					
PP	00536	00087	00322				
PRINT	00068	00395					
R15	00149	00116					
RESET	00438	00310	00424	00476			
RSECT	00577	00121	00384	00558			
RWL	0055[illegible]	00332	0041[illegible]	00[illegible]			

SETHI	00078	00062	00226	00579			
SIZE	00537	00323	00556				
SWAP2	00233	00067	00118	00241	00242	00430	00536
TAGS	00538	00271	00344	00348			
TPACK	00539	00267	00383				
TPK	00540	00263	00557				
UBFWA	00574	00446	00495				

```
FTN4,L
      SUBROUTINE GRAPH ( XMIN, XMAX, XSCAL,
     1                   YMIN,       YSCAL,
     2                   NOFN, F )
C
C   SUBROUTINE "GRAPH"
C
C
C   WRITTEN AT MCMASTER UNIVERSITY, FEBRUARY 1975
C
C   BY C. A. BRYCE, FOR A MSC. PROJECT.
C
C
C   "GRAPH" IS A UTILITY PROVIDED IN CONJUNCTION
C              WITH A PLOTTING PACKAGE USED ON
C              AN ELECTROSTATIC PRINTER, THAT
C              PERMITS THE PLOTTING OF FUNCTIONS
C
C              THIS ROUTINE WILL GRAPH FUNCTIONS
C              FROM "XMIN" TO "XMAX" USING A
C              SCALE OF "XSCAL", LENGTHWISE
C              DOWN THE PAGE. IT WILL USE
C              "YMIN" AND TAKING "YSCAL"
C              CALCULATE A "YMAX".  IN ADDITION
C              IT WILL GRAPH UP TO 10 FUNCTIONS
C              SIMULTANEOUSLY.
C
C
C   "GRAPH" USES 4 SUBROUTINES
C
C   1) EXEC  - THE SYSTEM EXECUTIVE IS USED TO
C              GENERATE NEEDED PAGE EJECTS
C
C   2) INSRT - A ROUTINE THAT ADDS NECESSARY BITS
C              INTO OUTPUT BUFFER "LINE()"
C
C   3) BFOUT - THE ROUTINE THAT PUTS "LINE()" OUT
C              TO THE PRINTER/PLOTTER
C
C   4) F     - THIS ROUTINE MUST BE SUPPLIED BY
C              THE USER, AND PASSED BY NAME TO
C              "GRAPH" THROUGH THE EXTERNAL "F".
C              THIS ROUTINE HAS 2 ARGUMENTS,
C              THE 1ST AN INTEGER ( < 11 )
C              THAT SPECIFIES WHICH FUNCTION TO
C              EVALUATE, AND THE 2ND A REAL
```

```
C                   WHICH IS THE VALUE TO BE USED IN THE
C                   PARTICULAR FUNCTION   THE USER
C                   SUPPLIES A MAIN PROGRAM THAT CALLS
C                   "GRAPH" & THE FUNCTION "F(I,X)" THAT
C                   IS IN TURN CALLED FROM "GRAPH"
C
C
C    METHOD
C
C       AFTER CALCULATING CONVERSION FACTORS
C    "DELTX" & "DELTY", REPORTING "XMIN" USED,
C    & PUTTING OUT TOP BORDER, "GRAPH" GETS
C    DOWN TO WORK AND BASICALLY BY TAKING
C    3 CONSECUTIVE  VALUES OF THE FUNCTIONS
C    Y-LAST, Y-PRESENT, & Y-NEXT, THE ROUTINE
C    INTERPOLATES BACK AND FORWARD TO GET A
C    FAMILY OF VALUES FOR THIS FUNCTION AND
C    THIS VALUE OF X   AS UP TO 10 FUNCTIONS
C    CAN BE PUT OUT AT ONCE, THE 3 VALUES
C    ARE SAVED IN IYL( ), IYP( ), AND IYN( )
C
C       IN ADDITION AT EACH VALUE OF X
C    THE SIDE BORDERS AND POSSIBLY THE X-AXIS
C    IS PLOTTED   ALSO THERE IS A CHECK TO
C    CATCH THE OCCURRENCE OF THE Y-AXIS, AND
C    PLOT IT
C
C       WHEN "XMAX" IS REACHED, THE VALUES OF
C    "YMIN", "YMAX", "XMAX", "XSCAL",
C    & OF "YSCAL" ARE REPORTED
C
C
C    KEY VARIABLES
C
C    XMIN    - REAL VARIABLE FOR MINIMUM X
C
C    XMAX    - REAL VARIABLE FOR MAXIMUM X
C
C    XSCAL   - REAL VARIABLE FOR X-SCALE TO USE
C
C    YMIN    - REAL VARIABLE FOR MINIMUM Y
C
C    YSCAL   - REAL VARIABLE FOR Y-SCALE TO USE
C
C    NOFN    - INTEGER NUMBER OF FUNCTIONS
C
```

```
C    F     - FUNCTION EXTERNAL PASSED TO "GRAPH"
C
C
         REAL      XMIN, XMAX, XSCAL
         REAL      YMIN,       YSCAL
         INTEGER   NOFN
         EXTERNAL F
C
C
C        LOCAL VARIABLES
C
C   MAX     - INTEGER CONSTANT FOR THE NUMBER
C             OF POINTS AVAILABLE IN LINE()
C             ( = 560 )
C
C   CNFCT   - REAL CONSTANT FOR CONVERSION
C             FACTOR NEEDED TO CONVERT X & Y
C             INTO INCHES IN PLOTTER UNITS
C             ( = 72.5 )
C
C   LINE()  - INTEGER BUFFER USED TO
C             STORE FUNCTION IMAGES?
C
C   DELTX   - REAL VALUE OF STEP 1 LINE
C             IN X-DIRECTION
C
C   DELTY   - REAL VALUE OF A STEP IN
C                Y-DIRECTION
C
C   YMAX    - LOCAL REAL VALUE CALCULATED
C             FROM "YMIN" & "YSCAL"
C
C   X       - REAL VALUE OF CURRENT X
C
C   XAXIS   - INDEX OF X-AXIS IN "LINE()"
C
C   XL      - REAL VALUE OF LAST X
C
C   J       - INDEX USED TO FLAG FUNCTIONS
C
C   Y       - INDEX USED FOR "LINE()"
C
C   IYL()   - INTEGER ARRAY SAVING LAST Y
C             FOR J'TH FUNCTION
C
C   IYP()   - INTEGER ARRAY SAVING PRESENT Y
```

```
C               FOR J'TH FUNCTION
C
C     IYN( )  - INTEGER ARRAY SAVING NEXT Y
C               FOR J'TH FUNCTION
C
C     IYPJ, IYPJB, & IYPJF
C             - INTEGER TEMPORARIES USED
C               DURING INTERPOLATION
C               PROOF.J  ( IYPJ  = IYP( J ) )
C
C
        INTEGER  LINE, IYL, IYP, IYN
        DIMENSION  LINE ( 35 ), IYL ( 10 )
        DIMENSION  IYP ( 10 ), IYN ( 10 )
        INTEGER  MAX, J, XAXIS, Y, IMIN, IMAX
        INTEGER  IYPJ, IYPJB, IYPJF
        REAL     DELTX, DELTY, YMAX, X, XL
C
C     INITIALIZE DATA
C
        DATA  MAX          500
        DATA  CNFCT       72.5
C
C
C*****************************************
C
C
C     START OF SUBROUTINE "GRAPH"
C
C
C     FIRST - CHECK TO SEE IF # OF FUNCTIONS IS
C             IN CORRECT RANGE -- [ 1, 10 ]
C             REPLY WITH APPROPRIATE ERROR MESSAGE
C             IF NECESSARY AND RETURN
C
C
        IF ( NOFN GT 0 AND NOFN LE 10 )  GO TO 10
C
        WRITE ( 6, 99 )  NOFN
  99    FORMAT ( 25H ERROR IN CALL TO "PRINT" ,
     1           25H # OF FUNCTIONS IN CALL = , I5 )
        RETURN
C
C
C*****************************************
C
```

```
C
C   FROM "XSCAL" & "YSCAL" DETERMINE THE
C   NECESSARY VALUES FOR "DELTX" & "DELTY"
C   RESPECTIVELY
C
C   ALSO DETERMINE THE PROPER INDEX FOR
C   "XAXIS"  IF "XAXIS" NOT IN [ 1, "MAX" ]
C   SET "XAXIS" TO 1, AND HIDE IT ON LEFT
C   BORDER
C
 10    DELTX = 1.0 / ( XSCAL + CNFCT )
       DELTY = 1.0 / ( YSCAL + CNFCT )
       XAXIS = 1
       Y = - IFIX ( YMIN / DELTY )
       IF ( Y.GE.1.AND.Y.LE.MAX ) XAXIS = Y
C
C   SKIP TO TOP OF NEXT PAGE WITH
C   EXECUTIVE CALL
C
C   THEN REPORT "XMIN" VALUE USED &
C   PUT OUT TOP BORDER
C
       CALL EXEC ( 3, 1110B, 63 )
C
       WRITE ( 8, 100 )  XMIN
 100   FORMAT ( 30X, 7HXMIN = , G10.3, / )
C
       DO 15  Y = 1, 35
 15    LINE ( Y ) = 177777B
       CALL BFOUT ( LINE )
C
C
C   INITIALIZE  "IYL()" & "IYP()" WITH
C       VALUES OF FUNCTIONS FOR X EQUAL TO
C       "XMIN"-1 & TO "XMIN" RESPECTIVELY
C
C
      XL = XMIN - DELTX
      X  = XMIN
C
       DO 20  J = 1, NOFN
       IYL(J) = IFIX( (F(J,XL)-YMIN)/DELTY )
 20    IYP(J) = IFIX( (F(J,X)-YMIN)/DELTY )
C
C
C   GET NEXT VALUE OF X, AND CALCULATE
```

```
C        "IYN( )" FOR EACH FUNCTION
C
C
 30    X = X + DELTX
C
       DO 31  J = 1, NOFN
 31    IYN(J) = IFI.. (F(J,X)-YMIN) DELTY )
C
C
C    IS THIS WHERE THE Y-AXIS BELONGS ???
C
C    IF  -  IS SITUATION --  X-DELTX   0 0 . X
C        -  X HAS CHANGED SIGN WHICH MEANS THAT
C              X + X-DELTX  IS  -TIVE
C
C    IF SO PUT OUT Y-AXIS
C
       IF ( X*(X-DELTX) GE 0 0 )  GO TO 33
C
       DO 32  Y = 1, 35
 32    LINE ( Y ) = 177777B
       GO TO 50
C
C
C*********************************************
C
C
C    FOR EACH FUNCTION PUT NEEDED POINTS
C    INTO "LINE()"
C
C    NOTE .
C      IN ORDER TO GET A CONTINUOUS GRAPH
C    FOR EACH FUNCTION,  "GRAPH" INTERPOLATES
C    BACKWARDS AND THEN FORWARDS HALFWAY.
C    AND IT IS THIS RANGE THAT IS INCLUDED
C    NOT JUST THE Y VALUE
C
C
 33    DO 40  J = 1, NOFN
       IYPJ = IYP ( J )
       IYPJB = IYPJ - (IYPJ-IYL(J))/2
       IYPJF = IYPJ + (IYN(J)-IYPJ)/2
       IMIN = MIN0 ( MAX, IYPJB, IYPJ, IYPJF )
       IMAX = MAX0 ( , 1, IYPJB, IYPJ, IYPJF )
C
       IF ( IMIN.LT.1    )  IMIN = 1
```

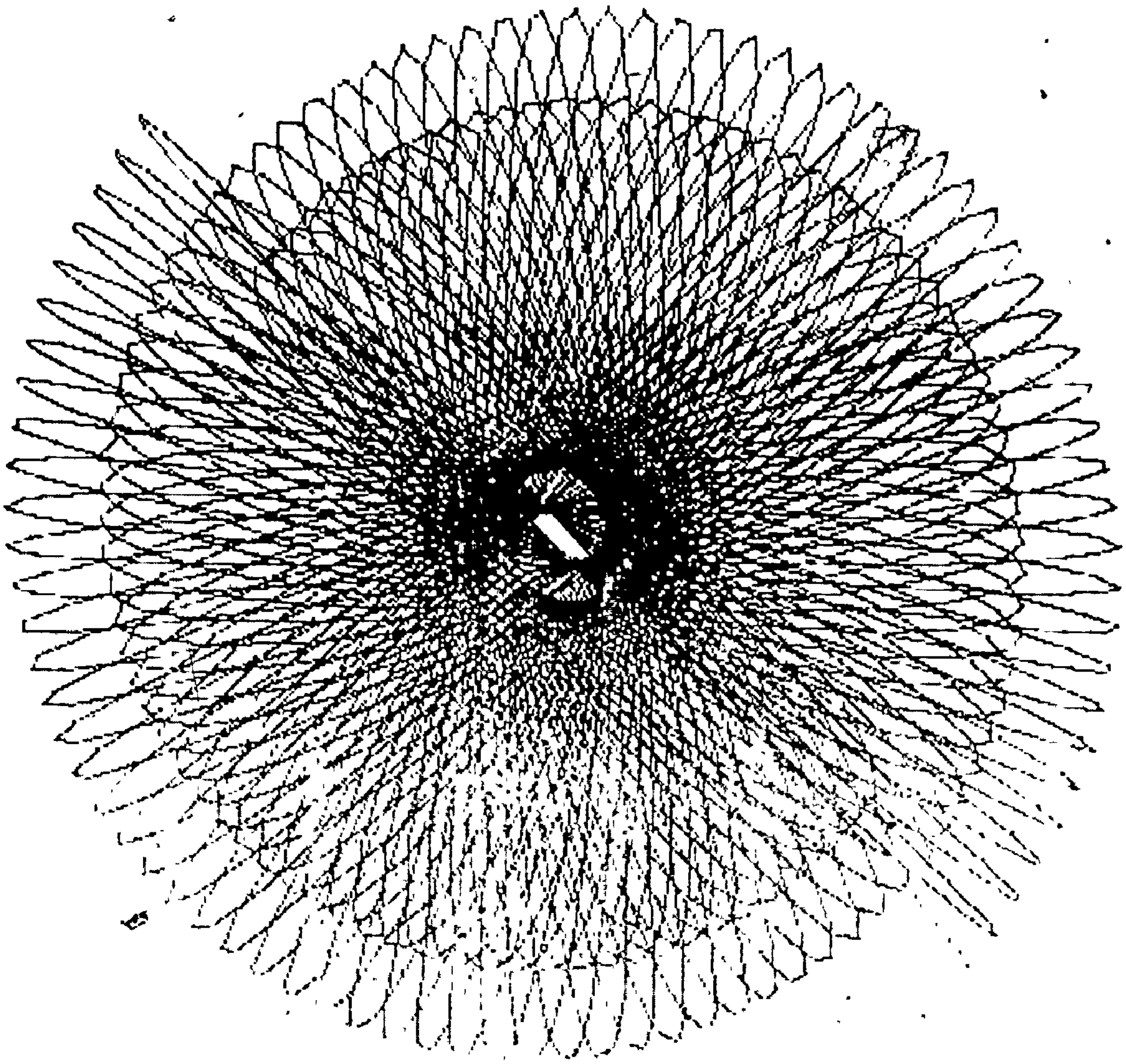
```
      IF ( IMAX GT MAX )  IMAX = MAX
C
      DO 35  Y = IMIN, IMAX )
 35   CALL INSRT ( LINE, Y )
C
 40   CONTINUE
C
C
C   NOW INCLUDE  THE X-AXIS, ALONG WITH
C      THE LEFT & RIGHT BORDERS
C
C
      CALL INSRT ( LINE, 1 )
      CALL INSRT ( LINE, XAXIS )
      CALL INSRT ( LINE, MAX )
C
C   BUFFER OUT CONTENTS OF "LINE()".
C
 50   CALL BFOUT ( LINE )
C
C
C   NOW SET  --  "IYL()" <== "IYP()"
C             &  "IYP()" <== "IYN()"
C
C   UNLESS ALL THE GRAPHING HAS BEEN
C   DONE ( X > XMAX ) PROCESS FOR
C   NEXT X  (  => 30 )
C
C
      DO 51  J = 1, NOFN
      IYL ( J ) = IYP ( J )
 51   IYP ( J ) = IYN ( J )
      IF ( X.LE.XMAX )  GO TO 30:
C
C
C   PUT OUT BOTTOM BORDER AND THEN
C
C   REPORT "YMIN", "XMAX", "YMAX",
C
C          . "XSCAL", AND "YSCAL", THAT
C
C              WAS USED TO CREATE GRAPH.
C
C   SKIP TO TOP OF NEXT PAGE AND
C   RETURN
C
```

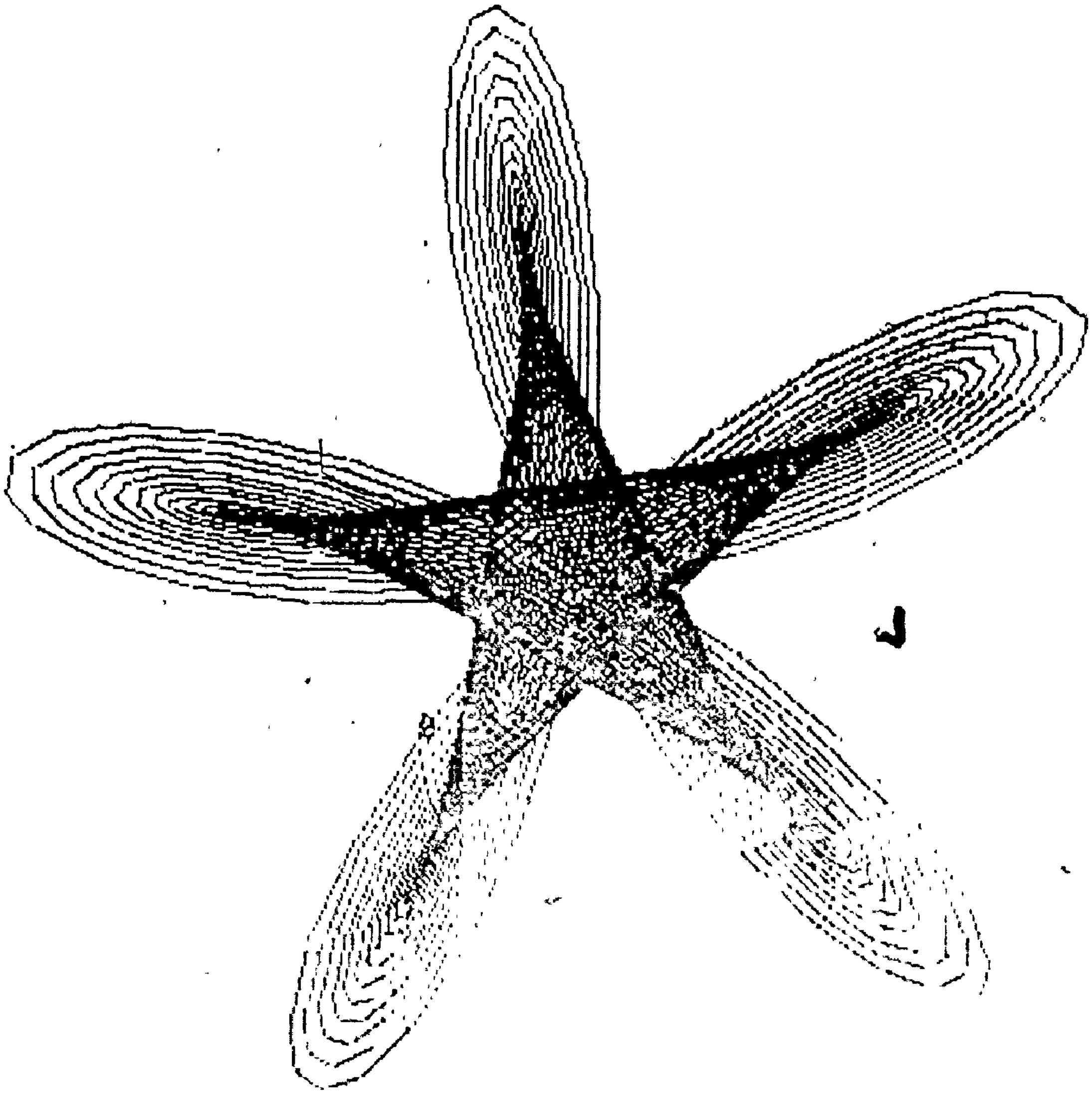
```
C
      DO 60  Y = 1, 35
 60   LINE ( Y ) = 177777B
      CALL BFOUT ( LINE )
C
      YMAX = YMIN + FLOAT(MAX)*DELTY
      WRITE ( 8, 101 )  XMIN, XMAX, YMAX
 101  FORMAT ( , 1X, 7HYMIN =  , G10 3,
     1         12X, 7HXMAX =  , G10 3,
     2         10X, 7HYMAX =  , G10 3 )
C
      WRITE ( 8, 102 )  XSCAL, YSCAL
 102  FORMAT ( 1X, /, 12H SCALE USED  ,
     1         23HIN X-DIRECTION(DOWN)   ,
     2         10H1 UNIT =   , G10 3,
     3          7HINCHES, , /, 12X,
     4         23HIN Y-DIRECTION(ACROSS)  ,
     5         10H1 UNIT =   , G10 3,
     6          7HINCHES   )
C
      CALL EXEC ( 3, 1110B, 63 )
C
      RETURN
C
C++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C   THIS IS THE END OF SUBROUTINE "GRAPH"
C
C
      END
      END$
```

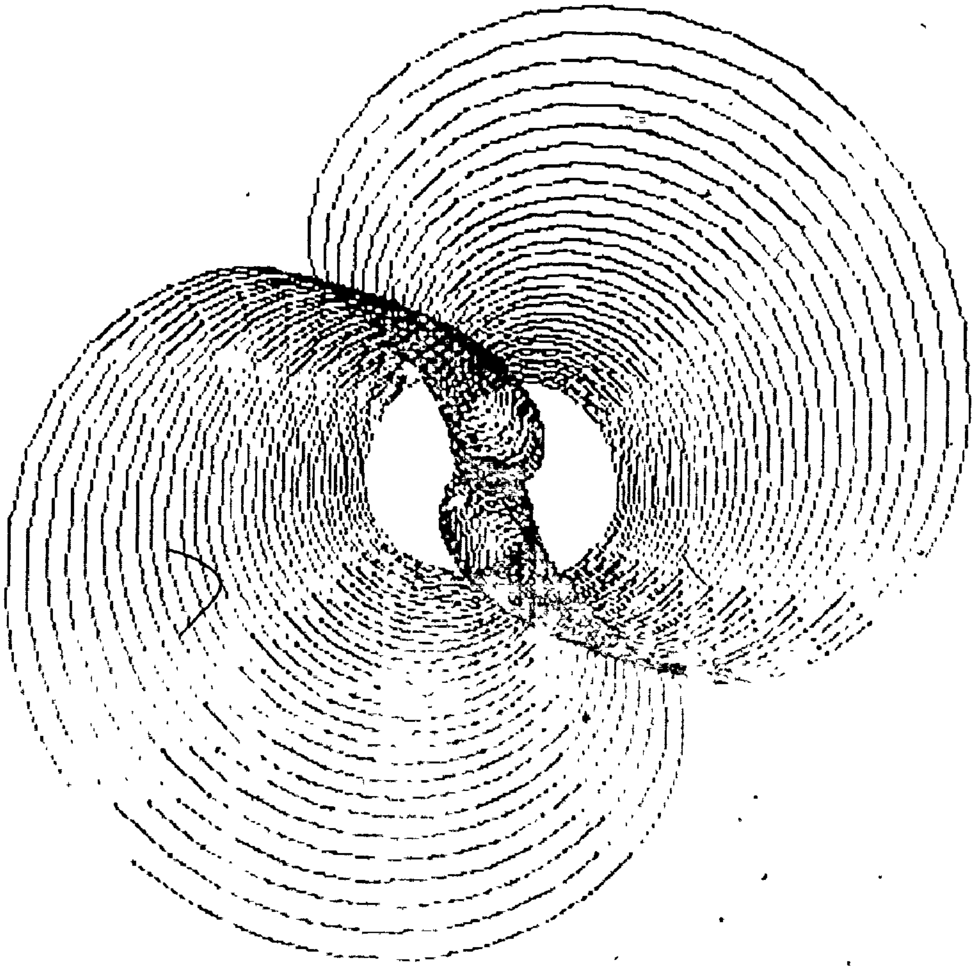
APPENDIX E

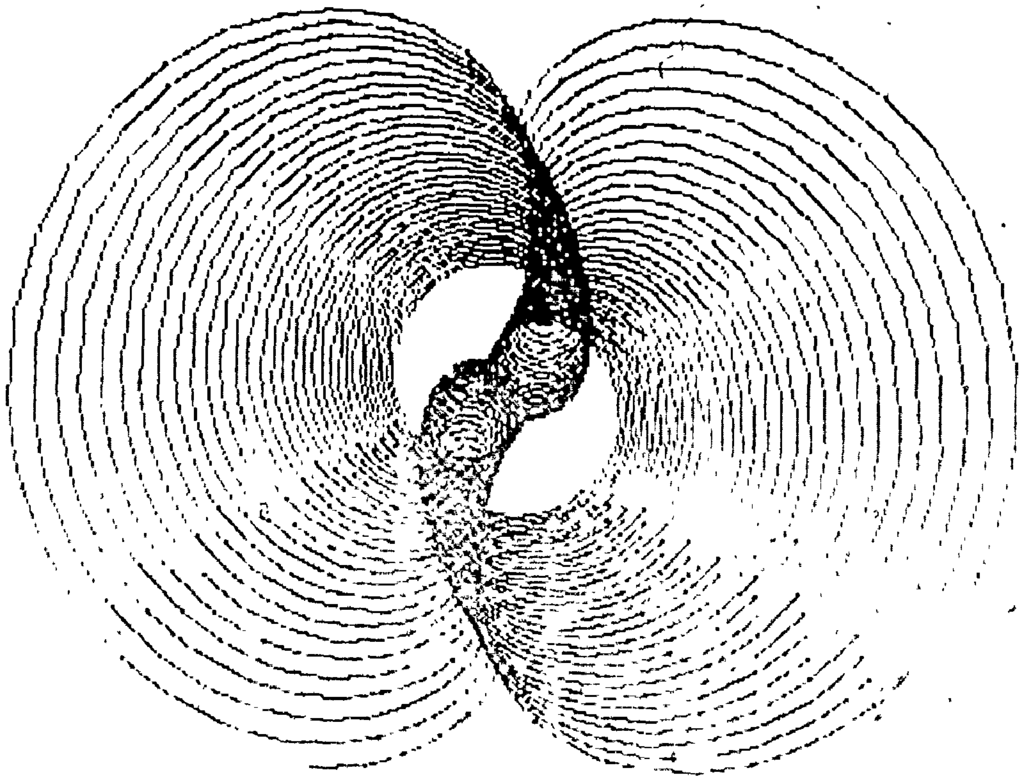
Further Examples of Plotter Output

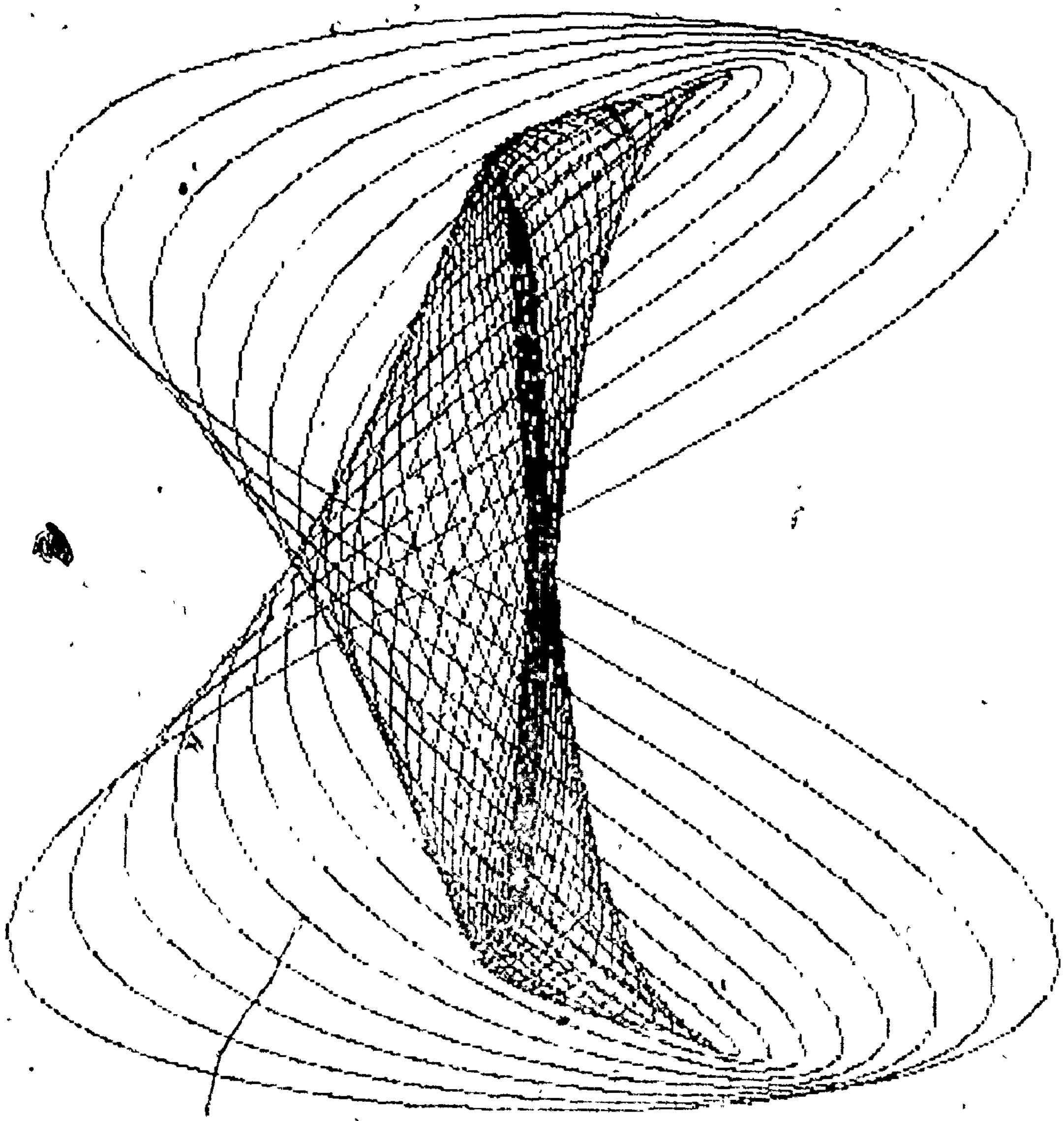
To further illustrate what the plot package is capable of at McMaster, seven more harmonographs (see 4.1.2.) are provided.

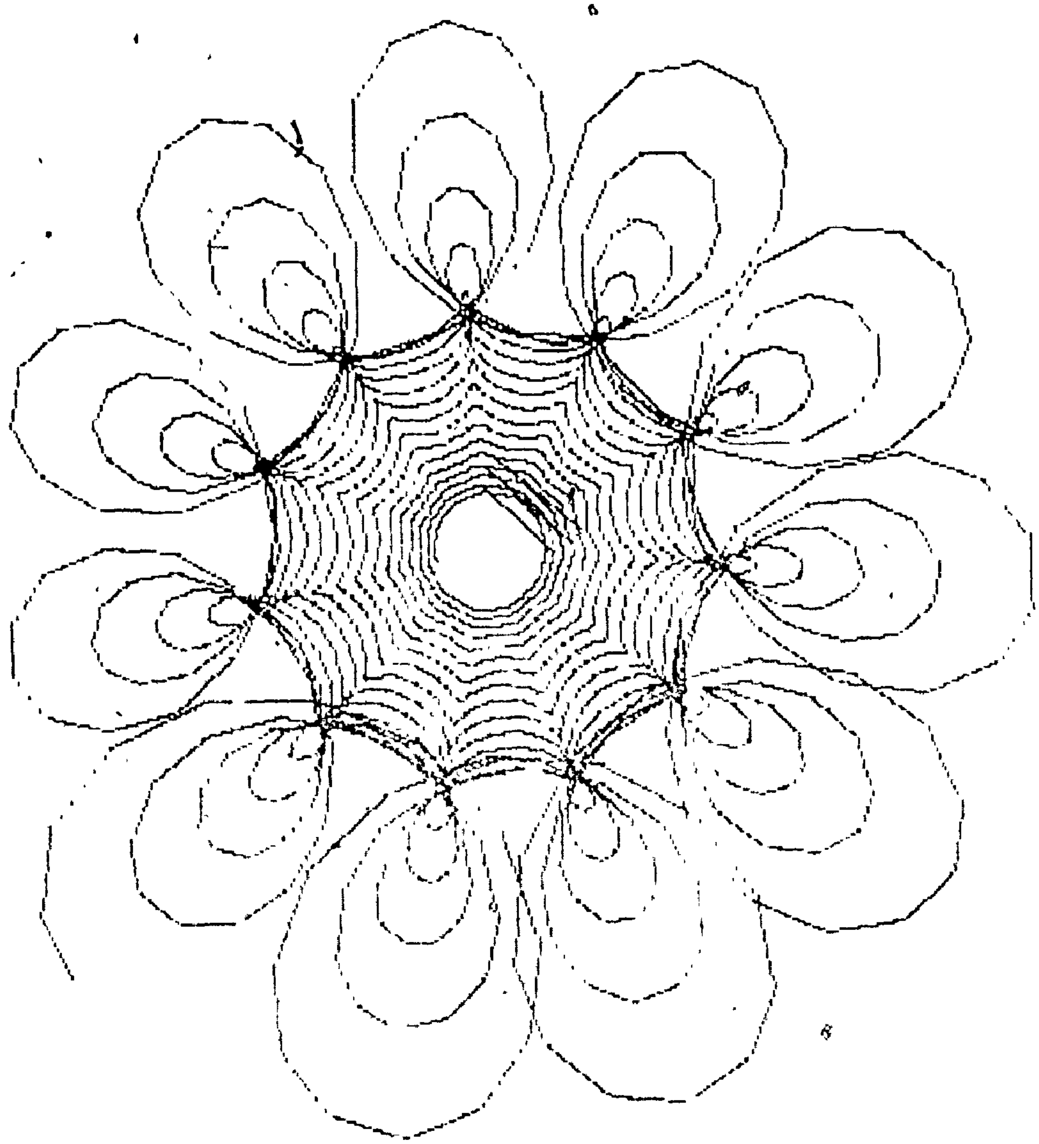












REFERENCES

1. Hewlett-Packard 2100A - Reference Manual,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 02100 - 90001)

2. Hewlett-Packard Assembler Manual,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 02116 - 9014)

3. Hewlett-Packard Fortran IV Reference Manual,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 5951-1321)

4. Moving-Head Disc. Operating System,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 02116 - 91779)

5. DOSM Operating System Listings,
   Hewlett-Packard Corporation,
   11000 Wolfe Road,
   Cupertino, California 95014.
   (HP 24225 - 6000X Rev.D)

6. Benson-Lehner Plotting System for CDC 6400
   Data Processing and Computing Centre,
   McMaster University,
   Hamilton, Ontario.
   (Publication 5.2.1.)

7. Software - Practice and Experience,
   Volume 2 (1972)
   Computer Recreations,
   Wiley Interscience.
   (Pages 293 to 297)

8. Investigations of Available Sorting Algorithms
   and their Behaviour for Different Orderings,
   A Project for the Applied Math. Dept. (1969-1970),
   McMaster University,
   Hamilton, Ontario.