STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM
REKENAFDELING

MR 73

SOME ALGOL PLOTTING PROCEDURES

J.A.Th.M. van Berckel
B.J. Mailloux

3rd EDITION

January 1970

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

Table of Contents

Foreword

Since March, 1964, a Calcomp 507 digital incremental plotter has been operational on the Electrologica X1 computer at the Mathematical Centre in Amsterdam. The following report describes the specifications of this plotter, and of a programming package developed for it. The procedures PLOT, PLOTFRAME, PLOTTEXT, PLOTAXIS, PLOTCURVE, MARKEDCURVE, SCALE, PLOTAXIS2, and PLOTPICTURE were all written in ALGOL 60 [1], and rigourously tested on the X1 in this form; the ALGOL texts of all these procedures are included in the present report. After this, the procedures PLOT, PLOTFRAME, PLOTTEXT, PLOTAXIS, PLOTCURVE, FIXPLOT, ABSFIXPLOT, and FLOPLOT were programmed in machine code for use with the MC I ALGOL system; these texts are not included in the report.

All the procedures except PLOTPICTURE may be regarded as more or less basic building blocks, whereas PLOTPICTURE is a completed "building". The "building blocks" can be used to construct extensions to the present system, such as logarithmic and polar scales, isolines, etc..

The authors wish to thank all members of the Computation Department of the Mathematical Centre for their participation in numerous fruitful discussions concerning this project. Also, we wish to thank Miss H. de Hoyer for her rapid and accurate typing of this report.

Erratum: The statement

alpha:=(phi + arctan(arg/sqrt(1 - arg ↑ 2))) × rad + 180;

in Sample Program 4 is incorrect: the phi should be changed to read phi1. However, this change makes very little difference in the drawing obtained.

1. Description of the Plotter

1.1 Introduction

The plotter consists of a drum over which paper passes from a supply to a take-up roll. A carriage, carrying a pen, runs parallel to the axis of the drum, a few millimeters from the paper. By means of special machine instructions in the X1, the drum may be rotated forward or backward by one tenth of a millimeter; similarly, the carriage, with the pen, may be moved the same distance to left or right; finally, the pen may be lowered to the surface of the paper, or lifted from it.

We may define a system of co-ordinates by supposing that we view the plotter from the front. Then, the +y direction is to the left, and -y to the right; +x is from front to back, and -x from back to front; +z is pen up, and -z is pen down.
The detailed specifications of the plotter follow

drum speed	300 increments/s
carriage speed	300 increments/s
pen up/down	10 operations/s
step size	.1 mm in x or y
step size	$.1 \times \sqrt{2}$ mm for x and y simultaneously
maximum length of paper	ca. 3650 cm (120 ft)
width of paper	ca. 30.5 cm (12 in)
width available for plotting	ca. 27.9 cm (11 in)

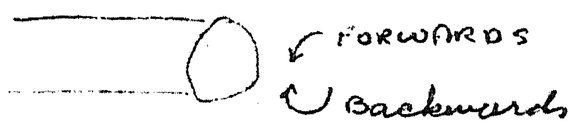
1.2 X1 Plotter Instructions

The six least significant bits of (A) or (S) can be sent to the plotter (PL), with or without inversion.

```
6Y 8 XP      +(A) ==> (PL)
7Y 8 XP      -(A) ==> (PL)
6Z 8 XP      +(S) ==> (PL)
7Z 8 XP      -(S) ==> (PL)
```

An action is defined only for the following values of the six bits reaching the plotter:

1	+x	(drum forwards)
2	-x	(drum backwards)
4	+y	(carriage left)
8	-y	(carriage right)
16	+z	(pen up)
32	-z	(pen down)
5	+x and +y	simultaneously
9	+x and -y	simultaneously
6	-x and +y	simultaneously
10	-x and -y	simultaneously



If a plotter order is given within 3 ms after another plotter order (100 ms after pen up or down), then the X1 delays the execution of the given instruction for the appropriate length of time. No interruption facilities are provided.

A special condition setting P version of the above four orders exists to determine whether the carriage is at the border of the available plotting area. If the carriage stands at the right-hand edge and an instruction to move it further right is given or at the left-hand edge and an instruction to move it further left is given, the condition register is set affirmative and no motion of the carriage takes place; in all other cases, the condition is set negative.

1.3 Manual Controls on the Plotter

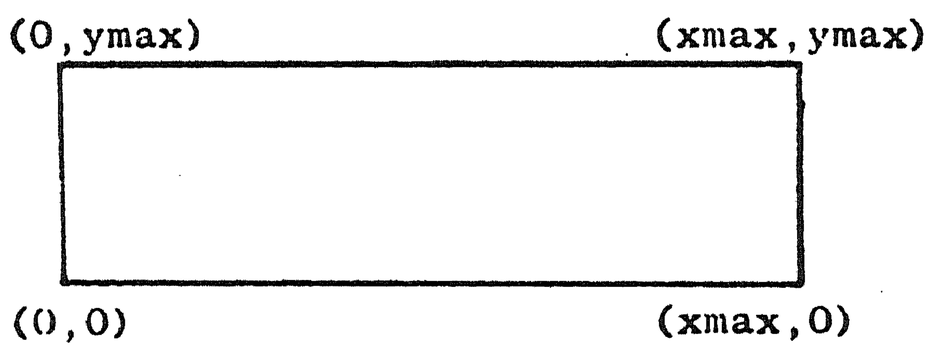
There are several knobs on the plotter to facilitate setting-up:

1. power on/off
2. drum one step forward or backward
3. carriage one step left or right
4. pen up/down
5. drum forward or backward at 100 steps/s
6. carriage left or right at 100 steps/s
7. chart drive on/off.

1.4 Scaling

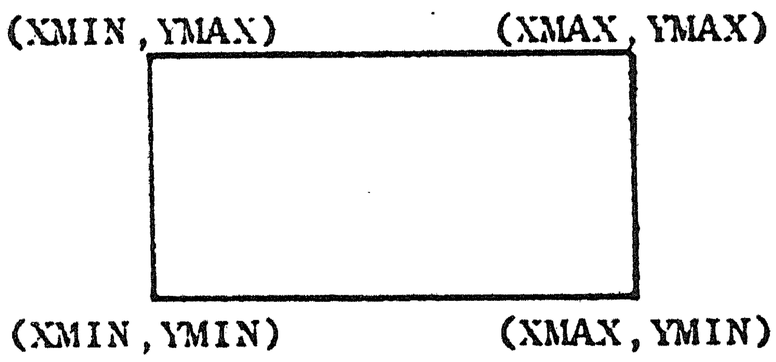
In the sequel, we will make use of certain conventions concerning scaling. For any plotting job, it is necessary to map from the units of the quantities used in the program (hereafter called "data units" or "dits") to some sort of physical "plotter units" ("plits"). We will usually use capital letters to represent variables expressed in data units, and small letters for those in plotter units.

We imagine a rectangle on the paper (the "plotter space") extending from the point (0,0) at lower left to (xmax, ymax) at upper right, thus:



The unit in this plotter space is 0.1 mm in both x and y directions.

The corresponding rectangle in the "data space" extends from (XMIN, YMIN) corresponding to (0,0) in plotter space to (XMAX, YMAX) corresponding to (xmax, ymax), thus:



Defining the scaling factors

$$Scx = (XMAX - XMIN)/xmax$$

and

$$Scy = (YMAX - YMIN)/ymax,$$

then arbitrary points (X,Y) in the data space and (x,y) in the plotter space are mapped onto one another as follows:

$$(X,Y) \Longleftrightarrow (x \times Scx + XMIN,\ y \times Scy + YMIN)$$

and

$$(x,y) \Longleftrightarrow (X - XMIN)/Scx,\ (Y - YMIN)/Scy).$$

We now proceed to the description of a number of standard procedures developed by the authors.

2. Users' Description of the Plotting Procedures

2.1 The Procedure PLOT

PLOT constitutes the nucleus of the entire set of plotter procedures; all the other procedures call upon it, either to accomplish some pen movement, etc., or to obtain information about the current pen position, scaling quantities, etc.

Within PLOT are stored the quantities: XMIN, YMIN, xmax, ymax, Scx, and Scy described above; xlast and ylast, representing the current x and y co-ordinates of the pen in plits; and the Boolean pen recording whether the pen is up (true) or down (false).

We now give the ALGOL heading of the procedure

```
real procedure PLOT(X,Y,IPEN); value X, Y, IPEN;
real X,Y; integer IPEN;
```

X and Y usually represent the co-ordinates of a point and may then be in either plits or dits. IPEN usually represents the desired vertical pen position according to a code described below.

(a) Calls to Cause Motion of the Pen ($0 < abs(IPEN) \leq 4$)

For all of these calls, the pen is moved from its current position (recorded in xlast and ylast in plits) in as straight a line as possible to the point represented by X and Y; according as IPEN is positive or negative, X and Y are in dits or plits, respectively. If the point given by X and Y does not lie within the frame, an error (no. 1-27) is signalled; if the program is then continued, the line will nevertheless be drawn to the prescribed point, unless it lies outside the dimensions of the paper.

The pen is up or down during the drawing of the line depending on the value of abs(IPEN):

if abs(IPEN) = 1, the pen is down;
2, the pen is up;
3, the vertical pen position is unaltered;
4, the pen is up, and the prescribed point may lie outside the frame in the +x direction.

In this last case (abs(IPEN) = 4), xlast and ylast are reset to zero, thus defining the origin of a new frame. If IPEN = -4 and X < 0, then the pen moves to the position (xmax + 3000, 0)- i.e. 30 cm beyond the frame.

After each of these calls, PLOT has the value zero.

(b) Calls to Obtain Information ($5 \leq$ IPEN ≤ 19)

After each of these calls, PLOT has some value which may be used by the calling program.

IPEN = 5 PLOT = the value of X (in dits) converted to plits
6 PLOT = the value of Y (in dits) converted to plits
7 PLOT = the value of X (in plits) converted to dits
8 PLOT = the value of Y (in plits) converted to dits

For the above four calls, the relevant quantity X or Y must lie within the frame or an error condition will be signalled; if the program is continued, the conversion is carried out nevertheless.

IPEN = 9 PLOT = the value of XMIN (in dits)
10 PLOT = the value of YMIN (in dits)
11 PLOT = the value of Scx
12 PLOT = the value of Scy
13 PLOT = the value of xmax (in plits)
14 PLOT = the value of ymax (in plits)
15 PLOT = the value of xlast (in plits)
16 PLOT = the value of ylast (in plits)
17 PLOT = the value of xlast converted to dits
18 PLOT = the value of ylast converted to dits
19 PLOT = +1 if the pen is up, -1 if the pen is down.

For the above 11 calls, the values of the parameters X and Y are irrelevant.

(c) Initialization Calls, etc. ($20 \leq abs(IPEN) \leq 25$)

IPEN = 20

This call is used to cause a single movement of the pen according to the value of X; the value of Y is irrelevant. The actions corresponding to the various values of X have already been given under the heading X1 Plotter Instructions. Note that the quantities xlast, ylast, and pen are not updated by this call.

abs(IPEN) = 21

This call causes XMIN to receive the value of X, and YMIN the value of Y; if IPEN = +21, the pen is raised, and the quantity pen is initialized.

IPEN = 22

This call causes Scx to receive the value of X, and Scy the value of Y.

abs(IPEN) = 23

This call causes xmax to receive the value of X, and ymax the value of Y; if IPEN = +23, then xlast and ylast are initialized, and the pen is moved to the point (0,0) in plits((XMIN, YMIN) in dits). The frame is thereby centered in the middle of the paper in the y dimension.

IPEN = 24

This call causes xlast to receive the value of X, and ylast the value of Y.

IPEN = 25

This call causes pen to be set true or false according as X is positive or not; the value of Y is irrelevant.

NOTE 1: Certain interlocks are built into the procedure to prevent the use of any quantities which are currently undefined. The initialization calls may be used at any time, but, for all other calls, (0 < abs(IPEN) $\leq$ 19), it is necessary that calls to initialize all of the quantities XMIN, YMIN, Scx, Scy, xmax, ymax, xlast, ylast, and pen have been given previously. The easiest way to accomplish this is to use PLOTFRAME.

NOTE 2: Notice carefully that after a call with IPEN = 20, the quantities xlast, ylast, and pen become undefined, and must be redefined appropriately before calls with 0 < abs(IPEN) $\leq$ 19 may be employed.

NOTE 3: Failure to observe the above points will result in an error condition (no. 1-26).

NOTE 4: A call of PLOT with an invalid value of IPEN will also result in an error signal (no. 1-25).

2.2 The Procedure PLOTFRAME

The purpose of this procedure is to assign values to the quantities XMIN, YMIN, xmax, ymax and to calculate Scx and Scy (these are stored within the procedure PLOT, q.v.). The frame is centered in the y direction on the paper, and the pen is left standing above the point (0,0) in the plotter space ((XMIN,YMIN) in data space).

We here give the ALGOL heading of the procedure, to show the input parameters

```
procedure PLOTFRAME(XMIN,YMIN,XMAX,YMAX,xmax,ymax);
value XMIN,YMIN, XMAX, YMAX, xmax, ymax;
real XMIN,YMIN,XMAX,YMAX; integer xmax,ymax;
```

xmax and ymax are given in plits, of course. If ymax is specified greater than 2750, an error (no. 1-30) is signalled, and, if the program is continued, ymax is set to 2750.

NOTE 1: PLOTFRAME only defines the frame and leaves the pen in the position described above; it does not draw any lines around the edge of the frame.

NOTE 2: Suppose we desire to leave a border around the actual plotting area (for scales, labels, etc.); that is, suppose we want a border of lb plits to the left, rb to the right, bb below, and ab above. Then, we may use the following program to define this frame:

```
Q: = (XMAX - XMIN)/xmax;
R: = (YMAX - YMIN)/ymax;
PLOTFRAME(XMIN - Q × lb, YMIN - R × bb,
          XMAX + Q × rb, YMAX + R × ab,
          xmax + lb + rb, ymax + bb + ab);
```

2.3 The Procedure PLOTTEXT

We here give only the ALGOL 60 procedure heading:

```
procedure PLOTTEXT(X, Y, angle, height, italicity, first, i, text);
value X, Y, first, i;
real X, Y, angle, height, italicity;
Boolean first;
integer i;
string text;
```

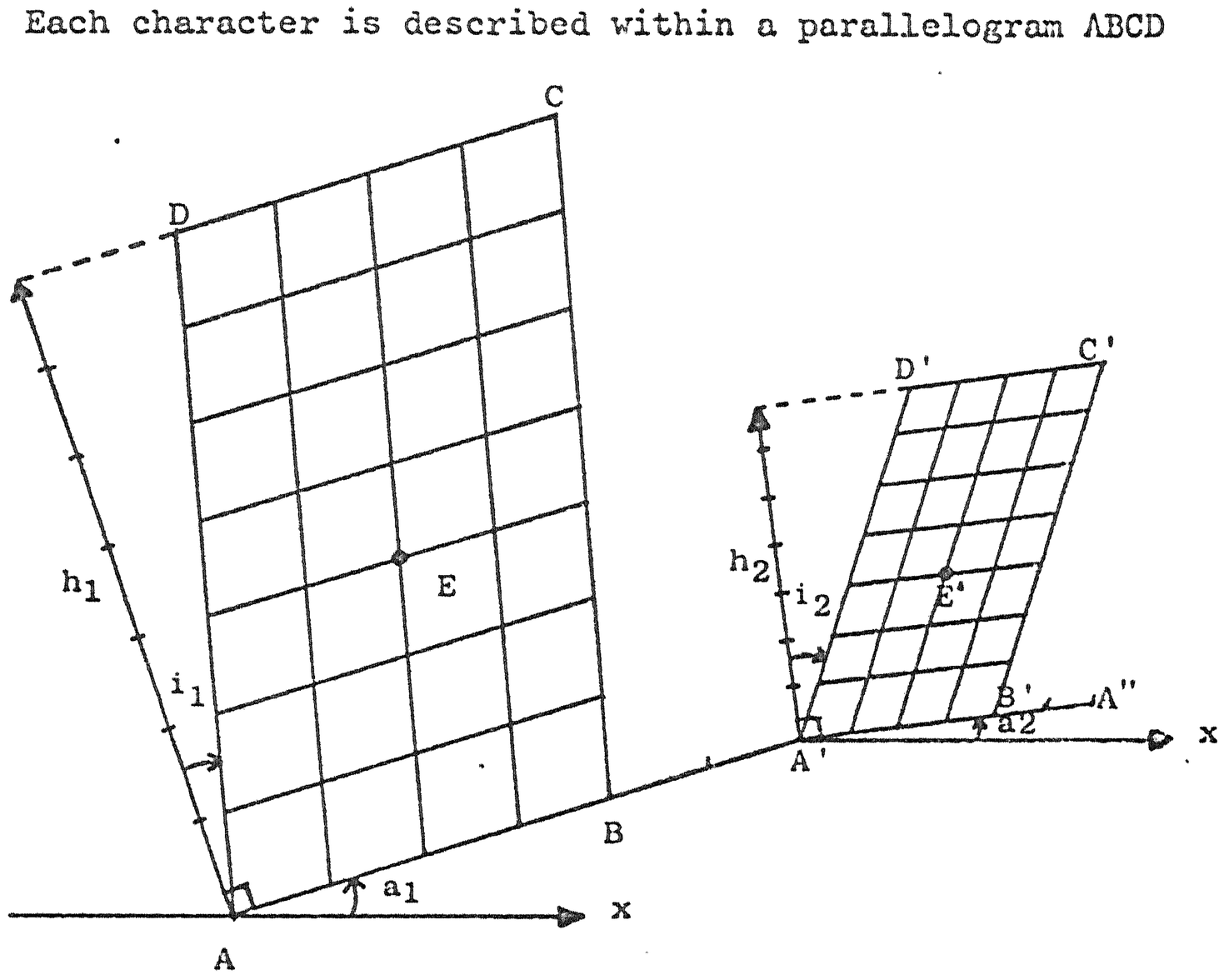
By means of the procedure PLOTTEXT, it is possible to draw a string of any ALGOL 60 basic symbols, with the exception of those delimiters represented by strings of bold-face or underlined letters. Furthermore, it is possible to precede the symbols of the (possibly empty) string by one of the special characters given below. The first nine of these characters are intended for the marking of points on curves, etc. Whether any preceeding character is drawn, and if so which one, is determined by the absolute value of the parameter i according to the following schedule:

abs(i) =		
0	}	no extra character precedes the string
1		
2	+	precedes the string .
3	×	"
4	✳	"
5	✳	"
6	Y	"
7	⧖	"
8	⧖	"
9	⊡	"
10	◈	"
11	⬡	"
12	⊓	"

The Form of the Characters

The shape of the plotted characters is determined by the parameters angle, height, and italicity in the following way:

Each character is described within a parallelogram ABCD



angle prescribes the angle a_1 (a_2), in degrees, measured counter-clockwise from the x axis;

abs(height) specifies the height h_1 (h_2) of the parallelogram, in plotter units;

italicity gives the value of the angle i_1 (i_2), in degrees, measured clockwise.

The Form of the String

The quantities height, angle, and italicity are evaluated precisely once for each character to be drawn, before the drawing of the character begins, and in the order given above (notice that these parameters do not appear in the value list, and are thus "called by name"). Thus, it is possible to vary these quantities during the plotting of the string.

After drawing each character, PLOTTEXT calculates the following "reference point"; that is, the position of the corner A' of the next parallelogram. This point lies on the continuation of AB at a distance of 6/7 h_1 from A.

The Position of the Reference Point of the First Character

This is determined by the quantities X, Y, height, first, and sign(i).

The Boolean first specifies whether a new reference point must be calculated (first ≡ true), or that the reference point calculated during the last call of PLOTTEXT, FIXPLOT, ABSFIXPLOT, or FLOPLOT is to be used (first ≡ false). For this purpose, PLOTTEXT maintains the co-ordinates of the reference point in two own variables (FIXPLOT, ABSFIXPLOT, and FLOPLOT function by calling PLOTTEXT, and do not require direct access to this information). Because of this scheme, it is possible to call any of these four procedures with first ≡ false provided that any one of them has previously been called with first ≡ true. Thus, text and numbers may follow directly after one another without the programmer having to concern himself with the calculation of the proper co-ordinates.

We consider now the case first ≡ true. Then, the sign of i determines whether the given co-ordinates X and Y specify the point A of the diagram ($i \geq 0$) or the point E($i < 0$). The case $i < 0$ is particularly important for the drawing of the special curve marking symbols described above; then X and Y do not specify the reference point, but it can easily be derived from them. The sign of height specifies whether X and Y are given in data units, and must be scaled to plotter units (height > 0), or are given directly in plotter units (height < 0).

In the case first ≡ false, the quantities X, Y, and sign(height) are completely irrelevant. The new reference point is simply derived from the last one calculated by PLOTTEXT.

For this derivation, not only the current sign of i, but also that of the previous call is of importance. If these signs are the same, then the new reference point is equal to the one previously calculated and stored by PLOTTEXT. If old i $\geq$ 0 and new i < 0, then the reference point left behind by PLOTTEXT indicates the point E of the new parallelogram. In this case, the following characters are shifted the appropriate amount down and to the left. If old i < 0 and new i $\geq$ 0, then an analogous shift up and to the right will automatically be performed.

NOTE 1: The pen is always left lowered on the paper.

NOTE 2: abs(i) > 12 or a call with first ≡ false which has not been preceded by a call with first ≡ true will lead to an error being signalled (no. 1-31).

NOTE 3: The basic symbols begin, end, own, Boolean, integer, real, array, switch, procedure, string, and label correspond with the special characters for abs(i) = 2 to 12 inclusive. The symbols goto, if, then, else, for, do, step, until, while, comment, value, true, and false, along with tab and carriage return, all correspond to a single space.

NOTE 4: The drawing of the characters for 2 $\leq$ abs(i) $\leq$ 10 ends at the point E of the parallelogram. Thus, for -10 $\leq$ i $\leq$ -2, the drawing ends at the given point X, Y (provided, of course, that first ≡ true).

2.4 The Procedure PLOTAXIS

The purpose of this procedure is to facilitate the drawing of co-ordinate axes. We here give the ALGOL procedure heading.

```
procedure PLOTAXIS(X, Y, angle, LENGTH, DL);
value X, Y, angle, LENGTH, DL ;
real X, Y, angle, LENGTH, DL;
```

The procedure draws a line beginning at the point given in X and Y, in the direction given by angle, for a distance given by LENGTH. Tick marks are made along this line at intervals given by DL.

X, Y, LENGTH, and DL are in dits if LENGTH > 0, and otherwise in plits; angle is given in degrees measured counter-clockwise from the +x axis on the paper.

If DL < 0, the tick marks are made 15 plits to each side of the line.
If DL > 0 (but DL $\leq$ abs(LENGTH)) then the ticks are made to one side of the line only (either above, or to the right, whichever is appropriate).
If abs(DL) > abs(LENGTH), no tick marks whatsoever are given,

If angle is other than a multiple of 90° and the scaling factors, Scx and Scy, are unequal, then no simple interpretation can be given to the quantities LENGTH and DL.

At exit from the procedure, the pen is always down, at the point (X + abs(LENGTH) × cos(angle), Y + abs(LENGTH) × sin(angle)).

2.5 The Procedure PLOTCURVE

This procedure is used to draw continuous curves through a set of given points. A special third degree interpolation formula is used.

We here give the ALGOL procedure heading

```
real procedure PLOTCURVE(X, Y, I); value X, Y, I;
real X, Y; integer I;
```

X and Y usually represent the co-ordinates (in dits) of one of the points through which the curve is to pass. The quantity I is used to specify whether we are beginning, continuing, or ending the curve, as follows.

I = 0

The desired initial value of the slope of the curve is given by X (in dits); the value of Y is irrelevant. This call serves to initialize the procedure to begin the interpolation process.

I = 1

This call is used to initialize the procedure when no initial slope is known; the values of X and Y are irrelevant.

I = 2

This call supplies the co-ordinates of one of the points through which the curve is to pass, in X and Y (in dits).

I = 3

This call is used to terminate the current curve; the values of X and Y are irrelevant.

I = 4

This call is used to terminate the current curve and to specify the desired value of the final slope at the last point given. X supplies the value of the slope (in dits) and the value of Y is irrelevant.

NOTE 1: The procedure is intended to be used by first giving a call with I = 0 or 1, followed by a number (at least 3) of calls with I = 2, followed by a call with I = 3 or 4. The pen position is not altered until the third call with I = 2; during this call, the pen moves to the first point given (pen up, of course!), and then draws the segment of the curve to the second point (pen down). During successive calls with I = 2, 3 or 4, the procedure first moves the pen to the last point drawn on the curve (pen up), if necessary, and then continues the curve to the point given in the previous call. Thus, it is possible to use other plotter procedures during the drawing of a curve (e.g. PLOT, PLOTTEXT, etc.).

NOTE 2: If a call with I = 2, 3 or 4 is given before a call with I = 0 or 1, or if a call with I = 3 or 4 is given before 3 calls with I = 2 have been given, or if a call with I ≠ 0, 1, 2, 3 or 4 is given, an error is signalled (no. 1-28).

NOTE 3: By slope, we mean dY/dX.

NOTE 4: After a call with I = 0 or 1 and after the first two calls with I = 2, PLOTCURVE has the value zero and the pen position is completely unaltered. After any other call, PLOTCURVE has the value of the slope of the curve at the previously given point and the pen is down at the previously given point.

NOTE 5: If any part of the curve falls outside the frame, an error is signalled (no. 1-27). If the program is continued, the curve will be drawn nevertheless.

NOTE 6: In order to guarantee proper interpolation, the following rule of thumb is suggested - Imagine straight lines drawn connecting the successive, given points on the curve. Then, for best results, the angle from each of these lines to the next should not exceed 20 degrees, and should preferably be less. If the curvature becomes too great, an error (no. 1-29) may be signalled.

2.6 The Procedure MARKEDCURVE

This procedure is identical in function to PLOTCURVE, except that, when the parameter I = 0 or 1, the parameter Y specifies a special marking symbol which is to be drawn to mark all of the given points. For each value of abs(Y), the marking symbol specified is the same as that for the corresponding value of abs(i) shown in PLOTTEXT.

2.7 The Procedures FIXPLOT, ABSFIXPLOT, FLOPLOT

All three of these procedures may be used to draw numbers on the paper. The three procedures differ only in the format in which the number is drawn.

We here give a typical ALGOL procedure heading.

```
procedure FIXPLOT(X, Y, angle, height, italicity, first, i, n, m,
                  number);
value X, Y, first, i, n, m, number;
real X, Y, angle, height, italicity, number;
integer n, m, i;
Boolean first;
```

The interpretation of X, Y, angle, height, italicity, first, and i is the same as for PLOTTEXT. In all cases, number is the number to be drawn; the interpretation of n and m depends on which of the three procedures is used.

FIXPLOT

The quantity number is drawn in fixed-point representation with n digits before the decimal point and m after. More exactly:

the sign of number (+ or -);
the integer part of number, in n digits;
a decimal point;
m digits of the fraction part of number;
a space.

If m = 0, the decimal point is suppressed.

Nonsignificant leading zeroes in the integer part are replaced by spaces, except in the units position when m = 0.

The number is correctly rounded in the last decimal place. If abs(number) $\geq$ 10 ↑ n, then "+inf" or "-inf" will be drawn followed by spaces, space permitting; otherwise, a shortened version, "+in", "-in", "+i" or "-i" will be drawn.

The total number of characters (including spaces) drawn is

$$m + n + 3 \quad \text{if} \quad m \geq 1$$

or

$$n + 2 \quad \text{if} \quad m = 0$$

The actual parameters for n and m may not be negative, and the case n = m = 0 is forbidden.

ABSFIXPLOT

This procedure is identical in function with FIXPLOT except that the sign of the number is replaced by a blank space.

FLOPLOT

The quantity number is drawn in floating-point format. The parameter n specifies the total number of digits in the mantissa, and m the number in the exponent. More precisely:

the procedure calculates a mantissa xi and an exponent d, such that

$$\text{number} = xi \times 10 \uparrow d,$$

where

$$.1 \leq \text{abs}(xi) < 1;$$

then, the following is drawn

the sign of xi;

a decimal point;

the first n digits of xi, correctly rounded in the last place;

$10^{}$

the sign of d;

the exponent d, in m (1, 2 or 3) digits, with leading zeroes replaced by spaces;

a space.

If $d \geq 10 \uparrow m$, then "+inf" or "-inf" is drawn (depending on the sign of number), followed by n+m+1 spaces,

If $d \leq -10 \uparrow m$ then a + or - sign is drawn (depending on the sign of number), followed by a decimal point, followed by n zeroes, followed by m+3 spaces.

n+m+5 characters are always drawn by the procedure, except in the case n = 0 when the decimal point is omitted, and m+4 characters are drawn.

The actual parameters for n and m must satisfy n $\geq$ 0 and m = 1, 2 or 3.

2.8 The Procedure SCALE

When plotting a graph, it is usual to divide the axes into a number of intervals of equal length, such that each division point represents some "nice number" of data units. If the minimum and maximum data values are known beforehand, it is usually a fairly simple matter to perform this subdivision manually; if, however, no a priori bounds are known, then some machine algorithm is required. SCALE is such an algorithm. Specifically, SCALE examines the data to be plotted, and produces as output a number of quantities which may be used as input parameters to PLOTFRAME, PLOTAXIS, etc.

Clearly, we must have a precise idea of what we mean by a "nice number". Hence, suppose we have a set of so-called "basic round numbers" r_j, $j = 1, 2, \ldots, m$, satisfying $1 \leq r_1 < r_2 < \ldots < r_m < 10$ and not necessarily all integers. Then, we will call any number $DL_{jk} = r_j \cdot 10^k$, for all integers k, a "round number". Thus, if we have the set (1, 1.25, 2, 2.5, 4, 5, 8) of basic round numbers (this is, in fact, the set which is used in the procedure), then the following are examples of round numbers in our sense:

0.1, 1, 10, 12.5, 20, 25, .25, 40, 5, 50, 80, etc.

Finally, by a "nice number", we shall mean any integer multiple of a round number.

We now define an optimal subdivision of an axis by the following four requirements:

(R1) The difference between consecutive division points must be a round number;

(R2) Each division point must be a nice number;

(R3) The data must fit into the allotted space; and,

(R4) Subject to the above three requirements, the data must occupy as much as possible of the allotted space.

The procedure SCALE operates on only one axis at a time. The programmer must supply, for example, the value of the x-co-ordinate of each data point to be plotted. SCALE then searches through these data to determine the maximum and minimum values present, and, applying the four requirements above, calculates parameters which determine an optimum subdivision of the relevant axis. If the minimum and maximum are already known from the previous calculation, then it is possible to provide only these two values as data to the procedure.

The heading of the procedure follows:

```
integer procedure SCALE(Ti, i, n, nint, mode, MIN, MAX, DL);
value n, nint, mode;
integer i, n, nint, mode;
real Ti, MIN, MAX, DL;
```

The parameters have the following interpretations:

Ti — is a real expression, called by name, and depending, in general, on i (Jensen device). It is used to give the successive values of one of the co-ordinates of the data to be plotted. Ordinarily, Ti will simply be a subscripted variable;

i — is a running variable, called by name, which assumes successively all integer values from 1 to n, inclusive.

n — is the number of co-ordinate values to be scaled.

nint — is the number of subintervals into which it is desired to divide the total interval. The precise interpretation of nint depends on the following parameter, viz. mode.

mode — may assume the values 0, 1, or 2, with the following meaning:

mode = 0 the interval is subdivided into precisely nint subintervals;

mode = 1 the interval is subdivided into nint or <u>fewer</u> subintervals. This freedom is used to further maximize the proportion of the available space which is actually used. The actual number of subintervals is between .625 × nint and nint.

mode = 2 the interval is divided into <u>approximately</u> nint subintervals (between .625 × nint and 1.6 × nint). This freedom is used to maximize the proportion of the available space which is actually used.

MIN is an output parameter giving the value of the first division point on the axis (i.e., the point with the smallest algebraic value);

MAX is an output parameter giving the value of the last division point on the axis (i.e., the point with the greatest algebraic value);
both MIN and MAX are nice numbers as defined above;

DL is an output parameter giving the difference in data units between any two successive division points; it is a round number as defined above.

The value of SCALE after a call is the number of subintervals actually used.

NOTE 1: If the minimum and maximum values on the axis are already known (from a previous calculation, perhaps) then only these values need be given to the procedure, instead of the entire set of data. This can be accomplished by using an expression of the following form for Ti:

<u>if</u> i = 1 <u>then</u> minimum <u>else</u> maximum

with n = 2.

NOTE 2: The boundaries given in the description of mode, for the actual number of subintervals used, are dependent upon the set of basic round numbers which are used. If the user replaces the given set (1, 1.25, 2, 2.5, 4, 5, 8) by another, these boundaries may well be altered.

NOTE 3: We now consider the question of what value of mode should be chosen in various circumstances. Apparently, there are two conflicting requirements which the programmer may wish to satisfy. The first of these is that he may require that the physical distance between successive division points be some fixed length (e.g. 1 or 2 cm, 1 inch, etc.), in order to facilitate interpolation using a ruler (with divisions of 1 cm, 1 inch, etc.). Opposed to this requirement, may be a desire to cover as much as possible of the plotting surface with the material which is to be plotted - that is, one wishes to avoid long regions at the beginning and end of the grid in which nothing is plotted.

For the following, let us refer to the total physical length of the axis as pl, and the physical length of each subinterval as dpl. Consider now the following situations:

(a) If precisely the given values of pl and dpl must be used, call SCALE with mode = 0, and nint = pl/dpl;

(b) If an upper bound for pl is given, but an exact value for dpl is required, call SCALE with mode = 1, and nint = pl/dpl;

(c) If an approximate value of pl is given, but an exact value for dpl is required, call SCALE with mode = 2, and nint = pl/dpl;

(d) If a precise value of pl is required, and it is desired to divide the axis into approximately some given number, nint, of subintervals, call SCALE with mode = 2, and the given, approximate, value of nint.

After the call(s) of SCALE, it will presumably be desired to call PLOTFRAME. The correct physical length to be used here (i.e. xmax or ymax) is calculated by multiplying the actual number of intervals (i.e. the value of SCALE) by dpl (in situation (a) above, this is the same as pl).

NOTE 4: The value of the parameter nint must be at least two.

2.9 The Procedure PLOTAXIS2

This procedure may be used to draw co-ordinate axes, complete with tick marks, and with numbers indicating the values corresponding to the tick marks. We here give the ALGOL procedure heading:

```
real procedure PLOTAXIS2 (MIN, MAX, DL, horizontal, OTHER);
value MIN, MAX, DL, horizontal, OTHER;
real MIN, MAX, DL, OTHER;
Boolean horizontal;
```

The procedure draws a line from (MIN, OTHER) to (MAX, OTHER) if horizontal ≡ true and from (OTHER, MIN) to (OTHER, MAX) if horizontal ≡ false. Tick marks of length 15 plits are drawn at an interval of DL from MIN to MAX, either above or to the right of the line, as the case may be. Following this, the values corresponding to the tick marks are drawn below the line or to the left of it. If possible (that is, if the numbers are sufficiently small with respect to the physical distance between tick marks), then the numbers are centred by each tick mark; if this is not possible, then the procedure skips over a few tick marks between numbers, and the tick mark corresponding to each number drawn is lengthened to 25 plits. Under certain circumstances, the procedure calculates, and draws, a power of ten, by which the figures drawn must be multiplied to obtain the actual values.

MIN, MAX, DL and OTHER are always given in data units. It is strongly recommended that MIN, MAX and DL be obtained as output from the procedure SCALE, or at least satisfy similar requirements. In particular, MAX - MIN must be an integer multiple of DL (except, possibly, for small rounding errors).

NOTE 1: If the number of digits to be plotted for any number along the axis is more than seven, then the plotting of all numbers along the axis will be suppressed. In practice, this will seldom prove to be a great restriction; for instance, the number

.00000000000123

has only 3 digits in our sense - it will be drawn as 1.23, and a power of ten = -12 will also be drawn along the axis; the number

1230000000000

also has only 3 digits - it will be drawn as 1.23, with a power of ten = +12; however, the numbers

12345678, 1234.5678, .12345678, etc.,

contain 8 digits, and cause all numbers along the axis to be suppressed.

This suppression is controlled by the statement

precision: = 7;

In general the value of precision should be the smaller of the following two implementation dependent quantities:

(a) The number of decimal digits in the integer capacity,

(b) two less than the number of decimal digits in the real number representation (two less in order to provide a few guarding digits).

NOTE 2: All digits drawn are 28 plits high. If some other height should be desired, this can be accomplished by altering the statement

h: = 4;

near the beginning of the procedure. The value of h is one seventh of the desired height.

NOTE 3: PLOTAXIS2 delivers as its value a co-ordinate which may be used for drawing a text beside the axis. If horizontal ≡ true, this is a Y value; otherwise, an X value. This co-ordinate guarantees a reasonable clearance between the text and the numbering of the axis, if a height of 28 plits is used for the text. In the case that the numbering is suppressed (NOTE 1), PLOTAXIS2 has the value of the parameter OTHER.

NOTE 4: It is important to remember, while calling PLOTFRAME, that it is necessary to leave a border all around the graph, in order to have room for the numbering, power of ten, and, possibly, the text. This border must have at least the following dimensions:

left (-x)	336 plits
right (+x)	132 plits
above (+y)	14 plits
below (-y)	100 plits

See the description of PLOTFRAME for a discussion of the method by which these borders may be obtained.

2.10 The Procedure PLOTPICTURE

This procedure may be used to plot complete graphs, including annotated axes. The user is not required to call (or even be aware of the existence of) any other plotter procedures.

The user is required to provide a set of points which are to be plotted (i.e. usually through which a curve is to be drawn), along with some information about the lengths and divisioning of the axes. From this information, PLOTPICTURE can perform the necessary scaling operations, draw X- and Y-axes complete with tick marks and numbering, and draw a curve through the given points. The given points may be marked with special marking symbols, and a text may be drawn by the axes to indicate their significance.

It is possible to draw an arbitrary number of curves (each marked with a unique symbol) with each axis pair. It is also possible to obtain an arbitrary number of Y-axes corresponding to a single X-axis. A system of axes and curves is here called a "picture", whence the name of the procedure.

We now give the ALGOL procedure heading:

```
procedure PLOTPICTURE (Xi, Yi, i, n, mark, deltamark, mode,
                       XMIN, XMAX, DX, xmax, xstring,
                       YMIN, YMAX, DY, ymax, ystring);
value n, mark, deltamark, mode, XMIN, XMAX, DX, xmax, YMIN,
      YMAX, DY, ymax;
real Xi, Yi, XMIN, XMAX, DX, YMIN, YMAX, DY;
integer i, n, mark, deltamark, mode, xmax, ymax;
string xstring, ystring;
```

The parameters have the following interpretations:

Xi and Yi are real expressions, called by name, depending, in general, on i (Jensen device). They give the X and Y co-ordinates, respectively of the i'th point which is to be plotted. Ordinarily, Xi and Yi will simply be subscripted variables (array elements);

i — is a running variable, called by name, which assumes, successively, all integer values between 1 and n, inclusive;

n — is the number of points to be plotted;

mark — specifies which one of nine special marking symbols, if any, is to be drawn to mark the points (Xi, Yi), according to the following table:

abs(mark) = 0,1	no symbol is drawn
2	+
3	×
4	✳
5	✳
6	Y
7	⧖
8	⧖
9	⊡
10	◇

deltamark — if deltamark $\geq$ 0, a curve is drawn passing through all n of the points (Xi, Yi) in succession. If deltamark < 0, the drawing of the curve is suppressed.

Furthermore, if deltamark $\neq$ 0, then every abs(deltamark)'th point, beginning with the first (i = 1), is marked with the symbol specified by the parameter mark; the intermediate points are evaluated, but not marked. If deltamark = 0, no marking at all takes place, but the curve is drawn. For example, if abs(deltamark) = 1, all points are marked; if abs(deltamark) = 2, every other point is marked; if abs(deltamark) = n-1, then only the first and last point are marked; if abs(deltamark) $\geq$ n, then only the first point is marked.

mode — may be thought of as actually composed of four parameters: kg, kd, kx and ky, thus

mode = 1000 × kg + 100 × kd + 10 × kx + ky.

That is, the four parameters are each one decimal digit long, and mode is obtained by writing these digits in order beside one another.

kg may have the values 0, 1, or 2. kg = 0 or 1 indicates that an entire, new "picture", complete with annotated axes is to be drawn; kg = 2 indicates that the new data is to be plotted in the old "picture", possibly with a new Y-axis. If kg = 1, then a rectangular grid is drawn over the plotting surface to the right of the Y-axis, and above the X-axis. The grid lines may be thought of as simply extensions to the tick marks which are normally placed by the axes (see later discussion). If kg = 0, then only the two grid lines on the border are drawn, complete with tick marks, thereby framing the picture in. If kg = 2, no grid lines at all are drawn.

kd may have the values 0, 1, 2, or 3. It specifies whether or not the programmer has provided begin and/or end slopes of the curve to be drawn.
kd = 0 indicates that no derivatives are given;
kd = 1 indicates that the initial derivative is given;
kd = 2 indicates that the final derivative is given;
kd = 3 indicates that both derivatives are given.
If the initial slope is given, PLOTPICTURE will call for it by setting the parameter i to zero, and evaluating the expression Xi. Similarly, the end slope is called for by setting i to zero and evaluating the expression Yi. Thus, a typical expression for Xi might be

if i = 0 then beginslope else X[i].

kx and ky are used to specify one of eight possible scalings for, respectively, the X- or Y-axis. They exercise influence upon the interpretation of all of the remaining parameters. Since the possibilities for the two axes are practically identical, we describe here the situation with respect to the Y-axis; the X-axis is handled similarly, except for one point which will be discussed later.

For ky = 0, 1, or 2, the values of the parameters YMIN and YMAX are completely ignored. The procedure evaluates the expression Yi for all values of i between 1 and n, and determines the minimum and maximum values which occur. This minimum and maximum are used for the further determination of an optimal scale, in a manner depending on the particular value of ky.

The programmer is given the option of avoiding the above (possibly slow) evaluations of Yi, if he already knows the minimum and maximum values assumed by Yi. This is accomplished by using ky = 3, 4, or 5, and by giving the known minimum and maximum values in YMIN and YMAX, respectively.

For $0 \leq ky \leq 5$, the value of DY specifies the desired physical distance (in units of one tenth of a millimeter (plits)), between tick marks on the Y-axis; ymax specifies the desired physical length of the entire Y-axis, in the same units; ymax must be an integer multiple of DY.

If ky = 0 or 3, then the Y-axis drawn has precisely the measurements specified by DY and ymax.

If ky = 1 or 4, then the distance between tick marks on the Y-axis will be precisely that specified by DY. However, the total physical length of the Y-axis will be chosen, between 0.625 × ymax and ymax, in such a manner that the axis be as "full" as possible.

By as "full" as possible, we mean that the data to be plotted extend from a position as near as possible to the bottom (left end) of the graph to a position as near as possible to the top (right end) of the graph; that is, the unused space is kept as small as possible.

If ky = 2 or 5, then the total physical length of the Y-axis is precisely that specified by ymax. The distance between tick marks is chosen between .625 × DY and 1.6 × DY in such a manner that the axis be as "full" as possible.

For the previous values of ky ($0 \leq ky \leq 5$), the procedure calculates a "nice number", less than or equal to the minimum value of Yi (either given or calculated), by the first (lowest) tick mark on the axis. Similarly, a "nice number" is chosen for the last (highest) tick mark (for a precise definition of "nice number", see the description of the procedure SCALE).

However, if the programmer himself wishes to specify precisely the beginning and ending point of the axis, he may do so by choosing ky = 6. Then the axis will be drawn with a physical length of precisely ymax, and the tick marks will be drawn at YMIN, YMIN + DY, YMIN + 2 × DY, ..., YMAX - DY, YMAX. The value of YMAX - YMIN must be an integer multiple of DY, except for possible small rounding errors.

For all of the above values of ky ($0 \leq ky \leq 6$), a new Y-axis, complete with numbering, is drawn to the left of the previous one. Also, the text specified by ystring is drawn by the axis.

If ky = 7 and kg = 2 (old picture), then the values of all the parameters YMIN, YMAX, DY, ymax, and ystring are completely ignored, and no new Y-axis is drawn. That is, the last-defined Y-axis is to be used for the plotting of the current set of data. If ky = 7, and kg = 0 or 1 (new picture), then a new Y-axis is drawn (the first one of the new picture), having the same length, subdivisioning, numbering, etc., as the immediately previous Y-axis (i.e. the last Y-axis of the previous picture), but with the current text specified by ystring.

The handling of the X-axis, and the parameters XMIN, XMAX, DX, xmax and xstring is, as previously mentioned, analogous with that of the Y-axis and the parameters YMIN, YMAX, DY, ymax and ystring. The one difference is that no provision is made for multiple X-axes in one picture. This is realized by the following rule: if kg = 2 (old picture), then the actual value of kx is ignored; the procedure carries on as though kx = 7 had been specified.

NOTE 1: The strings xstring and ystring may contain any valid ALGOL basic symbols. However, the following result in a blank (space): goto, if, then, else, for, do, step, until, while, comment, value, true, false, tab, and carriage return. Furthermore,

begin	gives	┼
end	"	╳
own	"	✳
Boolean	"	✳
integer	"	Y
real	"	⧖
array	"	⧖
switch	"	◱
procedure	"	◈
string	"	⬡
label	"	⊓

NOTE 2: The parameter ymax $\leq$ 2500; that is, the Y-axis may be no more than 25 cm long.

NOTE 3: Sufficient room is reserved to the left of each new picture to guarantee accomodation for at least ten Y-axes.

NOTE 4: If deltamark $\geq$ 0, it is required that n $\geq$ 3.

NOTE 5: The following table may be found useful in the construction of the parameter mode.

kg = 0 new picture, without grid
= 1 new picture, with grid
= 2 old picture (without grid)

kd = 0 no derivatives given
= 1 initial derivative given
= 2 final derivative given
= 3 initial and final derivatives both given

ky = 0 YMIN, YMAX irrelevant; DY exact physical length of intervals; ymax exact physical length of axis
= 1 YMIN, YMAX irrelevant; DY exact physical length of intervals; ymax upper bound to physical length of axis
= 2 YMIN, YMAX irrelevant; DY approximate physical length of intervals; ymax exact physical length of axis
= 3 YMIN, YMAX given; DY exact physical length of intervals; ymax exact physical length of axis
= 4 YMIN, YMAX given; DY exact physical length of intervals; ymax upper bound to physical length of axis
= 5 YMIN, YMAX given; DY approximate physical length of intervals; ymax exact physical length of axis
= 6 from YMIN, in steps of DY to YMAX; ymax exact physical length of axis
= 7 same as previous Y-axis

NOTE 6: We now give the order in which the expressions Xi and Yi are evaluated:

(a) if $0 \leq kx \leq 2$, and $kg \neq 2$, then Xi, i = 1, 2, ..., n;

(b) if $0 \leq ky \leq 2$, then Yi, i = 1, 2, ..., n;

(c) if kd = 1 or 3, then Xi, i = 0 - i.e. the initial slope;

(d) Xi, Yi, i = 1, 2, ..., n - i.e. X_1, Y_1, X_2, Y_2, ..., X_n, Y_n;

(e) if kd = 2 or 3, then Yi, i = 0 - i.e. the final slope.

NOTE 7: In order to guarantee proper interpolation of the curve between the given points (deltamark ≥ 0), the following rule of thumb is suggested - Imagine straight lines connecting the successive, given points on the curve. Then, for best results, the angle from each of these lines to the next should not exceed 20 degrees, and should preferably be less.

NOTE 8: The complete picture, including numbering and text, has the following maximum dimensions:

(a) height: physical length of Y-axis (i.e. ymax) + 114 tenths of a mm.

(b) width: physical length of X-axis (i.e. xmax) + 132 + (no. of Y-axes) × 336 tenths of a mm.

3. Theory of the ALGOL Procedures

3.1 Introduction

In the following ALGOL texts, we make use of the following typing conventions

÷	represents	ALGOL	60	÷
≠	"	"	"	'
≥	"	"	"	'

Furthermore, where the ALGOL concept own is employed, the so-called "static interpretation" [2] is intended.

The procedure STOP is common to several of the procedures. Its purpose is to signal error conditions which arise during the execution of the procedures, and then (possibly) return to the calling procedure. It accepts a single parameter, specified as type string, which indicates the nature of the error.

It is assumed that the following declaration appears in some block enclosing that in which the plotter procedures are declared:

```
Boolean state1, state2, state4, state8, state16,
        state32, state64, firstentrance;
```

and that the following statement has been executed before the first call of any of the plotter procedures:

```
firstentrance: = true;
```

This is used to monitor whether or not certain important variables have as yet had values assigned to them.

Note that users of the machine code procedures for the Electrologica X1 are not required to provide the above-mentioned declarations.

3.2 The Straight-line Approximation in PLOT

The procedure first converts (if necessary) the given points to plotter units. Then the total movement dx in the x-direction, and dy in the y-direction are calculated.

It is clear that every straight line which is not parallel to one of the basic plotting directions will nevertheless lie between a true x- or y-direction and one of the diagonals. Thus, we can approximate any arbitrary straight line by a suitable sequence of only two of the basic pen movements - a true x- or y-movement, and one of the diagonal movements.

Accordingly, we adopt the strategy of transforming the given straight line into the first octant, and simultaneously determining the two basic plotter motions involved. The determinition of the plotter motions is somewhat machine dependent (that is, it depends on the specific numerical codes corresponding to the several directions), and is not further discussed here.

The transformation to the first octant proceeds as follows: we first transform to the first quadrant by replacing dx and dy by their absolute values; then, if dy > dx, we interchange the values of dx and dy with one another, thereby arriving in the first octant. Note once again that, during these transformation steps, some here unspecified process must determine the codes for the two relevant plotter motions.

Now, at the i'th step forward in the x-direction (i.e. at x = i), the value of y is clearly

$$y_i = i \times \frac{dy}{dx} \,, \qquad 3.2.1$$

whereas the y-co-ordinate of the actual pen position has some value y_i^*. That is, the pen position deviates from the correct position by

$$dev_i = y_i - y_i^* \,. \qquad 3.2.2$$

Then, when we advance to the next value of x (= i + 1), the value of y is

$$y_{i+1} = (i + 1) \times \frac{dy}{dx} = y_i + \frac{dy}{dx}\ , \qquad 3.2.3$$

and the y-co-ordinate of the actual pen position is

$$y^*_{i+1} = y^*_i + g, \qquad 3.2.4$$

where g is either 0 or 1 (i.e. a step along the true direction or the diagonal, respectively). This gives rise to a new deviation

$$\begin{aligned} dev_{i+1} &= y_{i+1} - y^*_{i+1} \\ &= y_i + \frac{dy}{dx} - y^*_i - g \\ &= dev_i + \frac{dy}{dx} - g. \qquad 3.2.5 \end{aligned}$$

Now we must make this deviation as small as possible in absolute value by choosing g appropriately. Since there are only two possibilities, this is rather simple; we choose the true direction (g = 0) if

$$\left| dev_i + \frac{dy}{dx} \right| < \left| dev_i + \frac{dy}{dx} - 1 \right|. \qquad 3.2.6$$

Squaring both sides, we may remove the absolute value signs; thus, we choose the true direction if

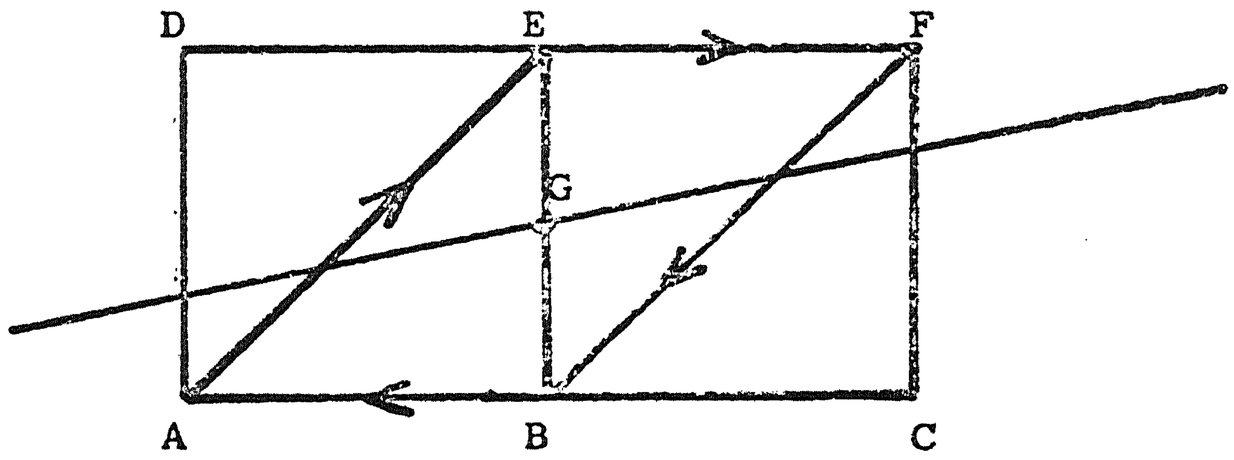
$$(dev_i + \frac{dy}{dx})^2 < (dev_i + \frac{dy}{dx})^2 - 2(dev_i + \frac{dy}{dx}) + 1; \qquad 3.2.7$$

i.e., if

$$dev_i + \frac{dy}{dx} < \frac{1}{2}\ . \qquad 3.2.8$$

Clearly, we choose the diagonal direction if the inequality in (3.2.8) is reversed. The question still remains as to which choice should be made in the case that both sides of (3.2.8) are equal, and this is not such a trivial matter as one might think. The difficulty arises if one wishes to retrace a section of straight line, but in the opposite direction, rather than to raise and lower the pen (at a total cost of at least 200 ms - equivalent in time to 60 ordinary steps).

Suppose that we choose the true direction if (3.2.8) is satisfied, and otherwise the diagonal direction (i.e. if $dev_i + dy/dx \geq 1/2$). Then, consider the following situation, where the line to be approximated passes exactly through G, the centre-point of BE.



If the pen begins at A, this choice will lead to a diagonal step to E, since equality holds, followed by a horizontal step to F. However, if the pen begins at F, (i.e. retracing the line in the opposite direction), a diagonal step is again taken, to B, followed by a horizontal step to A. Thus, the pen does not follow precisely the same path during the retrace as during the trace, and this may give rise to thick spots, or even open boxes in the line. This is not usually serious with a step length of .1 mm and a pen .4 mm thick, but with a step length of .01 inch and a finer pen, the result may be annoying.

Fortunately, the solution is very simple; when going from left to right, for instance, we use (3.2.8) as it stands, but when going from right to left, we replace the < by $\leq$ as the criterion for moving in the true direction. The direction (left to right or right to left) can be determined once and for all by examining the sign of dx before the transformation to the first octant.

In order to avoid working with floating-point numbers, we multiply the equations (3.2.5) and (3.2.8) by 2 × dx, obtaining

$$2 \times dx \times dev_{i+1} = 2 \times dx \times dev_i + 2 \times dy - 2 \times g \times dx, \qquad 3.2.5a$$

and

$$2 \times dx \times dev_i + 2 \times dy < dx \text{ (or } \leq). \qquad 3.2.8a$$

Thus, the final algorithm is as follows:

(a) Calculate dx and dy;
(b) Store the sign of dx;
(c) Replace dx by |dx| and dy by |dy|;
(d) If dy > dx then interchange dy and dx;
(e) During (c) and (d) determine the codes for the true and the diagonal direction;
(f) Calculate dx2: = 2 × dx and dy2: = 2 × dy;
(g) Set dev: = 0;
(h) Replace dev by dev + dy2;
(i) If dev < dx (or $\leq$ depending on original sign of dx) then perform a step in the true direction; and otherwise, a step in the diagonal direction followed by the subtraction of dx2 from dev;
(j) Repeat (h) and (i) dx times.

3.3 Some Notes Concerning PLOTTEXT

We consider here the question of the manner in which the various characters drawn by PLOTTEXT are described and how these descriptions are stored in a computer. For this purpose, we think in terms of a rectangle of size 7 by 4 plotter units. This rectangle may be subjected to later transformations (such as magnification, translation, etc.), but for our present purposes, we think only of a rectangle.

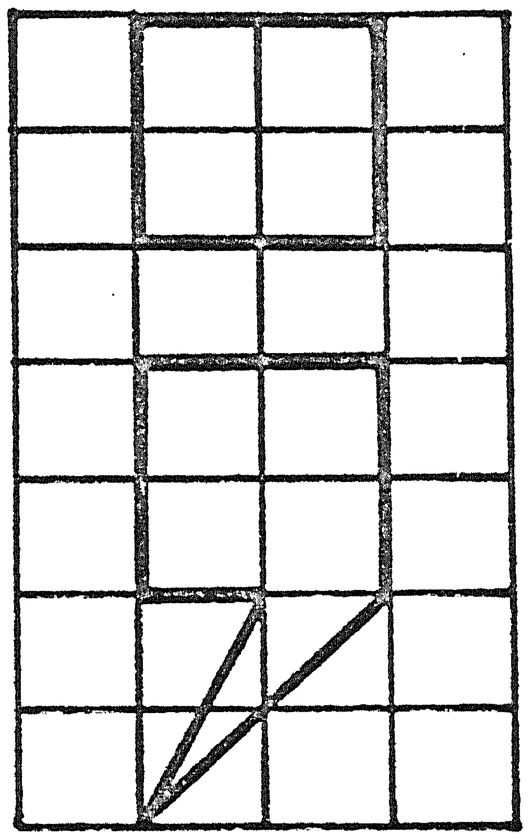
The rectangle is subdivided into squares of one plotter unit on each side, and the nodes of this grid have Cartesian co-ordinates associated with them in an obvious manner. Thus, the lower left-hand corner of the rectangle has co-ordinates (0,0), and the upper right-hand corner is (4,7).

Each character is described by a sequence of co-ordinate pairs in the rectangle. At the beginning of the character, the plotter moves (with pen up) to the point given by the first pair, and lowers the pen. Then, the pen is moved in a straight line to the point given by the following pair, and so on, until the entire character has been drawn.

Note that the values 5, 6, and 7 for the first co-ordinate of a co-ordinate pair do not specify points in the rectangle, but are used for the following special purposes. If the first co-ordinate is 5 or 6, this means that the entire rectangle is to be thought of as being shifted,up(6) or down (5) by the number of squares given by the second co-ordinate. If the first co-ordinate is 7 the pen is raised, and is moved to the position given by the next pair before being lowered again (the second co-ordinate is irrelevant if the first one is 7).

As an example, we give the description of the semicolon:

(5,2)	(1,0)	(3,2)	(3,4)	(1,4)	(1,2)	(2,2)
(1,0)	(7,0)	(1,5)	(1,7)	(3,7)	(3,5)	(1,5)



The ALGOL 60 procedure assumes the following declaration, global to PLOTTEXT:

```
integer array CX, CY[0:552], N, ENTRANCE[0:122];
```

The arrays CX and CY contain, respectively, the first and second co-ordinates of all the pairs required to describe all of the characters which can be drawn by the procedure. The integer representation of each string symbol (delivered by STRINGSYMBOL) is used to select a unique element from each of the arrays N and ENTRANCE. The element from N specifies the number of co-ordinate pairs which correspond to the given string symbol,and the element from ENTRANCE is the index of CX and CY in which the first pair can be found.

In machine code, of course, these arrays are actually an inherent part of the procedure, and are, furthermore, carefully packed to occupy the minimum of storage possible.

As an economy measure, certain of the characters have been described in common. For example, the characters comma and semicolon have the same ENTRANCE value, but a different N value (for comma, N = 8; for semicolon, N = 14).

In tables 1 and 2 we give the complete contents of the arrays described above.

Table 1 describes the contents of CX and CY. Any material which appears between apostrophes does not form a part of the array, but is included for clarification only. The string symbol which is being described (or its corresponding value of abs(i) in parentheses, where relevant)is given at the left. The co-ordinate pairs for this character follow immediately and continue until the end of the sequence, unless there appears the character $\wedge$, followed by another character, enclosed in apostrophes, indicating that the sequence is terminated for the character following the $\wedge$. The character $\vee$ indicates that the description of the character following the $\vee$ begins with the following co-ordinate pair.

Table 2 describes the contents of N and ENTRANCE. The first column gives the N value, and the second the ENTRANCE of the character given in the third column. In the fourth column, the integer representation of the character in X1 ALGOL is given. For the character corresponding to $2 \leq \text{abs}(i) \leq 12$ the values of abs(i) are given in an additional column.

Table 3 shows all the characters that can be drawn by PLOTTEXT, in boxes 14 by 24.5 mm (i.e., the distance between grid points is 3.5 mm).

3.4 Theory of PLOTCURVE

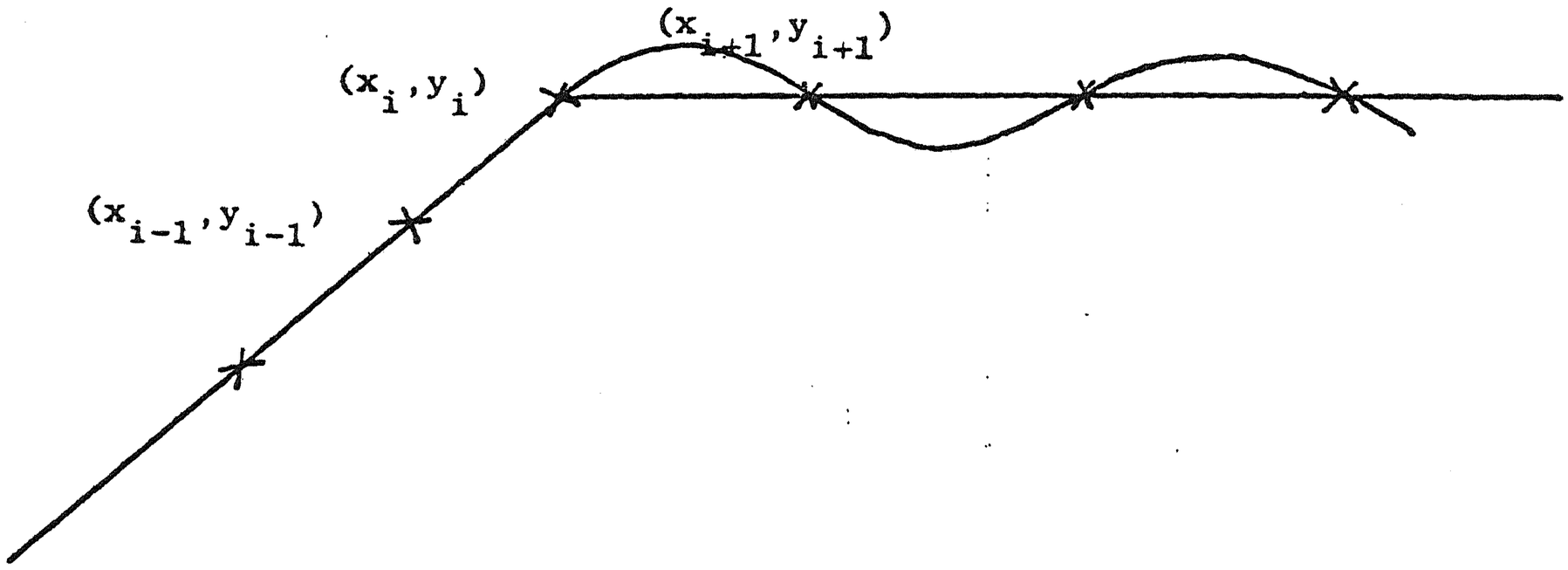
Introduction

The problem we wish to solve is the following: Given an ordered set of points (x_i, y_i) in the plane, to pass a "smooth" curve through these points successively; the problem is not one of interpolation in the usual sense, since we wish to admit the possibility that the curve may loop around arbitrarily.

Anticipating the fact that we shall have to be able to evaluate the interpolating curve once for each plotter step (i.e., about once every 3 1/3 ms) we are practically forced to restrict the candidates for this interpolating function to low-order polynomials, in order to achieve a decent speed on a medium-speed (or even high-speed) computer.

We will require at least the following two properties of the curve: (a) the curve must indeed pass through the given points, and in the given order; (b) the slope of the curve must be everywhere continuous, including, in particular, the given points.

The simplest polynomial that we can consider is the quadratic passing through the points (x_i, y_i) and (x_{i+1}, y_{i+1}) with the same slope at (x_i, y_i) as the quadratic in the previous interval from (x_{i-1}, y_{i-1}) to (x_i, y_i). The two conditions of passing through the two points, along with the condition on the derivative uniquely determine the three coefficients of the quadratic. The objection to this scheme, however, is that if there is a discrepancy ε between the calculated derivative at x_i and the "real" derivative, this discrepancy is propagated, with a change of sign; that is, errors are not damped out. The following diagram illustrates the sort of thing that can happen.



Thus, we require a scheme in which the curve between the points (x_i, y_i) and (x_{i+1}, y_{i+1}) does not depend in any way on too many of the past and future points. In the scheme which we have adopted, the points (x_{i-1}, y_{i-1}), (x_i, y_i), (x_{i+1}, y_{i+1}), and (x_{i+2}, y_{i+2}) are used to determine the curve (a cubic polynomial) between (x_i, y_i) and (x_{i+1}, y_{i+1}). In this way, propagation of errors is completely avoided.

The Interpolating Process

We first transform all the points to plotter units; that is, the interpolation is done in the plotter space, not the data space. Further, at the time we are considering the segment of curve from (x_i, y_i) to (x_{i+1}, y_{i+1}), we perform a translation of axes so that $(x_i, y_i) = (0,0)$. Also, for compactness, we relabel (x_{i+k}, y_{i+k}) as (x_k, y_k).

Now, we seek the coefficients of the polynomial

$$f(x) = ax^3 + bx^2 + cx + d \qquad 3.4.1$$

such that $f(x_0) = y_0$, $f(x_1) = y_1$,

$$f'(x_0) = y'_0, \text{ and } f'(x_1) = y'_1, \qquad 3.4.2$$

where y'_0 and y'_1 are the desired values of the slope at x_0 and x_1 (we discuss the choice of these slopes later).

A short computation yields

$$a = \frac{y'_0 + y'_1 - 2y_1/x_1}{x_1^2}$$

$$b = \frac{y_1 - ax_1^3 - y'_0 x_1}{x_1^2} \qquad 3.4.3$$

$$c = y'_0$$

$$d = 0.$$

We must calculate values for the slopes y'_0 and y'_1 in such a way that they depend upon only a small number of surrounding points. If y_1 is not an end point of the curve, the value we choose for y'_1 is the slope of the unique circle passing through (x_0, y_0), (x_1, y_1) and (x_2, y_2). This choice has the advantage of being symmetric, and independent of the system of axes.

Let us consider the circle

$$f(x,y) = x^2 + y^2 + Ax + By + C = 0, \qquad 3.4.4$$

passing through the three points. We thus have

$$f(0,0) = f(x_1, y_1) = f(x_2, y_2) = 0. \qquad 3.4.5$$

From the first of these, C = 0.

From the remaining two

$$x_1^2 + y_1^2 + Ax_1 + By_1 = 0, \qquad 3.4.6$$

$$x_2^2 + y_2^2 + Ax_2 + By_2 = 0, \qquad 3.4.7$$

and hence,

$$B = \frac{x_1x_2(x_2 - x_1) + x_1y_2^2 - x_2y_1^2}{x_2y_1 - x_1y_2}, \qquad 3.4.8$$

$$A = -(x_1^2 + y_1^2 + By_1)/x_1. \qquad 3.4.9$$

Now, we have

$$f(x,y) = x^2 + y^2 + Ax + By = 0.$$

Hence, by differentiation,

$$\frac{df}{dx} = 2x + 2y\,\frac{dy}{dx} + A + B\,\frac{dy}{dx} = 0,$$

and thus

$$\frac{dy}{dx} = -\frac{(2x + A)}{(2y + B)} \qquad 3.4.10$$

gives the slope of the circle through the three points at an arbitrary point (x,y) on the circle.

Thus, after some computation, we have

$$y_1' = \frac{dy}{dx}\bigg|_{x_1} = \frac{y_1(y_2(y_2 - y_1) + x_2(x_2 - 2x_1)) + x_1^2\,y_2}{x_1(y_2(y_2 - 2y_1) + x_2(x_2 - x_1)) + x_2y_1^2} \qquad 3.4.11$$

As value for y'_0, we use the value calculated for y'_1 one step previous, thus ensuring continuity of slope, as required. When we first begin, however, we have no previous value of y'_1 to use as our y'_0. Hence, we give the user the option of specifying the initial slope value if he knows it. If he does not, then we simply use the slope of the circle through the first three points, at the first of these. Thus, from (3.4.10), we use

$$y'_0 = \frac{dy}{dx}\bigg|_0 = \frac{y_1(x_2^2 + y_2^2) - y_2(x_1^2 + y_1^2)}{x_1(x_2^2 + y_2^2) - x_2(x_1^2 + y_1^2)} . \qquad 3.4.12$$

Similarly, when we come to calculate the curve between the last two points, we give the user the option of specifying the final slope. If he elects not to specify this, then we use the slope of the circle through the last three points, at the last point. For programming reasons, it turns out that at this time we have only the values of the last two points as (x_0, y_0) and (x_1, y_1), but we also have the slope y'_0 of the circle at (x_0, y_0). Thus, we must construct y'_1 from these data. From (3.4.10), we have, at (x_0, y_0)

$$y'_0 = \frac{dy}{dx}\bigg|_0 = -\frac{A}{B} , \qquad 3.4.13$$

and thus,

$$A = -By'_0 . \qquad 3.4.14$$

Substituting (3.4.14) into (3.4.6), we have

$$x_1^2 + y_1^2 - Bx_1y'_0 + By_1 = 0, \qquad 3.4.15$$

and hence,

$$B = \frac{x_1^2 + y_1^2}{x_1y'_0 - y_1} , \qquad A = \frac{-y'_0(x_1^2 + y_1^2)}{x_1y'_0 - y_1} . \qquad 3.4.16$$

Hence, substituting (3.4.16) into (3.4.10),

$$y_1' = \frac{dy}{dx}\bigg|_{x_1} = -\frac{(2x_1 + A)}{(2y_1 + B)}$$

$$= \frac{2x_1y_1 + y_0'(y_1^2 - x_1^2)}{2x_1y_1y_0' - (y_1^2 - x_1^2)} \quad . \qquad 3.4.17$$

This completes the definition of the interpolating curve except for one very important point - we have said that we shall have no objection to multivalued curves, and yet, we are using polynomials for interpolation and these are certainly single-valued. We get around this difficulty by the simple device of allowing either x or y to be the independent variable; in fact, we decide anew for each section of curve between two successive points, whether x or y is to be independent in that section.

To describe how this choice is made, we begin a brief description of the ALGOL procedure itself.

Within the procedure are declared the own variables x1, x2, y1, y2 and the non-own's x0 and y0, which contain the three points of current interest, scaled to plotter units. At each normal entry to the procedure, the point (x1, y1) is pushed down onto (x0, y0), (x2, y2) is pushed onto (x1, y1), and the scaled co-ordinates of the new point which has just been given, are assigned to (x2, y2). Furthermore, the own y1p (representing y_1') is pushed down onto y0p (representing y_0').

We then begin the translation of the given points relative to (x0, y0). This is performed by creating the new quantities xs1, ys1, xs2 and ys2 (xs0 and ys0 do not exist, being identically zero), corresponding to the quantities x_1, y_1, x_2, y_2, respectively, in equations (3.4.3), (3.4.11), (3.4.12) and (3.4.17). At the same time, we make a further transformation, so that xs1 and xs2 are values of the independent variable actually chosen, and ys1 and ys2 are values of the corresponding dependent variable; that is, for instance,

$$xs1 = x1 - x0$$

if x is the current independent variable, and

$$xs1 = y1 - y0$$

otherwise. In more detail, we make use of the own Boolean xisx with the interpretation that xisx ≡ true if x is the current independent variable, and xisx ≡ false otherwise. Initially, xisx ≡ true. When we come to calculate the scaled co-ordinates xs1, ys1, xs2, ys2, we first assume the choice of independent variable used in the previous step, thereby calculate the scaled quantities, and the derivate y1p (note that y0p and y1p are always the derivates with respect to the current independent variable).

Clearly, if we had used the other choice of independent variable, we would have the slopes 1/y0p, 1/y1p in place of y0p and y1p. Then, if

$$\max(|y0p|, |y1p|) \leq \max(|1/y0p|, |1/y1p|),$$

we accept the previous choice of independent variable as the current one; otherwise, we switch over, and recalculate the quantities xs1, etc., relative to the new independent variable. More intuitively, we may say that we choose the independent variable to minimize the greatest slope of the curve in the current segment.

When this decision has been completed, we continue on to the calculation of the polynomial coefficients themselves. In this connection,there is one subtle point to be discussed. Clearly, when we convert the given points from data units, the result will not, in general, be an integral number of plotter units.

Hence, we use the integer quantities xs1i and ys1i instead of xs1 and xs2 in the equations (3.4.3), in order to obtain a polynomial which passes precisely (aside from round-off error) through the grid-points nearest to the given points. This trick prevents possible discontinuities of one plotter step which otherwise would sometimes occur at the given points. However, for maximum accuracy (especially when the given points are fairly close together), we use the quantities xs1, ys1, xs2, ys2 in the equations (3.4.11), (3.4.12), and (3.4.17), which determine only the slope of the curve at the given points.

Following the Interpolation Polynomial with the Plotter

Having now defined mathematically the curve which we wish to draw, we turn our attention to the problem of following this curve with the (eight) basic motions made available to us by the plotter.

In the sequel, it will be important to remember that the quantities x, y, f, xs1i are relative to the current independent variable, chosen as described in the previous section.

We define the quantities x, y and f as follows:
x is the abscissa of the position of the pen; y is the corresponding ordinate; and f is the value of the polynomial at x. These three quantities are all set to zero at the beginning of the current section of curve (i.e. between two given points).

The quantity incrx is set to +1 or -1 according as we need to increase or decrease x to arrive at the desired endpoint of the current section (thus, incrx = sign(xs1i)).

In order to proceed from the current abscissa to the next following, we add incrx to x, and calculate the new value of f. We then calculate dif = f - y and absdif = abs(dif). Now we are

ready to determine the proper plotter action as follows: if absdif is less than one half plotter unit, the step is in a purely horizontal direction; if between a half and one and a half, the step is diagonal and is accompanied by an appropriate up-dating of y; if greater than one and a half, the step is vertical, accompanied by up-dating of y, and is followed by a return (to label "loop2" in the ALGOL procedure) to calculate a new dif, and repeat.

The problem of which action should be performed by the plotter is simplified by introduction of the array ACT, which contains integers corresponding to the various possible plotter motions. The element of ACT which is used may be described by the following ALGOL 60 expression:

```
ACT[(if absdif ≤ 0.5 then 0 else if absdif ≤ 1.5 then 8 else16)
   + (if xisx then 0 else 4)
   + (if incrx > 0 then 0 else 2)
   + (if dif > 0 then 0 else 1)]
```

In the ALGOL procedure, as much as possible of this subscript expression is evaluated before entering the loop. At the end of each step, a test is made to determine whether x = xs1i; if it does, the section of curve has been completed, and exit is taken from the procedure; otherwise, the next step is carried out.

Evaluating the Polynomial

In the previous section, we mentioned that we would evaluate the polynomial

$$f(x) = ax^3 + bx^2 + cx \qquad 3.4.18$$

once for each plotter step taken. Since each step of the plotter takes only 3.3 ms, we must certainly perform this polynomial evaluation in rather less than this length of time, in order to leave time for the decision described above, of which plotter movement to take.

On machines with fast floating-point hardware, one can evaluate (3.4.18) directly for each step. However, for other machines, we now give a scheme which can be carried out rapidly in fixed-point, although at the cost of some extra storage space for the procedure.

We begin by noticing the important fact that our evaluations of f(x) occur, not for arbitrary, but for successive values of x. Thus, we expect that, given f(x), the following value of f, f(x + h) (h = incrx = $\pm$ 1) should not be very greatly different. Accordingly, we attempt to calculate this difference.
Defining

$$\Delta f(x) = f(x + h) - f(x), \qquad 3.4.19$$

we have

$$f(x + h) = f(x) + \Delta f(x), \qquad 3.4.20$$

and, from (3.4.18) and (3.4.19),

$$\Delta f(x) = a(3x^2h + 3xh^2 + h^3) + b(2xh + h^2) + ch. \qquad 3.4.21$$

Thus, if we know f(x) and $\Delta f(x)$, we can easily calculate f(x + h) from (3.4.20).
Setting

$$\Delta^2 f(x) = \Delta(\Delta f(x)) = \Delta f(x + h) - \Delta f(x), \qquad 3.4.22$$

we have

$$\Delta f(x + h) = \Delta f(x) + \Delta^2 f(x), \qquad 3.4.23$$

and, from (3.4.21) and (3.4.22),

$$\Delta^2 f(x) = 6ah^2(x + h) + 2bh^2. \qquad 3.4.24$$

Setting

$$\Delta^3 f(x) = \Delta(\Delta^2 f(x)) = \Delta^2 f(x + h) - \Delta^2 f(x), \qquad 3.4.25$$

we have

$$\Delta^2 f(x + h) = \Delta^2 f(x) + \Delta^3 f(x), \qquad 3.4.26$$

and, from (3.4.24) and (3.4.25),

$$\Delta^3 f(x) = 6ah^3. \qquad 3.4.27$$

Thus, the following scheme suggests itself.Suppose that, at a given moment, we know $f(x)$, $\Delta f(x - h)$, $\Delta^2 f(x - 2h)$, and $\Delta^3 f$, then we can calculate,sequentially:

$$\begin{aligned} \Delta^2 f(x - h) &= \Delta^2(x - 2h) + \Delta^3 f, \\ \Delta f(x) &= \Delta f(x - h) + \Delta^2 f(x - h), \qquad 3.4.28 \\ f(x + h) &= f(x) + \Delta f(x). \end{aligned}$$

In this,way, we obtain $f(x + h)$ and the Δf and $\Delta^2 f$ required for the next step, at the cost, essentially, of three addition and store operations.

In order to get started, we know that $f(0) = 0$ from our choice of co-ordinate system. At $x = 0$, (3.4.28) becomes

$$\begin{aligned} \Delta^2 f(-h) &= \Delta^2 f(-2h) + \Delta^3 f, \\ \Delta f(0) &= \Delta f(-h) + \Delta^2 f(-h), \qquad 3.4.29 \\ f(h) &= f(0) + \Delta f(0). \end{aligned}$$

Thus, we require the values of $\Delta^2 f(-2h)$ and $\Delta f(-h)$ initially. From (3.4.21),

$$\Delta f(-h) = ah^3 - bh^2 + ch, \qquad 3.4.30$$

and from (3.4.24),

$$\Delta^2 f(-2h) = -6ah^3 + 2bh^2. \qquad 3.4.31$$

Now, recalling that h = incrx = $\pm$ 1, we have to calculate the following initial quantities:

$$\begin{aligned} f &= 0, \\ \text{delta} &= \text{incrx}(a - b \times \text{incrx} + c), \\ \text{delta2} &= 2b - 6a \times \text{incrx}, \\ \text{delta3} &= 6a \times \text{incrx}. \end{aligned} \qquad 3.4.32$$

If the process described above is carried out using fixed-point arithmetic, it will probably be sufficiently fast. However, the quantities f, delta, delta2, and delta3 are not, in general, integers, and hence the question of scaling comes up; that is, we must determine how many figures to keep after the decimal or binary point in order to assure that round-off errors do not accumulate disastrously. More precisely, we must assure that the value of f is never in error by more than half a plotter unit, since any error less than half a unit will never cause a deviation of the pen from the calculated curve of more than one plotter step; this we regard as acceptable.

Accordingly, we now begin a fixed-point error analysis of the process defined by equations (3.4.28).

By definition,

$$f(i \times \text{incrx}) = f_i = f_0 + \sum_{j=0}^{i-1} \Delta f_j . \qquad 3.4.33$$

However, this equation is only theoretically valid; computationally, we may write

$$f_i^* = f_0^* + \sum_{j=0}^{i-1}{}^{*} \Delta f_j^*, \qquad 3.4.34$$

where f_i^* and Δf_j^* represent the <u>calculated</u> values of f_i and Δf_j,

and the asterisk on the summation sign indicates that a rounding error is made, in general, with the addition of each new term in the sum.

By our choice of co-ordinate axes, $f_0^* = f_0 = 0$. Further, since we carry only a certain (as yet not specified) number of digits in f, a maximum rounding error of, say, ν, is committed at each addition of a $\Delta^* f_j$ to the sum. Hence,

$$|f_i^* - f_i| \leq i\nu + \sum_{j=0}^{i-1} |\Delta^* f_j - \Delta f_j|. \qquad 3.4.35$$

Now, theoretically,

$$\Delta f_j = \Delta f_{-1} + \sum_{k=-1}^{j-1} \Delta^2 f_k, \qquad 3.4.36$$

but computationally,

$$\Delta^* f_j = \Delta^* f_{-1} + \sum_{k=-1}^{j-1} \Delta^{2*} f_k. \qquad 3.4.37$$

Again, since we carry only a restricted number of digits in Δf, we commit a maximum rounding error of, say, μ, with each addition of a $\Delta^{2*} f_k$, and also in the initial truncation of $\Delta^* f_{-1}$. Thus,

$$|\Delta^* f_j - \Delta f_j| \leq |\Delta^* f_{-1} - \Delta f_{-1}| + \left|\sum_{k=-1}^{j-1} (\Delta^{2*} f_k - \Delta^2 f_k)\right| + (j+1)\mu$$

$$\leq (j + 2)\mu + \sum_{k=-1}^{j-1} |\Delta^{2*} f_k - \Delta^2 f_k|. \qquad 3.4.38$$

Once again, we have theoretically,

$$\Delta^2 f_k = \Delta^2 f_{-2} + \sum_{l=-2}^{k-1} \Delta^3 f, \qquad 3.4.39$$

but computationally,

$$\Delta^{2*}f_k = \Delta^{2*}f_{-2} + \sum_{l=-2}^{k-1}{}^{*} \Delta^3 f. \qquad 3.4.40$$

Since we carry only a finite number of figures in delta2 and delta3 (the same for both - there is no advantage in carrying more figures in delta3 than in delta2, since the rounding error is in any case effectively determined by the number of figures in delta2), we commit a maximum rounding error of, say, ε with each addition of $\Delta^3 f$ to the sum and also with the initial truncation of $\Delta^{2*}f_{-2}$. Thus,

$$|\Delta^{2*}f_k - \Delta^2 f_k| \leq |\Delta^{2*}f_{-2} - \Delta^2 f_{-2}| + \left|\sum_{l=-2}^{k-1} (\Delta^3 f_l - \Delta^3 f_l)\right| +$$

$$+ (k + 2)\varepsilon$$

$$\leq (k + 3)\varepsilon. \qquad 3.4.41$$

Hence, substituting (3.4.41) into (3.4.38),

$$|\Delta^* f_j - \Delta f_j| \leq (j+2)\mu + \varepsilon \sum_{k=-1}^{j-1} (k + 3)$$

$$= (j + 2)\mu + \frac{(j+1)(j+4)}{2}\varepsilon, \qquad 3.4.42$$

and substituting this into (3.4.35),

$$|f_i^* - f_i| \leq i\nu + \mu \sum_{j=0}^{i-1} (j+2)\mu + \frac{\varepsilon}{2} \sum_{j=0}^{i-1} (j+1)(j+4)$$

$$= i\nu + \frac{i(i+3)}{2}\mu + \frac{i(i+1)(i+5)}{6}\varepsilon$$

$$< i\nu + \frac{(i+3)^2}{2}\mu + \frac{(i+5)^3}{6}\varepsilon\ . \qquad 3.4.43$$

Now, we require that this error be less than 1/2; we decide, rather arbitrarily, to spread the error equally among the three terms of (3.4.43). Thus, we require at the endpoint of the current section ($i = |xs1i|$),

$$|xs1i|\nu < 1/6 \quad ===> \quad \nu < \frac{1}{6|xs1i|},$$

$$\frac{(|xs1i| + 3)^2}{2}\mu < 1/6 \quad ===> \quad \mu < \frac{1}{3(|xs1i| + 3)^2}, \qquad 3.4.44$$

$$\frac{(|xs1i| + 5)^3}{6}\varepsilon < 1/6 \quad ===> \quad \varepsilon < \frac{1}{(|xs1i| + 5)^3}.$$

At this point, we narrow our consideration to a binary computer; the necessary changes for a decimal machine are obvious, however.

If we maintain a certain number, say n, of bits after the binary point, then the maximum error, m, during addition, assuming a proper rounding process, is clearly

$$m = \frac{1}{2} \cdot 2^{-n} = 2^{-n-1}. \qquad 3.4.45$$

Taking base 2 logarithms on both sides, we have

$$\log_2 m = -(n + 1), \qquad 3.4.46$$

and therefore, the number of bits required for a rounding error no greater than m, is

$$n = -\log_2 m - 1. \qquad 3.4.47$$

Thus, the number of extra bits, n_f, required in f is at least

$$n_f = -\log_2 \nu - 1 = +\log_2(6|xs1i|) - 1,$$

and similarly,

$$n_{\Delta f} = -\log_2 \mu - 1 = +\log_2(3(|xs1i| + 3)^2) - 1, \qquad 3.4.48$$

and,

$$n_{\Delta^2 f} = n_{\Delta^3 f} = -\log_2 \varepsilon - 1 = +\log_2((|xs1i| + 5)^3) - 1.$$

In order to reduce the probability of any error of even as much as one plotter step, we arbitrarily carry two further guarding bits in the above quantities and obtain finally:

$$n_f = \log_2(6|xs1i|) + 1,$$

$$n_{\Delta f} = \log_2(3(|xs1i| + 3)^2) + 1, \qquad 3.4.49$$

$$n_{\Delta^2 f} = n_{\Delta^3 f} = \log_2((|xs1i| + 5)^3) + 1.$$

In ALGOL, the bits after the binary point are expressed by carrying the quantities as integers, multiplied by 2 to the appropriate n from (3.4.49). This requires the calculation of a few auxiliary quantities, but the details should be clear from an inspection of the ALGOL procedure declaration. Note that most of the divisions by powers of 2 may actually be carried out, in machine code, by corresponding right shift operations, and multiplications by means of left shifts.

Similarly, the base 2 logarithms may be sufficiently accurately evaluated, in machine code, by using the binary exponent of the corresponding normalized floating-point quantity.

3.5 Theory of SCALE

The Case mode = 0

We begin by formalizing the requirements R1 to R4 [3]. Defining MN and MX as respectively the minimum and maximum values of the parameter Ti, we have the following conditions:

(C1) $DL = r_j \times 10^k$, for some integer k;

(C2) $MIN = p \times DL$, for some integer p;

(C3) (a) $MN - DL < MIN \leq MN$; $(MN < MX)$;

(b) $MAX = MIN + nint \times DL \geq MX$;

(C4) Of all values of DL satisfying C1, C2, and C3, the one chosen is that which has the highest efficiency, defined as

$$eff(DL) = \frac{MX - MN}{nint \times DL}.$$

It is immediately obvious from C4 that the smallest of the DL satisfying C1, C2, and C3 is the one desired (i.e. because MX, MN and nint are constant for each given problem).

This suggests that we might proceed by searching through an ordered set of candidates for DL, which satisfy C1, beginning with the smaller members, and continuing through successively larger candidates, up to the first one which satisfies all of the conditions C2, C3a, and C3b. This candidate is then the desired DL, and satisfies all the requirements.

It is clear that if, for a given candidate DL, we choose

$$MIN = entier\left(\frac{MN}{DL}\right) \times DL, \qquad 3.5.1$$

that this value of MIN automatically satisfies C2 and C3a.

Thus, it only remains to see how large a set of candidate round numbers is required to ensure that C3b is satisfied.

From C3b and the right-hand inequality of C2, we have

$$MN + nint \times DL \geq MX,$$

and hence,

$$DL \geq \frac{MX - MN}{nint} . \qquad 3.5.2$$

Consider the inequality

$$10^k \leq \frac{MX - MN}{nint} < 10^{k+1} . \qquad 3.5.3$$

We attempt to find a value of k satisfying (3.5.3). Taking logarithms, we have

$$k \leq \log_{10} \left(\frac{MX - MN}{nint} \right) < k + 1. \qquad 3.5.4$$

From the right-hand inequality of (3.5.4),

$$k > \log_{10} \left(\frac{MX - MN}{nint} \right) - 1, \qquad 3.5.5$$

and thus,

$$\log_{10} \left(\frac{MX - MN}{nint} \right) - 1 < k \leq \log_{10} \left(\frac{MX - MN}{nint} \right). \qquad 3.5.6$$

Now

$$k = entier \left(\log_{10} \left(\frac{MX - MN}{nint} \right)\right) \qquad 3.5.7$$

satisfies the inequality (3.5.6) exactly. Thus, with this choice of k, if we begin the search at

$$Rmin = 1 \times 10^k, \qquad 3.5.8$$

and continue sufficiently far, then we are sure that the first candidate satisfying (3.5.2) will be encountered.

Next, we consider what is meant by "sufficiently far". From C3b and the left-hand inequality of C2, we have that C3b is certainly satisfied if

$$MN - DL + nint \times DL \geq MX;$$

that is, if

$$DL \geq \frac{MX - MN}{(nint - 1)} \quad . \qquad 3.5.9$$

That is, the search may be discontinued at any round number

$$Rmax \geq \frac{MX - MN}{(nint - 1)}$$

$$= \frac{MX - MN}{nint} \times \frac{nint}{nint - 1} \quad . \qquad 3.5.10$$

Result (3.5.9) indicates that for nint = 1, the search may continue indefinitely. In fact, the requirements C2 and C3 may easily be shown to be in general inconsistent for nint = 1. Thus, we require nint $\geq$ 2.

With this requirement, we then have

$$\frac{nint}{nint - 1} \leq 2, \qquad 3.5.11$$

with the equality holding only for nint = 2.

Thus, from the right-hand inequality of (3.5.3), and from (3.5.10) and (3.5.11), we may be assured that the first round number satisfying

$$Rmax \geq 2 \times 10^{k+1} = 20 \times 10^{k} \qquad 3.5.12$$

will certainly satisfy C3b.

Accordingly, if we expand our set of "basic round numbers" r_j as follows

$$1, 1.25, 2, 2.5, 4, 5, 8, 10, 12.5, 20, 25 \qquad 3.5.13$$

(the 25 is included in order to be absolutely safe against rounding errors), then the following algorithm will solve the problem:

```
k: = entier(log10((MX - MN)/nint));
for rj: = 1, 1.25, 2, 2.5, 4, 5, 8, 10, 12.5, 20, 25 do
begin DL: = rj × 10 ↑ k;
      if DL > (MX - MN)/nint then
      begin MIN: = entier(MN/DL) × DL;
            MAX: = MIN + nint × DL;
            if MAX ≥ MX then goto out
      end
   end;
out:
```

The Case mode = 1

As motivation for the inclusion of the extra features provided by mode = 1, we consider the problem defined by

$$MN = -40.1,\ MX = 35,\ nint = 15. \qquad 3.5.14$$

Application of the algorithm described above yields

$$MIN = -48,\ DL = 8,\ \text{and } MAX = 72, \qquad 3.5.15$$

with an efficiency of only 63 %. The reason for this rather low efficiency is easy to discover - namely, there are four "empty" intervals from 40 to 72. This is a consequence of requiring precisely 15 intervals.

If, however, the user is willing to accept some number of intervals,

$$ni \leq nint, \qquad 3.5.16$$

and to replace the condition C3b by

(C3c) $$MAX = MIN + ni \times DL \geq MX,$$

then it will often be possible to obtain a higher efficiency, now defined as

$$eff(DL) = \frac{MX - MN}{ni \times DL}, \qquad 3.5.17$$

by simply discarding the empty intervals.

From C3c, we have that

$$ni \geq \frac{MX - MIN}{DL} \quad , \qquad 3.5.18$$

and the value we choose, which certainly satisfies (3.5.18), is

$$ni = entier\left(\frac{MX - MIN}{DL} \right) + 1. \qquad 3.5.19$$

Applying this to the problem (3.5.14), we obtain the solution

$$MIN = -48,\ DL = 8,\ MAX = 40,\ ni = 11, \qquad 3.5.20$$

with an efficiency of 85 %.

We shall now obtain a lower bound for ni. Consider

$$r_{j-1}\ 10^k < \frac{MX - MN}{nint} \leq r_j\ 10^k, \qquad 3.5.21$$

where $r_j\ 10^k$ is the round number actually used as DL, and $r_{j-1}\ 10^k$ is the last round number which was too small to satisfy C3b. From (3.5.18), and the left-hand inequality of (3.5.21), we have

$$ni > nint \times \frac{r_{j-1}}{r_j} \quad . \qquad 3.5.22$$

In our chosen set of basic round numbers,

$$\frac{r_{j-1}}{r_j} = .8 \quad \text{or} \quad .625 \qquad 3.5.23$$

of which the second is smaller, and hence ni is always larger than .625 × nint. Hence, for mode = 1,

$$.625 \times nint < ni \leq nint. \qquad 3.5.24$$

The Case mode = 2

The left-hand inequality of (3.5.21) leads us to wonder if we might not obtain still higher efficiency by using DL1 = $r_{j-1} \times 10^k$

instead of DL = $r_j \times 10^k$ (i.e. by using the largest round number which fails to satisfy C3b), and then calculating a

$$MIN1 = \text{entier}\left(\frac{MN}{DL1}\right) \times DL1, \qquad 3.5.25$$

and a

$$ni1 = \text{entier}\left(\frac{MX - MIN1}{DL1}\right) + 1, \qquad 3.5.26$$

and finally, a

$$MAX1 = MIN1 + ni1 \times DL1. \qquad 3.5.27$$

In the procedure SCALE, for the case mode = 2, we calculate the quantities described above, and compare the efficiency of the solution so obtained with that obtained for the case mode = 1; we then choose the more efficient of the two.

We now obtain bounds for the number of intervals which this process may produce. Clearly, if the mode = 1 solution is more efficient, we have from (3.5.24), the lower bound

$$.625 \times nint < \text{number of intervals}. \qquad 3.5.28$$

Now, the efficiency of the choice DL1 is

$$eff(DL1) = \frac{MX - MN}{ni1 \times DL1}, \qquad 3.5.29$$

and, since DL1 is chosen only if eff(DL1) > eff(DL), i.e. only if

$$\frac{eff(DL1)}{eff(DL)} = \frac{ni \times DL}{ni1 \times DL1} > 1, \qquad 3.5.30$$

therefore

$$ni1 < ni \times \frac{DL}{DL1}$$

$$< nint \times \frac{r_j \times 10^k}{r_{j-1} \times 10^k}$$

$$< 1.6 \times nint. \qquad 3.5.31$$

Thus, combining (3.5.28) and (3.5.31),

$$.625 \times nint < \text{number of intervals} < 1.6 \times nint. \qquad 3.5.32$$

We have now demonstrated the three possible modes of the procedure SCALE, each of which gives higher efficiency than the previous one, at the price of allowing the procedure more freedom in the choice of the number of intervals.

3.6 Theory of PLOTAXIS2

In this section we discuss the method by which several of the features of PLOTAXIS2 are accomplished.

We introduce the concept of the "position number" of a digit, relative to the decimal point, within a number. For instance, in the number

12345.6789,

the position of the 5 is 0, of the 4 is 1, of the 3 is 2, of the 6 is -1, of the 7 is -2, etc. From the figure

```
. . . X  X  X  X  X  .  X  X  X  X  X . . .
      4  3  2  1  0     -1 -2 -3 -4 -5
```

it is easy to see that the position number of the left-most non-zero digit of an arbitrary number x is the characteristic of the base 10 logarithm of the absolute value of x. That is, naming this position number λ,

$$\lambda = \text{entier}(\log_{10} |x|). \qquad 3.6.1$$

The problem of finding the position of the last (i.e. right-most, non-zero) digit of an arbitrary real number is not well-defined. In the context of the procedure PLOTAXIS2, however, we are interested in the position of the last "significant" digit of the number, since we usually wish to draw only the "significant" digits of the numbers by the tick marks on the axis.

This intuitive meaning of "significant" may be made clearer by considering DL, the interval between tick marks. Ideally, DL contains only one, two, or three significant digits (see the description of the procedure SCALE). Clearly, since all of the numbers to be drawn by the tick marks are integer multiples of DL, then all digits to the right of the last significant digit

of DL are theoretically zeroes, and need not (usually) be drawn; note that because of rounding errors, there will usually be at the end of the number, a sequence of zeroes or nines followed by some non-zero digits, but as long as we do not draw them, no harm will be done.

Thus, we must determine the position of the least significant digit of DL. This is accomplished as follows:
we calculate

$$\delta = DL \times 10^{(2 - \text{entier}(\log_{10}(DL)))}; \qquad 3.6.2$$

that is, an integer less than 1000. Then the number of significant digits of DL is

$$\nu = 3 - \text{number of trailing zeroes of } \delta, \qquad 3.6.3$$

and the position of the least significant digit is then

$$\begin{aligned} k2 &= \text{entier}(\log_{10}(DL)) - (\nu - 1) \\ &= \text{entier}(\log_{10}(DL)) - 2 + \text{number of trailing zeroes of } \delta. \end{aligned} \qquad 3.6.4$$

The determination of the number of trailing zeroes of an arbitrary integer is accomplished in PLOTAXIS2 by the integer procedure zeroes.

Now, the position of the left-most digit of the number with the greatest number of digits is

$$k1 = \text{entier}(\log_{10}(\max(|MAX|, |MIN|))), \qquad 3.6.5$$

and the total number of significant digits in this number is

$$k3 = k1 - k2 + 1. \qquad 3.6.6$$

We now consider the question of the drawing of a power of 10 along with the other numbers. One may be tempted to do this whenever there are zeroes between the decimal point and the significant digits of the number (i.e. whenever $k2 > 0$ or $k1 < -1$).

In PLOTAXIS2, however, the power of 10 is drawn only if the number of significant digits plus the number of zeroes between the point and the significant digits exceeds 5 (the value 5 is rather arbitrarily chosen). More precisely, a power of 10 is provided if

$$((k2 > 0) \wedge (k1 \geq 5)) \vee ((k1 < -1) \wedge (k2 \leq -5)). \qquad 3.6.7$$

At this point, we know the number of characters (including possible decimal point and minus sign) which have to be drawn possibly for all the numbers and certainly for the largest. Thus we know the physical length of the numbers to be drawn; dividing this physical length (plus a suitable safety margin) by the physical distance between successive tick marks, we obtain the quantity step with the interpretation that if step $\leq$ 1 there is sufficient room between successive tick marks for each number, and that otherwise, at least (step -1) tick marks must be skipped over between two successive numbers.

However, there may be objections to simply beginning at MIN and drawing at each following step'th position. For instance, consider the case MIN = 0, DL = 2, and step = 4; that is,

0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, ...

Here, we would have the numbers 0, 8, 16, 24, ... whereas, if we use step = 5, we have, instead, the numbers 0, 10, 20, 30, ... The latter choice appears esthetically more desirable. Consider also the case MIN = -25, DL = 25, and step = 4; that is,

-25, 0, 25, 50, 75, 100, 125, 150, 175, 200, 225, ...

Here, we would have -25, 75, 175, ..., whereas, if we begin at 0, and keep the step = 4, we have 0, 100, 200, ..., etc., and again, this seems esthetically attractive.

It would thus appear, from the preceding paragraph, that we wish to see, as far as possible, numbers which are integer multiples of powers of ten. Clearly, if step $\leq$ 1, there is no problem,

but if step > 1, then we must consider precise criteria for choosing a revised step, and a beginning position other than MIN.

We now digress briefly to consider some simple number theory. Each number to be drawn is of the form

$$MIN + i \times DL,$$

but since MIN is itself an integer multiple of DL, we can write all the numbers to be drawn in the form

$$N_j = j \times DL, \qquad 3.6.8$$

with some appropriate bounds for j.

Suppose that we have found by some means that the last k significant digits of the number N_n are zeroes; that is,

$$N_n = n \times DL = \alpha \times 10^{k-k2}, \qquad 3.6.9$$

where α is an integer. Suppose further that ν is the <u>smallest</u> (positive) integer such that

$$N_{n+\nu} = N_n + \nu \times DL = \beta \times 10^{k-k2}, \qquad 3.6.10$$

where β is some integer; that is, $N_{n+\nu}$ is the next number after N_n of which the last k (or more) significant digits are zeroes. We then consider the numbers

$$N_{n+m\nu} = N_n + m\nu \times DL, \qquad 3.6.11$$

for all integers m. From (3.6.9) and (3.6.10), we have that

$$\nu \times DL = (\beta - \alpha) \times 10^{k-k2} \qquad 3.6.12$$

and hence, from (3.6.11) and (3.6.12),

$$N_{n+m\nu} = (\alpha + m(\beta - \alpha)) \times 10^{k-k2}$$

$$= \delta \times 10^{k-k2}, \qquad 3.6.13$$

where δ is clearly again an integer.

That is, all of the numbers $N_{n+m\nu}$ have the property of having zeroes for at least the last k significant digits.

Furthermore, there are no other numbers with this property; for, suppose that N_p is a number that does have the property. Then, clearly all numbers $N_{p+m\nu}$ also have the property, but one of these lies closer to N_n than $N_{n+\nu}$, contrary to the assumption that $N_{n+\nu}$ was the closest.

The significance of this is that two numbers with at least a prescribed number of trailing zeroes, and between which no other number has at least that prescribed number of trailing zeroes, determine a sequence which includes all (and only) numbers with at least as many zeroes.

In the ALGOL text of PLOTAXIS2 between the labels CYCLE and gotthem, the following process is performed:

(a) The unique smallest cycle which has a length (i.e. ν) at least as large as step is calculated by means of a rather complicated loop. The loop produces pos, the index of a number in the cycle, and the cycle-length cycle;

(b) Further, we calculate the smallest integer divisor of cycle which is at least as great as step (in the program, this is called itemp);

(c) In the case that no sufficiently large cycle exists, we simply multiply the largest cycle which does exist by successive integers 2, 3, ... until it is large enough;

(d) DL is replaced by itemp × DL, and appropriate new values of MIN and MAX are calculated;

(e) Control is transferred back to the beginning of the procedure in order that the decision about the drawing of a power of 10 may be reviewed (that is, it may no longer be necessary).

The iteration mentioned in (e) is performed only once. After this, control is directed to gotthem, where the actual drawing of the numbers and power of 10 is performed.

4. Sample Application Programs

Note that the declarations of the plotter procedures (and the declarations of state1, state2, etc.), have, for reasons of economy, been omitted in the listings of the sample programs.

4.1 Circle and Ellipse

In data units the co-ordinates of 32 points on the circle r = 500 are calculated and then plotted by PLOTCURVE; then, a square is drawn around the circle.

This is done twice, the first time with equal scaling factors Scx and Scy, and the second time with Scx = 2 × Scy. With equal scaling factors the circle and square appear as a circle and square on the paper; however, in the other case, they appear as an ellipse and a rectangle.

4.2 Error Damping by PLOTCURVE

This program demonstrates that, although a discontinuity of a curve cannot be reproduced very precisely by PLOTCURVE, it nevertheless has no influence on the interpolation process at a distance from the discontinuity.

Specifically, we have two straight lines making an angle of 45, 90, and 135 degrees with each other. PLOTCURVE follows the first line faithfully to the last point before the "kink", and from the first point thereafter.

At the same time, the program demonstrates the use of MARKEDCURVE, and some of the features of PLOTTEXT.

4.3 Demonstrate PLOTPICTURE

The first call of PLOTPICTURE draws an axis system, complete with rectangular grid, and two cycles of the curve $y = \sin(2\pi x)$. The curve is given by means of the co-ordinates of 65 points equally spaced in x, and by the initial and final slopes (both equal to 2π). Every fourth given point is marked with one of the special marking symbols. The scaling of the X-axis has been completely specified (kx = 6) by the requirements that it be 16 cm (1600 plits) long, and with vertical grid lines at the points 0, .25, .5, ..., 2. The scaling of the Y-axis has been specified by requiring it to be 12 cm long, and with horizontal grid lines every 1 cm; the minimum and maximum values of the function are estimated (rather conservatively!) to be -1.01 and +1.01 respectively. Appropriate texts are drawn beside the axes; notice that a few non-ALGOL characters are represented in the strings by corresponding word-delimiters.

The second call draws two cycles of the curve $y = \cos(2\pi x)$ using the axis system, etc. established by the previous call.

Following this, values of x and $\cot(2\pi x)$ are calculated for $x = 1/64, 2/64, \ldots, 31/64$, and stored in arrays.

The third call of PLOTPICTURE draws a new Y-axis, and one cycle of the curve $y = \cot(2\pi x)$. The curve is defined by the previously calculated and stored co-ordinate pairs. The scaling of the Y-axis is determined by the requirements that it be 12 cm long, with a tick-mark every 1 cm, and that the y-values must "fit" on the axis, with as little wasted space as possible.

The following three calls (in the for-loop) reproduce the curve produced by the third call, at positions successively farther to the right. The axis system defined and drawn by the third call is used by these last three calls.

4.4 Digits of π in a Spiral

We choose (because of esthetic considerations which are not important here), the exponential spiral

$$r(\phi) = rmax \times e^{p\phi}\ , \qquad 4.4.1$$

with rmax = 650 plotter units, p = .023, and with ϕ taking on values from 0 to approximately -32π.

The idea is to write the first 1000 digits of the decimal expansion of π with the feet of the digits following the spiral. The height of each digit is to be some fixed proportion, say 75 %, of the available distance to the digits from the previous revolution; that is, the height of a digit at the position ϕ of the spiral is to be

$$h(\phi) = .75 \times (r(\phi + 2\pi) - r(\phi))$$

$$= .75 \times rmax \times e^{p\phi} \times (e^{2p\pi} - 1), \qquad 4.4.2$$

and the breadth of the digit (including the space to its successor) is

$$b(\phi) = \frac{6}{7} h(\phi) = c \times e^{p\phi}, \qquad 4.4.3$$

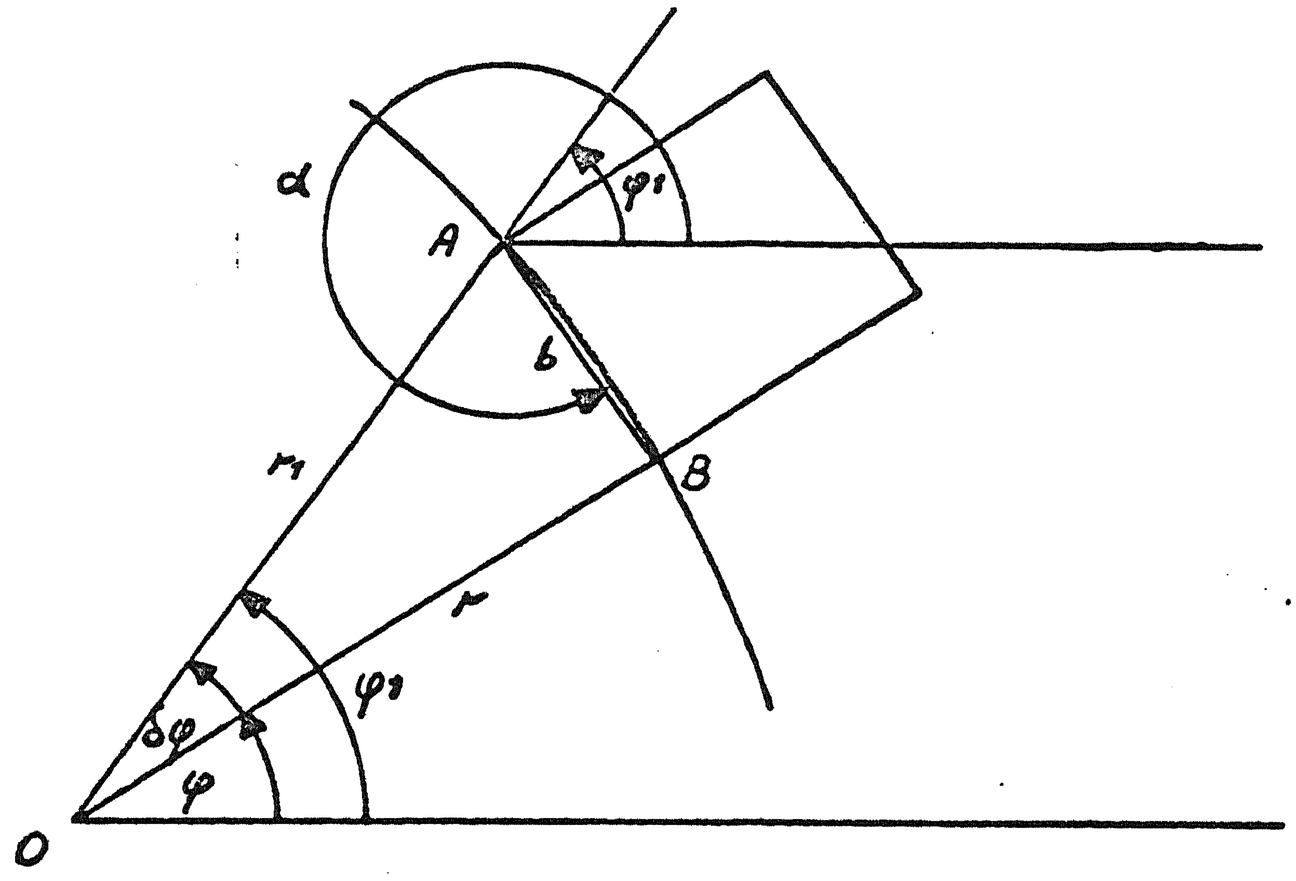
where

$$c = \frac{6}{7} \times .75 \times rmax \times (e^{2p\pi} - 1). \qquad 4.4.4$$

Because of the transcendental character of the spiral, we can not make direct use of the above formulas; instead, we proceed approximately.

Referring to the diagram, we assume that the radius r_1 and angle ϕ_1 have been calculated during the drawing of the previous digit. We then calculate the <u>desired</u> breadth of the new digit as

$$b_1 = ce^{p\phi_1}\ . \qquad 4.4.5$$



Using the approximation

$$r_1 \delta\phi = b_1, \qquad 4.4.6$$

we calculate

$$\delta\phi = b_1/r_1 = ce^{p\phi_1}/r_1, \qquad 4.4.7$$

and

$$\phi = \phi_1 - \delta\phi, \qquad 4.4.8$$

and

$$r = rmax \times e^{p\phi}. \qquad 4.4.9$$

This implies that the actual breadth of the digits (i.e., the chord AB) will be somewhat different from the desired breadth b_1, but the difference will not be readily visible to the eye. Nevertheless, we must calculate the actual breadth b (in order to calculate the actual height).

From the cosine law of plane trigonometry,

$$b = \sqrt{r^2 + r_1^2 - 2rr_1\cos(\delta\phi)}. \qquad 4.4.10$$

We also require the angle α. Clearly,

$$\alpha = \phi_1 + \pi + \measuredangle BAO, \qquad 4.4.11$$

and, from the sine law

$$\frac{\sin(\measuredangle BAO)}{r} = \frac{\sin(\delta\phi)}{b}\ ; \qquad 4.4.12$$

i.e.

$$\measuredangle BAO = \arcsin\left(\frac{r}{b}\sin(\delta\phi)\right). \qquad 4.4.13$$

Furthermore, the arcsin can be calculated from

$$\arcsin(arg) = \arctan\left(\frac{arg}{\sqrt{1 - arg^2}}\right). \qquad 4.4.14$$

Clearly, the height of the digit is

$$h = \frac{7}{6} \times b. \qquad 4.4.15$$

Finally, the values of r and ϕ must be preserved as r_1 and ϕ_1 for the follcwing digit.

The above mentioned process is performed by the procedure height, which is called by PLOTTEXT just before drawing each digit.

Note that the string to be plotted (the digits of π) contains a blank or carriage return after each ten digits. Since we do not wish to draw these spaces, we set height to zero for each eleventh character, with the help of the counter j.

4.5 Draw all PLOTTEXT Characters

This program was used to produce Table 3, showing all of the 102 characters which can be drawn by PLOTTEXT. Note that all three pages of the table are produced at the same time.

```
real procedure PLOT(X,Y,IPEN); value X, Y, IPEN; integer IPEN; real X,Y;
begin   integer x, y, xy, dx, dy, dx2, dy2, i, dev, absipen; boolean right;
        own integer xlast, ylast, xmax, ymax; own boolean pen; own real Scx, Scy, XMIN, YMIN;
        switch switch:= IPEN0,IPEN1,IPEN2,IPEN3,IPEN4,IPEN5,IPEN6,IPEN7,IPEN8,IPEN9,IPEN10,IPEN11,IPEN12,
            IPEN13,IPEN14,IPEN15,IPEN16,IPEN17,IPEN18,IPEN19,IPEN20,IPEN21,IPEN22,IPEN23,IPEN24,IPEN25;
        comment MOVE(alpha) is a boolean procedure (X1-code) which moves the pen in the direction specified
                by the integer alpha. MOVE is false if the plotter carriage is against one of the edges of the
                paper and we are attempting to go further against that edge, otherwise true.
                The following values for alpha are allowed:
                    1       + x direction
                    2       - x direction
                    4       + y direction
                    8       - y direction
                   16       + z direction (pen up)
                   32       - z direction (pen down)
                    5       + x and + y direction simultaneously
                    9       + x and - y direction simultaneously
                    6       - x and + y direction simultaneously
                   10       - x and - y direction simultaneously;
if firstentrance then firstentrance:= state1:= state2:= state4:= state8:= state16:= state32:= state64:= false;
absipen:= abs(IPEN);
if absipen ≥ 20 ∨ absipen = 0 then goto switch[(if absipen > 25 then 0 else absipen) + 1];
if ¬(state1 ∧ state2 ∧ state4 ∧ state8 ∧ state16) then STOP(‡1 - 26‡);
```

comment PLOT second page;

```
        if IPEN > 0 ∧ IPEN ≤ 6 then begin x:= (X - XMIN)/Scx; y:= (Y - YMIN)/Scy end
                               else if absipen ≤ 8 then begin x:= X; y:= Y end;
        goto switch[absipen + 1];
IPEN0 : STOP(⌊1 - 25⌋);
IPEN1 : if   pen then begin MOVE(32); pen:= false end; goto IPEN3;
IPEN2 : if ¬ pen then begin MOVE(16); pen:= true  end;
IPEN3 : if x < 0 ∨ y < 0 ∨ y > ymax ∨ (x > xmax ∧ absipen < 4) then STOP(⌊1 - 27⌋);
        dx:= x - xlast; dy:= y - ylast; xlast:= x; ylast:= y; right:= dx > 0;
        if dx < 0 then begin dx:= -dx; x:= 2 end else x:= 1;
        if dy < 0 then begin dy:= -dy; y:= 8 end else y:= 4; xy:= x + y;
        if dy > dx then begin x:= y; i:= dx; dx:= dy; dy:= i end;
        dx2:= dx × 2; dy2:= dy × 2; dev:= 0;
        for i:= 1 step 1 until dx do
        begin dev:= dev + dy2; if (if right then dev < dx else dev ≤ dx) then MOVE(x)
                                                                        else begin MOVE(xy); dev:= dev - dx2 end
        end;
        if absipen = 4 then xlast:= ylast:= 0; goto end;
IPEN4 : if IPEN < 0 ∧ x < 0 then begin x:= xmax + 3000; y:= 0 end; goto IPEN2;
IPEN5 : PLOT:= x; PLOT51: if x < 0 ∨ x > xmax then STOP(⌊1 - 27⌋); goto endd;
IPEN6 : PLOT:= y; PLOT61: if y < 0 ∨ y > ymax then STOP(⌊1 - 27⌋); goto endd;
IPEN7 : PLOT:= x × Scx + XMIN; goto PLOT51;  IPEN8 : PLOT:= y × Scy + YMIN; goto PLOT61;
```

```
comment PLOT third page;

IPEN9 : PLOT:= XMIN; goto endd;                    IPEN10: PLOT:= YMIN; goto endd;
IPEN11: PLOT:=   Scx; goto endd;                   IPEN12: PLOT:=   Scy; goto endd;
IPEN13: PLOT:= xmax; goto endd;                    IPEN14: PLOT:= ymax; goto endd;
IPEN15: PLOT:= xlast; goto endd;                   IPEN16: PLOT:= ylast; goto endd;
IPEN17: PLOT:= xlast × Scx + XMIN; goto endd;      IPEN18: PLOT:= ylast × Scy + YMIN; goto endd;
IPEN19: PLOT:= if pen then 1 else -1; goto endd;
IPEN20: x:= X; MOVE(x); state8:= state16:= false; goto end;
IPEN21: XMIN:= X; YMIN:= Y; state1:= true;
        if IPEN > 0 then begin state16:= pen:= true; MOVE(16) end; goto end;
IPEN22: Scx:= X; Scy:= Y; state2:= true; goto end;
IPEN23: xmax:= X; ymax:= Y; state4:= true;
        if IPEN > 0 then begin   state8:= true; xlast:= ylast:= 0; EDGE: if MOVE(8) then goto EDGE;
                                 for i:= (2794 - ymax) : 2 step - 1 until 1 do MOVE(4)
                         end;
        goto end;
IPEN24: xlast:= X; ylast:= Y; state8:= true; goto end;
IPEN25: pen:= X > 0; state16:= true;
end:    PLOT:= 0;
endd:
end PLOT;
```

```
procedure PLOTFRAME(XMIN,YMIN,XMAX,YMAX,xmax,ymax); value XMIN, YMIN, XMAX, YMAX xmax,ymax;
        real  XMIN, YMIN, XMAX, YMAX; integer xmax, ymax;
begin   if ymax > 2750 then begin STOP(¢1 - 30¢); ymax:= 2750 end;
        PLOT(XMIN,YMIN,21);
        PLOT((XMAX - XMIN)/xmax,(YMAX - YMIN)/ymax,22);
        PLOT(xmax,ymax,23)
end PLOTFRAME;
```

```
procedure PLOTTEXT(X,Y,angle,height,italicity,first,i,text); value X, Y, first, i;
        real X, Y, angle, height, italicity; Boolean first; integer i; string text;
begin   integer IPEN, j, k, n, absi, xki; real hecosan, hesinan, x, y, xk, yk, an, he, it, abshe, tanit, cosit;
        own real x0, y0; own Boolean shift;
        comment The integer procedure STRINGSYMBOL(j,text) is a code procedure which delivers as its value
        the integer representation of the j'th ALGOL symbol in the string text, j counting from zero. It delivers
        the value 255 in place of the last closing string quote;
        j:= -1; absi:= abs(i);
        if absi > 1 then begin if absi > 12 then goto MISTAKE; absi:= absi + 102; goto SYMBOL end;
STRING:         j:= j + 1; absi:= STRINGSYMBOL(j,text); if absi = 255 then goto end;
SYMBOL:         he:= height/7; abshe:= abs(he); an:= angle × .0174532925199; it:= italicity × .0174532925199;
        hecosan:= abshe × cos(an); hesinan:= abshe × sin(an);
        cosit:= cos(it); tanit:= sin(it)/cosit; cosit:= sign(cosit);
        if j ≤ 0 then   begin   if first
                        then    begin  first:= shift:= false; state64:= true;
                                if he < 0
                                then begin x0:= X; y0:= Y end
                                else begin x0:= PLOT(X,0,5); y0:= PLOT(0,Y,6) end
                        end
                        else    if ¬state64 then MISTAKE: begin STOP(‡1 - 31‡); goto end end;
                        if i < 0 ∧ ¬shift
                        then    begin  x0:= x0 - (2 × hecosan - 3 × hesinan) × cosit;
                                y0:= y0 - (3 × hecosan + 2 × hesinan) × cosit; shift:= true
                        end
```

```
                    else    if i ≥ 0 ∧ shift then
                            begin   x0:= x0 + (2 × hecosan - 3 × hesinan) × cosit;
                                    y0:= y0 + (3 × hecosan + 2 × hesinan) × cosit; shift:= false
                            end
                end if j ≤ 0;
IPEN:= -2; x:= x0; y:= y0; k:= ENTRANCE[absi]; n:= k + N[absi] - 1;
for k:= k step 1 until n do
begin   xki:= CX[k]; yk:= CY[k] × cosit;
        if xki < 5
then    begin  xk:= xki + yk × tanit;
               PLOT(x + xk × hecosan - yk × hesinan,y + yk × hecosan + xk × hesinan, IPEN);
               IPEN:= -1
        end
else    if xki = 7
        then    IPEN:= -2
        else    if xki = 6
                then    begin  x:= x - yk × (hesinan - tanit × hecosan);
                               y:= y + yk × (hecosan + tanit × hesinan)
                        end if xki = 6
```

```
                                    else    begin   x:= x + yk × (hesinan - tanit × hecosan);
                                                    y:= y - yk × (hecosan + tanit × hesinan)
                                            end if xki = 5
        end for k;
        x0:= x0 + 6 × hecosan; y0:= y0 + 6 × hesinan; goto STRING;
end:
end PLOTTEXT;
```

```
procedure PLOTAXIS(X,Y,angle,LENGTH,DL); value X, Y, angle, LENGTH, DL; real X, Y, angle, LENGTH, DL;
begin   real sina, cosa, markx, marky, temp, dlsina, dlcosa, absdl;
        angle:= angle × 0.0174532925199; sina:= sin(angle); cosa:= cos(angle);
        if abs(sina) < 2↑(-16) then sina:= 0 else if abs(cosa) < 2↑(-16) then cosa:= 0;
        if LENGTH < 0  then     LENGTH:= -LENGTH
                       else     begin   X:= (X - PLOT(0,0,9))/PLOT(0,0,11);
                                        Y:= (Y - PLOT(0,0,10))/PLOT(0,0,12);
                                        temp:= sqrt((PLOT(0,0,11) × cosa)↑2 + (PLOT(0,0,12) × sina)↑2);
                                        LENGTH:= LENGTH/temp; DL:= DL/temp
                                end;
        PLOT(X,Y,-2); absdl:= abs(DL); if absdl > LENGTH then goto end;
        temp:= if sina > cosa then 15 else -15;
        markx:= temp × sina; marky:= -temp × cosa; dlsina:= absdl × sina; dlcosa:= absdl × cosa;
LOOP:   PLOT(X + markx,Y + marky,-1); if DL < 0 then PLOT(X - markx,Y - marky,-3);
        PLOT(X,Y,-3);
        LENGTH:= LENGTH - absdl; if LENGTH ≥ -1 then begin  X:= X + dlcosa; Y:= Y + dlsina;
                                                            PLOT(X,Y,-3); goto LOOP
                                                end
                                                else LENGTH:= LENGTH + absdl;
end:    PLOT(X + LENGTH × cosa,Y + LENGTH × sina,-1)
end PLOTAXIS;
```

```
real procedure PLOTCURVE(X, Y, I); value X, Y, I; real X, Y; integer I;
begin   real temp, temp2, a, b, sc1, sc2, xs1, xs2, ys1, ys2, max1, max2, y0p, x0, y0;
        integer  x, y, which, which1, which2, which3, dif, absdif, incrx, f, delta,
                 delta2, delta3, x0i, y0i, xs1i, ys1i, ln1, ln2, ln3, isc1, isc2, isc3,
                 isc4, isc5, isc6, isc7, absxs1, xsmin, ysmin, xsmax, ysmax;
        Boolean small, huge, firsttry, oksofar;
        own real y1p, scalef, x1, x2, y1, y2;
        own integer count;
        own Boolean deriv, xisx;
        own integer array ACT[0:23];
        real procedure log2(x); real x; log2 := ln(x) / ln(2);
        PLOTCURVE := 0;
INIT:   if I ≤ 1 then   begin   if I < 0 then oops:begin STOP(‡1-28‡); go to exit end;
                                deriv := I = 0; xisx := true;
                                count := 0; x1 := x2 := y1 := y2 := 0;
                                ACT[0] := ACT[1] := ACT[20] := ACT[22] := 1;
                                ACT[2] := ACT[3] := ACT[21] := ACT[23] := 2;
                                ACT[4] := ACT[5] := ACT[16] := ACT[18] := 4;
                                ACT[8] := ACT[12]:= 5;
                                ACT[10]:= ACT[13]:= 6;
                                ACT[6] := ACT[7] := ACT[17] := ACT[19] := 8;
                                ACT[9] := ACT[14]:= 9;
                                ACT[11]:= ACT[15]:=10;
```

```
comment PLOTCURVE second page;

                                        scalef := PLOT(0, 0, 11) / PLOT(0, 0, 12);
                                        if deriv then y0p := X × scalef;
                                        state32 := true; go to exit
                                end if I ≤ 1;
        CHECK:if I > 4 ∨ ¬state32 ∨ (I ≥ 3 ∧ count ≠ 4) then go to oops;
        SHIFT:  x0 := x1; y0 := y1;
                x1 := x2; y1 := y2;
NEWPOINT:       if I = 2 then
                begin   x2 := (X - PLOT(0, 0, 9)) / PLOT(0, 0, 11);
                        y2 := (Y - PLOT(0, 0,10)) / PLOT(0, 0, 12)
                end;
                if count < 3 then begin   count := count + 1;
                                          go to if count = 3 then position else exit
                                  end
                             else         y0p := y1p;
position:       x0i := x0; y0i := y0;
                if x0i ≠ PLOT(0, 0, 15) ∨ y0i ≠ PLOT(0, 0, 16) then PLOT(x0i, y0i, -2);
PENDOWN:        if PLOT(0, 0, 19) > 0 then PLOT(x0i, y0i, -1);
                firsttry := true;
```

```
comment PLOTCURVE third page;

      scale:  if xisx then      begin   xs1 := x1 - x0; ys1 := y1 - y0; xs2 := x2 - x0; ys2 := y2 - y0;
                                        xsmin := -x0i; xsmax := PLOT(0, 0, 13) - x0i;
                                        ysmin := -y0i; ysmax := PLOT(0, 0, 14) - y0i
                                end
                        else    begin   xs1 := y1 - y0; ys1 := x1 - x0; xs2 := y2 - y0; ys2 := x2 - x0;
                                        xsmin := -y0i; xsmax := PLOT(0, 0, 14) - y0i;
                                        ysmin := -x0i; ysmax := PLOT(0, 0, 13) - x0i
                                end;
              if firsttry then
      SLOPE:begin       if I = 3
      then    begin    temp := 2 × xs1 × ys1;
                       temp2 := ys1 ↑ 2 - xs1 ↑ 2;
                       y1p := (temp + y0p × temp2) / (temp × y0p - temp2)
              end
      else    begin    if I = 4
              then     y1p := if xisx then X × scalef else 1 / (X × scalef)
              else     begin   temp := xs2 × (xs2 - xs1) + ys2 × (ys2 - ys1);
                               y1p := (ys2 × xs1 ↑ 2 + ys1 × (temp - xs1 × xs2))
                                    / (xs2 × ys1 ↑ 2 + xs1 × (temp - ys1 × ys2))
                       end
              end;
```

```
comment PLOTCURVE fourth page;

                    if count = 3 then
                    begin   count := 4; if ¬deriv then
BEGINSLOPE:                 begin   temp := xs1 ↑ 2 + ys1 ↑ 2;
                                    temp2:= xs2 ↑ 2 + ys2 ↑ 2;
                                    y0p := (ys1 × temp2 - ys2 × temp)
                                         / (xs1 × temp2 - xs2 × temp)
                                end
                    end;
                    max1 := abs(y0p); temp := abs(y1p);
INTERCHANGE:        if max1 > temp then max2 := 1 / temp
                              else  begin   max2 := 1 / max1; max1 := temp end;
                    if max1 > max2 then   begin   xisx := ¬xisx; y0p := 1 / y0p; y1p := 1 / y1p;
                                                  firsttry := false; go to scale
                                          end
                    end if firsttry;
POLYNOM:            temp := entier(x1 + .5) - x0i;
                    temp2:= entier(y1 + .5) - y0i;
                    xs1i := if xisx then temp else temp2;
                    ys1i := if xisx then temp2 else temp;
                    incrx := sign(xs1i);
                    temp := xs1i ↑ 2;
                    temp2 := y1p - 2 × ys1i / xs1i + y0p;
```

comment PLOTCURVE fifth page;

```
                a := temp2 / temp;
                b := (ys1i - xs1i × (temp2 + y0p)) / temp;
                x := y := f := 0;
                PLOTCURVE := (if xisx then y1p else 1 / y1p) / scalef;
                absxs1 := abs(xs1i);
SCALING:        if absxs1 = 0 then begin if count = 3 then count := 2; isc6 := 1; go to escape end;
                ln1 := log2(6 × absxs1) + 1;
                ln2 := log2(3 × (absxs1 + 3) ↑ 2) + 1;
                ln3 := log2((absxs1 + 5) ↑ 3) + 1;
                sc1 := 2 ↑ ln2;
                sc2 := 2 ↑ ln3;
                isc1 := 2 ↑ (ln3 - ln2 - 1);
                isc2 := isc1 × 2;
                isc3 := 2 ↑ (ln2 - ln1 - 1);
                isc4 := isc3 × 2;
                isc5 := 2 ↑ (ln1 - 1);
                isc6 := isc5 × 2;
                isc7 := isc5 + isc6;
```

```
comment PLOTCURVE sixth page

                    Overflow of any of the quantities f, delta, delta2, or delta3
                    is to be understood to result in an appropriate call of STOP,
                    followed by a transfer of control to the label escape;
DIFFERENCES:  delta := (a - b × incrx + y0p) × sc1 × incrx;
              delta2 := (2 × b - 6 × a × incrx) × sc2;
              delta3 := 6 × a × sc2 × incrx;
              which1 := (if xisx then 1 else 5) - incrx;
              which2 := which1 + 8;
              which3 := which1 + 16;
              ysmin := ysmin × isc6;
              ysmax := ysmax × isc6;
              oksofar := true;
    loop:     x := x + incrx;
              delta2 := delta2 + delta3;
              delta := delta + (delta2 + isc1 × sign(delta2)) : isc2;
              f := f + (delta + isc3 × sign(delta)) : isc4;
    loop2:    dif := f - y;
              absdif := abs(dif);
              small := absdif ≤ isc5; huge := absdif > isc7;
              which := if small then which1 else if huge then which3 else which2;
ACTION:       PLOT(ACT[if dif < 0 then which + 1 else which], 0, 20);
```

```
comment PLOTCURVE seventh page;

            if small then go to test;
            y := y + isc6 × sign(dif);
            if oksofar ∧ (y < ysmin ∨ y > ysmax) then begin STOP(‡1-27‡); oksofar := false end;
            if huge then go to loop2;
test:       if oksofar ∧ (x < xsmin ∨ x > xsmax) then begin STOP(‡1-27‡); oksofar := false end;
            if x ≠ xs1i then go to loop;
escape:     if xisx then PLOT(x0 + x, y0 + y : isc6, 24) else PLOT(x0 + y : isc6, y0 + x, 24);
            PLOT(-1, 0, 25);
            if I ≥ 3 then state32 := false;
exit:
end PLOTCURVE;
```

```
real procedure MARKEDCURVE(X, Y, I); value X, Y, I; real X, Y; integer I;
begin   own real X1, Y1; own integer i; own Boolean mark;
        MARKEDCURVE := PLOTCURVE(X, Y, I);
        if I ≤ 1 then     begin   mark := false; i := -abs(Y) end
                 else     begin   if mark then PLOTTEXT(X1, Y1, 0, 28, 0, true, i, ⏀)
                                          else mark := true;
                                  X1 := X; Y1 := Y
                          end
end MARKEDCURVE;
```

```
integer procedure SCALE(Ti,i,n,nint,mode,MIN,MAX,DL); value n, nint, mode;
        integer i, n, nint, mode; real Ti, MIN, MAX, DL;
        begin   integer ni, ni1; real MN, MX, temp, P, r, r1, min, min1;
                i:= 1; MN:= MX:= Ti; for i:= 2 step 1 until n do
                begin   temp:= Ti; if temp > MX then MX:= temp else if temp < MN then MN:= temp end;
                temp:= (MX - MN)/nint; P:= 10↑entier(ln(temp)/ln(10)); temp:= temp/P; r1 := .8;
                for r:= 1, 1.25, 2, 2.5, 4, 5, 8, 10, 12.5, 20, 25 do
                begin   if r < temp
                        then    r1:= r
                        else    begin  r:= r × P; min:= entier(MN/r) × r; if min + nint × r ≥ MX then
                                       begin   ni:= if mode = 0 then nint else entier((MX - min)/r + 1);
                                               if mode = 2 then
                                       begin   r1:= r1 × P; min1:= entier(MN/r1) × r1;
                                               ni1:= entier((MX - min1)/r1 + 1);
                                               if ni × r > ni1 × r1 then
                                               begin r:= r1; min:= min1; ni:= ni1 end
                                       end;
                                       goto out
                                end;
                                r1:= r/P
                        end
        end;
out:    DL:= r; MIN:= min; MAX:= min + ni × r; SCALE:= ni
end SCALE;
```

```
real procedure PLOTAXIS2(MIN,MAX,DL,horizontal,OTHER); value MIN, MAX, DL, horizontal, OTHER;
real  MIN, MAX, DL, OTHER; Boolean horizontal;
begin   integer k1, k2, k3, pt, n, m, p, q, step, cycle, pos, i, j, j2, h, h6, h7, precision, itemp;
        real factor, ln10, eps, dl, temp, min, max, other, x, y; Boolean tick, neg;
        integer procedure log10(x); real x; log10:= entier(ln(x)/ln10);
        real procedure S(U,i); value i; real U; integer i; S:= (U - PLOT(0,0,i + 4))/PLOT(0,0,i + 6);
        integer procedure zeroes(i); value i; integer i;
        begin   integer i1, k;
                if i = 0
                then    zeroes:= k3
                else    begin   k:= 0;
                ZA:     i1:= i : 10; if i = i1 × 10 then begin i:= i1; k:= k + 1; goto ZA end;
                        zeroes:= k
                end
        end zeroes;
        precision:= 7; h:= 4; h6:= 6 × h; h7:= 7 × h; ln10:= ln(10); tick:= false;
        if horizontal then PLOTAXIS(MIN,OTHER,0,MAX - MIN,DL) else PLOTAXIS(OTHER,MIN,90,MAX - MIN,DL);
AA:     k1:= log10(DL × 1.01); k2:= k1 + zeroes(DL × 10↑(2 - k1)) - 2; eps:= .5 × 10↑k2;
        k1:= log10(abs(if abs(MIN) > abs(MAX) then MIN else MAX) + eps);
        k3:= k1 - k2 + 1; if k3 > precision then begin PLOTAXIS2:= OTHER; goto exit end;
        if (k1 ≥ 5 ∧ k2 > 0) ∨ (-1 > k1 ∧ -5 ≥ k2)
        then begin pt:= k1; n:= 1; m:= k1 - k2; factor:= 10 ↑ (-pt) end
        else begin pt:= 0; n:= if k1 < 0 then 0 else k1 + 1; m:= if k2 < 0 then -k2 else 0; factor:= 1 end;
```

```
comment PLOTAXIS2 second page;

        q:= (MAX - MIN)/DL; dl:= DL/PLOT(0,0,if horizontal then 11 else 12);
        n:= n + (if m = 0 then 0 else m + 1) + (if MIN < -eps then 1 else 0);
        step:= (if horizontal then (n + 2) × 6 else 21) × h/dl + .5; if step < 2 then goto gotthem;
CYCLE:  j := j2 := pos := -1; for  i := 0 step 1 until q do
        begin   itemp := zeroes(abs(MIN + i × DL) × 10 ↑ (-k2)); if itemp ≥ j ∧ j2 ≥ j then
                begin   cycle := i - pos; if cycle < step
                        then    j := j + 1
                        else    for i := cycle : 2 step -1 until 1 do
                                begin   itemp := cycle : i;
                                        if cycle = itemp × i ∧ itemp ≥ step then go to gothim
                                end
                end;
                if itemp > j2 then begin pos := i; j2 := itemp end
        end;
        itemp := cycle;  BB:    itemp := itemp + cycle; if itemp < step then go to BB;
gothim: temp:= (MIN + pos × DL); DL:= itemp × DL;
        MIN:= temp - pos : itemp × DL; MAX:= temp + (q - pos) : itemp × DL; tick:= true; goto AA;
gotthem: if horizontal
        then begin min:= S(MIN,5); max:= S(MAX,5); other:= S(OTHER,6); y:= other - 8 × h; pos:= y - 14 × h end
        else begin min:= S(MIN,6); max:= S(MAX,6); x:= other:= S(OTHER,5); pos:= x - (n + 2) × h6 end;
```

```
comment PLOTAXIS2 third page;

        eps := eps × factor; for i:= 0 step 1 until q do
        begin   if horizontal
                then begin x:= max - i × dl; if tick then begin PLOT(x,other,-2); PLOT(x,other + 25,-1) end end
                else begin y:= max - i × dl; if tick then begin PLOT(other,y,-2); PLOT(other + 25,y,-1) end end;
                temp:= (MAX - i × DL) × factor; neg:= temp < -eps; temp:= abs(temp);
                if temp < eps
                then begin n:= 1; p:= 0 end
                else begin n:= log10(temp + eps) + 1; p:= m; if n < 0 then n:= 0 end;
                j:= n + (if p = 0 then 1 else p + 2);
                itemp:= x - (if horizontal then (j - (if neg then 1 else 0))/2 else j) × h6;
                if neg  then    FIXPLOT(itemp,y,0,-h7,0,true,-1,n,p,-temp)
                        else ABSFIXPLOT(itemp,y,0,-h7,0,true,-1,n,p, temp);
POWEROFTEN:     if i = q ∨ 2 ∧ pt ≠ 0 then
                begin   p:= if horizontal then 0 else 90;
                        n:= log10(abs(pt) + .5) + 1; j:= (min + max - (n + 3) × h6)/2;
                        PLOTTEXT(if horizontal then j else pos, if horizontal then pos else j,p,-h7,0,true,-1,↓× 10↑);
                        FIXPLOT(0,0,p,-h7,0,false,-1,n,0,pt); pos:= pos - 14 × h; pt := 0
                end
        end for i;
        PLOTAXIS2:= if horizontal then PLOT(0,pos - 3 × h,8) else PLOT(pos + 3 × h,0,7);
exit:
end PLOTAXIS2;
```

```
procedure PLOTPICTURE(Xi,Yi,i,n,mark,deltamark,mode,XMIN,XMAX,DX,xmax,xstring,YMIN,YMAX,DY,ymax,ystring);
value n, mark, deltamark, mode, XMIN, XMAX, DX, xmax, YMIN, YMAX, DY, ymax;
real Xi, Yi, XMIN, XMAX, DX, YMIN, YMAX, DY;
integer i, n, mark, deltamark, mode, xmax, ymax;
string xstring, ystring;
begin   real X, Y, X1, Y1; integer kg, kd, kx, ky, j; Boolean new, increase, curve;
        own real XMN, XMX, XDL, YMN, YMX, YDL, P, Q, R; own integer xint, yint, xmx, ymx;

        kg:= mode : 1000; mode:= mode - kg × 1000;
        kd:= mode :  100; mode:= mode - kd ×  100;
        kx:= mode :   10;     ky:= mode - kx ×   10; new:= kg < 2;
scalex: if new ∧ kx ≠ 7 then
        begin   if kx = 6
                then    begin  XMN:= XMIN; XMX:= XMAX; XDL:= DX; xint:= (XMX - XMN)/XDL end
                else    begin  xint:= if kx > 2
                               then SCALE(if j = 1 then XMIN else XMAX,j,2,xmax/DX,kx - 3,XMN,XMX,XDL)
                               else SCALE(Xi,i,n,xmax/DX,kx,XMN,XMX,XDL);
                               if kx = 1 ∨ kx = 4 then xmax:= xint × DX
                        end;
                P:= (XMX - XMN)/xmax; xmx:= xmax + 5200
        end;
```

```
comment PLOTPICTURE second page;

scaley: if ky ≠ 7 then
        begin   if ky - 5
                then    begin  YMN:= YMIN; YMX:= YMAX; YDL:= DY; yint:= (YMX - YMN)/YDL end
                else    begin  yint:= if ky > 2
                               then SCALE(if j = 1 then YMIN else YMAX,j,2,ymax/DY,ky - 3,YMN,YMX,YDL)
                               else SCALE(Yi,i,n,ymax/DY,ky,YMN,YMX,YDL);
                               if ky = 1 ∨ ky = 4 then ymax:= yint × DY
                        end;
                Q:= (YMX - YMN)/ymax; ymx:= ymax + 250
        end;
        if new ∨ ky ≠ 7 then
begin   if new then begin R:= XMN; PLOT(0,0,21); PLOT(0,2750,23) end;
        PLOT(XMN - 5000 × P,YMN - 200 × Q,-21); PLOT(P,Q,22); PLOT(xmx,ymx,-23);
        R:= PLOTAXIS2(YMN,YMX,YDL,false,R); PLOTTEXT(R,YMN,90,28,0,true,0,ystring);
        R:= R - 84 × P;
        if new then
        begin   Y:= PLOTAXIS2(XMN,XMX,XDL,true,YMN); PLOTTEXT(XMN,Y,0,28,0,true,0,xstring);
        grid:   if kg = 1
                then    begin  X1:= XMN; Y1:= YMN end
                else    begin  X1:= XMX - 15 × PLOT(0,0,11); Y1:= YMX - 15 × PLOT(0,0,12) end;
```

```
comment PLOTPICTURE third page;

                increase:= false;
horiz:          for j:= 1 step 1 until yint do
                begin   increase:= ¬increase; Y:= YMN + j × YDL;
                        if j = yint ∧ kg = 0 then begin X1:= XMN; increase:= false end;
                        PLOT(if increase then X1 else XMX,Y,2);
                        PLOT(if increase then XMX else X1,Y,1)
                end;
                increase:= true;
vert:           for j:= 1 step 1 until xint do
                begin   increase:= ¬increase; X:= XMN + j × XDL;
                        if j = xint ∧ kg = 0 then begin Y1:= YMN; increase:= false end;
                        PLOT(X,if increase then Y1 else YMX,2);
                        PLOT(X,if increase then YMX else Y1,1)
                end
        end if new
end if new ∨ ky ≠ 7;
curve:= deltamark ≥ 0; deltamark:= abs(deltamark);
if curve then   begin   if kd = 0 ∨ kd = 2
                        then    PLOTCURVE(0,0,1)
                        else    begin i:= 0; PLOTCURVE(Xi,0,0) end
                end;
```

```
comment PLOTPICTURE fourth page;

        j:= (if deltamark = 0 then -n else deltamark) - 2; mark:= -abs(mark);
        for i:= 1 step 1 until n do
        begin   j:= j + 1; X:= Xi; Y:= Yi;
                if curve then PLOTCURVE(X,Y,2);
                if j = deltamark then begin j:= 0; PLOTTEXT(X1,Y1,0,28,0,true,mark,◁▷) end;
                X1:= X; Y1:=Y
        end;
        if curve then   begin   if kd ≤ 1
                                then    PLOTCURVE(0,0,3)
                                else    begin i:= 0; PLOTCURVE(Yi,0,4) end
                        end;
        if j + 1 = deltamark then PLOTTEXT(X1,Y1,0,28,0,true,mark,◁);
        PLOT(xmx,0,-2)
end PLOTPICTURE;
```

```
begin   comment Sample program 1: Circle and Ellipse;
        integer i; real twopi, breadth;

        twopi:= 2 × 3.141592654;
        for breadth:= 1400, 700 do
        begin   PLOTFRAME(-700,-700,700,700,breadth,1400);
                PLOTTEXT(0,0,0,70,0,true,-5,◁);
                PLOTCURVE(10,0,0); comment initial slope infinite;
                for i:= 0 step 1 until 32 do PLOTCURVE(500 × cos(twopi × i/32),500 × sin(twopi × i/32),2);
                PLOTCURVE(10,0,4); comment final slope infinite;
        box:    PLOT( 500,-500,2); PLOT(-500,-500,1);
                PLOT(-500, 500,3); PLOT( 500, 500,3); PLOT( 500,-500,3);
        clear:  PLOT( 700,-500,2)
        end;
        PLOT(-1,0,-4)
end
```

```
begin   comment Sample Program 2: Error Damping by PLOTCURVE;
        real rad, angle, sina, cosa, X; integer inc;
        PLOTFRAME(-100, -100, 2200, 1400, 2300, 1500);
        rad := 3.141592654 / 180; X := 0;
        for angle := 45, 90, 135 do
        begin   PLOTTEXT(X, 400, 0, 28, 0, true, 0, ⟨angle of kink =⟩);
                FIXPLOT(0, 0, 0, 28, 30, false, 0, 3, 0, angle); PLOTTEXT(0, 0, 0, 28, 0, false, 11, ⟨⟩);
                MARKEDCURVE(0, 6, 1);
                for inc := 0 step 100 until 500 do MARKEDCURVE(X + inc, 500, 2);
                X := X + 500;
                sina := sin(angle × rad); cosa := cos(angle × rad);
                for inc := 100 step 100 until 500 do MARKEDCURVE(X + inc × cosa, 500 + inc × sina, 2);
                MARKEDCURVE(0, 0, 3);
                X := X + 300
        end for angle;
        PLOTTEXT(0, 200, 0, 56, 0, true, 0, ⟨Sample Program 2: Error Damping by PLOTCURVE⟩)
end
```

```
begin   comment Sample program 3: Demonstrate PLOTPICTURE with sin cos cot;
        real twopi, x, delta; integer i; Boolean first; real array X, cot[1:31];

        twopi := 2 × 3.141592654;
plotsin: PLOTPICTURE(if i = 0 then twopi else (i - 1)/32, if i = 0 then twopi else sin(twopi × (i - 1)/32), i, 65,
                     10, 4, 1363,
                         0,     2, 0.25, 1600, ‡x‡,
                        -1.01, 1.01,  100, 1200, ‡procedure is sin(2labelx), switch is cos(2labelx)‡);

plotcos: PLOTPICTURE(if i = 0 then 0 else (i - 1)/32, if i = 0 then 0 else cos(twopi × (i - 1)/32), i, 65,
                       9, 4, 2377, 0, 0, 0, 0, ‡‡, 0, 0, 0, 0, ‡‡);

        for i := 1 step 1 until 31 do
        begin   X[i] := i / 64; x := twopi × i / 64;
                cot[i] := cos(x) / sin(x)
        end for i;
        first := true;
plotcot: for delta := 0.0, 0.5, 1.0, 1.5 do
        begin   PLOTPICTURE(X[i] + delta, cot[i], i, 31, 7, 4, if first then 2070 else 2077,
                                  0, 0,    0,     0, ‡‡,
                                  0, 0, 100, 1200, ‡real is cotan(2labelx) = cos(2labelx)/sin(2labelx)‡);
                first := false
        end for delta
end
```

```
begin   comment Sample program 4: Demonstration of PLOTTEXT, 1000 decimals of pi in an exponential spiral;
        integer j; real r, r1, phi, phi1, deltaphi, c, p, rmax, b, arg, alpha, twopi, rad;
        real procedure height;
        begin   if j = 10 then begin j := 0; b := 0; goto exit end;
                j := j + 1; deltaphi := c × exp(p × phi1) / r1; phi := phi1 - deltaphi; r := rmax × exp(p × phi);
                b := sqrt(r ↑ 2 + r1 ↑ 2 - 2 × r × r1 × cos(deltaphi)); arg := r × sin(deltaphi) / b;
                alpha := (phi + arctan(arg / sqrt(1 - arg ↑ 2))) × rad + 180; r1 := r; phi1 := phi;
        exit:   height:= b × 7/6
        end calculation of alpha and height;
        twopi := 2 × 3.141592654; rad := 360 / twopi; p := .023; r1 := rmax := 650; phi1 := 0; j := 8;
        c := (6/7) × .75 × rmax × (exp(twopi × p) - 1); PLOTFRAME(-1350, -1350, 1350, 1350, 2700, 2700);
        PLOTTEXT(rmax, 0, alpha, height, 0, true, 0, ⊧3.
        1415926535 8979323846 2643383279 5028841971 6939937510 5820974944 5923078164 0628620899 8628034825 3421170679
        8214808651 3282306647 0938446095 5058223172 5359408128 4811174502 8410270193 8521105559 6446229489 5493038196
        4428810975 6659334461 2847564823 3786783165 2712019091 4564856692 3460348610 4543266482 1339360726 0249141273
        7245870066 0631558817 4881520920 9628292540 9171536436 7892590360 0113305305 4882046652 1384146951 9415116094
        3305727036 5759591953 0921861173 8193261179 3105118548 0744623799 6274956735 1885752724 8912279381 8301194912
        9833673362 4406566430 8602139494 6395224737 1907021798 6094370277 0539217176 2931767523 8467481846 7669405132
        0005681271 4526356082 7785771342 7577896091 7363717872 1468440901 2249534301 4654958537 1050792279 6892589235
        4201995611 2129021960 8640344181 5981362977 4771309960 5187072113 4999999837 2978049951 0597317328 1609631859
        5024459455 3469083026 4252230825 3344685035 2619311881 7101000313 7838752886 5875332083 8142061717 7669147303
        5982534264 2675546873 1159562863 8823537875 9375195778 1857780532 1712268066 1300192787 6611195909 2164201989⊧);
                PLOT(-1,0,-4)
end
```

```
begin   comment Sample program 5: Draw all PLOTTEXT characters;
        integer boxes, box, i, j, X, Y, x, y, x1, y1, x2, y2;
        PLOTFRAME(0, -200, 10000, 1800, 10000, 2000);
        Y := 1540; for boxes := 30, 30, 22, 20 do
        begin   X := 0; Y := Y - 385; for box := 1 step 1 until boxes do
                begin   if box = 11 ∨ box = 21 then X := X + 1050; for i := 0 step 1 until 7 do
                        begin   y := Y + i × 35; y1 := if i = 0 then y else y - 8;
                                y2 := if i = 7 then y else y + 8; for j := 0 step 1 until 4 do
                                begin   x := X + j × 35; x1 := if j = 0 then x else x - 8;
                                        x2 := if j = 4 then x else x + 8;
                                        PLOT(x1, y, 2); PLOT(x, y, 1); PLOT(x, y1, 3);
                                        PLOT(x, y2, 3); PLOT(x, y, 3); PLOT(x2, y, 3);
                                end for j
                        end for i;
                        X := X + 210
                end for box
        end for boxes;
        PLOTTEXT(0, 1155, 0, 245, 0, true, 0, <0123456789      PpQqRrSsTt      ,.₨:()[]⊄⊅⊀);
        PLOTTEXT(0,  770, 0, 245, 0, true, 0, ⊀AaBbCcDdEe      UuVvWwXxYy      :=⊀);
        for i := 2 step 1 until 10 do PLOTTEXT(0, 0, 0, 245, 0, false, i, ⊄⊅);
        PLOTTEXT(0,  385, 0, 245, 0, true, 0, ⊀FfGgHhIiJj      Zz+-×/:⩚⩓≥      ⊀);
        for i := 11, 12 do PLOTTEXT(0, 0, 0, 245, 0, false, i, ⊄⊅);
        PLOTTEXT(0,    0, 0, 245, 0, true, 0, ⊀KkLlMmNnOo      =≪⊀⩛⩔¬=;⊀);   PLOT(-1, 0, -4)
end
```

'Table 1'

'0'	1	0	0	1	0	6	1	7	3	7	4	6	4	1	3	0								
	1	0	'^ 0'		7	0	2	2	4	0	'^Q'													
'1'	2	0	2	7	1	6																		
'2'	0	5	0	6	1	7	3	7	4	6	4	4	0	1	0	0								
	4	0																						
'3'	0	1	1	0	3	0	4	1	4	3	3	4	1	4	3	4								
	4	5	4	6	3	7	1	7	0	6	'^3'		0	5	1	4								
	0	3	0	1	'^8'																			
'4'	3	0	3	7	0	3	4	3																
'5'	0	1	1	0	3	0	4	1	4	3	3	4	0	4	0	7								
	4	7																						
'6'	0	3	1	4	3	4	4	3	4	1	3	0	1	0	0	1								
	0	6	1	7	3	7	4	6																
'7'	0	6	0	7	4	7	1	0																
'8'	'see 3'																							
'9'	'see g'																							
'a'	4	0	4	3	3	4	1	4	0	3	0	1	1	0	3	0								
	4	1	4	4	'Aa'		4	7	'Ad'															
'b'	0	7	0	0	'Vo'		0	1	1	0	3	0	4	1	4	3								
	3	4	1	4	0	3	'Ab'		0	1	'Ao'													
'c'	4	1	3	0	1	0	0	1	0	3	1	4	3	4	4	3								
	'Ac'		4	2	0	2	'Ae'																	
'd'	'see a'																							
'e'	'see c'																							
'f'	0	0	0	4	2	4	0	4	0	6	1	7	3	7	4	6								
'g'	5	3	'V9'		0	1	1	0	3	0	4	1	4	6	3	7								
	1	7	0	6	0	4	1	3	3	3	4	4												
'h'	0	7	'V n r'		0	4	0	0	0	3	1	4	3	4	4	3								
	'Ar'		4	0	'Ah n'																			
'i'	1	0	3	0	2	0	2	4	1	4														
'j'	'see y'																							
'k'	1	7	1	0	1	1	4	4	2	2	4	0												
'l'	'see I'																							
'm'	0	0	0	4	0	3	1	4	2	3	2	0	2	3	3	4								
	4	3	4	0																				
'n'	'see h'																							
'o'	'see b'																							
'p'	5	3	0	0	0	7	0	6	1	7	3	7	4	6	4	4								
	3	3	1	3	0	4																		
'q'	5	3	4	0	4	7	4	6	3	7	1	7	0	6	0	4								
	1	3	3	3	4	4																		
'r'	'see h'																							
's'	0	1	1	0	3	0	4	1	3	2	1	2	0	3	1	4								
	3	4	4	3																				
't'	1	7	1	4	0	4	2	4	1	4	1	1	2	0	3	0								
	4	1																						
'u'	0	4	0	1	1	0	3	0	4	1	4	0	4	4										
'v'	0	4	2	0	4	4																		
'w'	0	4	0	1	1	0	2	1	2		2	1	3	0	4	1								
	4	4																						

```
'x'     5  1    'V x (2) (3) (4)'  0  1    4  5    2  3    0  5    4  1
        'Λ x x'  2  3    'Λ(2)'     'V - + (1)'      0  3    4  3    'Λ-'
        2  3    'Λ(3)'    2  5    2  1    'Λ+'     2  3    'Λ(1) (4)'
'y'     5  3    0  1     1  0    3  0    4  1    4  7    'Λj'     4  4
        3  3    1  3     0  4    0  7
'z'     0  4    4  4     0  0    4  0
'A'     0  0    0  4     4  4    4  6    3  7    1  7    0  6    0  4
        4  4    4  0
'B'     0  0    0  7     3  7    4  6    4  5    3  4    0  4    3  4
        4  3    4  1     3  0    0  0
'C'     4  5    4  6     3  7    1  7    0  6    0  1    1  0    3  0
        4  1    4  2     'ΛC'     4  3    3  3    'ΛG'
'D'     0  0    0  7     3  7    4  6    4  1    3  0    0  0
'E'     'see F'
'F'     4  7    0  7     0  4    3  4    0  4    0  0    'ΛF'     4  0
        'ΛE'
'G'     'see C'
'H'     0  0    0  7     0  4    4  4    4  7    4  0
'I'     1  0    3  0     2  0    2  7    1  7    'ΛI'     3  7    'ΛI'
'J'     'see U'
'K'     0  0    0  7     0  3    4  7    1  4    4  0
'L'     0  7    0  0     4  0    'ΛL'     4  7    0  7    'ΛO'
'M'     0  0    0  7     2  4    4  7    4  0
'N'     0  0    0  7     4  0    4  7
'O'     'see L'
'P'     0  0    0  7     3  7    4  6    4  5    3  4    0  4    'ΛP'
        3  4    4  3     4  0    'ΛR'
'Q'     'see 0'
'R'     'see P'
'S'     0  1    1  0     3  0    4  1    4  3    3  4    1  4    0  5
        0  6    1  7     3  7    4  6
'T'     2  0    2  7     0  7    4  7
'U'     0  7    'VJ'      0  1    1  0    3  0    4  1    4  7    'ΛUJ'
'V'     0  7    2  0     4  7
'W'     0  7    1  0     2  4    3  0    4  7
'X'     0  0    2  4     0  7    2  4    4  7    2  4    4  0
'Y'     2  0    2  4     0  7    2  4    4  7
'Z'     0  7    4  7     0  0    4  0
'+'     'see x'
'-'     'see x'
'x'     'see x'
'/'     0  0    4  7
':'     1  2    1  0     3  0    3  2    1  2    'Λ.'     7  0    1  4
        1  6    3  6     3  4    1  4    'Λ:'     7  0    0  3    4  3
        'Λ:'
'⅄'     2  0    2  6     0  4    2  6    4  4
'>'     'see ≥'
'≥'     0  5    4  3     0  1    'Λ >'    7  0    'V<'     0  0    4  0
        'Λ≥'     7  0    'V<'     4  1    0  3    4  5    'Λ<<'
```

'='	'see ≠'																
'≤'	'see ≥'																
'≮'	'see ≯'																
'≠'	0 2	4 2	7 0	4 4	0 4	'∧='	7 0	4 6									
	0 0	'∧≠'															
'⌐'	0 4	4 4	4 2														
'∧'	0 0	2 6	4 0														
'∨'	0 6	2 0	4 6														
'¬'	0 1	3 1	4 2	4 4	3 5	0 5											
'≡'	0 1	4 1	7 0	4 3	0 3	7 0	0 5	4 5									
','	'see ;'																
'.'	'see :'																
'10'	5 2	0 2	1 3	1 0	7 0	2 0	4 0	4 3									
	2 3	2 0															
':'	'see :'																
';'	5 2	1 0	3 2	3 4	1 4	1 2	2 2	1 0									
	'∧,'	7 0	1 5	1 7	3 7	3 5	1 5	'∧;'									
':='	1 2	1 1	0 1	0 2	1 2	7 0	2 2	4 2									
	7 0	4 4	2 4	7 0	1 4	0 4	0 5	1 5									
	1 4																
'('	4 0	3 1	3 6	4 7													
')'	0 0	1 1	1 6	0 7													
'['	4 0	2 0	2 7	4 7													
']'	0 0	2 0	2 7	0 7													
'≮'	6 2	2 3	2 5	4 7	3 5	4 5	4 3	2 3									
'≯'	6 2	0 3	1 5	0 5	0 7	2 7	2 5	0 3									
'(2)'	'see x'																
'(3)'	'see x'																
'(4)'	'see x'																
'(5)'	'see x'																
'(6)'	2 1	2 3	0 5	2 3	4 5	2 3											
'(7)'	0 1	'∨(8)'	4 1	0 5	4 5	0 1	2 3	'∧(7)(8)'									
'(8)'	'see (7)'																
'(9)'	2 1	0 1	0 5	4 5	4 1	2 1	2 3										
'(10)'	2 1	0 3	2 5	4 3	2 1	2 3											
'(11)'	6 2	0 5	1 4	2 4	3 5	3 6	2 7	1 7									
	0 6	0 5															
'(12)'	1 0	1 4	0 4	4 4	3 4	3 1	4 0										

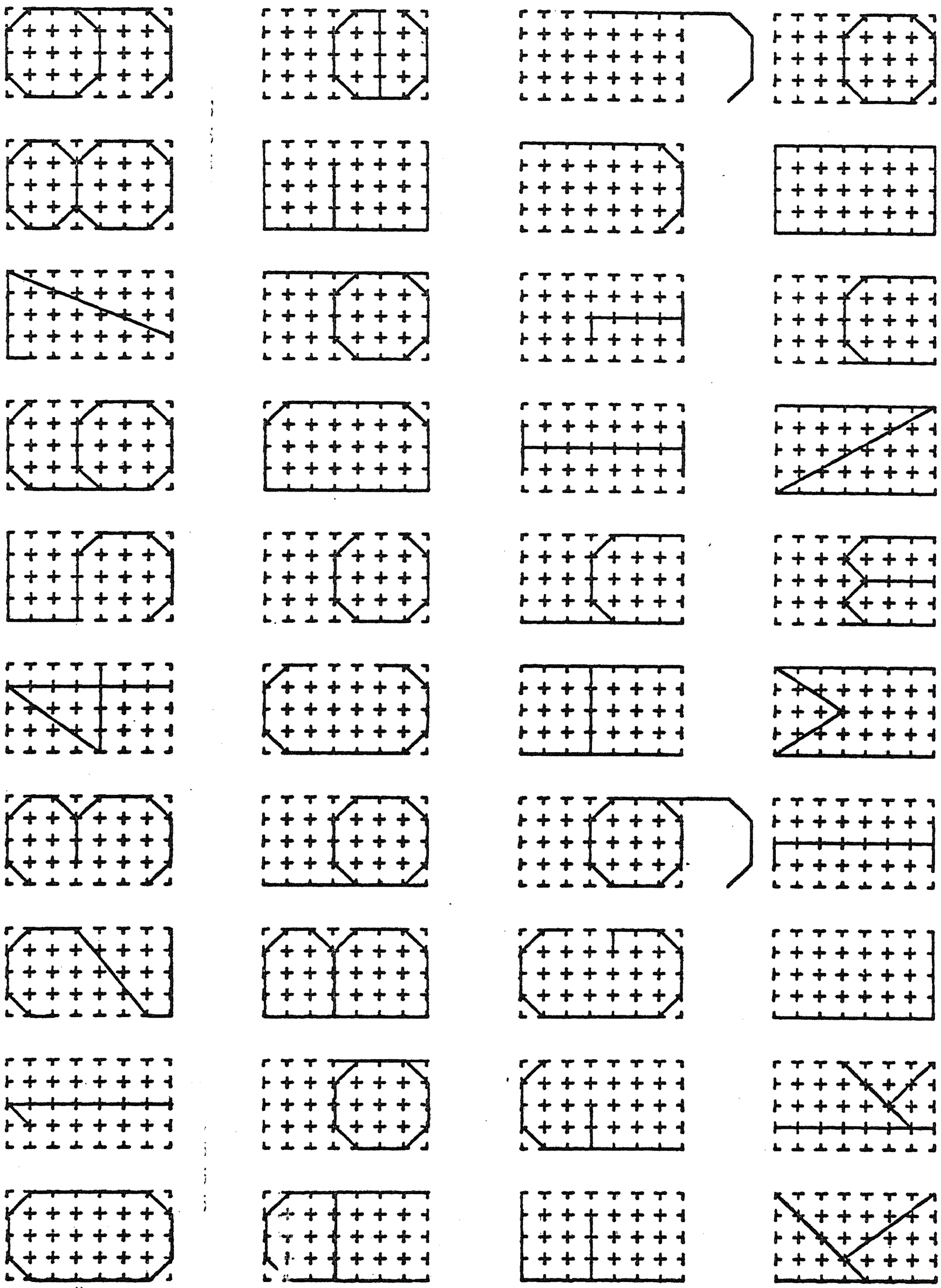
'Table 2'

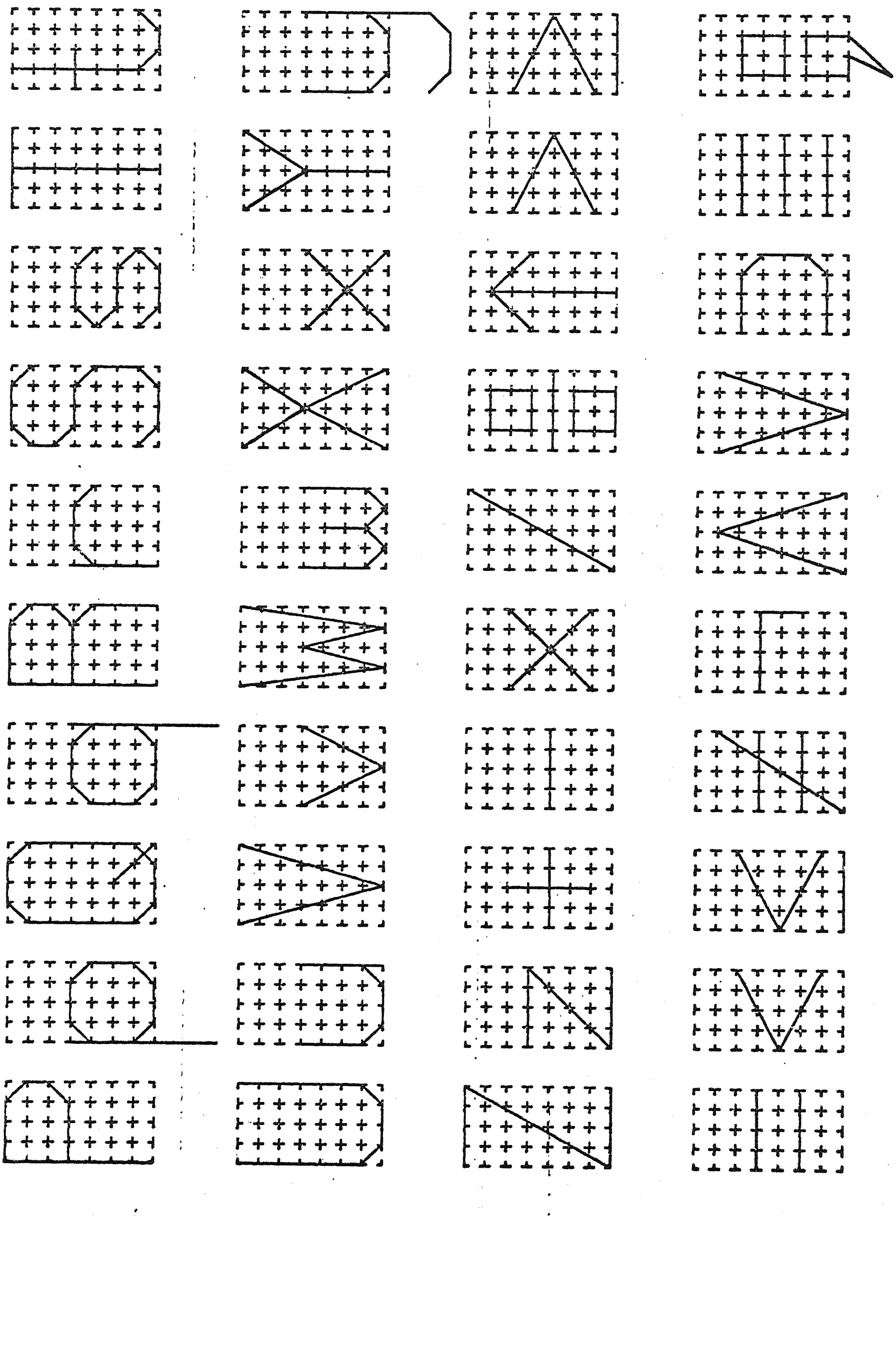
9	0	'0	0'
3	12	'1	1'
9	15	'2	2'
13	24	'3	3'
4	41	'4	4'
9	45	'5	5'
12	54	'6	6'
4	66	'7	7'
17	24	'8	8'
12	111	'9	9'
10	70	'a	10'
10	81	'b	11'
8	92	'c	12'
11	70	'd	13'
10	92	'e	14'
8	102	'f	15'
13	110	'g	16'
8	123	'h	17'
5	131	'i	18'
6	225	'j	19'
6	136	'k	20'
5	294	'l	21'
10	142	'm	22'
7	124	'n	23'
9	83	'o	24'
11	152	'p	25'
11	163	'q	26'
6	124	'r	27'
10	174	's	28'
9	184	't	29'
7	193	'u	30'
3	200	'v	31'
9	203	'w	32'
6	212	'x	33'
11	225	'y	34'
4	236	'z	35'
0	0	'	36'
10	240	'A	37'
12	250	'B	38'
10	262	'C	39'
7	274	'D	40'
7	281	'E	41'
6	281	'F	42'
12	262	'G	43'
6	288	'H	44'
6	294	'I	45'
5	347	'J	46'
6	300	'K	47'
3	306	'L	48'
5	311	'M	49'

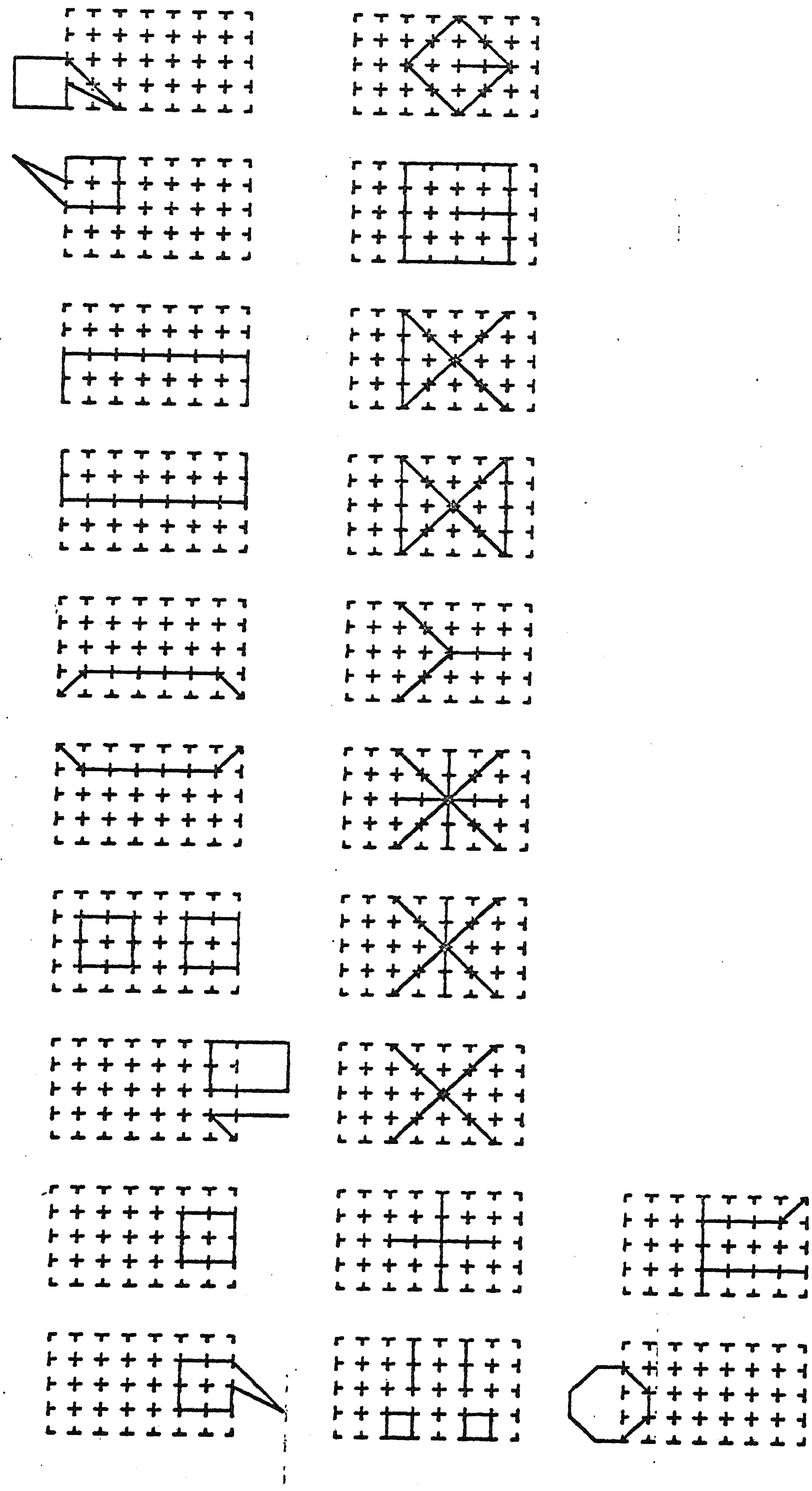
```
 4      316      'N          50'
 5      306      'O          51'
 7      320      'P          52'
12        0      'Q          53'
10      320      'R          54'
12      330      'S          55'
 4      342      'T          56'
 6      346      'U          57'
 3      352      'V          58'
 5      355      'W          59'
 7      360      'X          60'
 5      367      'Y          61'
 4      372      'Z          62'
 0        0
 5      219      '.          64'
 2      219      '_          65'
 5      213      '×          66'
 2      376      '/          67'
14      378      ':          68'
 5      392      '≠          69'
 3      397      '>          70'
 6      397      '≥          71'
 5      407      '=          72'
 6      401      '≤          73'
 3      404      '<          74'
 8      407      '≠          75'
 3      415      '¬          76'
 3      418      '∧          77'
 3      421      '∨          78'
 6      424      '¬          79'
 8      430      '≡          80'
 0        0      'goto       81'
 0        0      'if         82'
 0        0      'then       83'
 0        0      'else       84'
 0        0      'for        85'
 0        0      'do         86'
 8      448      ',          87'
 5      378      '.          88'
10      438      '10         89'
11      378      ':          90'
14      448      ';          91'
17      462      ':=         92'
 0        0      'space      93'
 0        0      'step       94'
 0        0      'until      95'
 0        0      'while      96'
 0        0      'comment    97'
 4      479      '(          98'
 4      483      ')          99'
```

```
4       487     '[              100'
4       491     ']              101'
8       495     '<              102'
8       503     '>              103'
6       219     'begin          104     2'
6       213     'end            105     3'
9       213     'own            106     4'
12      213     'Boolean        107     5'
6       511     'integer        108     6'
6       517     'real           109     7'
5       518     'array          110     8'
7       523     'switch         111     9'
6       530     'procedure      112     10'
10      536     'string         113     11'
7       546     'label          114     12'
0       0       'value          115'
0       0       'true           116'
0       0       'false          117'
0       0       'tab            118'
0       0       'new line       119'
0       0       '               120'
0       0       '''             121'
0       0       'stopcode or ?  122'
```

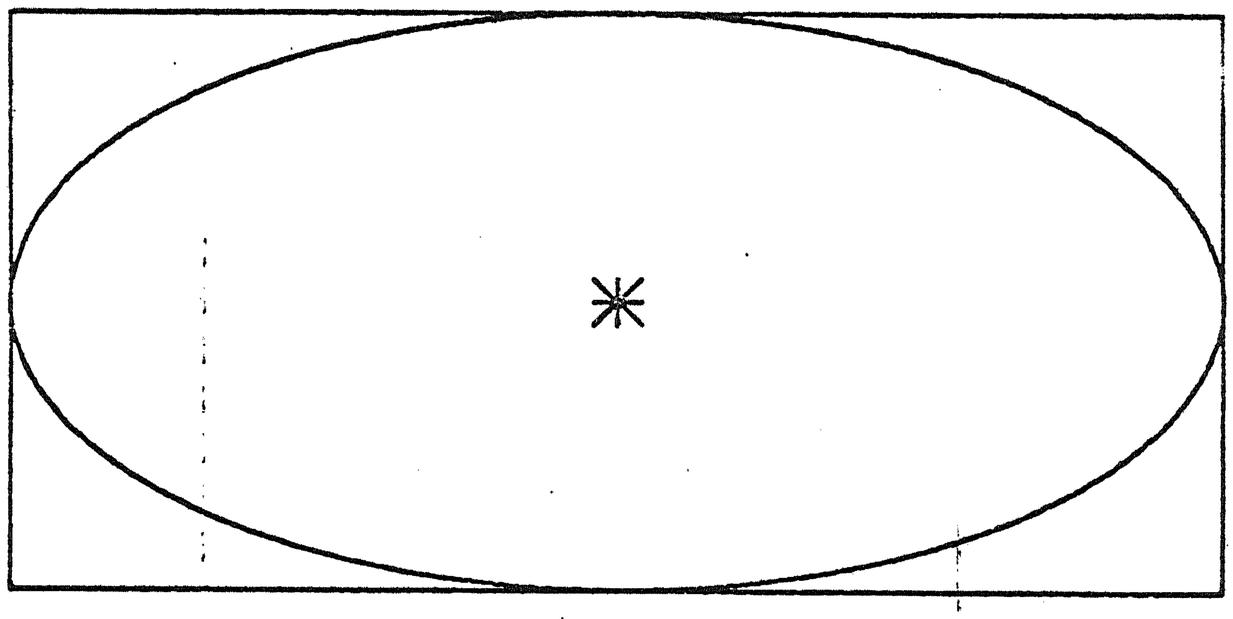
Table 3: Output from Sample Program 5

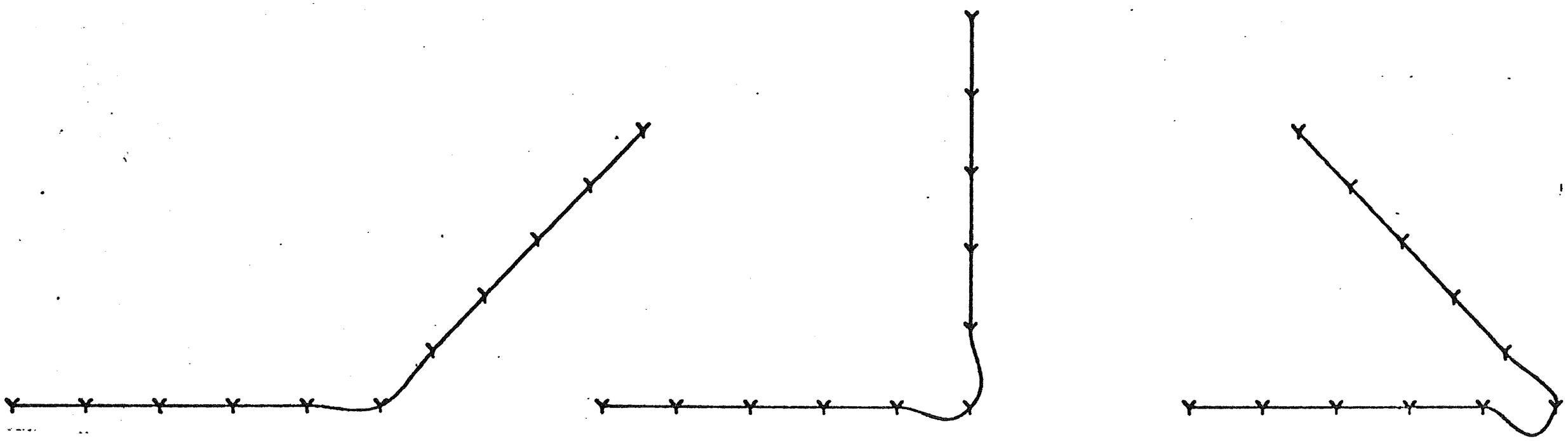






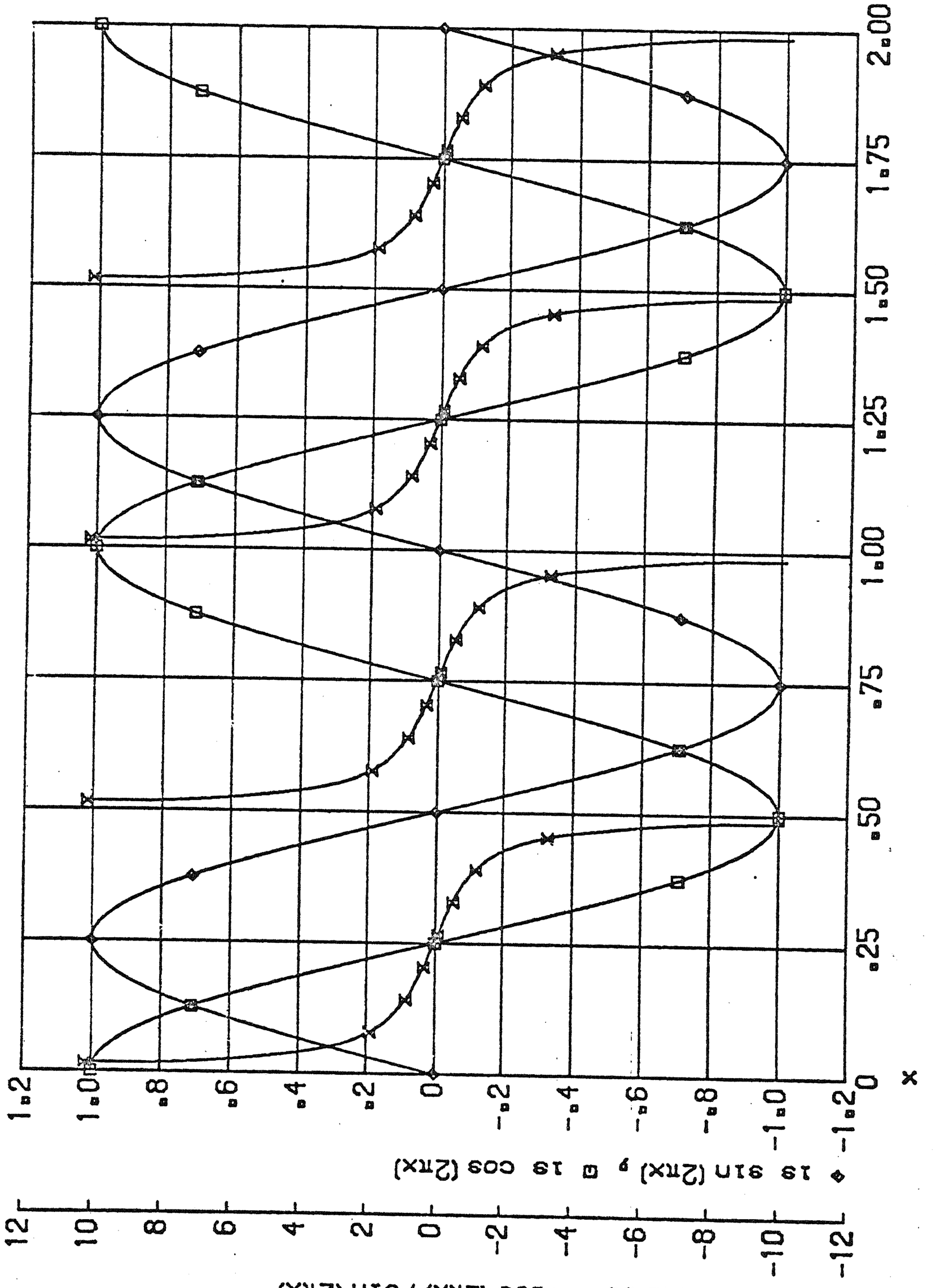
Output from Sample Program 1



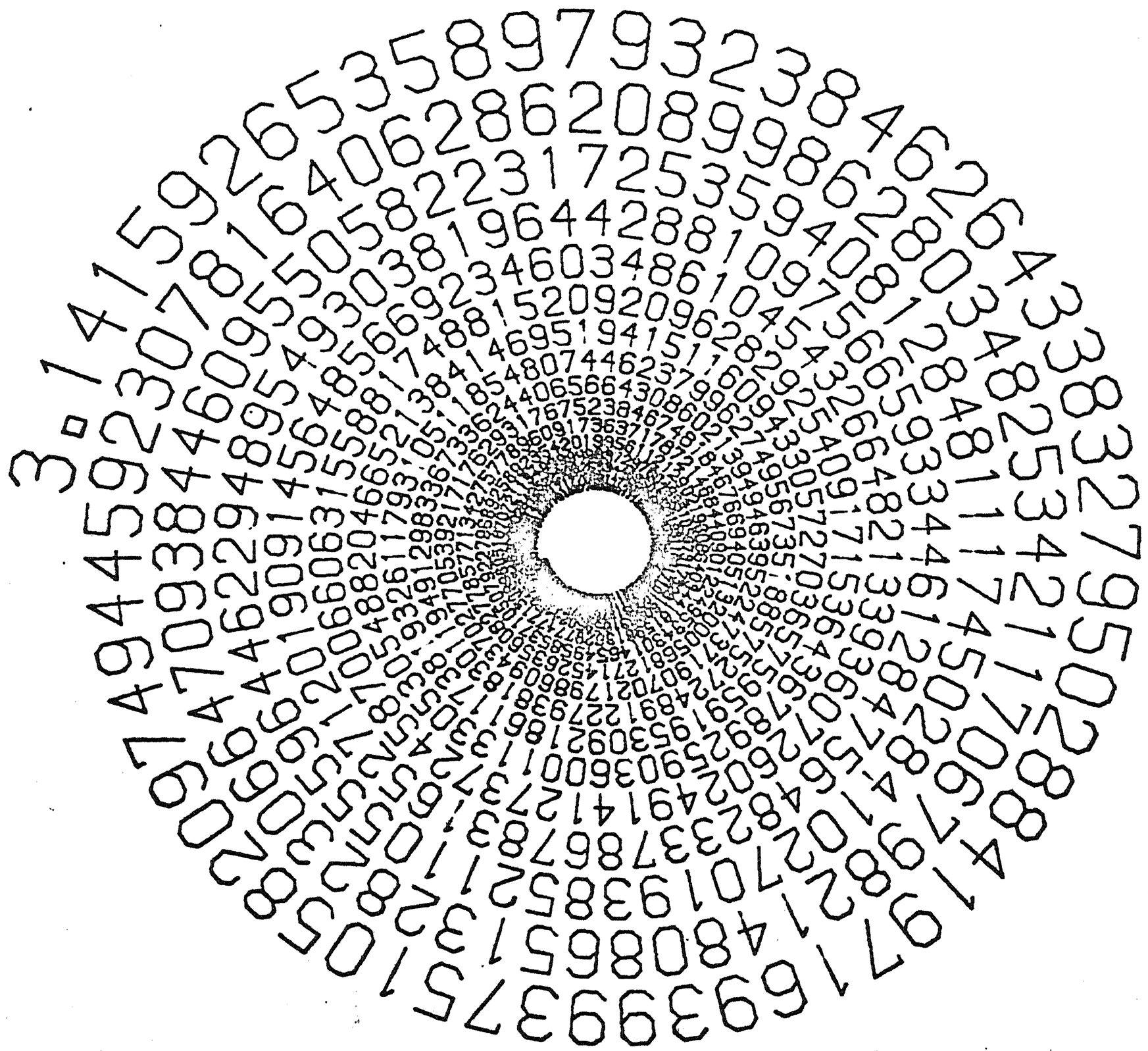


Sample Program 2: Error Damping by PLOTCURVE

Output from Sample Program 3



Output from Sample Program 4



References

[1] P. Naur, et. al., Revised Report on the Algorithmic Language ALGOL 60, C. ACM 6(1963), 1-17.

[2] Suggestions on ALGOL 60 (ROME) Issues, C. ACM 6(1963), 20-23.

[3] T. Giammo, A Mathematical Method for the Automatic Scaling of a Function, J. ACM 11, 1(Jan. 1964), 79-83.