

# MINC-11

## Book 4: MINC Graphic Programming

November 1978

This manual teaches concepts of graphic programming for MINC systems, and it explains the use of the MINC graphic routines. All readers of this manual should first read *Book 1: Introduction to MINC*. All readers should be familiar with MINC BASIC, which is documented in *Book 2: MINC Programming Fundamentals* and *Book 3: MINC Programming Reference*.

Order Number AA-D574A-TC

MINC-11

VERSION 1.0

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

First Printing, November 1978

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1978 by Digital Equipment Corporation

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
MINC-11	DECSYSTEM-2020	

# CONTENTS

## PREFACE

## PART 1 INTRODUCTION TO COMPUTER GRAPHICS

<b>CHAPTER 1 COMPUTER GRAPHICS WITH MINC</b>	<b>1</b>
ROUTINE STATEMENTS IN IMMEDIATE MODE	3
<b>CHAPTER 2 PROPERTIES OF THE MINC TERMINAL</b>	<b>5</b>
TEXT POSITIONS AND GRAPHIC COORDINATES	6
Text Positions	6
Graphic Coordinates	6
SUBDIVIDING THE SCREEN	7
The Scrolling Area	8
Scrolling Modes	9
Graph Regions	9
DISPLAYING TEXT	12
Character Mode	12
Display Mode	13
GRAPH NUMBERS	14
THE GRAPHIC MEMORY	14
Erasing a Graph	14
Making a Graph Invisible	15
THE STANDARD INITIAL STATE	15
<b>CHAPTER 3 INTRODUCTION TO THE GRAPH ROUTINE</b>	<b>19</b>
INITIAL PROCEDURES	19
Creating a Data Structure	19
First Display	21
Omitted Arguments	22
The Option String	22
Clearing the Screen	23
Displaying Two Graphs	23

## CONTENTS

OPTIONAL FEATURES	23
The Graph Number Argument	23
The BRANDS Option	24
The SHADE Option and the Shade Line Argument	25
Erasing a Graph	26
Making a Graph Invisible	27
The Number Argument	28
Autoscaling	29
The Options GRID, TICKS, UNITS, HLINES, and VLINES	30
The EXACT Option	33
Strip-Chart Mode	35
The Start X Argument	39
The Increment Argument	40
The Start Index Argument	41
INDEX ARRAYS	42
Using Index Arrays	44
Indexing a Single Point	44
Multiple Indexing	46
<b>CHAPTER 4 PICTORIAL EXAMPLES</b>	<b>51</b>
Simple Point-Plot of Entire Array	52
Simple Point-Plot of Entire Array, with X Coordinates	53
Simple Point-Plot of Array Subset	54
Bargraph of Array Subset	55
Shaded Point-Plot of Entire Array	56
Shaded Point-Plot of Array Subset, with Brands	57
Point-Plot with Exact Axis Units	58
Point-Plot of Every Second Value	59
Interpolated Point-Plot of Every Twentieth Value	60
Interpolated Point-Plot with Shading	61
Interpolated Point-Plot, Shaded about $Y = 0$	62
Bargraph of Array Subset, Shaded about $Y = 0$	63
Bargraph of Array Subset, Shaded about $Y = 1$	64
Simple Point-Plot of Entire Array, with Vertical Grid Lines	65
Simple Point-Plot of Entire Array, without Horizontal Grid Lines	66
Simple Point-Plot of Entire Array, without Axis Units	67
Simple Point-Plot of Entire Array, Labeled with LABEL Routine	68
Replacement of X Units with Text Label	69
Simple Point-Plot of Entire Array, with Marked Value	70
<b>CHAPTER 5 ARRAYS AND DATA STRUCTURES</b>	<b>71</b>
ARRAYS	71
One-Dimensional Arrays	72
Two-Dimensional Arrays	73
VIRTUAL ARRAY FILES	77
Displaying Virtual Array Files	78

**PART 2 ROUTINES**

**INTRODUCTION TO PART 2 83**  
**BARGRAPH 89**  
**BOX 90**  
**CHAR\_MODE 93**  
**DISPLAY\_CLEAR 96**  
**DISPLAY\_MODE 98**  
**DUAL\_MOVE 101**  
**ERASE\_GRAPH 106**  
**ERASE\_TEXT 112**  
**FIND\_POINT 116**  
**GET\_CURSOR and MOVE\_CURSOR 119**  
**GRAPH and BARGRAPH 122**  
**GRAPH\_INIT 134**  
**GRID 136**  
**HLINE and VLINE 138**  
**HTEXT and VTEXT 141**  
**LABEL 144**  
**LIGHTS 147**  
**MAP\_TO\_GRAPH 149**  
**MAP\_TO\_TEXT 152**  
**MOVE\_CURSOR 155**  
**POINT 156**  
**PUT\_SYMBOL 160**  
**REGION 163**  
**ROLL\_AREA 165**  
**SET\_BAR 167**  
**SHADE 169**  
**TEXT\_INIT 171**  
**TEXT\_LINE 172**  
**VIEW 175**  
**VLINE 176**  
**VTEXT 177**  
**WIDE\_LINE 178**  
**WINDOW 180**

**INDEX 185**



Figure 1.	MINC Terminal Screen, Showing Maximum Scrolling Area	8
2.	MINC Terminal Screen, Showing the Three Graph Regions	10
3.	MINC Character Modes	12
4.	Primary Setup Mode Display	16
5.	Secondary Setup Mode Display	17
6.	A Simple Sine Graph	21
7.	A Graph with Brands (Lower Region)	24
8.	Two Shaded Graphs, with and without Shade Lines	25
9.	Shaded Graph without the LINES Option (Lower Region)	26
10.	Result of VIEW ("INVISIBLE", 2)	27
11.	GRAPH Display of 20 Points	29
12.	Result of GRID and V LINES Options	31
13.	Result of GRID plus -UNITS Options	32
14.	Result of GRAPH Statement without EXACT Option	34
15.	Result of GRAPH Statement with EXACT Option	35
16.	Static Display of a 1401-Point Array	36
17.	First 511 Points of Array M	37
18.	First 500 Points, Beginning of Strip-Chart Operation	37
19.	Last 512 Points, End of Strip-Chart Operation	38
20.	Logarithmic X Axis	40
21.	Sine (Upper) and Cosine (Lower) Waves	41
22.	Index Array before Execution	42
23.	Index Array after Execution	43
24.	Beginning of Indexing Operation	45
25.	Simple Point-Plot of Entire Array	52
26.	Simple Point-Plot of Entire Array, with X Coordinates	53
27.	Simple Point-Plot of Array Subset (50 Points)	54
28.	Bargraph of Array Subset (10 Points)	55
29.	Shaded Point-Plot of Entire Array	56
30.	Shaded Point-Plot with Brands (20 Points)	57
31.	Point-Plot with Exact Axis Units	58
32.	Point-Plot of Every Second Value	59
33.	Interpolated Point-Plot of Every Twentieth Value	60
34.	Interpolated Point-Plot with Shading	61
35.	Interpolated Point-Plot, Shaded about $Y = 0$	62
36.	Bargraph of Array Subset, Shaded about $Y = 0$	63
37.	Bargraph of Array Subset, Shaded about $Y = 1$	64

## FIGURES

## CONTENTS

- Figure 38. Simple Point-Plot of Entire Array, with Vertical Grid Lines 65
- 39. Simple Point-Plot of Entire Array, without Horizontal Grid Lines 66
- 40. Simple Point-Plot of Entire Array, without Axis Units 67
- 41. Labeled Point-Plot of Entire Array 68
- 42. Replacement of X Units with Text Label 69
- 43. Simple Point-Plot of Entire Array with Marked Point 70
- 44. A Representation of the Array T1 (5,4) 73
- 45. Allocation of the Array A1 (1,2) 74
- 46. Points Stored in a Two-Dimensional Array 75
- 47. Display of a Two-Dimensional Array 76
- 48. Display of a Virtual Array File 80
- 49. Result of Two BOX Statements 90
- 50. Sample Character Modes 94
- 51. A Long-Format Graph 98
- 52. Original Graph 106
- 53. Result of ERASE\_GRAPH (“-ALL, POINTS”) 107
- 54. Result of ERASE\_GRAPH (“-ALL, GRID”) 107
- 55. Result of ERASE\_GRAPH (“-ALL, HLINE”,,1.00) 108
- 56. SHADE Option with a 100-Point Graph 126
- 57. SHADE and LINES Options with a 100-Point Graph 127
- 58. Sine Wave with GRID Display 136
- 59. Typical HTEXT and VTEXT Displays 141
- 60. Result of the LABEL Routine 144
- 61. Programmable LED Options 148
- 62. PUT\_SYMBOL Characters 160
- 63. TEXT\_LINE Display with Stair-Step Effect 174
- 64. Regular and Double-Width Characters 178



## PREFACE

This manual shows you how to use all the MINC graphic routines, beginning with those used most frequently.

Part 1 of this manual is written so that you can run sample programs on the MINC system while you read. If you do not have a MINC system immediately available, you may find it useful to read the definition of each routine (Part 2) as a preparation. Each routine has its own reference section in Part 2, and the sections are ordered alphabetically by routine name.

If you begin with Part 1, put your system diskette in the MINC and start the machine, because you will want to try examples as you read.

This manual, and the programming examples within it, are written with the assumption that you know something about the BASIC programming language. If you are not familiar with BASIC, read *Book 2: MINC Programming Fundamentals*, before going any further.

Even if you know BASIC, be sure to read Chapter 2, "Properties of the MINC Terminal."



**PART 1**

**INTRODUCTION TO  
COMPUTER  
GRAPHICS**



# CHAPTER 1

## COMPUTER GRAPHICS WITH MINC

Computer graphics is a process that uses computers to create pictures. Just as the computer can be used for a wide variety of computational and control tasks, computer graphics is used for a variety of pictorial tasks.

In a laboratory with a MINC computer system, the typical use of graphics is to display data that are read from a laboratory instrument or data that are computed by a program.

Graphic programming is generally done by writing computer programs in a special language. The language is usually one of the standard programming languages that has been extended with additional statements that are specialized to graphics. In MINC programming, these additional statements are called *routines*, because they perform routine tasks associated with graphic displays. MINC routines are simple to use and have names that help define their functions. When you use a routine in a program, the statement containing the routine name is known as a *routine statement*. A routine statement looks like this in a MINC program:

```
<line number> <routine name>(<arguments>)
```

The words between the angle brackets (<>) represent the three components that are common to routine statements.

1. The *line number* has the same significance in graphic programming as in other MINC programming: it tells MINC the order in which statements should be executed.

2. The *routine name* is a word or series of words that identifies the routine you wish to use. MINC graphic programming is done with a set of 33 special routines that you can use in a MINC program. Each of these routines has a name that describes its function. For instance, BARGRAPH is the name of the routine that makes bargraphs from your data. The routine that erases text from the screen is named ERASE\_TEXT. Notice that when routine names consist of more than one word, you must connect the words with an underscore character (\_). You make the underscore character by holding down the SHIFT key on your terminal and pressing the hyphen key.
3. The *arguments* for the routine statement are a set of values that tell the routine exactly what to do. Some routines always perform the same function, in which case they have no arguments at all. Most routines, though, can perform many variations of the same general function, and they need a list of arguments to define the specific operation you want.

Nearly all the routines that use arguments also have *default conditions* for arguments. A default condition, or simply, a *default*, occurs when you leave an argument out of a routine statement. When you leave out the argument, the routine makes an assumption about the effect you want. The default conditions for the routines are documented in Part 2 of this manual. In every case, the defaults select the most powerful application of the routine; this allows you to avoid naming a long list of arguments in a statement, while still taking advantage of the routine's full capabilities.

When you include a list of arguments in a routine statement, the individual arguments are separated by commas, and the entire list is enclosed by parentheses, as shown in the form above.

This is a sample graphic routine statement as it might appear in a MINC program:

```
10 GRAPH("SHADE",10,,Y(0),,2)
```

The line number is 10. The name of the routine is GRAPH. The arguments are "SHADE", a string expression; the number 10; the expression Y(0); and the number 2. The exact meanings of these arguments will be discussed later in Part 1. Notice that in two places in the GRAPH statement, there are pairs of commas with nothing between them. These commas indicate that two

arguments are missing, and the GRAPH statement will use the default values for them. Actually, the GRAPH routine can accept up to eight arguments, but because the seventh and eighth arguments come after the 2, they do not need to be represented by missing commas. Argument lists in MINC are like decimal digits, in these two ways:

1. The interpretation of an argument depends on its *position* in the list. For instance, the argument "SHADE" must always be in the first position in a GRAPH statement. This is similar to a decimal number such as 1234.567, in which the position of the digit 3 identifies it as the "tens" digit.
2. Missing arguments, like missing decimal digits, need be represented only if they come *between* other arguments. Thus, the pairs of commas in the sample GRAPH statement assure that the subsequent arguments are interpreted correctly by MINC. The pairs of commas are similar to zeros in a decimal number. In the number 12300.456, the zeros are significant; in the number 123.45600, they are not and may be omitted.

The line number shown in the previous example is required only when you use a routine in a program. You can also use routines in immediate mode simply by typing the routine name and arguments without the line number. For instance, this dialog, in which your responses are in red, will first erase graphic material from the screen and then will erase text from the screen:

```
READY
ERASE_GRAPH
```

```
READY
ERASE_TEXT
```

```
READY
```

## ROUTINE STATEMENTS IN IMMEDIATE MODE





## CHAPTER 2 PROPERTIES OF THE MINC TERMINAL

The MINC terminal was designed as an integral part of the MINC system, and it has many features that are not found in most computer terminals. The special properties of interest to graphic programmers are listed here.

1. Separate systems of *text coordinates* and *graphic coordinates*, so that you can address the terminal in the coordinate system most appropriate to the type of display you desire.
2. A programmable *scrolling area*, which lets you define the portion of the screen to be used for ordinary text (such as system messages and program writing).
3. Programmable *graph regions*, which let you define the portions of the screen to be used for graphic displays.
4. Programmable *character modes*, which let you decide whether characters in any text display will be displayed as normal, flashing, underlined, boldface, or reverse video characters. (Combinations of these modes are also allowed.)
5. Programmable *display modes*, which let you define the width of the screen, the background color (black or white), and the scrolling mode (smooth or jump).
6. Selectable *graph numbers*, which let you display two graphs simultaneously.

7. An internal *graphic memory*, which holds graphic information within the terminal itself and allows graphs to be erased or changed without affecting the workspace used by MINC BASIC.
8. A *standard initial state*, which guarantees that the terminal has a known set of characteristics when you turn on its power (as when starting the MINC system).

## TEXT POSITIONS AND GRAPHIC COORDINATES

The MINC terminal can, at any given time, display up to 24 rows of 132 characters. It can simultaneously display up to two graphs, each of which can contain 512 displayed points.

This section describes the two different systems that you use to actually display text and graphs on the screen.

### Text Positions

When you display text on the screen with the graphic routines, you refer to a *text position*. The MINC terminal can display up to 24 lines of text. Each line can contain either 80 or 132 characters, depending on your choice when you use the DISPLAY\_MODE routine.

The word *row* refers to the horizontal strip on the MINC screen in which a *line* of text can be displayed. Similarly, the word *column* refers to the vertical strip in which a character is displayed.

When you use a text-display routine (such as HTEXT) that takes row-column arguments, the row argument should be an integer in the range 1-24. The column argument should be an integer in the range 1-80 or 1-132, depending on the screen width you have selected with the DISPLAY\_MODE routine.

### Graphic Coordinates

When you display points or graphs with the graphic routines, you refer to screen positions with *graphic coordinates*. Graphic coordinates, or X-Y coordinates, are the actual numbers that result from a computation or from reading data from an instrument. These numbers may be integers, or they may be real numbers.

Unlike text positions, graphic coordinates have no fixed relationship to physical screen positions. For instance, the point at coordinates  $X=.5, Y=.25$  could appear anywhere on the screen. The only fixed relationship that applies to graphic coordinates is their relative position. That is, the point at coordinates  $(.5,.25)$  is placed on the screen so that it is clearly at a "lower" coordinate position than, say,  $(1.5,1.25)$ .

For a single graph, the relationship that defines placement of coordinates at physical screen positions is called the *window* of the graph. This term is used because the graph is like a window through which you view a portion of the X-Y coordinate space.

The window establishes the ranges of X and Y coordinates that can appear in a particular graph.

You can define the window of a graph with the WINDOW routine (see Part 2 for a definition of WINDOW). With the routines GRAPH and BARGRAPH, you can use a feature called *auto-scaling*, which sets the window automatically, based on the minimum and maximum values of the data set you wish to display.

To summarize:

1. When you want to display a text string or expression, the text-display routines are used. They take *row and column* arguments in the range 1-24 and 1-132, respectively.
2. When you want to display graphical information (such as a point), the graphic display routines are used. These routines take *X and Y* arguments, the range of which depends on the current window. The GRAPH and BARGRAPH routines, which can use the autoscaling feature, also accept arguments that mark the beginning of entire arrays of X and Y coordinates. These arguments, called the *start X* and *start Y* arguments, are discussed in Chapter 3, "Introduction to the GRAPH Routine."

You will be using the terminal both for the display of graphical information and for the display of text that ordinarily appears on a programming terminal, such as program listings, requests for input from a MINC program, and messages printed by a program with the PRINT statement. Because you will commonly display these two kinds of information at the same time, the MINC system must use different areas of the screen for the different types of display. Doing so prevents such text as input requests (created by the INPUT statement of BASIC) and error messages from getting lost in a complicated graph. In this section, text of this type is called *ordinary text*.

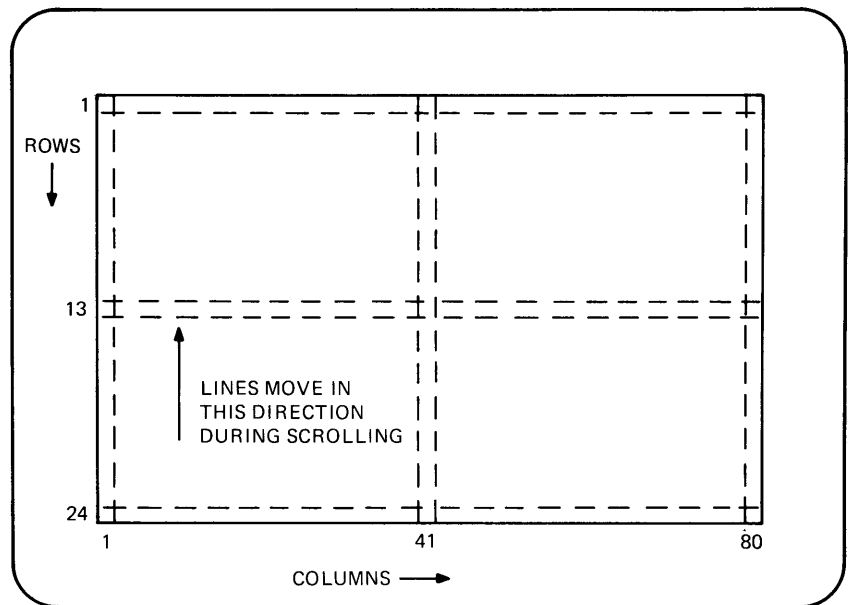
## SUBDIVIDING THE SCREEN

The MINC graphic system gives you simple mechanisms with which to subdivide your terminal screen. The section of the

screen you reserve for ordinary text is the *scrolling area*. The section you reserve for a graph is the *graph region*.

**The Scrolling Area**

The terminal has a maximum scrolling area of 24 rows, with 80 or 132 columns in a row (see Figure 1).



MR-1621

Figure 1. MINC Terminal Screen, Showing Maximum Scrolling Area

If the computer sends the terminal 25 lines of text for display, line 1 is displayed at the top of the scrolling area (row 1), and the screen fills up with text until line 24 appears on row 24. When line 25 is received, it appears on row 24, at the bottom of the screen. Lines 1-24 move up one position, causing line 1 to disappear from the top of the screen. This movement is called *scrolling* because it resembles a scroll of printed lines being cranked past your field of vision.

With the `ROLL_AREA` routine, you can change the size and placement of the scrolling area to make it occupy any part of the screen. For instance, you could use the top half of the screen for a graph region and the bottom half for the scrolling area.

To demonstrate this to yourself, type the following statements on your MINC terminal.

```
ROLL_AREA(21,24)
```

```
DIRECTORY
```

With the `ROLL_AREA` statement, you have restricted the scrolling area to the bottom four rows of the screen. This `ROLL_AREA` statement also protects any text outside rows 21-24 from further scrolling. Notice that the cursor is now inside the bottom four rows, even if it was at the top of the screen before you typed the `ROLL_AREA` statement. As always, the cursor is showing you where the next line of ordinary text will appear.

The scrolling area is a type of receptacle for error messages, programming dialogs, and similar entries that you type by hand or that MINC sends to the terminal. These are all examples of ordinary text.

Many of the routines defined in Part 2 accept row and column numbers as arguments. These routines can display strings of characters (or special symbols) at any row-column position on the screen, whether or not the row is inside the scrolling area.

You can control the type of scrolling, or *scrolling mode*, with the `DISPLAY_MODE` routine. In *jump* scrolling mode, a new line appears all at once at the bottom of the scrolling area, so that the text already on the screen jumps upward by one row. *Smooth* scrolling mode has a more continuous appearance, because the new line scrolls onto the screen more gradually.

## Scrolling Modes

If your MINC system is now running, you can also demonstrate the graph regions to yourself. Type the red parts of this dialog:

```
READY
REGION('UPPER')
```

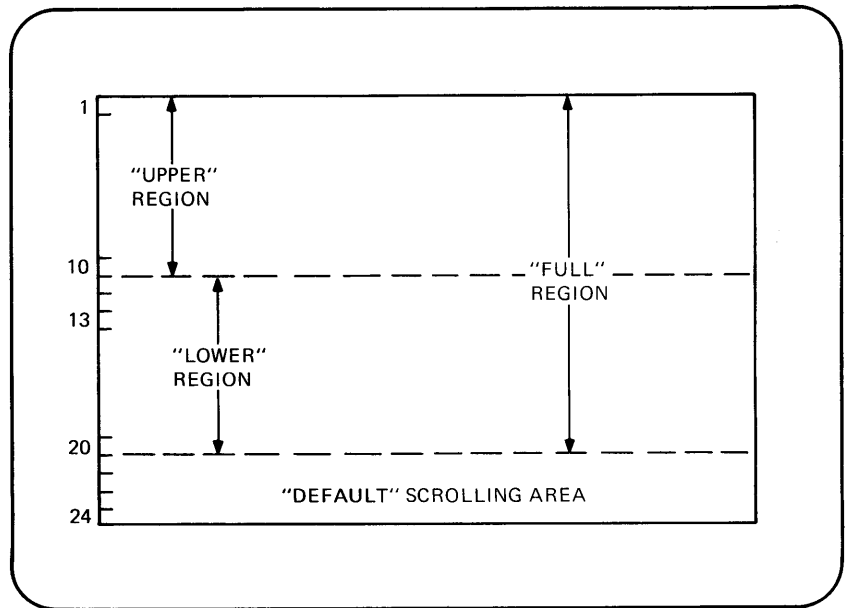
```
READY
REGION('LOWER')
```

```
READY
REGION('FULL')
```

Each time you type a new `REGION` statement, horizontal lines appear, marking the top and bottom of the named graph region.

Figure 2 shows all the possible divisions of the MINC screen into graph regions.

## Graph Regions



MR-1622

Figure 2. MINC Terminal Screen, Showing the Three Graph Regions

The main characteristics of graph regions are as follows.

1. There are only three kinds of graph region: the "upper region," which corresponds to rows 1-10 on the screen; the "lower region," which corresponds to rows 11-20; and the "full region," which corresponds to rows 1-20. You cannot make a graph region of any other size.
2. In each region, the bottom text row is reserved by the graphic routines for the display of X axis units. The reserved row is row 20 for the full and lower regions, and row 10 for the upper region.
3. You can define two graph regions at the same time. MINC can then display two totally independent graphs on the screen simultaneously, either in separate regions or in the same region.
4. You can never display a graph that covers all the rows on the screen. Even the full region extends only through row 20, so that the bottom four rows are reserved for a scrolling area.
5. Some of the graphic routines begin their display operations by erasing all the text in their graph region. For this reason,

the REGION routine resets the scrolling area automatically to keep it from overlapping the graph region. After you have defined a graph region with the REGION routine, you should use `ROLL_AREA` with care. If you were to define a scrolling area that overlapped an existing graph region, subsequent graphic routines could erase important messages from the screen.

The REGION routine associates a graph region with a particular graph number. (Graph numbers are discussed in a later section.) With the aid of REGION, you can display two graphs in the same region or you can use different regions for different graphs.

When you choose a graph region with a REGION statement, all graphic and text material is erased from the region, in preparation for the display of new material.

REGION also redefines the scrolling area so that it no longer occupies a part of the screen being used for a graph region.

For example, the statement

```
REGION("UPPER",1)
```

causes subsequent routines to display graph 1 in the upper region. The statement also resets the scrolling area to rows 11-24. In this respect, the REGION statement performs the same operation as the statement

```
ROLL_AREA(11,24)
```

Finally, this sample REGION statement erases all previous graphic and text material from rows 1 to 10 — the area corresponding to the upper region.

The REGION routine, and the expression "graph 1," are discussed in more detail in later sections.

### NOTE

REGION acts only on subsequently displayed graphs. That is, if you display graph 1 in the full region, you cannot change its placement with a later REGION statement.

When you are finished with this demonstration, type

DISPLAY\_CLEAR

to clear all text from the screen and to reset the scrolling area to the full 24 lines.

## DISPLAYING TEXT

There are two general properties, or modes, that affect the appearance of text strings on the MINC terminal screen. *Character mode* affects the appearance of all characters displayed on the screen. *Display mode* affects the general appearance and properties of the screen itself. These modes are controlled by the graphic routines CHAR\_MODE and DISPLAY\_MODE, respectively.

### Character Mode

Characters can be displayed in the “normal” manner, or as bold-face, reverse video, underlined, or flashing characters. Combinations of these properties are also allowed. Figure 3 shows the variety of character modes available on the MINC terminal.

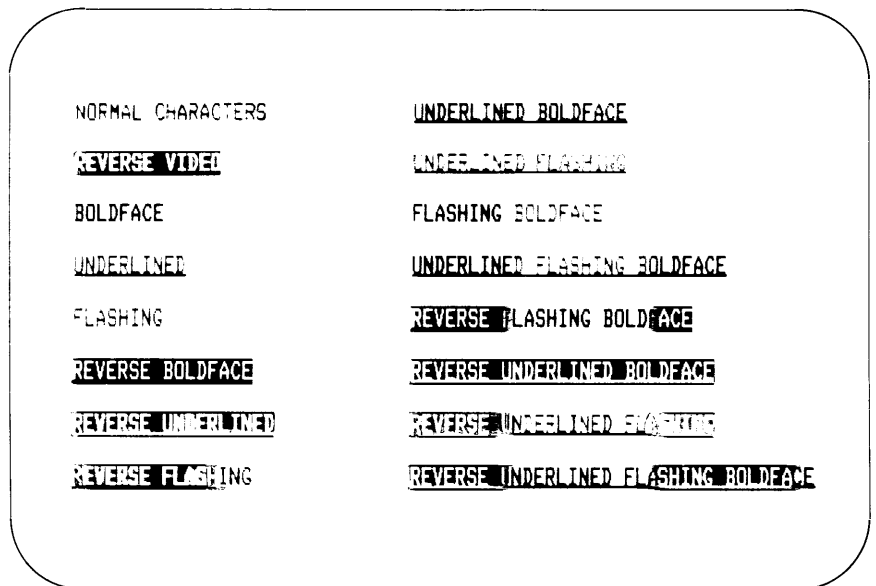


Figure 3. MINC Character Modes

The character mode is set by the CHAR\_MODE routine. Any time you use CHAR\_MODE, the character mode changes immediately — even in the middle of a row. Notice that in Figure 3 characters of different modes appear on the same row, demonstrating this property.



If you want to demonstrate the `CHAR_MODE` routine to yourself, type this statement on the MINC terminal:

```
CHAR_MODE("REVERSE")
```

Notice that the `READY` message, which appears after you type the statement, is in reverse video characters. All text that you type will now be displayed in this mode until you change it. To change back to normal character mode, type

```
CHAR_MODE("-REVERSE")
```

`CHAR_MODE` changes the character mode of all characters subsequently displayed on the screen. The new mode remains in effect until you change it with another `CHAR_MODE` statement.

Many other routines allow a “local” character mode setting. A local setting affects only the text string (or special symbol) displayed by the routine; subsequent text is not affected. Figure 3 was created by one of these routines, `HTEXT`. Other routines that allow a local character mode are `BOX`, `LABEL`, `PUT_SYMBOL`, `TEXT_LINE`, and `VTEXT`. If you want to examine the definitions of these routines, see the appropriate sections of Part 2.

The MINC terminal has the following characteristics, called *display modes*, that apply to the entire screen. The `DISPLAY_MODE` routine controls all these properties.

## Display Mode

1. Background color for text displays — the `BRIGHT` option displays all text as black characters on a white background. (The normal screen, with white letters on a black background, is selected by the option “`-BRIGHT`”.)
2. Scrolling mode — the `JUMP` option changes to “jump” scrolling instead of the normal, or smooth, scrolling. (Smooth scrolling is selected by the option “`-JUMP`”.)
3. Screen width — the `LONG` option changes the screen width to 132 columns. (The normal screen width, 80 columns, is selected by the option “`-LONG`”.)

When you use `DISPLAY_MODE` with the `LONG` option, subsequent graphs appear in a long graph format. The `LONG` option does not increase the number of points that can appear in a graph, but it does increase the number of numerical units and tick marks used to mark the graph’s axes.

## GRAPH NUMBERS

The MINC system allows you to display two completely distinct graphs at the same time. When you do so, you and your program distinguish the two graphs by number. There are only two graph numbers in MINC graphic programming: 1 and 2.

A graph number is really just a label that refers to a particular picture. Graph 1, for instance, could refer to a graph displayed in the full region, or it could apply to a graph that occupies only the upper or lower region.

With most of the graphic routines you can include a graph number in the statement optionally; if you do not include the graph number in such routines, they refer to graph 1 by default.

Many of the routines used to change the appearance of existing graphs also allow you to specify a graph number of 0. A graph number of 0 makes the routine statement apply to both graph numbers.

## THE GRAPHIC MEMORY

The MINC terminal displays graphs by storing information in an internal graphic memory. This memory resides in the MINC terminal, not within the MINC workspace that is used to store programs. Having a separate graphic memory allows you to display and erase graphs without changing the contents of the workspace.

You will find this feature extremely useful. It allows you to calculate or otherwise acquire a set of data, and to then display the same data in various graphic formats. In the section of this book titled "Introduction to the GRAPH Routine," this technique is used extensively. After creating a set of computed data, you will, in that section, display the data many different ways by typing graphic routine statements in immediate mode.

### Erasing a Graph

You can control the appearance of graphs either by making them invisible or by erasing them.

When you erase a graph, the graph not only disappears from the screen but also is erased from the graphic memory. Erasure is controlled by the ERASE\_GRAPH routine, which allows you to erase an entire graph from the screen or to erase only selected parts of the graph.

ERASE\_GRAPH does not erase the contents of variables or arrays in MINC's workspace. You therefore can repeat the original graph-drawing statement, or you can display the graph in a different form, without having to rerun the program that computed the original data.

When you make a graph invisible, the graphed points disappear from the screen, but the data remain in the graphic memory. Therefore, you can redisplay the same graph without having to repeat the graph-drawing statement. Visibility and invisibility are controlled by the VIEW routine, defined in Part 2.

### Making a Graph Invisible

#### NOTE

The effect of VIEW on the picture is more limited than the effect of ERASE\_GRAPH. The "ALL" option of ERASE\_GRAPH erases everything in the specified graph region. The VIEW routine only blanks out the *points* used to make the graph; VIEW does not remove the axes, axis units, or grid features.

The MINC terminal has a feature called *setup mode* that allows you to define, save, and recall the terminal's characteristics. You can find a general discussion of setup mode in Book 7. This section defines a subset of the setup mode parameters called the *standard initial state*. In order to use the terminal for graphic programming, you must set the following parameters to standard values:

### THE STANDARD INITIAL STATE

1. Background color (digit 3 of group 1 on "SETUP B" display) should be BLACK (the 0 setting).
2. Screen width (the 9 key on "SETUP A" display) should be 80 columns.
3. Scrolling mode (digit 1 of group 1, "SETUP B" display) should be SMOOTH (the 1 setting).
4. Escape sequence type (digit 3 of group 2, "SETUP B" display) should be set to ANSI (the 1 setting).

The standard initial state is in effect when your terminal arrives from the factory. However, the terminal gives you the ability to change these parameters to new settings. Therefore, because another user may have changed the saved parameters, you should

check the parameters periodically to make sure they are in the standard initial state.

The following procedures will ensure that these parameters are set properly. When you follow these procedures, the terminal power must be ON, although you do not need to insert or run the MINC system diskette.

1. With the terminal on, press the SETUP key at the upper left corner of the keyboard. The screen changes to show the display in Figure 4.
2. While holding down the SHIFT key, press the R key. Your screen should now be identical to Figure 4. (The combination SHIFT/R recalled the parameters stored in the terminal.)

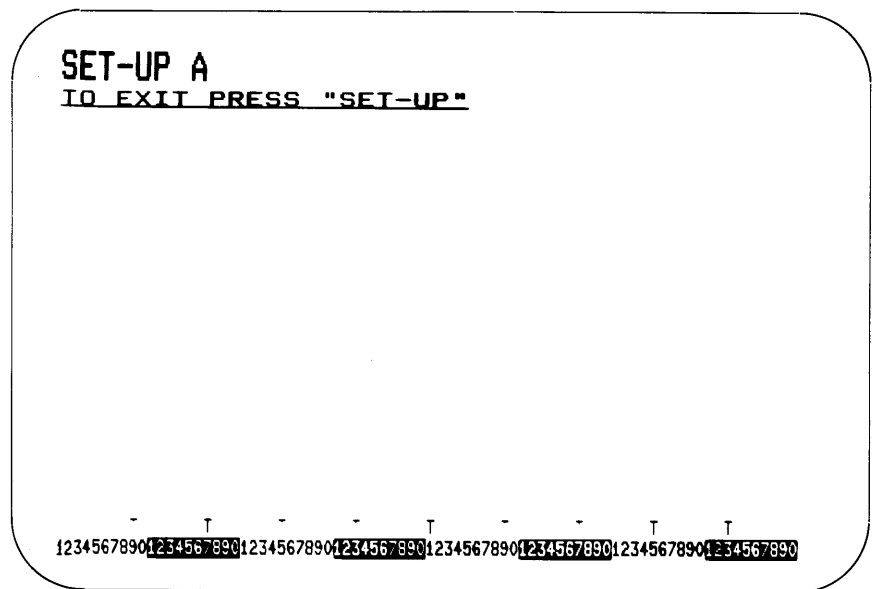


Figure 4. Primary Setup Mode Display

Compare carefully the row of numbers at the bottom of your screen with the corresponding numbers in Figure 4. This row should have 8 groups of 10 digits, marking 80 columns on the screen; if so, proceed with the next step. If instead your terminal shows 132 columns, press the 9 key. Now your screen should have 80 columns.

3. Press the 5 key. Your screen should change to the secondary setup mode display shown in Figure 5.

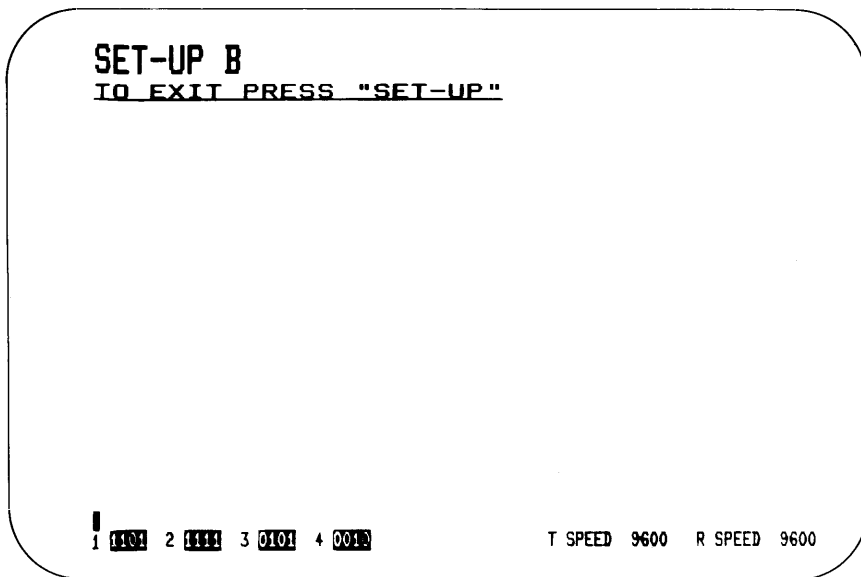


Figure 5. Secondary Setup Mode Display

Figure 5 shows some of the displayed digits in red ink. Compare these features carefully with the corresponding parts of your display. If the elements of your display are different, you must reset them.

To reset a setup parameter, press either the left-arrow key (←) or the right-arrow key (→). When you press these keys, the cursor will move back and forth above the rows of digits. When the cursor is directly over the digit you must change, press the 6 key. If the digit was previously 0, it will now be 1, and vice versa. Compare your screen again to Figure 5.

4. When the digits on your screen match those shown in Figure 5, your terminal is in the standard initial state.

Now hold the SHIFT key down and press the S key. The message WAIT will appear briefly on your terminal and then disappear. The SHIFT/S operation has saved the setup parameters in your terminal's internal memory. The parameters will remain there unless you change them again and save a new set with another SHIFT/S operation.

Once setup parameters have been saved in the terminal's memory, they remain there even when the terminal's power is turned off. When the power is turned on again, the saved parameters are recalled from the memory and put into effect.

## *GRAPHIC PROGRAMMING*

The MINC system also has a routine, `TEXT_INIT`, which recalls the parameters saved in the terminal's memory. `TEXT_INIT` is described in Part 2 of this manual. This routine can be very useful in graphic programming, because during a programming session, the setup parameters can be changed either by certain routines or by your reentering setup mode and changing them directly. You can use `TEXT_INIT` at any time to return the setup parameters to a known state.

# CHAPTER 3

## INTRODUCTION TO THE GRAPH ROUTINE

This section contains examples that are highly interactive with the MINC system itself, illustrating a wide variety of related graphic operations. By concentrating on the use of the GRAPH routine, this section introduces most of the features of graphic programming with MINC. Try to finish all the suggested exercises in this section during a single session at the MINC terminal. Such a session takes from one to three hours.

To get the most benefit from this section, you should be familiar with the BASIC programming language. If you do not have previous experience with BASIC, you should read *Book 2: MINC Programming Fundamentals* before going further.

To begin this session, sit down at the MINC terminal and start the system. If you are unsure about how to do so, read *Book 1: Introduction to MINC*. When you have the system going, the word

READY

should be on the screen, meaning that the system is ready for your first command.

Your first task is to create some sample data that can be displayed on the screen later. As you probably know, analog signals can be represented by the sine and cosine functions. Since MINC can easily calculate and store large arrays of sine and cosine values, we will use the sine function to create the first sample

**INITIAL  
PROCEDURES**

**Creating a Data  
Structure**

data. To create the data structure for the examples, type the red portion of the following dialog.

```
READY
NEW MDEMO

READY
10 DIM X1(199),F1(10),Y1(199)
20 FOR I=0 TO 199
30 Y1(I)=SIN(I*PI/50)
40 X1(I)=LOG10(I+1)
50 NEXT I
60 END
SAVE

READY
RUNNH
```

You have just created and run a very short BASIC program called MDEMO. Pause a moment and consider what this program is doing. The NEW command creates a new program called MDEMO. Line 10 creates three arrays. Lines 20-50 load 200 sine values into the array Y1 and 200 logarithms into array X1. Because no values were assigned to the array F1, its 11 elements all contain zeros.

PI is a predefined constant in MINC programming; the expression  $I*PI$  is divided by 50 so that the expression  $I*PI/50$  will go through the range  $0-2\pi$  nearly twice during the FOR/NEXT loop. The array Y1 will therefore contain Y values corresponding to two full cycles of a sine wave.

The values in the arrays X1 and Y1 are now available in your workspace. You can display these values either by adding graphic routine statements to MDEMO and running it again, or by writing routine statements in immediate mode, with reference to Y1. Because we are interested in the effects of single routines during most of this session, we will rely primarily on simple routine statements in immediate mode. Using immediate mode will also eliminate our waiting for new values to be computed each time we want a new display.

The arrays will retain these same values unless you change their elements with assignment statements, erase the program workspace with the SCRATCH command, or perform some other operation (such as turning off the power to MINC) that destroys the data.



We will begin the display examples with the GRAPH routine, which, as you will soon see, gives you a multitude of choices for display formats.

Type this line:

**First Display**

```
GRAPH(...Y1(0))
```

If you typed the GRAPH statement correctly, your screen now looks like the one shown in Figure 6.

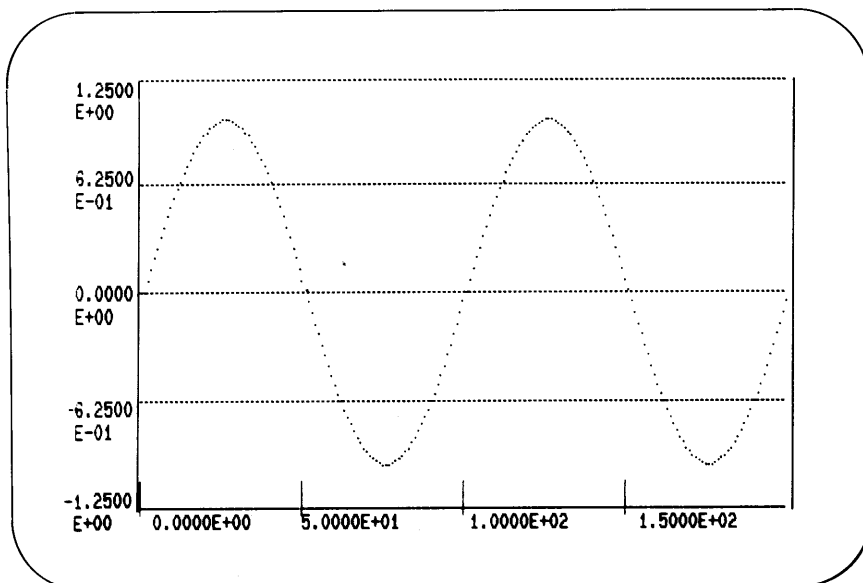


Figure 6. A Simple Sine Graph

### NOTE

Do not type commands to MINC while a graph or other figure is being drawn on the screen. If you do so, MINC will not display the characters you type on the screen, and the command may be ignored.

An exception to this rule is the command CTRL/C. If you type CTRL/C once or twice, the graph-drawing process will stop and the READY message will reappear.

The NO SCROLL key, when first pressed, will “freeze” a graph-drawing procedure. When the key is pressed a second time, the graph-drawing resumes.

If your screen does not look like Figure 6, examine your GRAPH statement. Are there any typing errors? The GRAPH statement you type must have exactly the right number of commas and parentheses. If you find a mistake, retype the GRAPH statement. (You should also be sure that you typed the MDEMO program correctly.)

### Omitted Arguments

You may be surprised that such a small number of statements created this complex picture. For one thing, you took advantage of all the default conditions allowed by the GRAPH routine. The three commas in the GRAPH statement are there to hold the place of missing arguments, much as the zeros in the number .0005 hold the places of missing digits. In other words, a routine statement with two leading commas has a very different meaning from a statement with three leading commas, just as .005 and .0005 represent different numeric values.

The complete form of a GRAPH statement is as follows:

```
GRAPH(option,number,start X,start Y,increment,shade line,  
graph number,start index)
```

The blue ink shows the GRAPH arguments that are optional. In your first use of the GRAPH statement, you omitted them all.

When you leave out arguments in a routine statement, the MINC system can make reasonable assumptions (called *default conditions*) about the intended appearance of the graphic display. The GRAPH routine has only one required argument, called the *start Y* argument. You supplied the element Y1(0). There are seven optional arguments that are legal in a GRAPH statement. From the form of the GRAPH statement that you just typed, you can see that you included only three leading commas, because three of the optional arguments come before the start Y argument. The other four follow start Y, and they do not need to be marked with commas. This, too, is analogous to zeros in a decimal number, since .005 and .0050 have the same meaning.

### The Option String

The first missing argument in the above GRAPH statement — that is, the argument that would come before the first comma — is the option string.

The *option string* is a single word or series of words, enclosed in quotation marks. If the string contains a series of words, you must separate them with commas.

**Clearing the Screen**

Now we will use the GRAPH\_INIT routine, which erases the graphic memory. Then we will be ready for a new graph. Type

```
GRAPH_INIT
```

Notice that the GRAPH\_INIT statement erased the graph, but left the text on the screen. GRAPH\_INIT also resets the scrolling area to rows 1-24. Now type

```
ERASE_TEXT
```

to erase the text from the screen.

You can also clear the screen by typing

```
DISPLAY_CLEAR
```

The DISPLAY\_CLEAR routine is equivalent to typing GRAPH\_INIT followed by ERASE\_TEXT.

Now we will set the system up to display two graphs at once, in different graph regions. Type the red part of this dialog:

```
READY
REGION("UPPER",1)
```

```
READY
REGION("LOWER",2)
```

Notice first that these two REGION statements have option strings. UPPER refers to the upper graph region. LOWER refers to the lower region. Only one word is allowed in the option string for REGION. The only other allowable word is FULL, which refers to the full graph region. If you omit the option word from a REGION statement, FULL is used by default.

The numbers 1 and 2 are graph numbers. MINC allows a maximum of two graphs to appear on the screen at once. Therefore, the REGION statements have told MINC to put graph 1 in the upper region and to put graph 2 in the lower region.

The GRAPH routine accepts a graph number as one of its optional arguments. When the graph number is left out of a GRAPH statement — or any other statement in which it is optional — the graph number has a default value of 1. Thus, your first picture was graph 1, displayed in the full graph region.

**Displaying Two  
Graphs****OPTIONAL FEATURES****The Graph Number  
Argument**

To redisplay the same picture in the upper region, type

```
GRAPH(...,Y1(0))
```

The upper region now should contain a smaller version of your sine graph. Notice that we used the same array, Y1, since its sine values are still intact in your workspace.

Now type

```
GRAPH("BRANDS,SHADE,LINES",30,,Y1(0)...5,2)
```

Your screen should now look like the one shown in Figure 7.

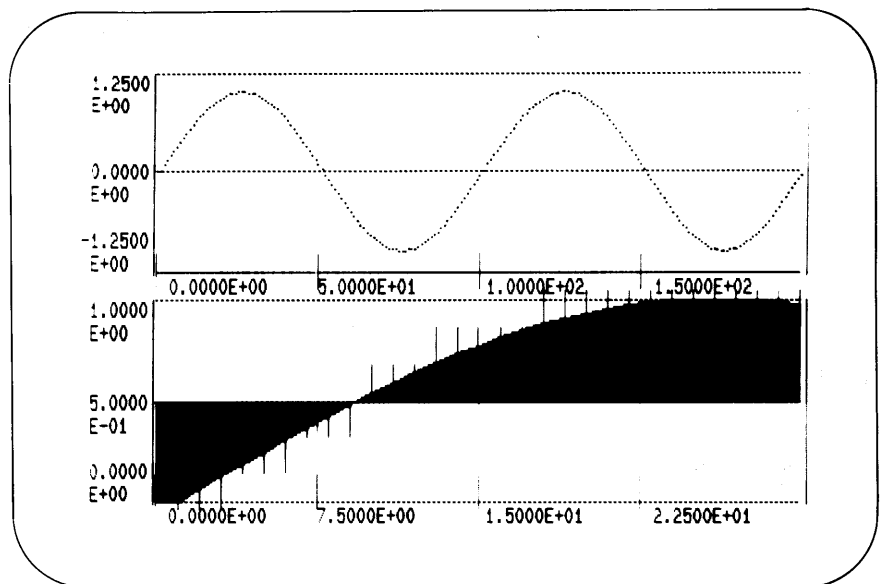


Figure 7. A Graph with Brands (Lower Region)

The figure in the bottom half of the screen is graph 2 — the last number in the GRAPH statement is the graph number 2.

### The BRANDS Option

Your second GRAPH statement includes a three-word option string. The word BRANDS creates the small vertical line segments that appear at regular points along graph 2. The BRANDS option is used to mark the exact horizontal location of data that might otherwise be hard to notice in a complicated picture. Notice that there are 30 brands in graph 2; each brand marks the position of one datum. The number 30 in the statement is called the *number argument*; this GRAPH statement plotted the first 30 data from array Y1. The number argument is explained later in more detail. The points that fall between the

brands in graph 2 are interpolated line segments created by the LINES option. This option is also explained later.

The shaded appearance of graph 2 is produced by the second option word, SHADE. SHADE tells MINC to shade a graph by drawing vertical lines from the graphed points to a particular horizontal line called the *shade line*. In this case, the shade line is at  $Y = .5$  — that is the purpose of the argument .5 in the second GRAPH statement.

### The SHADE Option and the Shade Line Argument

Try this statement, which omits the shade line argument:

```
GRAPH("SHADE,LINES",30,,Y1(0),...1)
```

Notice that, because you changed the graph number to 1, this statement erased the upper region and then displayed a new graph (see Figure 8).

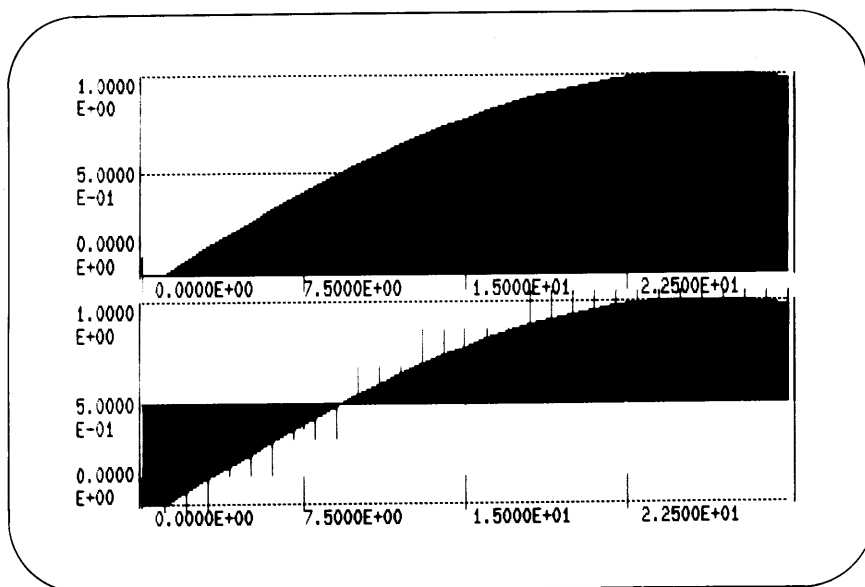


Figure 8. Two Shaded Graphs, with and without Shade Lines

As you see, the graph in the upper region is shaded from the bottom of the upper region. If you include “SHADE” in the option string, but omit the shade line argument, the bottom of the graph region is always used for the shade line. In this case, the bottom of the region corresponds to the Y coordinate 0.

Finally, type this statement, which displays only the 30 points, without the interpolated lines (see Figure 9):

```
GRAPH("SHADE",.30,,Y1(0),,2)
```

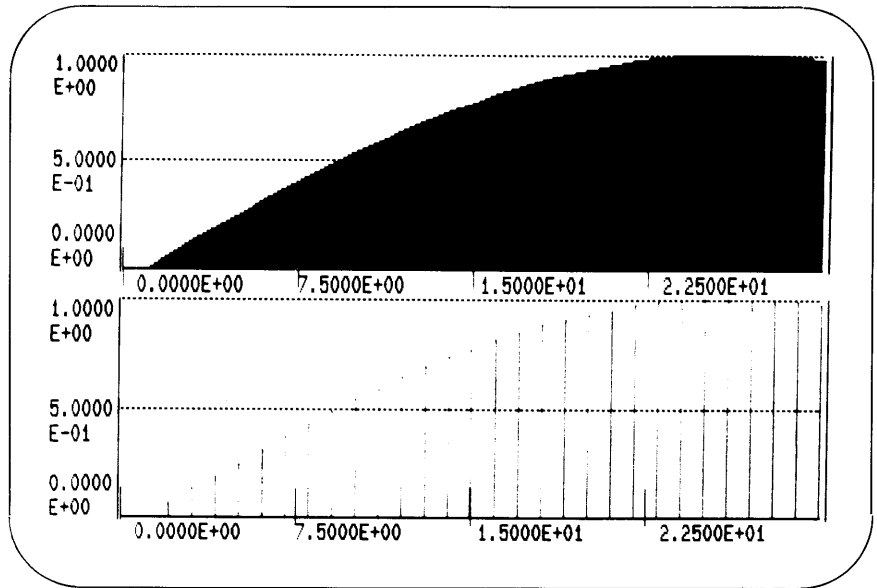


Figure 9. Shaded Graph without the LINES Option (Lower Region)

As shown in Figure 9, the shading loses some of its effectiveness when there are only a few points on the graph. For this reason, the SHADE and LINES options are often used together, with the BRANDS option serving to distinguish the actual data points from the interpolated points created by LINES.

### Erasing a Graph

You have just seen one method of eliminating an old graph: by using GRAPH again and replacing the old graph with a new one. You can also simply erase the graph in a particular region without displaying a new one. Examine, but *do not type* this statement:

```
ERASE_GRAPH("ALL",,2)
```

This statement would remove the shaded graph from the lower region, leaving graph 1 in the upper region. Notice that the ERASE\_GRAPH statement has the option word "ALL". That means "erase all the features of graph 2." As discussed in Part 2 of this manual, ERASE\_GRAPH can be used to remove some features from a graph without affecting the others. Notice also that the graph number (2) was included as the last argument in ERASE\_GRAPH. You could have left the graph number out, but then MINC would have used the default condition and erased graph 1. You want to avoid mistakes of that kind, since

once ERASE\_GRAPH is used on a graph, the graph can be recovered only by redrawing it. With the simple case we are using, redrawing is a minor problem (you have already done it in this session). In real working situations you could conceivably lose valuable information by an accidental erasure.

Partly to protect against accidental erasures, MINC has a routine called VIEW, which can make a graph disappear temporarily without erasing it from the graphic memory. Try the VIEW routine on graph 2 by typing

```
VIEW("INVISIBLE",2)
```

Notice that the graph points, lines, and brands have disappeared from graph 2, leaving only the features shown in Figure 10.

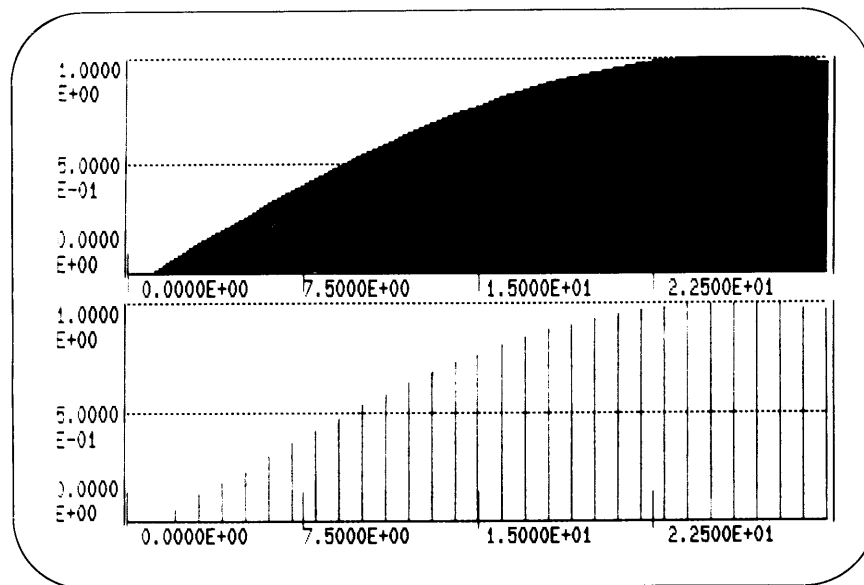


Figure 10. Result of VIEW("INVISIBLE",2)

You may already have realized that VIEW is especially useful for displaying two graphs sequentially in the full graph region. That is, if the two graphs make too confusing a picture, you can remove one of them without erasing it. Then you can put the invisible graph back on the screen with another simple statement. Type

```
VIEW("-INVISIBLE",2)
```

Notice that the points have been restored to graph 2. This VIEW

### Making a Graph Invisible

statement shows another important feature of MINC routines: In many cases, you can reverse the effect of an option word by preceding it with a minus sign (-). In this case, the option “-INVISIBLE” means “make the invisible graph visible again.” (When you omit the graph number from a VIEW statement, graph 1 is affected by default. When it is included, the graph number comes after the option word.)

The primary difference between VIEW(“INVISIBLE”) and ERASE\_GRAPH is that the graph number used in a VIEW statement remains “in use.” That is, the graph affected by the VIEW statement is preserved in the graphic memory, and you can therefore redisplay it with another VIEW statement. However, the fact that the graph number is in use does not mean that it is illegal in other statements. For example, type

```
VIEW(“INVISIBLE”)
```

to make graph 1 invisible. But, instead of restoring graph 1, type

```
GRAPH(“POINTS”,,Y1(0))
```

This statement has replaced the shaded graph in the graphic memory with an unshaded graph containing all 200 points. The GRAPH statement also negated the previous INVISIBLE option, so that the new graph appeared immediately on the screen.

Now let’s examine another optional GRAPH argument. First, type

```
DISPLAY_CLEAR
```

to erase the graphic memory and clear the screen.

### The Number Argument

The next optional argument we will examine is the *number argument*, which follows the option string. So far, most of our graphs used the entire contents of the array Y1. All the graphs were plotted with the value in Y1(0) as the Y coordinate of the first point. When we omitted the number argument, the GRAPH routine displayed all the Y coordinates from Y1(0) to the end of the array, which in this case is Y1(199). The number argument, which can assume any value from 0 to the entire array dimension, allows you to select the exact number of points to display. The following statement demonstrates the number argument.

```
GRAPH(“LINES,BRANDS”,20.,Y1(100),,2)
```



The result should look like Figure 11.

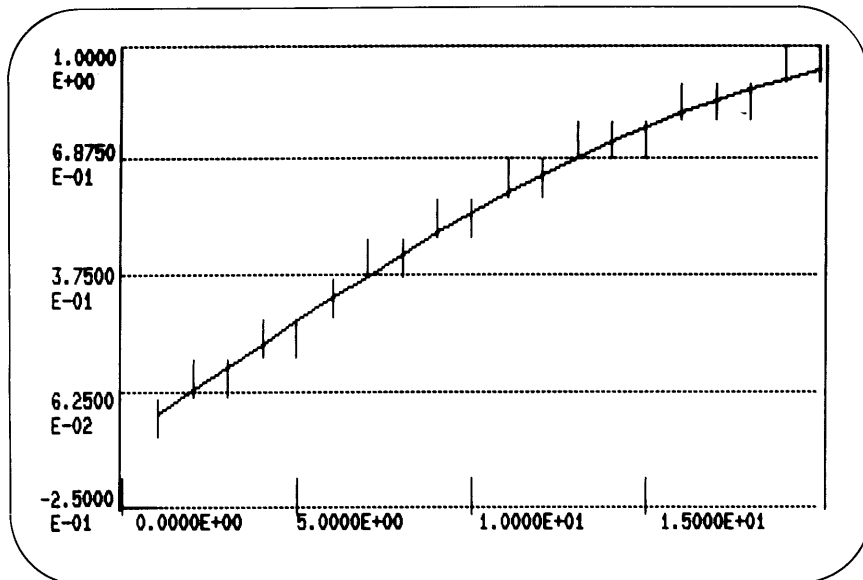


Figure 11. GRAPH Display of 20 Points

Notice that only 20 points appear on this new graph. The positions of the points are marked by the brands. Note also that this new graph is displayed in the full graph region. That is because you typed a `DISPLAY_CLEAR` statement, which, in addition to erasing the graphic memory, also restored both graph 1 and graph 2 to the full region.

Before we proceed, let's digress for a moment and discuss the MINC feature known as *autoscaling*.

### Autoscaling

Autoscaling is a feature of the MINC routines `GRAPH` and `BARGRAPH`. Its purpose is to ensure that the display of a certain set of data is as visually appealing and as easily interpreted as possible.

When you supply a set of data to `GRAPH` and `BARGRAPH` — that is, by supplying the start Y argument and the number argument—the routines detect the largest and smallest numerical values in the set. The two routines then adjust the largest and smallest *axis labels* so as to ensure that the full range of input values appears on the screen. In the usual mode, the minimum and maximum values on the axis labels slightly exceed the limits of the input values. (For more details on this adjustment procedure, see the description of the `EXACT` option.)

Because of autoscaling, the display is expanded to fill the graph

region, even when the data occupy a very small range, such as 0 to .001. The axes are automatically relabeled to retain the numerical information in the displayed graph. You have already seen this effect when you plotted the 20-point sine graph in the previous example.

Autoscaling works the same way for data that have a very large range between the maximum and minimum values. The autoscaling is independent on the two axes, so that the X values can occupy a very small range, and the Y values, a very large range.

However, when the GRAPH or BARGRAPH routine is presented with input sets that exceed 512 elements in length MINC can fit only 512 points on the screen at once. Although the routines will not prevent you from naming larger arrays, the resulting display may not be a true representation of your data unless you take extra measures. These measures are simple, but we will discuss them later. Until then we will continue using arrays of fewer than 512 elements.

You can override the autoscaling feature of GRAPH (and BARGRAPH) with the WINDOW routine. WINDOW sets fixed limits for data that can appear on a particular graph; data that fall outside such a window are legal but are not displayed on the graph.

Autoscaling is used by GRAPH and BARGRAPH in the following three cases.

1. You have not used WINDOW at all before using GRAPH or BARGRAPH.
2. You included no X or Y arguments in the last WINDOW statement before the GRAPH or BARGRAPH statement.
3. You have used the GRAPH\_INIT or the DISPLAY\_CLEAR routine before using GRAPH or BARGRAPH.

For more details on the WINDOW routine, see Part 2.

We will now return to our discussion of optional arguments.

**The Options GRID,  
TICKS, UNITS,  
HLINES, and VLINES**

As you have seen on previous graphs (for example, Figure 11), all the displays created by GRAPH had a superimposed pattern of horizontal lines, tick marks (the short vertical lines on the X axis), and numeric labels on both axes. The pattern is called a *grid*, and its purpose is to make the graphs easier to interpret.

These grids are created by the option word GRID. You have not needed to include this word in your GRAPH statements, because it is included by default.

The GRID option is associated with four other options: TICKS, UNITS, H LINES, and V LINES. Except for V LINES, these options simply refer to the individual features of the grid. To demonstrate how these options work, we will initialize the graphic memory and then display a variety of graphs in the full graph region. Type

```
READY
DISPLAY_CLEAR

READY
GRAPH("GRID,VLINES",,,Y1(0))
```

The resulting display is shown in Figure 12.

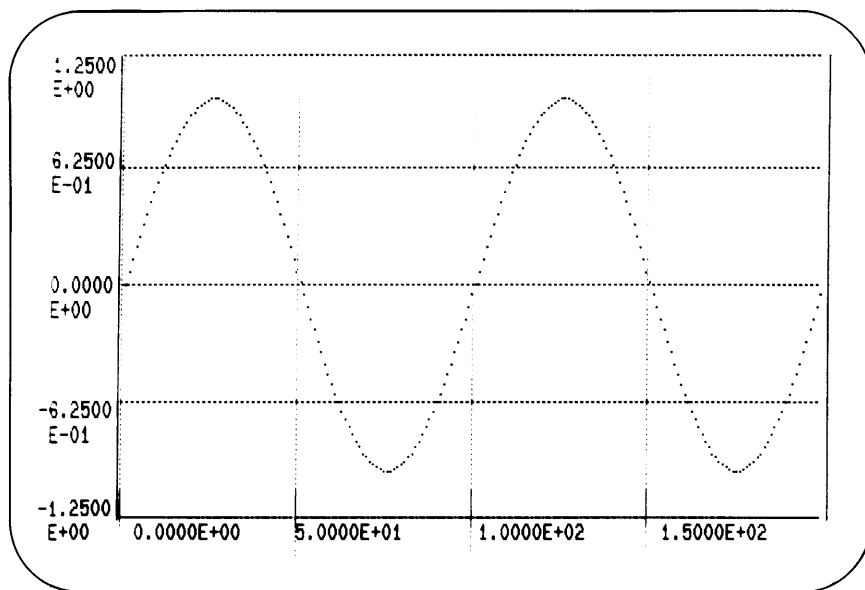


Figure 12. Result of GRID and V LINES Options

As you see from the display, the V LINES option replaces the tick marks with vertical “grid lines” that extend the height of the screen. In many cases, your graphs will be more easily interpreted if you use the V LINES option.

#### NOTE

The vertical grid lines ALWAYS have a height equal to the height of the full screen, *including the scrolling*

*area*. If you display two different graphs in the upper and lower graph regions, the vertical grid lines from each region will extend through the other region as well. In some cases, this property can lead to confusing pictures. In other cases, it permits you to make a useful visual comparison of the X data on two graphs.

V LINES can be used only when the GRID option is also included in the routine statement, either explicitly or by default. A statement such as

```
GRAPH("-GRID,HLINES,VLINES,UNITS",,,Y1(0))
```

is "legal," in that it produces no error message, but it does not display horizontal lines, vertical lines, or units.

It is legal to include H LINES, U NITS, and T I C K S in a statement in which GRID is *not* negated, but they do not add anything to the display.

When you display a graph or bargraph in the full region, either four or five horizontal lines appear; in the upper or lower region, three lines appear.

Now try this statement, which creates horizontal lines and tick marks but omits the axis units:

```
GRAPH("GRID,-UNITS",,,Y1(0))
```

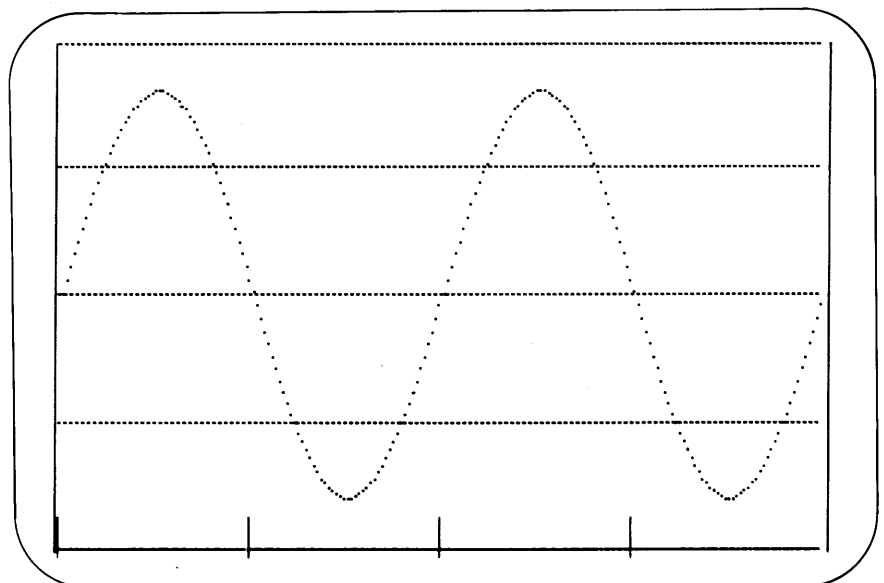


Figure 13. Result of GRID plus -UNITS Options

The screen should now resemble Figure 13.

As you see, these options produce horizontal lines and tick marks identical to those produced by your previous statements. However, the interpretive value of these features is minimal, because they are not labeled with axis units.

Let's summarize this information about grids:

1. The GRID option, by default, produces horizontal grid lines, tick marks, and axis units. The axis units appear at the intersections of these grid features with the X and Y axes.
2. If and only if the GRID option is present, the VLNES option produces vertical grid lines in place of the tick marks.
3. When GRID is negated, the horizontal lines, tick marks, and axis units cannot be produced.
4. The options HLNES, TICKS, and UNITS are useful only in their negative forms. To display grid features separately, you must use such forms as -UNITS to *subtract* the unwanted grid features.

### The EXACT Option

As you read in the section on autoscaling, the GRAPH routine normally does not use the highest and lowest values in your input arrays as the largest and smallest axis labels. Instead, the GRAPH routine makes the numerical range of the axis labels slightly larger than the numerical range of the input, in order to make the axis units in the display reasonably "round" numbers — for instance, 1.25 rather than 1.2533. Generally, this adjustment makes the grid more useful for interpreting the numeric values of particular points, and, by reducing the number of decimal digits, it makes the axis labels fit and look better on the display.

Consider this example:

```
READY
NEW EXACT
```

```
READY
10 DIM Z1(199)
20 FOR I=1 TO 199
30 Z1(I)=SIN(I*PI/420)
40 NEXT I
```

## GRAPHIC PROGRAMMING

```
50 END  
SAVE
```

```
READY  
RUNNH
```

After this calculation, the array elements Z1(1) and Z1(199) do not contain either 0 or 1. The value in Z1(1) is the sine of  $1/420 \times \pi$ , or  $7.47991 \times 10^{-3}$ . The value in Z1(199) is the sine of  $199/420 \times \pi$ , or .996617. However, if Z1 is plotted with the GRAPH routine, the smallest and largest Y axis labels are 0 and 1 (see Figure 14).

Type

```
GRAPH(...,Z1(1))
```

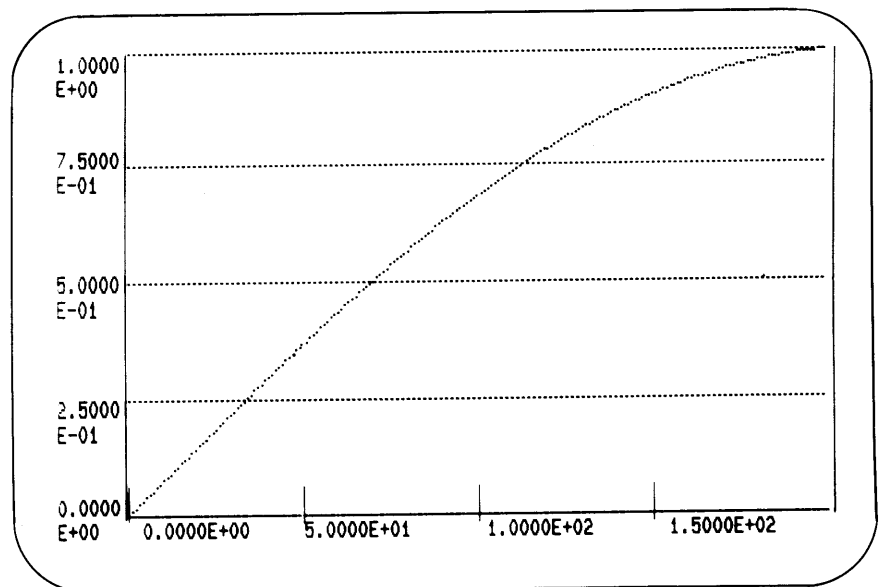


Figure 14. Result of GRAPH Statement without EXACT Option

There may be cases in which you do want to display the exact numerical values of your largest and smallest input data. The EXACT option is used for this purpose:

```
GRAPH("EXACT" ...,Z1(1))
```

Notice that the Y axis units in Figure 15 include only four decimal digits. The fourth digit to the right of the decimal point is the result of rounding less significant digits.

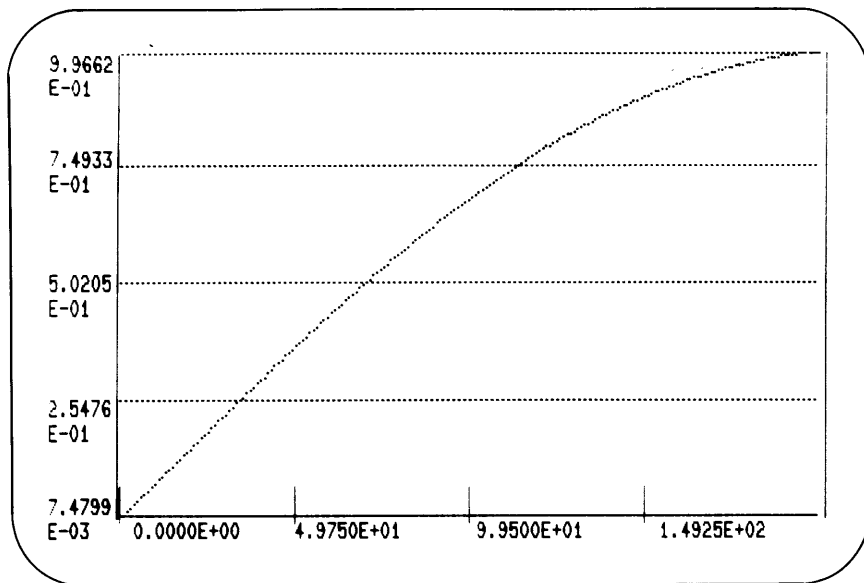


Figure 15. Result of GRAPH Statement with EXACT Option

The MINC terminal uses a feature called strip-chart mode for the display of large data sets.

### Strip-Chart Mode

The GRAPH and BARGRAPH routines, in their normal mode, will let you specify arrays of more than 512 points. However, the GRAPH and BARGRAPH routines do not produce true representations of such arrays in their normal mode of operation.

As an example, let's consider the array created by this short program:

```
10 DIM M(1400)
20 FOR I=0 TO 1400
30 M(I)= SIN(I*PI/25)*COS(I*PI/400)
40 NEXT I
50 GRAPH(...,M(0))
```

The array M contains 1401 values that represent a sine wave modulated by a cosine wave of much lower frequency. The display created by the GRAPH statement in line 50 is shown in Figure 16.

As you can see from Figure 16, the static display is not a good representation of array M. The autoscaling feature of the GRAPH routine has set the limits of the graph to -1 and 1 in the

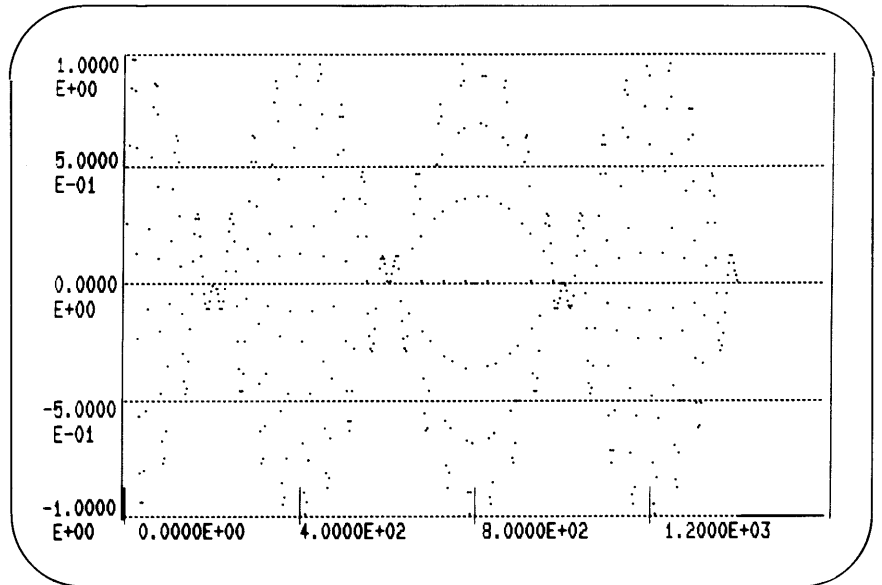


Figure 16. Static Display of a 1401-Point Array

Y direction, and to 0 and 1600 in the X direction. However, because only 512 points can appear at once, some of the points between 0 and 1401 have been dropped from the display, and the display has lost resolution. The points have been dropped at regular intervals, of course. That is why the general form, or “envelope,” of the low-frequency cosine wave is visible in Figure 16. However, little or no detail of the sine wave is visible.

The combination of limits set by the autoscaling feature (0, -1, 1600, and 1) is called the *window* of the graph. Points that fall within the window of a graph are displayed; points that fall outside the window are not displayed. The problem we have examined in Figure 16 is due to the fact that the autoscaling feature deliberately sets the window to *ensure* that the largest and smallest values are displayed.

In the simple case, the window limits are set automatically by the autoscaling feature. However, MINC contains a routine called WINDOW that lets you set explicit limits for the window; consequently, WINDOW also disables the autoscaling feature of the GRAPH and BARGRAPH routines.

Let us first use WINDOW to turn off the autoscaling feature; this action alone will allow us to see the true representation of *part* of array M. Type the red commands:

```
READY
WINDOW(0,-1,511,1)
```



```
READY
GRAPH(.511.,M(0))
```

The result is shown in Figure 17.

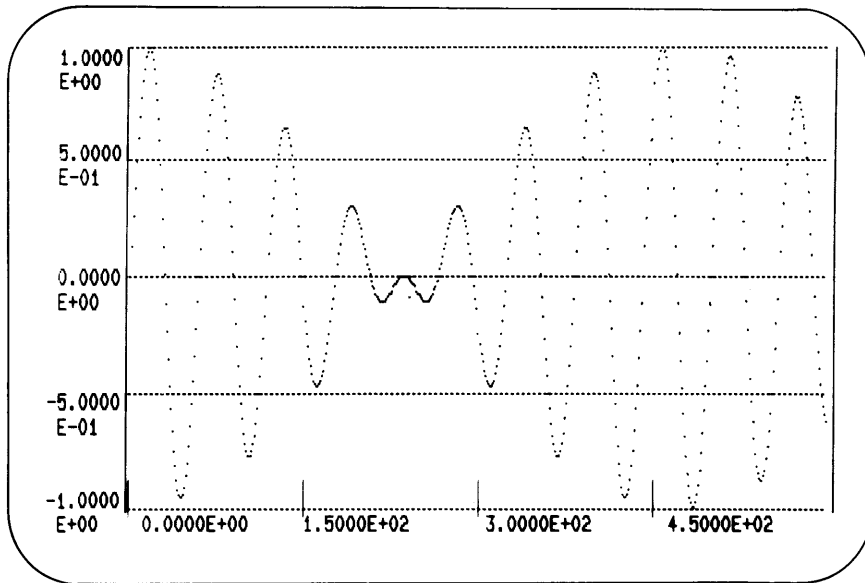


Figure 17. First 511 Points of Array M

Figure 17 shows a true representation of the first 511 points in array M. The features of both the sine and cosine functions are now clearly visible. The points between 512 and 1401 were not displayed because a number argument of 511 was used.

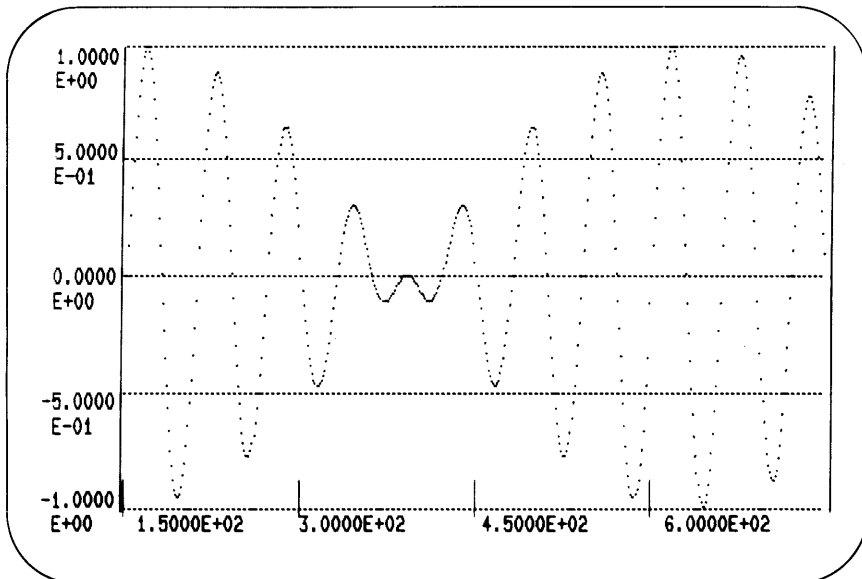


Figure 18. First 500 Points, Beginning of Strip-Chart Operation

Strip-chart mode will give us the ability to see all 1401 points in a continuous flow. Type the statement

```
GRAPH('MOVE',500,,M(0))
```

The result is shown in Figure 18.

To see the rest of the points, type

```
GRAPH('MOVE',901,,M(500))
```

The final display is shown in Figure 19.

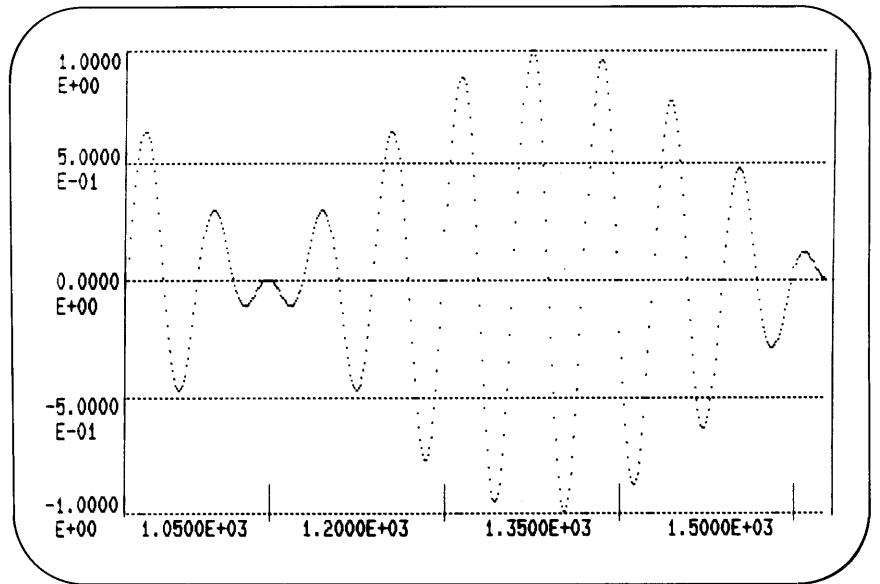


Figure 19. Last 512 Points, End of Strip-Chart Operation

The effect you observed was the strip-chart action. The second GRAPH statement plotted the remaining 901 points in array M, beginning with element M(500). As points beyond the old window limit (511) were added, they appeared at the right edge of the screen, and the old points shifted to the left.

In effect, strip-chart mode changes the window dynamically to allow new points to be added.

To leave strip-chart mode, type

```
DISPLAY_CLEAR
```

the only means of leaving strip-chart mode and returning to static displays. These routines also cancel previous WINDOW statements, which reinstitute autoscaling by the GRAPH and BARGRAPH routines.

You can also cancel a previous WINDOW statement by using the WINDOW routine again, without an argument list:

```
READY
WINDOW
```

```
READY
```

### NOTE

When you use strip-chart mode, you may notice that the screen flickers while the moving graph is being displayed. This effect occurs when your window setting (via the WINDOW routine) and the values contained in the array combine to cause points to overwrite old points.

A sample case occurs when your WINDOW statement sets the X range to 0-1.5, the first GRAPH statement displays the first 600 points, and all of the first 600 points have X coordinates between 0 and 1.5. You can correct the condition in this case by resetting the window to a lower maximum X value.

The flickering effect also occurs in some cases when the LINES option is combined with the MOVE option.

In any case, the flickering is a normal effect and does not indicate a malfunction of your terminal.

Now we are ready to examine the next optional argument. It follows the number argument and is called the *start X* argument. The start X argument is an element of an array of X coordinates. In the MDEMO program you ran at the beginning of this session, you defined a 200-element array called X1. In the same program, the elements of X1 were filled with the logarithms of the integers from 1 to 200. Before continuing, type RUN MDEMO to recreate the arrays X1, Y1, and F1.

### The Start X Argument

Now type

```
GRAPH("SHADE,LINES,BRANDS",50,X1(0),Y1(0))
```

The result looks like Figure 20.

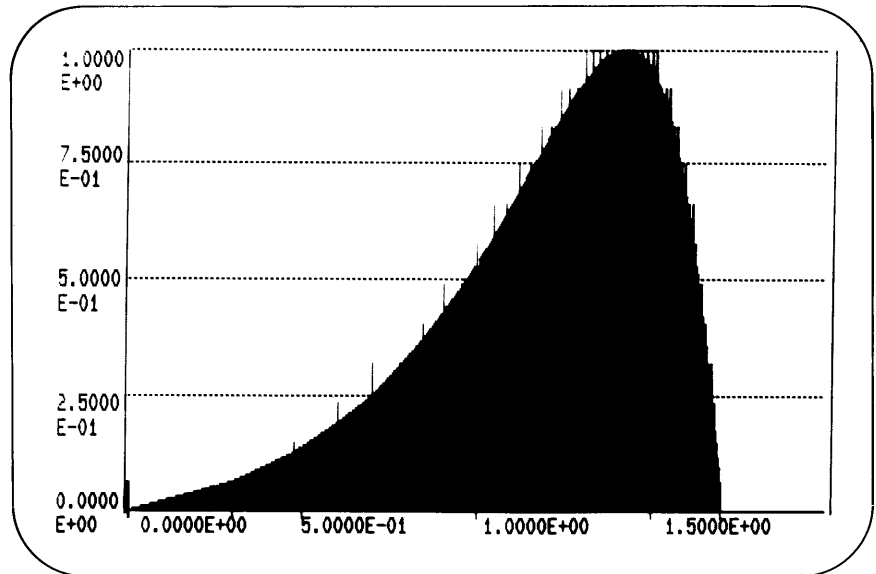


Figure 20. Logarithmic X Axis

As you can see, supplying X information to GRAPH allows you to plot nonlinear axes. Notice also that the autoscaling feature has expanded the graph to fit the full graph region, even though the values of X1(49) and Y1(49), the last data to be plotted, are very close to 0.

### The Increment Argument

We will now investigate the fifth argument of the GRAPH routine, which is the increment argument. In the examples you have seen, you have been giving GRAPH the array element at which to begin extracting data (the start Y argument) and the number of points to plot (the number argument). You have not supplied the increment argument, and the value 1 was therefore used by default. This meant that, when plotting 50 points starting at Y1(0), GRAPH plotted the data in elements Y1(0) through Y1(49). The increment, which can be any integer from 1 to 32767, allows you to precisely control the array elements used to construct a graph. Consider (and type) this example.

```
READY
SCRATCH
```

```
READY
10 DIM Y(200)
20 FOR I=1 TO 200 STEP 2
30 Y(I-1)=SIN(I*PI/100)
40 Y(I)=COS(I*PI/100)
50 NEXT I
```

```

60 REGION("UPPER",1)
70 REGION("LOWER",2)
80 GRAPH("SHADE,LINES,BRANDS",100,,Y(0),2,,1)
90 GRAPH("SHADE,LINES,BRANDS",100,,Y(1),2,,2)
100 LABEL("BOLD","SINE WAVE","GRAPH 1",1)
110 LABEL("BOLD","COSINE WAVE","GRAPH 2",2)
120 END
RUNNH

```

The resulting display should look like Figure 21.

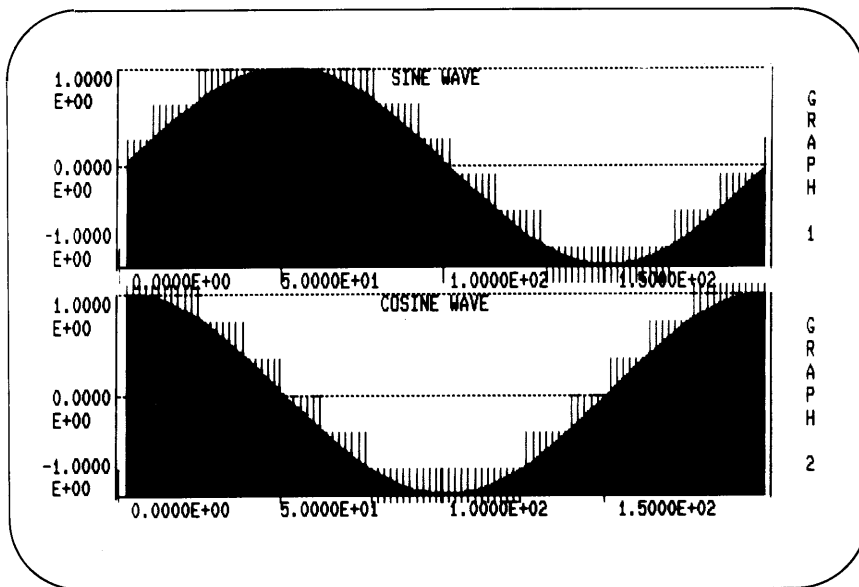


Figure 21. Sine (Upper) and Cosine (Lower) Waves

The increment argument has enabled you to display two waveforms from data in a single array. The sine and cosine waves were chosen for convenience.

A realistic application for the increment is analog-to-digital conversion of output from a two-channel laboratory instrument. Some lab module routines (described in *Book 6: MINC Lab Module Programming*) handle such an application by storing digital values from the two channels in alternate elements of a single array.

There is still one GRAPH argument to be examined: the *start index* argument. This argument is the first element of the *index array*. In the program MDEMO, which you ran at the beginning of this session, you created the 11-element array F1 for use as the index array. We have not used F1 yet, and we have deferred until now the discussion of the index array as a data structure.

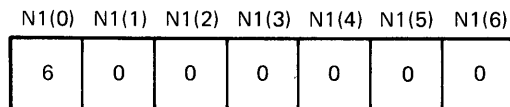
### The Start Index Argument

**INDEX ARRAYS**

Index arrays are used only with the GRAPH and BARGRAPH routines. Index arrays have the following structural and functional differences from the arrays of X and Y coordinates.

1. The purpose of the index array is to let you quickly find the coordinates of an interesting-looking feature of a GRAPH or BARGRAPH display. For example, you might want to quickly find the numerical values of local maxima in a waveform. You may use the index array to find the coordinates of such points *in the original arrays of X and Y coordinates*.
2. Index arrays cannot exceed 11 elements in length.
3. You assign a value to only one element of an index array before using it in a GRAPH or BARGRAPH statement. You always assign this value to the first element of the index array — that is, to the element named in the start index argument of your GRAPH or BARGRAPH statement.
4. During the execution of the statement, MINC *writes* values into the index array. Any or all values in the index array may be changed by the routine, including the value you placed in the first element.
5. The value written into the first element by the routine is an *index count* — it counts the elements of the index array that are changed during the execution of the routine.
6. Except for the index count, the values written into the array are indices. Indices are always either 0 or some positive integer.

Figure 22 shows a diagram of a typical (7-element) index array that is ready to be used in a GRAPH or BARGRAPH statement.



MR-1627

Figure 22. Index Array before Execution

The 6 in element N1(0) was inserted by the user, perhaps with a simple assignment statement such as N1(0)=6. The assignment statement is made *before* N1(0) is named in a GRAPH or BARGRAPH statement. Therefore, the value in N1(0) is not yet

the index count. Instead, the value 6 counts the maximum number of indices that the forthcoming routine can insert in the index array. Because there are 6 elements remaining after N1(0), the entire index array (except N1(0)) is now available for indices.

Now consider this example, but do not type it:

```
GRAPH("POINTS,INDEX",200,X1(5),Y1(10),...,N1(0))
```

This statement displays 200 points, using 200 elements of X1 and Y1 as the X and Y coordinates of the points. X1(5) contains the first X coordinate and Y1(10) contains the first Y coordinate.

N1(0) is the first element of the index array (which was filled with the value 6). Figure 23 shows how the index array might look after the above GRAPH statement is executed.

N1(0)	N1(1)	N1(2)	N1(3)	N1(4)	N1(5)	N1(6)
4	20	10	101	11	0	0

MR-1628

Figure 23. Index Array after Execution

First, notice that the first element now contains a 4; even though 6 elements were available, only 4 were changed. Notice also that this lets you know that the 0s in the last two elements are NOT indices but rather are the same values that existed in N1(5) and N1(6) *before execution* of the GRAPH routine.

The new values in elements 1-4 are indices. Each index refers to some interesting point that the user of the program selected from the displayed graph. The purpose of indexing is to help determine the coordinates of such points.

Each index in N1 gives the *ordinal number* of a displayed point, minus 1. If the index of a point is 0, then the point was the first point displayed by the GRAPH or BARGRAPH statement. The coordinates of the first displayed point are, as you know, in the array elements named in the start X and start Y arguments.

The index values begin with 0 (instead of 1) for a simple reason: this convention lets you determine the location of a point's coordinates in the X and Y arrays simply by adding the index of the point to the element number of the start X or start Y argument.

For example, consider element N1(1) in Figure 23. After the GRAPH statement, this element contains 20. The start X and start Y arguments in the GRAPH statement were X1(5) and Y1(10), respectively. During the execution of the GRAPH statement the element N1(1) was assigned the value 20 in order to *index* the coordinates of an interesting point. The coordinates of the point of interest are in X1(5+20) and Y1(10+20), that is, X1(25) and Y1(30).

As you may have gathered from this discussion, indices are never negative, nor do they ever have values larger than the maximum dimension of the X and Y arrays.

### Using Index Arrays

Although we have covered the structure and purpose of index arrays, the most important question still remains: Exactly *how* do those indices get stored in the index array? To answer this question, let's return to our example session with the GRAPH routine.

To store indices of displayed points, you need to include two additional items in the GRAPH statement. One is the start index argument, which is the last argument of the GRAPH routine. The second item is the option word INDEX. Let's examine the effects of these two items with a sample GRAPH statement.

To be sure that you still have the necessary program in your workspace, type

```
RUN MDEMO
```

(If, for some reason, MDEMO has been erased from your diskette, type NEW MDEMO, and then retype the program as shown at the beginning of this chapter.)

To begin the indexing operation, type

```
F1(0)=2
```

### Indexing a Single Point

You have now assigned the value 2 to the first element of your index array. Therefore, your next GRAPH statement can store the indexes of up to two points. Now type the GRAPH statement itself:

```
READY  
DISPLAY_CLEAR
```

```
READY  
GRAPH("INDEX",X1(5),Y1(5),...,F1(0))
```



The result is shown in Figure 24.

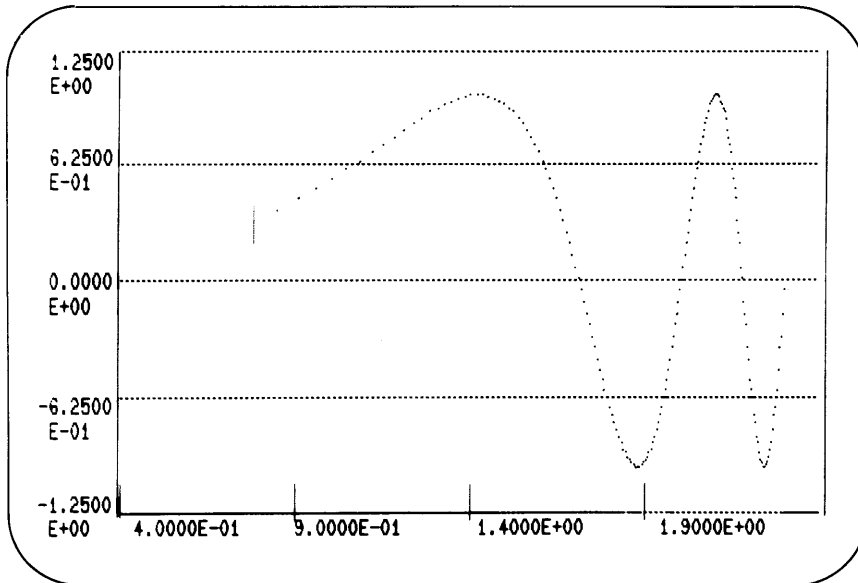


Figure 24. Beginning of Indexing Operation

As you see, Figure 24 shows a sine wave, distorted from its ordinary shape by having been plotted with a logarithmic X axis. Looking at the display screen, you will also notice a small flashing brand. Try holding down the left-arrow ( $\leftarrow$ ) and right-arrow ( $\rightarrow$ ) keys on your MINC keyboard. Notice that they make the flashing brand move to the left or right, following the contour of the sine wave. This flashing brand is used to mark the position of the point you want to index. Even though you have preset the element F1(0) to 2, you will index only one point in this operation.

With the arrow keys, move the flashing brand to the first (left-most) peak of the sine wave. When you have the flashing brand where you want it, press the S key on your keyboard. The flashing brand disappears, showing you that the index of the branded point is now stored in F1(1).

Now that you have indexed the point, press the **RET** key to end the indexing operation.

#### NOTE

For convenience, we have been using immediate mode in our examples. If you perform this type of indexing operation within a program, the program flow is suspended, and the GRAPH (or BARGRAPH) statement

remains in control until you press `(RET)`. When you press `(RET)`, the program flow resumes at the first statement following the GRAPH or BARGRAPH statement.

Examine the index count by typing

```
PRINT F1(0)
```

The result is 1. Previously, you assigned the value 2 to this element, to allow for two indices. The value 1 is a reminder that you actually stored only one index.

To examine the *coordinates* of the indexed point, type

```
READY  
PRINT "X COORDINATE: ";X1(5+F1(1))
```

```
READY  
PRINT "Y COORDINATE: ";Y1(5+F1(1))
```

The value 5 in the preceding statements is, of course, the element number of your start X and start Y arguments.

## Multiple Indexing

You have now used the indexing feature in a restricted application — to store the index of a single point. However, the maximum size for an index array is 11 elements, meaning that as many as 10 points can be indexed in a single operation. To make this exercise complete, we must now try a multiple-indexing operation.

Begin by typing

```
READY  
RUNNH
```

```
READY  
F1(0)=3
```

These two statements are all that you need to do in preparation. The only purpose of the RUNNH statement was to reinitialize the array F1 so that all of its elements would contain the value 0. Then the assignment statement preset the number of allowable indices to 3.

Now retype the GRAPH statement:

```
GRAPH("INDEX",,X1(5),Y1(5),,,,F1(0))
```

At this point, the display on your screen is identical to the one shown previously (Figure 24). You can now proceed with the multiple-indexing operation.

The numeric keys 1 through 0 above your keyboard represent the last 10 elements in the index array F1. (The 0 key represents the *last* element, F1(10).) To store or change an index, press the key that represents either the next available element, or one of the elements already used. The phrase “next available element” is used because you are not allowed to “skip” elements that have not been used. For instance, try pressing the 2 key. The buzzer sounds, meaning that you selected an improper element in the index array. (You skipped F1(1), which has not been used yet.)

The buzzer also sounds if you press a numeric key any time you have a flashing brand on the screen.

The available elements must be used in the correct order, because after the indexing operation is finished, you want to be sure that the index count not only gives the number of stored indices but also implies *which elements* of the index array were used.

By pressing each of the numeric keys, verify that none of the index array elements are in use. The flashing brand that is currently on the screen is for the index that will be stored in F1(1), the first available element.

With the arrow keys, move the new flashing brand to the first peak of the sine wave and then press the S key. The first index is now stored in F1(1).

Now, instead of pressing **(RET)**, press the 2 key. A new flashing brand appears, showing you that the next index element, F1(2), is now available. Move the flashing brand to the second peak of the sine wave and press the S key to store the second index.

To store the third index, press the 3 key, move the flashing brand all the way to the right end of the sine wave and press the S key. (Notice that the bell sounds if you attempt to go off the right edge of the graph; the same effect occurs at the left edge.) F1(3) now contains the index of the last point on the display.

DO NOT press **(RET)** yet, because there is one more feature to be demonstrated. First, try pressing the numeric keys from 4 to 0.

The buzzer sounds each time, because your preset value in F1(0) provides only three index elements for your use.

However, you can *change* any or all of the indices you have stored. We will change just one of them to demonstrate. Currently, F1(2) contains the index of the second peak of the sine wave. We will change it to the index of the first point on the screen. Begin by pressing the 2 key, which corresponds to F1(2). Notice that the buzzer does not sound, because this index element is still available. Now simply move the flashing brand to the left end of the sine wave, and press the S key. The element F1(2) has been changed; it now contains the index of the first point that was displayed. In general, you are allowed to repeat this change operation as often as you like; it is particularly useful for correcting minor errors, such as overshooting the point of interest and indexing some adjacent point instead.

Instead of changing any more indices, press **(RET)** to end the indexing operation. Then type

```
PRINT F1(2)
```

The index stored in F1(2) is 0, because you changed it to the index of the first point on the graph.

To examine the entire index array, type

```
FOR I=0 TO 10 \ PRINT "ELEMENT ",I," ":F1(I) \ NEXT I
```

The result is:

```
ELEMENT 0: 3  
ELEMENT 1: 20  
ELEMENT 2: 0  
ELEMENT 3: 194  
ELEMENT 4: 0  
ELEMENT 5: 0  
ELEMENT 6: 0  
ELEMENT 7: 0  
ELEMENT 8: 0  
ELEMENT 9: 0  
ELEMENT 10: 0
```

Element 0 contains a 3, because you used all three of the elements that you specified with the previous statement F1(0)=3. You also know that the indices were stored only in elements 1-3 and that the 0s in elements 4-10 are not indices but only the initial values of these elements. Element 1 contains the index 20,

corresponding to the first peak of the sine wave. The 0 in element 2 is an index, corresponding to the first point displayed. Element 3 contains the index of the last point displayed.



## CHAPTER 4 PICTORIAL EXAMPLES

This section summarizes some types of display that you can create with MINC graphic routines. Each picture is accompanied by a short program that creates the display. If you like, you can reproduce the pictures yourself by typing in the sample program.

Many of the examples use this program to create the data:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
```

The examples that use lines 10-50 to create data contain a single graphic statement in line 60.

If you are running several of these examples in a single session with MINC, you can enter and run this program once, to create the data in arrays X and Y. To display each picture, you can simply type the graphic statement as shown in line 60, in response to the READY message. If you leave the line number (60) out of this statement, the graphic routine is executed immediately (immediate mode). If you include the number 60 at the beginning, the routine statement is added to the program as line 60, and you must then type RUN or RUNNH to create the display (program mode). Program mode will take longer, because the X and Y values will be recalculated each time.

**Simple Point-Plot of Entire Array**

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH(...,Y(0))
```

Display shown in Figure 25.

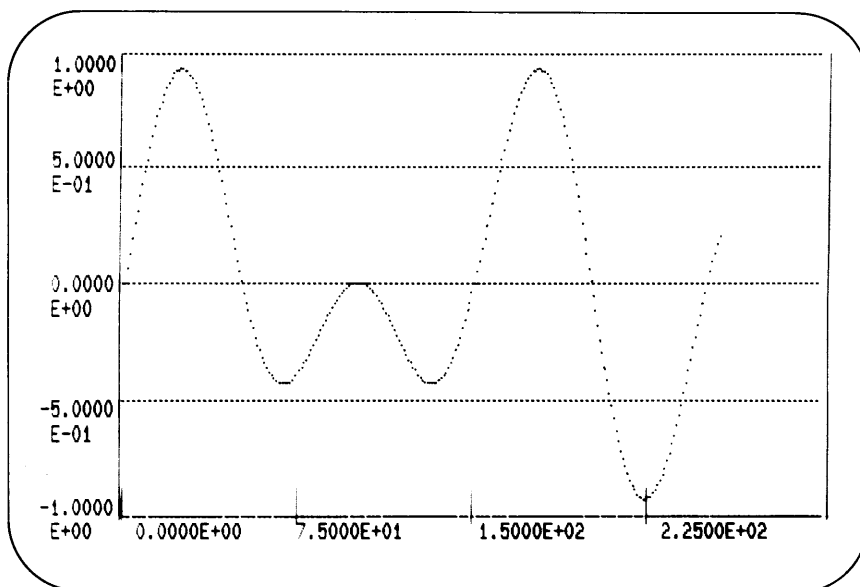


Figure 25. Simple Point-Plot of Entire Array



**Program:**

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH(.,X(0),Y(0))
    
```

**Simple Point-Plot of  
Entire Array, with X  
Coordinates**

Display shown in Figure 26.

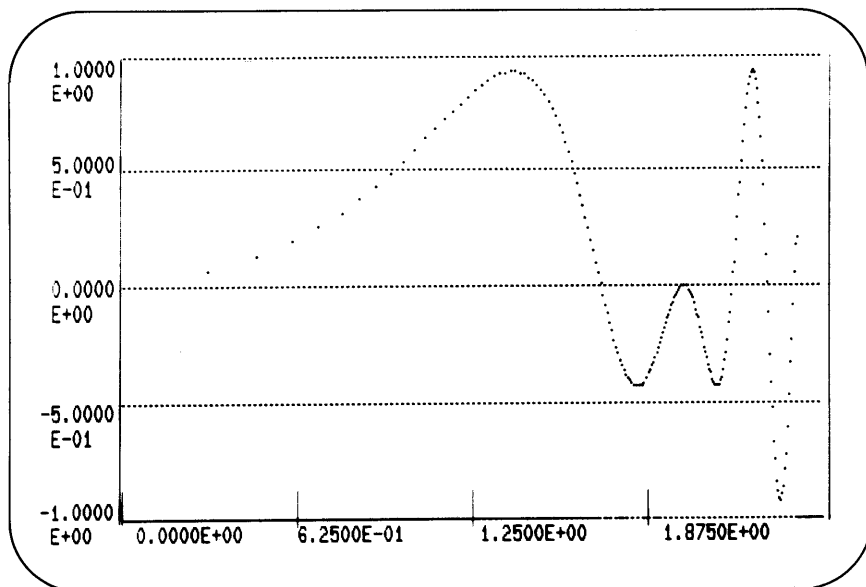


Figure 26. Simple Point-Plot of Entire Array, with X Coordinates

## GRAPHIC PROGRAMMING

### Simple Point-Plot of Array Subset

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH(.50,X(0),Y(0))
```

Display shown in Figure 27.

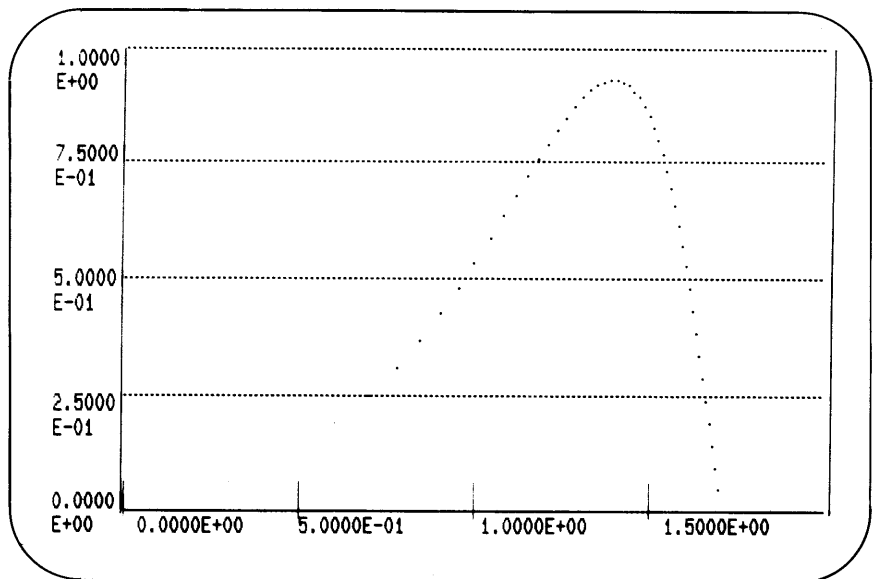


Figure 27. Simple Point-Plot of Array Subset (50 Points)

**Bargraph of Array Subset**

Program:

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 BARGRAPH(,10,X(0),Y(0))
    
```

Display shown in Figure 28.

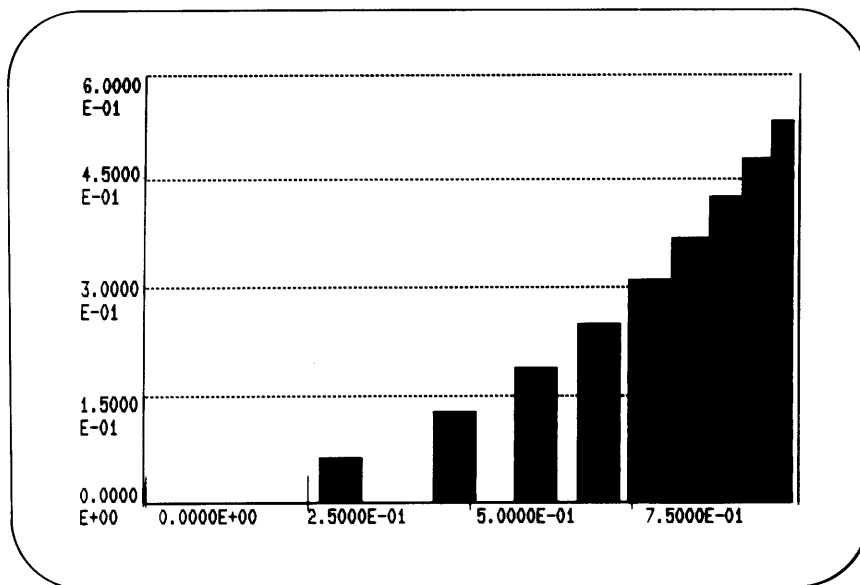


Figure 28. Bargraph of Array Subset (10 Points)

**Shaded Point-Plot of Entire Array**

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("SHADE",,X(0),Y(0))
```

Display shown in Figure 29.

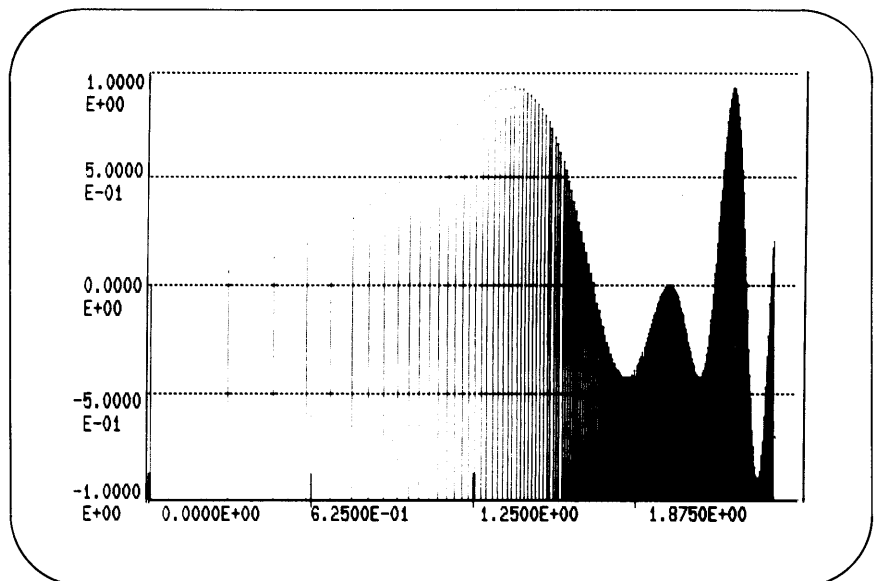


Figure 29. Shaded Point-Plot of Entire Array

**Shaded Point-Plot of  
Array Subset, with  
Brands**

Program:

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("SHADE,LINES,BRANDS",20,X(0),Y(0))
    
```

Display shown in Figure 30.

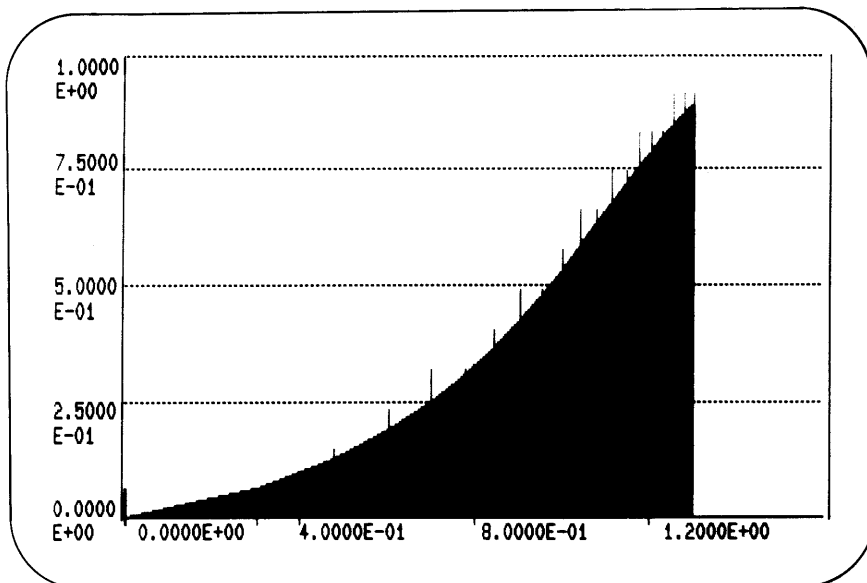


Figure 30. Shaded Point-Plot with Brands (20 Points)

## GRAPHIC PROGRAMMING

### Point-Plot with Exact Axis Units

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("EXACT",,X(0),Y(0))
```

Display shown in Figure 31.

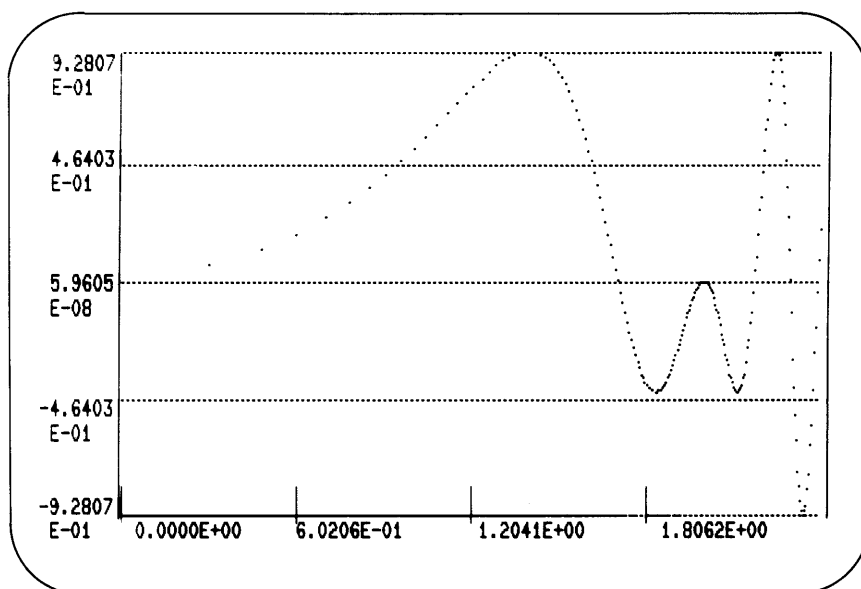


Figure 31. Point-Plot with Exact Axis Units

Program:

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH(,X(0),Y(0),2)
    
```

**Point-Plot of Every  
Second Value**

Display shown in Figure 32.

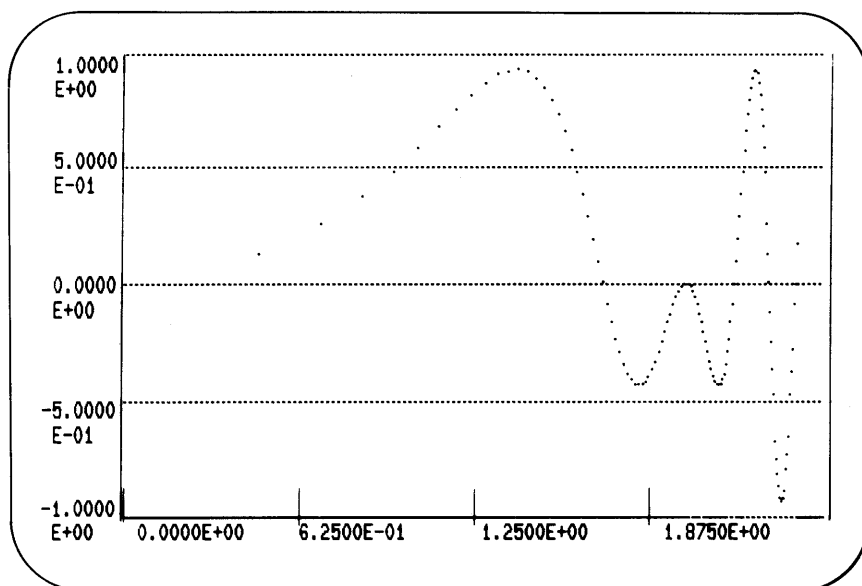


Figure 32. Point-Plot of Every Second Value

## GRAPHIC PROGRAMMING

### Interpolated Point-Plot of Every Twentieth Value

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("LINES,BRANDS",,X(0),Y(0),20)
```

Display shown in Figure 33.

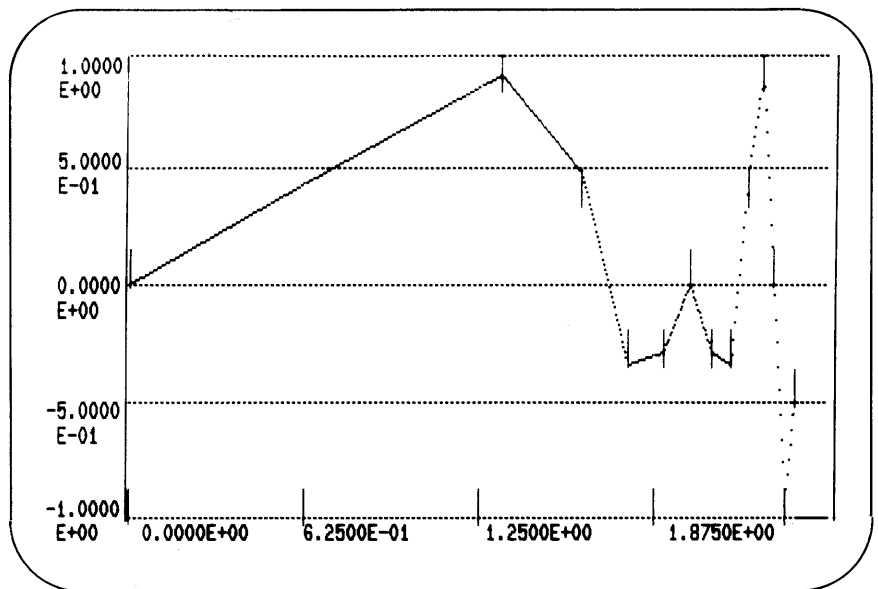


Figure 33. Interpolated Point-Plot of Every Twentieth Value



Program:

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("LINES,BRANDS,SHADE",,X(0),Y(0),20)
    
```

**Interpolated Point-Plot  
with Shading**

Display shown in Figure 34.

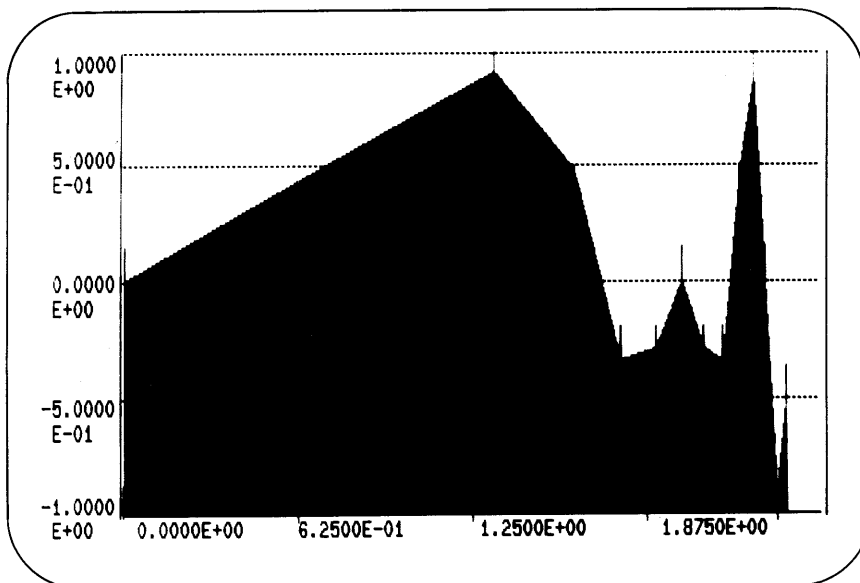


Figure 34. Interpolated Point-Plot with Shading

## GRAPHIC PROGRAMMING

### Interpolated Point-Plot, Shaded about Y = 0

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("LINES,BRANDS,SHADE",,X(0),Y(0),20,0)
```

Display shown in Figure 35.

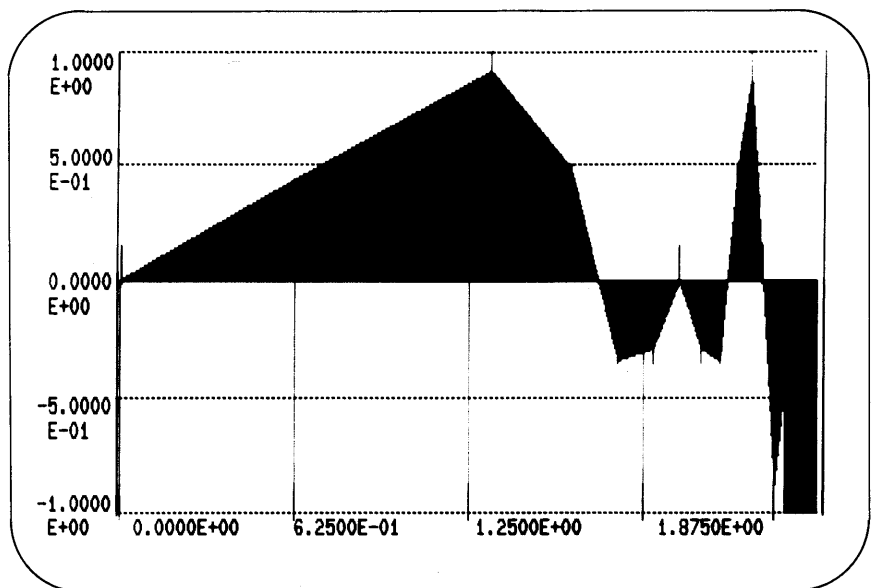


Figure 35. Interpolated Point-Plot, Shaded about Y = 0

Program:

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 BARGRAPH("SHADE",10,X(0),Y(0),20,0)
    
```

**Bargraph of Array  
Subset, Shaded about  
Y = 0**

Display shown in Figure 36.

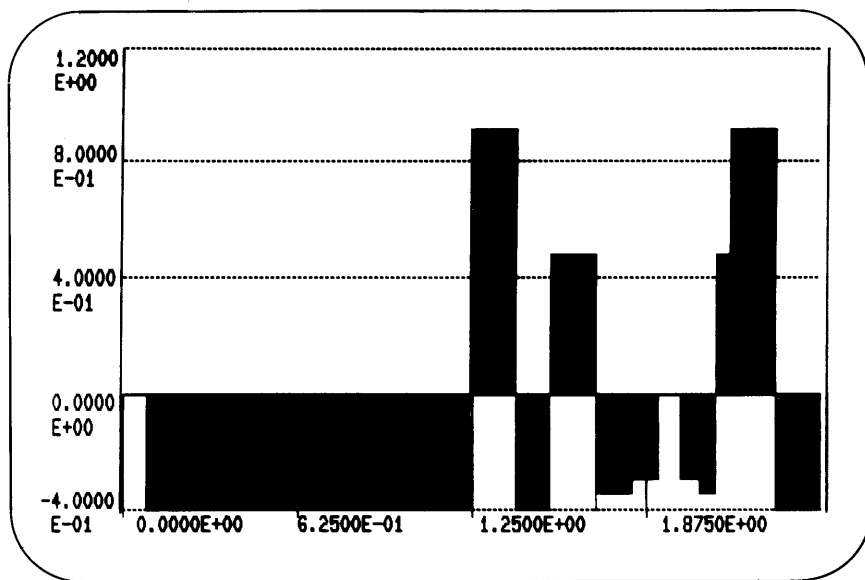


Figure 36. Bargraph of Array Subset, Shaded about Y = 0

GRAPHIC PROGRAMMING

**Bargraph of Array  
Subset, Shaded about  
Y = 1**

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 BARGRAPH("SHADE",10,X(0),Y(0),20,1)
```

Display shown in Figure 37.

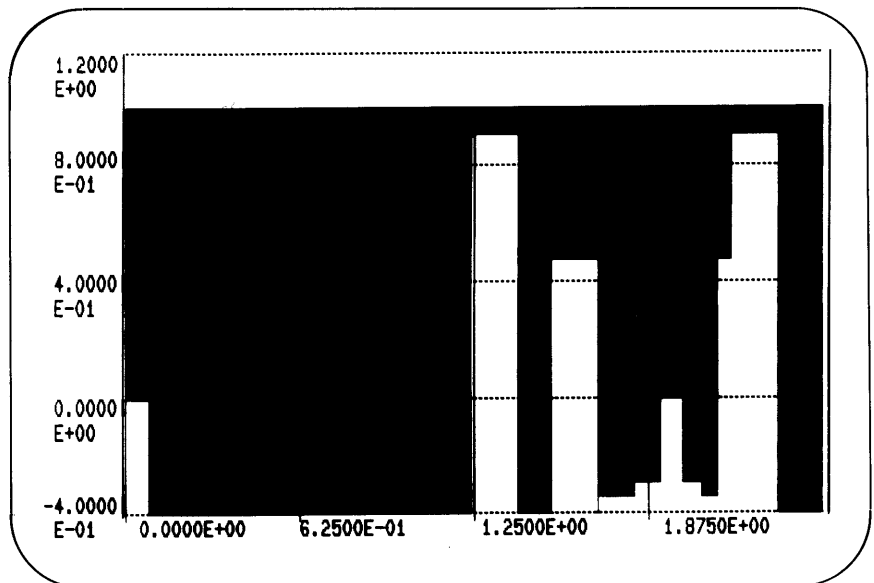


Figure 37. Bargraph of Array Subset, Shaded about Y = 1

Program:

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("VLINES",,X(0),Y(0))
    
```

**Simple Point-Plot of  
Entire Array, with  
Vertical Grid Lines**

Display shown in Figure 38.

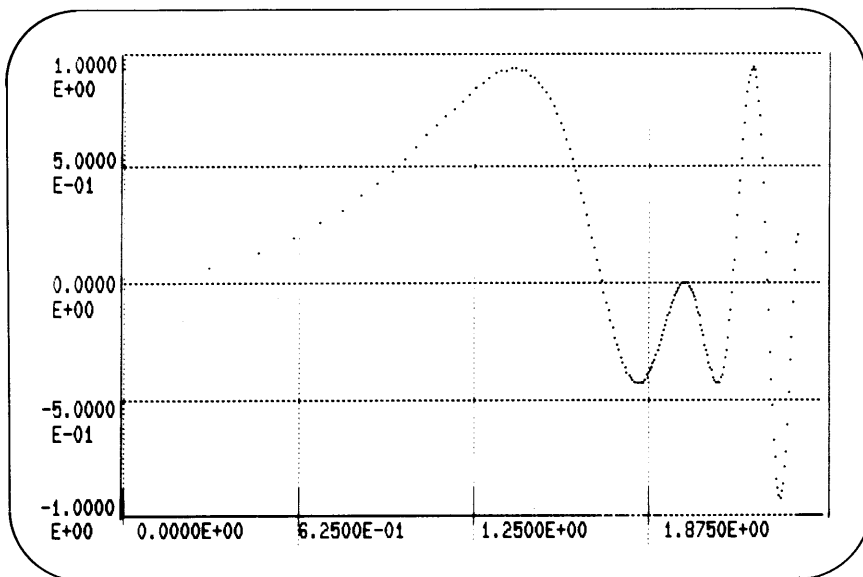


Figure 38. Simple Point-Plot of Entire Array, with Vertical Grid Lines

**Simple Point-Plot of Entire Array, without Horizontal Grid Lines**

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("-HLINES",X(0),Y(0))
```

Display shown in Figure 39.

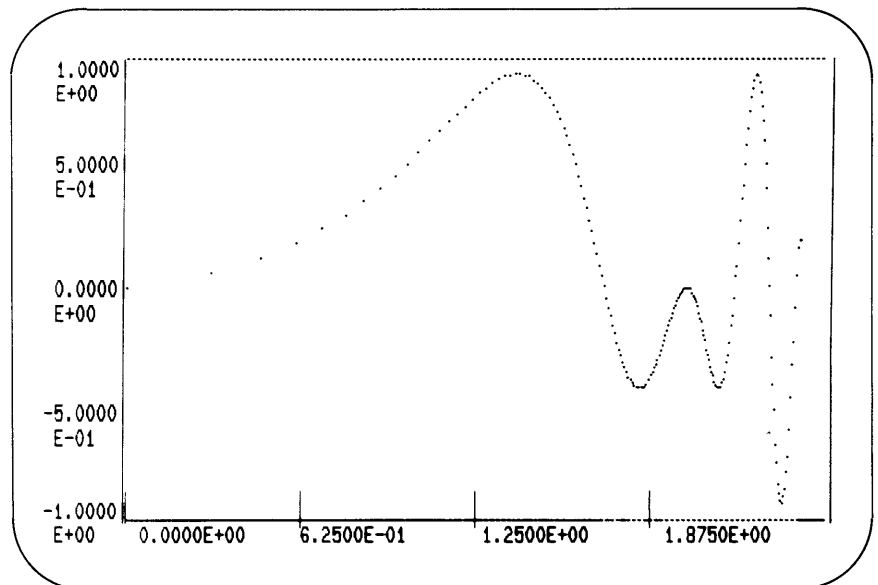


Figure 39. Simple Point-Plot of Entire Array, without Horizontal Grid Lines

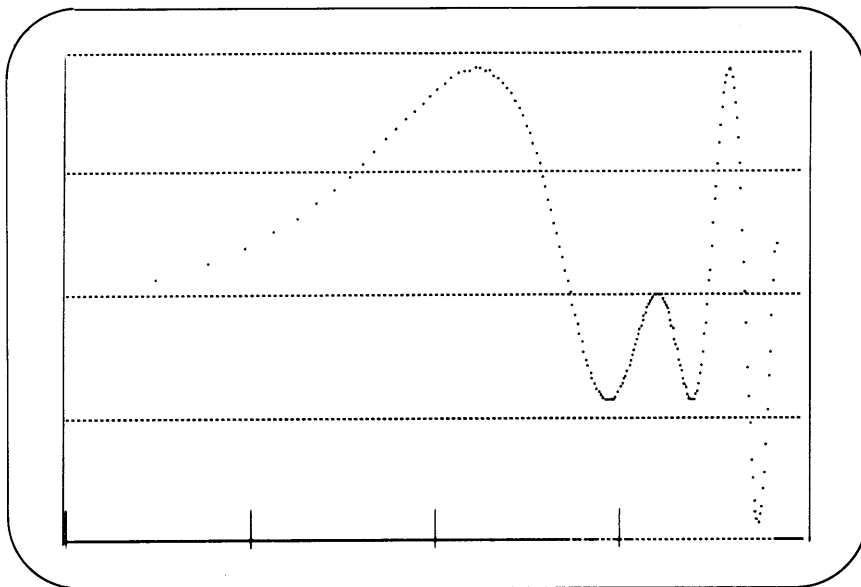
**Program:**

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH("-UNITS",X(0),Y(0))
    
```

**Simple Point-Plot of  
Entire Array, without  
Axis Units**

Display shown in Figure 40.



**Figure 40. Simple Point-Plot of Entire Array, without Axis  
Units**

## GRAPHIC PROGRAMMING

### Simple Point-Plot of Entire Array, Labeled with LABEL Routine

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH(,X(0),Y(0))
70 LABEL("BOLD","Base 10 Logarithms","Modulated Sine Wave")
```

Display shown in Figure 41.

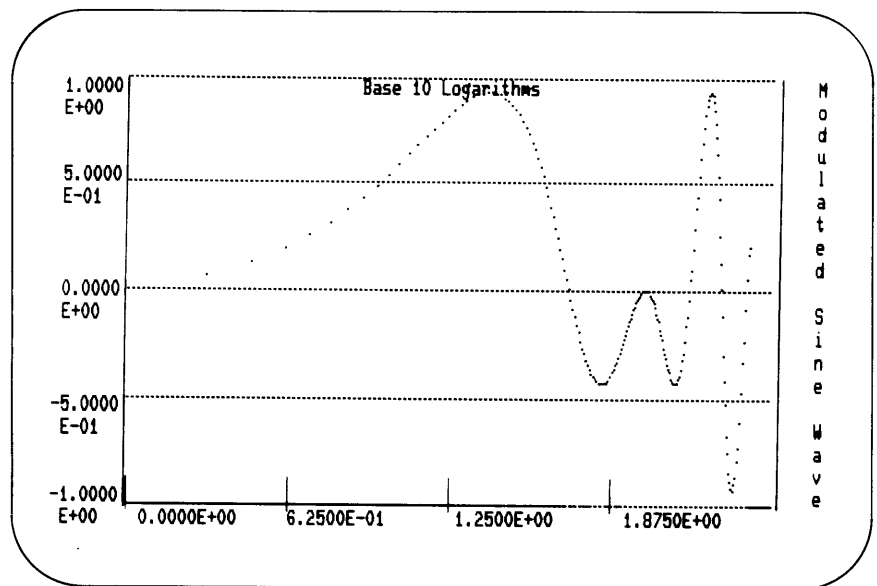


Figure 41. Labeled Point-Plot of Entire Array



Program:

```

10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH(,X(0),Y(0))
70 REMARK X UNITS (FULL REGION) BEGIN AT ROW 20, COLUMN 9
80 ERASE_TEXT("-TEXT,ROW",20,9,72)
90 HTEXT("BOLD",20,10,"Scroll-Proof Position for X Labels")

```

**Replacement of X  
Units with Text Label**

Display shown in Figure 42.

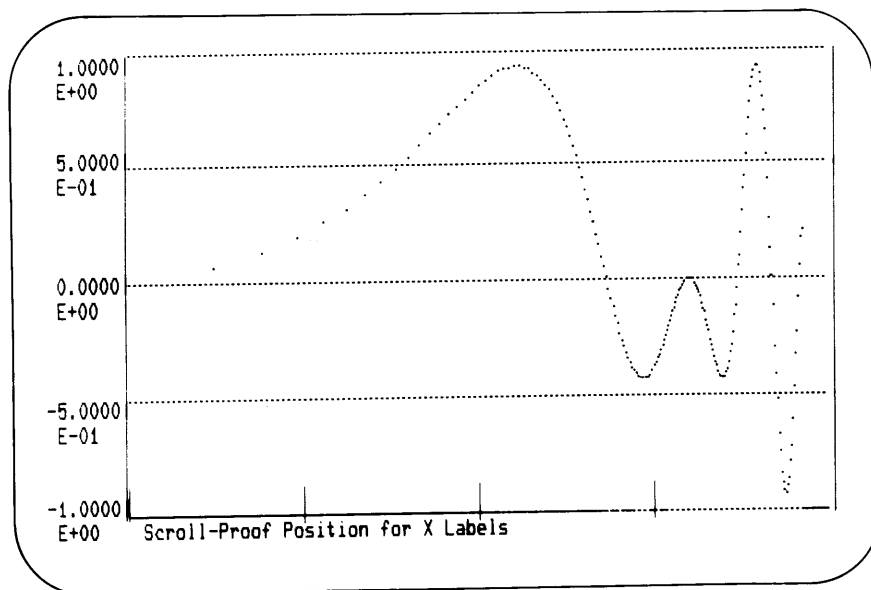


Figure 42. Replacement of X Units with Text Label

**Simple Point-Plot of Entire Array, with Marked Value**

Program:

```
10 DIM X(255),Y(255)
20 FOR I=0 TO 255
30 X(I) = LOG10(I+1)
40 Y(I) = SIN(I*PI/50)*COS(I*PI/200)
50 NEXT I
60 GRAPH(,X(0),Y(0))
70 REMARK 35th point will be marked with a reverse cross
75 X1=X(34) \ Y1=Y(34)
80 MAP_TO_TEXT(X1,Y1,R,C)
90 PUT_SYMBOL("BOLD,REVERSE,CROSS",R,C)
100 REMARK Print values from array
110 MOVE_CURSOR(R,C+2)
120 PRINT "X = "; X1; ", Y = "; Y1
130 REMARK Return to scrolling area
140 MOVE_CURSOR
```

Display shown in Figure 43.

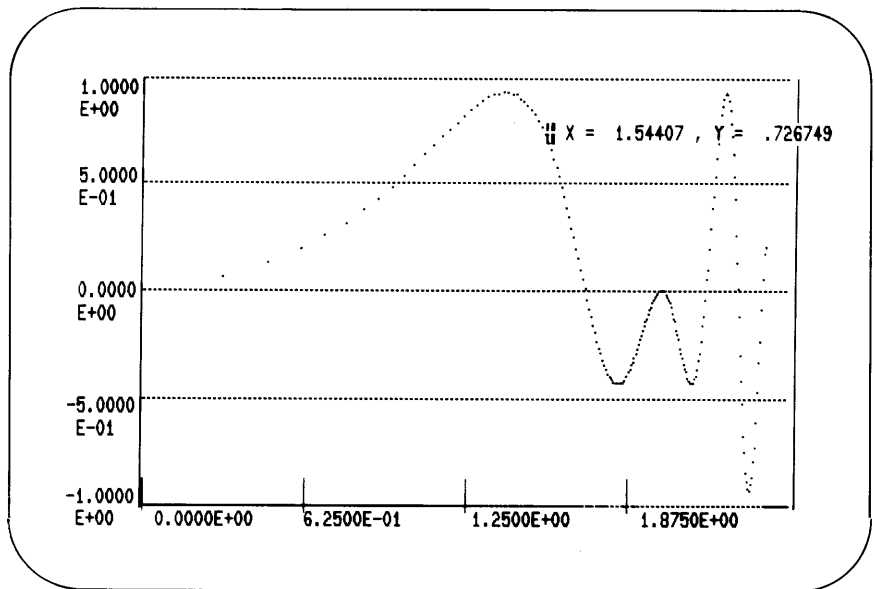


Figure 43. Simple Point-Plot of Entire Array with Marked Point

## CHAPTER 5 ARRAYS AND DATA STRUCTURES

With the MINC system, the actual display of data is done by the MINC graphic routines. The only jobs that you must do are to decide which data to display in a particular operation, and then to put your data in the proper type of structure.

The data structure used by the majority of MINC graphic routines is the array of the BASIC programming language. Except in one special case (the index array) you use arrays to store the X and Y coordinates of points or other figures that you want to display on the screen of your terminal. The graphic routines display the picture you want when you supply them with the array elements that contain the coordinates of, for example, a point.

### ARRAYS

In this section, we will consider the use of one- and two-dimensional arrays as *data structures* for the MINC graphic routines. Because it is a special case the index array is discussed as part of the introduction to the indexing feature of the GRAPH and BARGRAPH routines (see “Introduction to the GRAPH Routine”).

The graphic routines are more difficult to use with two-dimensional arrays than with one-dimensional arrays. Although the practice is allowable and predictable, you should, for simplicity's sake, avoid the use of two-dimensional arrays unless your application *requires* that you use them.

**One-Dimensional  
Arrays**

The MINC graphic routines are defined elsewhere in this manual. At this point, consider the following statements as examples of how to refer to arrays in a routine statement.

Example: POINT(X1(0),Y1(0))

Result:

This statement displays a single point on the screen. The POINT routine finds the X and Y coordinates of the point in element 0 of the arrays X1 and Y1, respectively.

One way to display 200 points would be to repeat the POINT statement 200 times. On each repetition of the statement, you would, in effect, specify different elements of the arrays X1 and Y1, in order to give each point a unique X and Y coordinate. In the following example, assume that X1 and Y1 are 200-element arrays created with the DIM statement, and that each element of X1 and Y1 has been assigned a value.

Example:

```
FOR M=0 TO 199 \ POINT(X1(M),Y1(M)) \ NEXT M
```

Result:

Since POINT is repeated 200 times in the FOR-NEXT loop, 200 points are displayed on the screen. The index of the FOR-NEXT loop is also the number of each array element used in each POINT statement. Thus, all 200 pairs of your X and Y coordinates are extracted from X1 and Y1 and used to display the points.

As you will see, using a statement such as POINT in this manner is also more complicated to write down than the alternative. The alternative is to use a more comprehensive routine, which can handle your arrays more conveniently.

Example:

```
GRAPH(,X1(0),Y1(0))
```

Result:

This GRAPH statement has the same effect as the FOR/NEXT loop shown previously. In this case, however, X1(0) and Y1(0) are simply the first pair of coordinates to be used — in this case, they are also the first elements in their arrays. They are called

the *start X* and *start Y arguments*, respectively. Once it knows the start X and start Y arguments, this form of the GRAPH statement plots all the remaining points by taking successive elements of the arrays X1 and Y1. GRAPH does not stop until it reaches the end of one of the arrays.

Therefore, with this GRAPH statement, you do not need to know how long the arrays are or how many points to plot. You are required only to know where in the arrays GRAPH should start and to arrange the arrays so that the X and Y coordinates match up properly. Note that the start X and start Y arguments do not need to have the same array element number. The two arguments could, for instance, be X1(100) and Y1(1). Neither is it necessary for the two arrays to have the same length. The GRAPH routine stops when it reaches the end of the shorter array.

**Two-Dimensional  
Arrays**

In MINC BASIC, you can create two-dimensional arrays with such statements as DIM T1(5,4)

This statement creates a 30-element array. You can put values in this array by giving an element number for each dimension, in statements such as

T1(1,2)=2

A two-dimensional array resembles a table, as shown in Figure 44.

	0	1	2	3	4
0	T1(0,0)				T1(0,4)
1					
2					
3					
4					
5	T1(5,0)				T1(5,4)

MR-1624

Figure 44. A Representation of the Array T1(5,4)

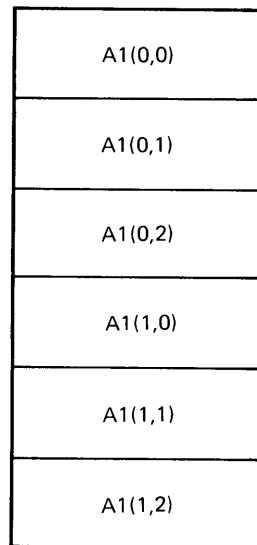
The table is an appropriate and simple structure for visualizing the individual elements in a two-dimensional array. However, if you use two-dimensional arrays with the MINC graphic routines, you must also remember the order in which successive elements are allocated in the MINC workspace. Within the workspace, array elements are not physically stored as shown in Figure 44. Instead, the elements are stored linearly, according to a fixed pattern that allows MINC to determine whether a given element belongs to one or the other dimension.

As the array elements are allocated, the second dimension varies most rapidly.

Thus, the statement

```
DIM A1(1,2)
```

allocates memory for the A1 elements in the order shown in Figure 45.



MR-1625

Figure 45. Allocation of the Array A1(1,2)

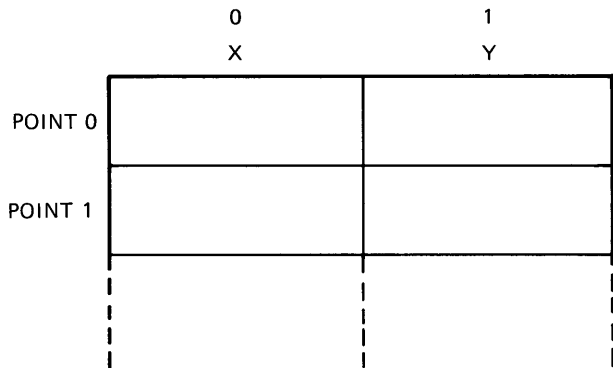
Notice that in Figure 45, all the “row 0” elements of A1 are allocated together, and then the elements of row 1 are allocated. Thus, the second dimension (the “columns”) varies most rapidly. Even though it is often convenient to think of A1 as a table with 2 rows and 3 columns, you must remember that *in the workspace*, it looks like Figure 45.

The MINC graphic routines display data from a two-dimensional array by varying the second dimension most rapidly. That is, the statement

```
GRAPH(,,A1(0,0))
```

displays all the information in the array A1, treating the values in both dimensions as Y coordinates. The elements of A1 are displayed in the order shown in Figure 45. As you can see, the GRAPH statement is fairly simple when you use a two-dimensional array for Y coordinates only. Of course, there would be no advantage over one-dimensional arrays in this case.

The familiar visualization of a two-dimensional array as a table suggests that such an array could be used to store X and Y coordinates. You could use one dimension to identify particular points; this dimension would equal the number of points you wish to display. The second dimension would then be used to store the information related to each point; if the information consists of an X coordinate and a Y coordinate, the second dimension has two elements for each element of the first dimension. The "table" for a set of points then resembles Figure 46.



MR-1626

Figure 46. Points Stored in a Two-Dimensional Array

The following example program, called TWOWAY, shows a two-dimensional array being used to store X and Y coordinates for 200 points. As the program shows, the GRAPH statement becomes more complicated in this case, although it does produce the correct result (Figure 47).

```
READY
NEW TWOWAY
```

## GRAPHIC PROGRAMMING

```
READY
5 REMARK Create a two-dimensional array E1.
10 DIM E1(199,1)
15 REMARK Fill the array with X and Y coordinates.
20 FOR I=0 TO 199
25 REMARK X values are stored in the 0 elements.
30 E1(I,0)=LOG10(I+1)
35 REMARK Y values are stored in the 1 elements.
40 E1(I,1)=SIN(I*PI/50)
50 NEXT I
51 REMARK Display the points in array E1.
52 REMARK The increment argument of 2 assures
53 REMARK the correct interpretation of X and Y values.
60 GRAPH("EXACT,LINES,SHADE",,E1(0,0),E1(0,1),2,0)
SAVE

READY
RUNNH
```

The resulting display is shown in Figure 47.

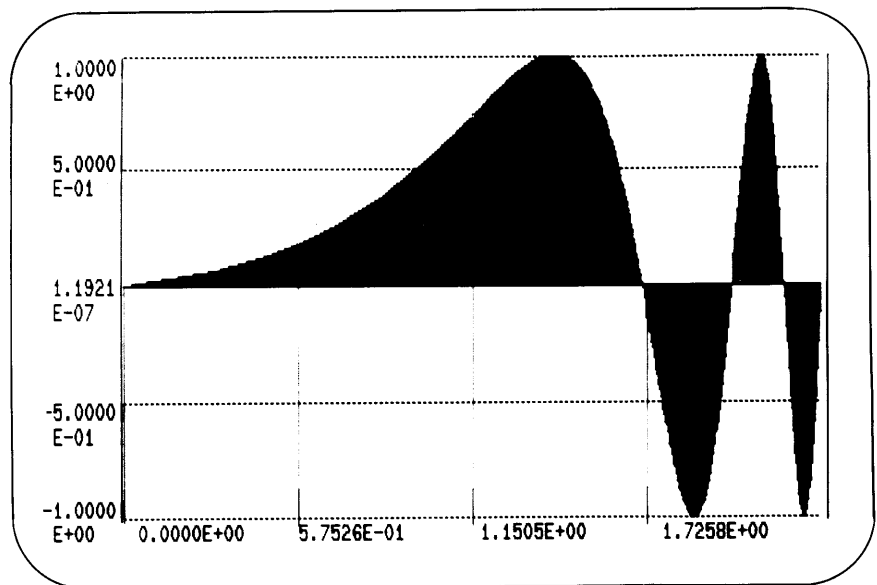


Figure 47. Display of a Two-Dimensional Array

The program `TWOWAY` creates a table of 200 points (line 10). The X coordinates are the logarithms of the integers from 1 to 200 (line 30). The Y coordinates are the sines of the angles between (approximately) 0 and  $4\pi$  (line 40).

The `GRAPH` statement in line 60 plots all the points in array `E1`. The element `E1(0,0)` is used as the start X argument, because it



contains the X coordinate of the first point. The element  $E1(0,1)$  is used as the start Y argument, because it contains the Y coordinate of the first point. The next-to-last argument in the GRAPH statement is the *increment* argument, for which a value of 2 is used. Recall that the E1 elements are stored in the workspace in the following order:

$E1(0,0)$   $E1(0,1)$   $E1(1,0)$   $E1(1,1)$   $E1(2,0)$  . . .

The value 2 for the increment argument tells GRAPH to skip an array element between each successive point. Therefore,  $E1(0,0)$  is the first X coordinate, and  $E1(1,0)$  is the second X coordinate.  $E1(0,1)$  is the first Y coordinate, and  $E1(1,1)$  is the second Y coordinate.

The last argument in the GRAPH statement (0) is the shade line argument. This argument shades the graph about the line  $Y = 0$ .

You will soon be using the MINC system to display sets of data that you have created by computation (in a program) or that you have acquired by reading values from laboratory instruments (as described in Books 5 and 6). As you attempt to display larger and larger data sets, you will eventually encounter the limitation imposed by using arrays. Because arrays are created and stored in the MINC workspace, they come with a built-in size limitation. You can determine the size of the available workspace at any time by typing the red part of this dialog:

```
READY
LENGTH
2500
```

```
READY
```

The number with which MINC responds, 2500 in this example, is the current length of the workspace in “words” of memory. MINC uses one word of memory to store the value of an integer and two words to store real numbers. This use of memory means that with a workspace of 2500 words, you can create an array of 2500 integers or 1250 real numbers.

In the laboratory, it is likely that you will want to process much larger sets of data than are allowed by these limitations. For such purposes, MINC provides an additional data structure known as the *virtual array file*.

## VIRTUAL ARRAY FILES

Virtual array files are created by a special form of the DIM statement:

```
DIM <file number>,<name> ( <dimension> )
```

As shown in the above example, the DIM statement also makes reference to a *file number*. As discussed in Book 2, a file number appears in an OPEN statement, and essentially tells MINC where data can be recorded or retrieved.

In the case of virtual array files, the OPEN statement and the DIM statement create a data structure *on a MINC diskette*, rather than in the MINC workspace. Conceivably, this data structure can occupy an entire diskette, meaning that the virtual array file can contain the equivalent of 240,000 words of memory. Translated into numbers of stored values, this means an array-like structure containing 240,000 integers or 120,000 real numbers.

### Displaying Virtual Array Files

You can easily display virtual array files with the POINT routine, using its MOVE option. The MOVE option causes the graph to move to the left on the screen as more than 512 points are displayed. This action is called *strip-chart mode*, because it resembles the action of a strip-chart recorder.

Strip-chart mode is explained in detail in Chapter 3. If you want to demonstrate strip-chart mode with virtual array files, type the following example. If you run this example, you should first put an empty (initialized) diskette in drive. 1. If the diskette has never been used, initialize it with the command

```
INITIALIZE SY1:
```

The program VIRTUE takes several minutes to run.

```
READY
NEW VIRTUE

READY
5 REMARK Create virtual array file
10 OPEN "SY1:VIRTUE" AS FILE #2
20 DIM #2,D(4000)
25 REMARK Clear screen
30 DISPLAY_CLEAR
35 REMARK Display advisory message
40 HTEXT("BOLD,FLASH",10,35,"FILLING VIRTUAL ARRAY FILE")
45 REMARK Put data in virtual array file
```

```

50 FOR I=0 TO 4000
60 D(I)=SIN(I*PI/200)*COS(I*PI/125)
70 NEXT I
75 REMARK Set window
80 WINDOW("EXACT",0,-1,500,1)
85 REMARK Display the grid
90 GRID
95 REMARK Display the points from the file
100 FOR I=0 TO 4000
110 POINT("MOVE,UNITS",I,D(I))
120 NEXT I
125 REMARK Label the finished graph
130 LABEL("BOLD","VIRTUAL ARRAY FILE",CLK$)
135 REMARK Close the virtual array file
140 CLOSE #2
150 END
SAVE

READY
RUNNH

```

The program VIRTUE performs the following tasks:

1. Opens a file called VIRTUE.DAT on SY1:, which is assumed to be an empty diskette created by the INITIALIZE command (see Books 2 and 3 for a description of INITIALIZE).
2. Creates a virtual array file called D that will be stored on SY1: with the name VIRTUE.DAT.
3. Computes 4000 real numbers and stores each number in a separate element of D. During the computation, the message FILLING VIRTUAL ARRAY FILE flashes on the screen.
4. When the computation is finished, the program sets the window (line 80) and requests a grid for the graph (line 90). (The routines WINDOW and GRID are described in Part 2 of this manual.)
5. Lines 100-120 display all the points from the virtual array file.
6. Line 130 labels the final display when all the points have been plotted. The title VIRTUAL ARRAY FILE is centered at the top of the graph region. The time of day (requested by the string function CLK\$) appears vertically at the right edge of the graph region.

- Line 140 closes the virtual array file on the diskette. The data are now stored permanently, although you can change their values by reopening the file.

The final display created by the program VIRTUE is shown in Figure 48.

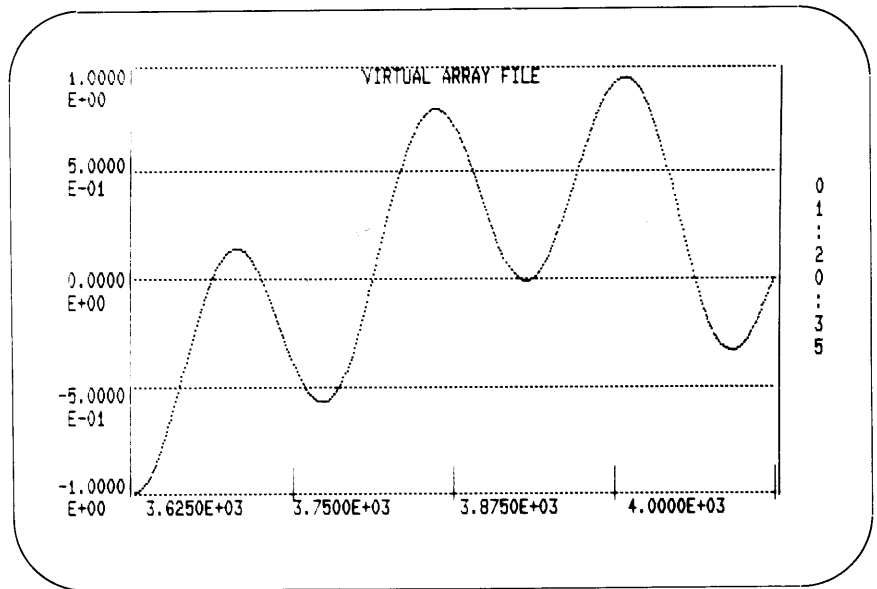


Figure 48. Display of a Virtual Array File

# **PART 2**

# **ROUTINES**



## INTRODUCTION TO PART 2

Part 2, which comprises the remainder of this manual, describes each of the graphic routines available in the MINC system. Each routine has a section of its own, and the sections are ordered alphabetically by routine name.

Some of the routines are related so closely that they are described together in single sections. GRAPH and BARGRAPH are described together; so are HLINE and VLINE; HTEXT and VTEXT; and GET\_CURSOR and MOVE\_CURSOR.

Each section in Part 2 contains subsections that cover the features of the routine in detail. These sections, and their contents, are as follows:

1. "Operation" describes the general purpose of the routine.
2. "Examples" gives short examples of the routine in use. Each example is followed by a brief explanation of the effects of the routine statement.
3. "Statement Form" gives the correct form, or syntax, for the routine. In the statement form, words and phrases are used to represent the routine's arguments.

Arguments that are required (and have no default setting) are shown in black ink. Optional arguments are shown in blue ink.

Note that many routines contain an option argument. In actual use, the value of this argument is an alphabetic string. One way to represent a string in a routine statement is by inserting a word or list of words that is enclosed by quotation marks. Another way to represent a string in a routine statement is by inserting the name of a *string variable* (example: S\$) to which you have previously assigned the appropriate string value (example: LET S\$="EXACT,SHADE,LINES,BRANDS"). String variables are useful if you want to make several routine statements that use the same option words. Once you have assigned the proper values to the string variable, you can simply name the variable in each routine statement and can avoid retyping the full list of words each time.

Notice that in the example LET statement, the list of option words is surrounded by quotation marks. When you use a string variable such as S\$, you do not need to put quotation marks around the variable name. For this reason, the "Statement Form" sections do not show quotation marks around the option word. You must remember to put them in, however, when you put the explicit option string in the routine statement.

If you need more information on the use of strings and string variables, see *Book 2: MINC Programming Fundamentals* and *Book 3: MINC Programming Reference*.

4. "Argument Descriptions" contains definitions and explanations of each argument that can appear in the routine statement. The arguments are described in the order in which they appear in the statement form. In some sections, such as the section on GRAPH and BARGRAPH, the argument descriptions are rather long; that is because the description of arguments, for these and other routines, contains most of the "fine detail" about the routines' operations. You should read the argument descriptions carefully to get a full understanding of the way the routines work.
5. "Related Routines" compares and contrasts the operation of the routine under discussion with other routines that have a similar function. This is another important section, because such routines as VIEW and ERASE\_GRAPH have a similar visual effect even though the similarity is really only superficial. This section is a good place to look when a routine



does not produce the result you expected, or when the previous sections on the routine suggest that the routine does not perform the operation you need.

6. “Restrictions” lists the specific things that the routine *does not do*. To some extent, this section anticipates the questions you have about the effects a routine may have, for instance, “Can I use GRAPH\_INIT to erase text from the screen?” This section also describes some operations that may appear to be valid but are in fact illogical; one example of this type is the inclusion of two contradictory option words in the same routine statement. In most cases, logical errors cause the routine statement to fail and produce an error message that tells where you made the mistake. There are some cases in which logic errors do not stop the routine and thus produce no error message. Thus, the “Restrictions” section is a good source of information when you are sure you are using the correct routine (as verified by “Related Routines”), you received no error message, and you still did not produce the display you wanted.
  
7. “Error Conditions” describes the common errors that occur with a particular routine. This section gives the text of error messages produced by the routine, which are accompanied by a short description of the condition that created the error. Unless a solution is obvious from the error message itself, the section also recommends a solution.

All the errors described are known as *fatal errors*. This phrase means that the error actually stopped the routine. Error messages are preceded by the letters “?MINC-F-”, in which the letter F designates a fatal error. Fatal errors can generally be corrected by repeating the routine statement in the proper form. Fatal errors do not change, or “corrupt,” the data that you are trying to display.

Some errors are not fatal and thus produce no error message. You can usually clarify these conditions by referring to the “Restrictions” section.

Many system messages and error messages besides those related to graphic programming are produced by MINC. For a list of system messages, see *Book 8: MINC System Index*.

The MINC system has 33 graphic routines that allow you to create and change complex pictures with single routine statements. The following list briefly describes what the graphic routines do.

1. `BARGRAPH` has the same features as `GRAPH`, but it displays data in bargraph form instead of point-plot form.
2. `BOX` draws a box on the screen.
3. `CHAR_MODE` controls the appearance of groups of displayed characters.
4. `DISPLAY_CLEAR` performs the actions of `ERASE_TEXT` and `GRAPH_INIT` in a single statement, erasing all text from the screen and clearing the graphic memory.
5. `DISPLAY_MODE` controls the width, background color, and scrolling mode of the screen.
6. `DUAL_MOVE` displays two graphs in strip-chart mode.
7. `ERASE_GRAPH` erases part or all of the graphic material on the screen and also erases the data from the graphic memory.
8. `ERASE_TEXT` erases part or all of the textual material from the screen but does not return the terminal to the state defined in setup mode.
9. `FIND_POINT` determines the X-Y coordinates of an interesting point on a graph.
10. `GET_CURSOR` and `MOVE_CURSOR` determine the current position of the cursor and move the cursor to a new position, respectively.
11. `GRAPH` displays data as a set of points on an X-Y plot (that is, a "point-plot"). `GRAPH` automatically scales the data, puts a grid on the screen, and puts numerical units on the X and Y axes. With the same routine, you can index the coordinates of interesting points, shade parts of the graph, and create a moving graph that simulates the output of a strip-chart recorder.
12. `GRAPH_INIT` erases the graphic memory, clearing all

graphics from the screen and exiting from strip-chart mode.

13. GRID plots a grid of horizontal lines and tick marks on the screen.
14. HLINE and VLINE plot horizontal and vertical lines on the screen, respectively.
15. HTEXT and VTEXT display horizontal and vertical text strings, respectively.
16. LABEL puts labels on the X and Y axes of either a graph or a bar chart.
17. LIGHTS controls four of the light-emitting diodes on the terminal keyboard.
18. MAP\_TO\_GRAPH translates a screen position from row-column coordinates to X-Y coordinates.
19. MAP\_TO\_TEXT translates a screen position from X-Y coordinates to row-column coordinates.
20. POINT plots a single point on the screen.
21. PUT\_SYMBOL displays one of a set of special symbols at a particular row-column position.
22. REGION sets the placement of the graphic displays on the screen.
23. ROLL\_AREA sets the placement of the scrolling area on the screen.
24. SET\_BAR can be used together with BARGRAPH to control the exact widths of individual bars in a BARGRAPH chart. If you use BARGRAPH without SET\_BAR, the widths of the bars are set automatically.
25. SHADE shades sections of a graph.
26. TEXT\_INIT erases all text and returns the terminal to the state defined in setup mode.
27. TEXT\_LINE draws lines on the screen from one row/column position to another.

## *GRAPHIC PROGRAMMING*

28. `VIEW` makes graphed data visible or invisible.
29. `WIDE_LINE` controls the width of characters on a particular row.
30. `WINDOW` sets the numerical limits of the X and Y coordinates for a graphic display.

## **BARGRAPH**

See "GRAPH and BARGRAPH."

# BOX

**Operation**

BOX draws rectangular boxes on the MINC screen with special text characters.

**Examples**

ROLL\_AREA(21,24)

BOX(1,1,20,70)

draws a box with corners at row 1, column 1, and row 20, column 70 (the outer box shown in Figure 49).

BOX("BOLD,REVERSE",5,5,16,60)

draws a box with corners at row 5, column 5, and row 16, column 60 (the inner box in Figure 49). This inner box is drawn with reverse boldface characters.

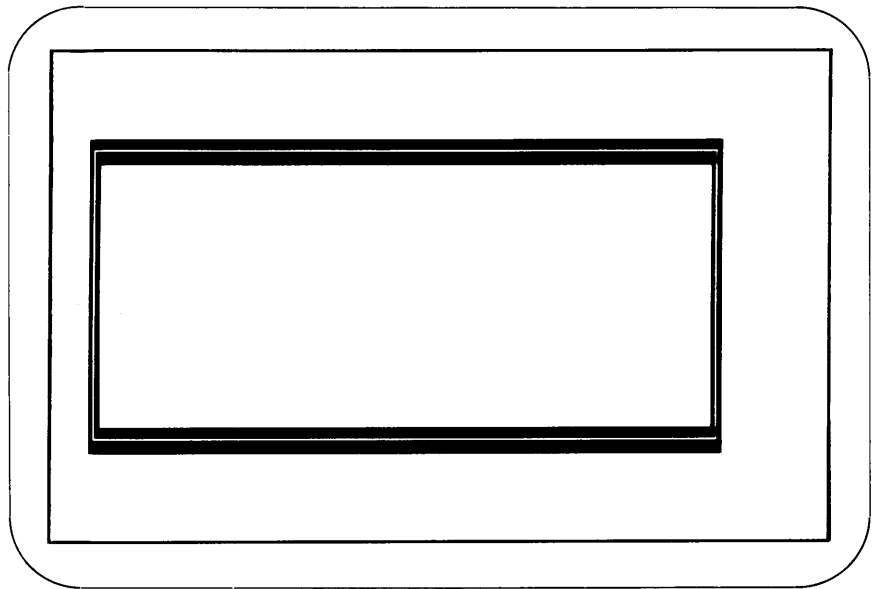


Figure 49. Result of Two BOX Statements

**Statement Form**

BOX(option,row 1,column 1,row 2,column 2)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	B,U,F,R,[-]I	-I
row 1,row 2	numeric expression	Integer in range 1-24	Row containing cursor
column 1, column 2	numeric expression	Integer in range 1-132	Column containing cursor

Argument  
Descriptions

**no arguments** Although there are default values for every argument, you cannot use BOX without arguments (see “Restrictions” below).

**option** Your choice of the options INVISIBLE, FLASHING, BOLD, REVERSE, and UNDERLINE. INVISIBLE displays spaces instead of line characters, erasing any characters previously drawn in the path of the box. Because INVISIBLE displays spaces, the combination of INVISIBLE and REVERSE *does* produce a visible result — a box of reverse video spaces, without the line-drawing characters normally used by BOX. The other four options describe the character mode that applies to the box. (For a description of character modes, see CHAR\_MODE.)

If you omit the option word, “-INVISIBLE” is used by default.

**row 1** row number of one corner of the box; must be in the range 1-24. If you omit row 1, the row in which the cursor is currently located is used by default.

**column 1** column number of one corner of the box; must be in the range 1-80 or 1-132. If you omit column 1, the column in which the cursor is currently located is used by default.

**row 2** row number of the opposite corner of the box; must be in the range 1-24. If you omit row 2, the row in which the cursor is currently located is used by default.

**column 2** column number of the opposite corner of the box; must be in the range 1-80 or 1-132. If you omit column 2, the column in which the cursor is currently located is used by default.

1. The LONG option of the DISPLAY\_MODE routine sets the column number range to 1-132 and enables you to enter BOX arguments in this range.
2. The extent of figures drawn by BOX is not affected by previous statements, including ROLL\_AREA or other routines that restrict the size of the scrolling area.

## Related Routines

1. All graph-drawing statements, including GRAPH and BARGRAPH, begin their operation by clearing their graph region of text. Therefore, when you want to put a box around some feature of a graph, you must use BOX *after* you use the graph-drawing routine.

## Restrictions

## **BOX**

2. BOX uses special text characters to draw lines. Therefore, any part of the box that extends into your scrolling area will scroll along with the other text in the scrolling area. You can use the `ROLL_AREA` routine to protect the box from this scrolling action.
3. The two column positions specified in a BOX statement cannot have common rows OR common columns. An error message appears when any BOX statement has `row 1=row 2` or `column 1=column 2`. Note that this occurs if you use BOX either with no arguments at all or with the option word alone, since all the rows and columns would default to the cursor position.

### **Error Conditions**

1. Error message: `ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED`

You have specified an illegal option word.

2. Error messages: `BAD ROW SPECIFICATION` or `BAD COLUMN SPECIFICATION`

One or more of the row and column numbers you entered are outside their legal range.

3. Error message: `SPECIFIED CORNERS ARE NOT DIAGONALLY OPPOSITE`

You have entered the same value for both row numbers or the same value for both column numbers (or both). This condition is also created by omitting all the row and column numbers from the statement.



# CHAR\_MODE

## Operation

As its name suggests, CHAR\_MODE sets the mode in which characters appear on the screen. A mode is a temporary state that remains in effect only until you change it.

The modes set by CHAR\_MODE remain in effect until they are changed by another CHAR\_MODE statement.

CHAR\_MODE is used most commonly to draw attention to a block of text on the screen, by making the text look different from the rest of the screen.

CHAR\_MODE provides the only means of controlling character modes of text that is printed by the PRINT and PRINT USING statements.

CHAR\_MODE("BOLD,REVERSE,UNDERLINE")

## Examples

sets the character mode of all subsequent characters to boldface, reverse video, underlined characters. This mode remains in effect until it is canceled by a TEXT\_INIT statement or another CHAR\_MODE statement.

CHAR\_MODE(option)

## Statement Form

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]B,[-]R,[-]F,[-]U	No change

**no arguments** Using CHAR\_MODE with no arguments has no effect on the current character mode.

## Argument Descriptions

**option** One or more of the following option words enclosed in quotation marks and separated by commas.

1. REVERSE specifies that all the following text output statements will display reverse video characters. Reverse video characters appear within a small area with the background color opposite that of the rest of the screen.
2. BOLD specifies that all the following text output statements will display boldface characters.

## CHAR\_MODE

3. UNDERLINE specifies that all the following text output statements will display underlined characters.
4. FLASH specifies that all the following text output statements will display flashing characters.

Figure 50 shows the appearance of various character modes.

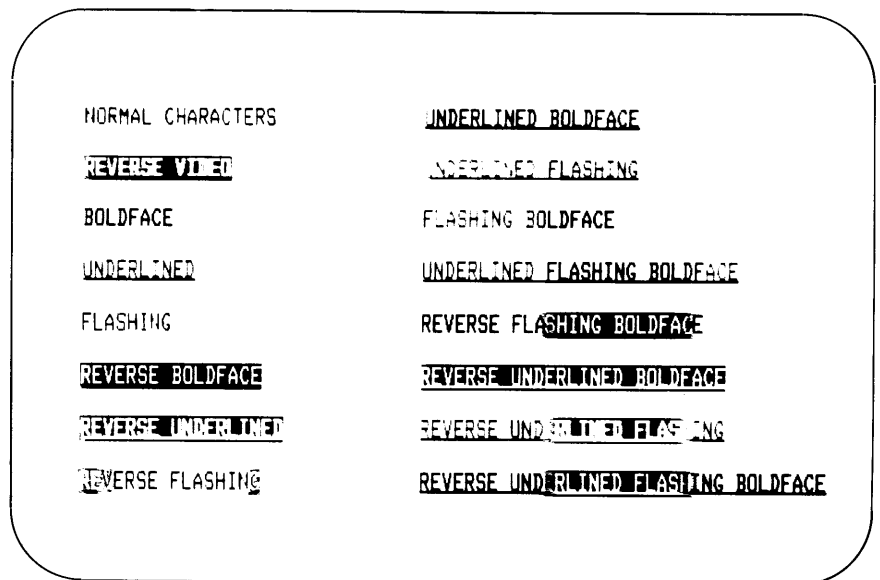


Figure 50. Sample Character Modes

### Related Routines

1. For the purpose of photographing the screen or using a screen copier, you may want to enable the interlacing feature in setup mode. Interlacing enhances the appearance of characters for photographic purposes because it reduces the amount of background color in each character. See “Setup Mode” in Book 7 for information about interlacing.
2. The TEXT\_INIT routine not only erases all text but also nullifies any previous CHAR\_MODE statements. TEXT\_INIT restores the stored setup mode parameters, which were defined at the factory and may be changed by a save (SHIFT/S) operation during setup mode. The saving of setup parameters is described in *Book 7: Working with MINC Devices*.
3. The DISPLAY\_MODE routine has an option (BRIGHT) that affects the appearance of characters on the screen. This option is far superior to the REVERSE option of CHAR\_MODE as a means of changing the entire screen to a

## CHAR\_MODE

white background. However, CHAR\_MODE can also display reversed characters after a DISPLAY\_MODE("BRIGHT") statement, in which case the reversed characters appear as white characters on small black backgrounds.

4. The LABEL, HTEXT, VTEXT, BOX, PUT\_SYMBOL, and TEXT\_LINE routines have all the character mode options of CHAR\_MODE; however, the character modes specified in these routines affect only a single text string or figure rather than all subsequent text. Therefore, these routines are preferable to CHAR\_MODE when you want to highlight a single entity.

1. Because CHAR\_MODE affects all subsequent text output, its effects are not limited to a particular part of the screen. The modes affect both the scrolling area and (conceivably) the graph regions.
2. Although the CHAR\_MODE options can be specified in any combination, the UNDERLINE option tends to reduce the readability of text that is also in reverse video.

### Restrictions

Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

### Error Conditions

You have attempted to specify an option that is illegal for CHAR\_MODE. The only legal options are BOLD, UNDERLINE, FLASH, and REVERSE.

# DISPLAY\_CLEAR

<b>Operation</b>	DISPLAY_CLEAR erases all text from the screen and erases all graphic displays. A single DISPLAY_CLEAR statement is fully equivalent to the following two statements:  ERASE_TEXT("TEXT") GRAPH_INIT
<b>Examples</b>	None included.
<b>Statement Form</b>	DISPLAY_CLEAR
<b>Argument Descriptions</b>	No arguments allowed.
<b>Related Routines</b>	<ol style="list-style-type: none"><li>1. DISPLAY_CLEAR erases both text and graphics in a single statement. To erase the text without erasing the graphics, use the ERASE_TEXT routine by itself. To erase the graphics without affecting the text, use either the ERASE_GRAPH or GRAPH_INIT routine by itself.</li><li>2. DISPLAY_CLEAR does not cancel the current character mode, nor does it recall the stored setup mode parameters. If you want to recall the stored parameters while erasing text from the screen, use the TEXT_INIT routine.</li><li>3. DISPLAY_CLEAR, like GRAPH_INIT, not only erases a graph from the screen but also cancels previous REGION and ROLL_AREA statements. To erase a graph without canceling these previous statements, use the ERASE_GRAPH routine.</li><li>4. DISPLAY_CLEAR, GRAPH_INIT, and ERASE_GRAPH all erase a graph from the screen and from the graphic memory of the terminal. To redisplay the graph, you must then repeat the graph-drawing statements that produced it originally. To remove a graph temporarily, use the VIEW routine. The graph can then be redisplayed immediately by another single VIEW statement.</li></ol>

## DISPLAY\_CLEAR

None.

**Restrictions**

**Error message: SYNTAX ERROR; CANNOT TRANSLATE THIS STATEMENT**

**Error Conditions**

You have either misspelled `DISPLAY_CLEAR` or have made some other syntax error, such as including an argument list. Repeat the statement.

# DISPLAY\_MODE

## Operation

DISPLAY\_MODE controls the background color, width in columns, and scrolling mode of the entire screen.

DISPLAY\_MODE can change these characteristics whether you use it before or after you actually display text. Note, however, that the LONG and -LONG options erase existing text and graphs from the screen.

## Examples

```
DISPLAY_MODE("BRIGHT")
```

sets the screen to display black characters on a white background.

```
DISPLAY_MODE("-BRIGHT, LONG, JUMP")
```

returns to displaying white-on-black characters, sets the screen to a width of 132 columns, and changes the scrolling mode to jump scrolling.

```
READY
```

```
DISPLAY_MODE("LONG")
```

```
READY
```

```
GRAPH(,, Y(0))
```

```
READY
```

produces a "long-format" graph such as is shown in Figure 51.

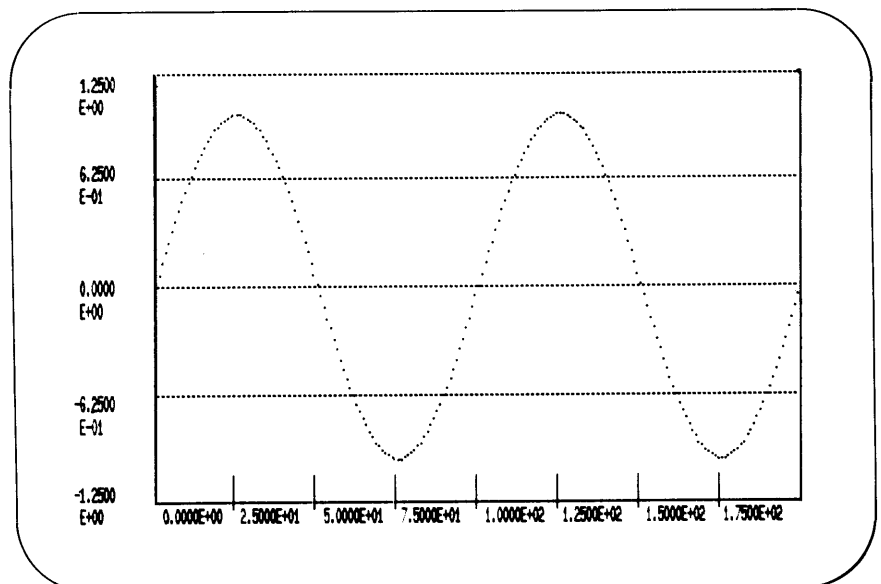


Figure 51. A Long-Format Graph

DISPLAY\_MODE(option)

Statement Form

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]L, [-]J [-]B	No change

**no arguments** Using DISPLAY\_MODE with no arguments is legal but has no effect on the screen.

Argument Descriptions

**option** One or more of the following option words, enclosed in quotation marks.

1. BRIGHT sets the background color of the screen to white, making displayed characters black. Negated by -BRIGHT. If you omit both these options, the background color is not changed.
2. LONG sets the screen width to 132 columns instead of the normal 80. Negated by -LONG. LONG and -LONG also erase all text from the screen and reset the scrolling area to rows 1-24. If you omit both these options, the screen width is unchanged.

LONG also performs an effective DISPLAY\_CLEAR statement, erasing graphs and text from the screen.

3. JUMP sets the scrolling mode to “jump,” so that a new line of text appears instantaneously in the scrolling area. The negation, -JUMP, sets the scrolling mode to “smooth,” so that a new line of the text scrolls onto the screen in smaller increments, giving the appearance of smooth motion. If you omit both these options, the scrolling mode is unchanged.

1. The three properties controlled by DISPLAY\_MODE can also be controlled in setup mode. However, you should use only DISPLAY\_MODE to set the background color and screen width, so that the setup mode parameters remain in the standard initial state. For example, none of the routines will accept a column number in the range 81-132 unless you have previously used DISPLAY\_MODE with the LONG option. These routines will fail if you have set the screen width to 132 via setup mode.

Related Routines

2. DISPLAY\_MODE(“LONG”) allows other graphic routines to use rows 81-132.

## DISPLAY\_MODE

3. The TEXT\_INIT routine eliminates the effects of all previous DISPLAY\_MODE statements and returns the terminal to the state saved in setup mode. This state may be the same as was defined at the factory. For details on saving and recalling setup mode parameters, see *Book 7: Working with MINC Devices*.

### Restrictions

In order for the DISPLAY\_MODE routine to work consistently with other routines, you must set the terminal to the standard initial state in setup mode (see Part 1).

### Error Conditions

Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have entered an illegal option word.



# DUAL\_MOVE

DUAL\_MOVE puts two graphs on the screen in strip-chart mode. It is the only means of doing so.

**Operation**

DUAL\_MOVE('DISTANCE',,1,Y1(0),Y2(0))

**Examples**

displays the entire contents of arrays Y1 and Y2 in strip-chart mode. The value of 1 for the start X argument is interpreted by DUAL\_MOVE as the distance between adjacent points on the graphs, where 1 is the smallest possible distance and 512 the largest. The increment argument was not included; therefore, all elements of both arrays are displayed. Array Y1 is displayed on the region defined for graph 1; array Y2 is displayed on the region defined for graph 2.

DUAL\_MOVE('-DISTANCE',100,X(0),Y1(0),Y2(0),2)

displays 100 points on each graph. In this case, the start X argument is the first element of array X. The values in array X will form the X axis for both graphs. Because the increment argument is 2, only the even-numbered elements are displayed from arrays X, Y1, and Y2.

DUAL\_MOVE(option,number,start X,start Y1,start Y2,increment)

**Statement Form**

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]D	-D
number	numeric expression	positive integer <= dimension of the smaller y array	smaller number of elements be- tween start Y1/ start Y2 and end of array Y1/Y2
start X	name of array element, or numeric expression	name of element in legally dimensioned array, or integer in range 1-512 (with D option), or real number	1 (with D option)  default not allowed with -D option
start Y1, start Y2	name of array element, or numeric expression	name of element in legally dimensioned array, or any legal numeric expression	not allowed
increment	numeric expression	integer in range 1-32,767	1

## DUAL\_MOVE

### Argument Descriptions

**option** A single word, enclosed in quotation marks, that specifies the proper interpretation of the start X argument. **DISTANCE** makes **DUAL\_MOVE** interpret start X as the distance in raster units between adjacent points on the screen. In other words, when you specify the **DISTANCE** option, start X must be an integer in the range 1-512. The option **-DISTANCE**, which is chosen by default when you omit the option word, makes **DUAL\_MOVE** interpret start X as the first element of an array of values that will be the start X coordinates for both graphs. This array must have a dimension at least as large as the two Y arrays.

**number** The number of points to plot on both graphs; must be equal to or less than the dimensions of the two Y arrays, if arrays are used for the graph 1 and graph 2 values.

If you do not include the number argument, **DUAL\_MOVE** will begin plotting from the array elements you specify and will continue to the ends of the arrays. If the arrays have different sizes, the length of the smallest array is used.

**start X** When the **-DISTANCE** option is used, start X is an element of the array of X values to use in plotting both graphs. When the **-DISTANCE** option is used, you must include the start X argument; there is no default value in this case, and omitting the start X argument causes an error. **DUAL\_MOVE** will use the value in this array element as the X coordinate of the first point on both graphs. In this case, the point is displayed only if the value of start X is within the window established for graph 1.

When the **-DISTANCE** option is used, **DUAL\_MOVE** uses the X window of graph 1 for *both graphs*.

When the **DISTANCE** option is used, start X must be an integer in the range 1-512, indicating the distance in raster units between adjacent points on the two graphs. In this case, you can omit the start X argument altogether, and a value of 1 will be used by default. A value of 1 simply means that the points will be as close together as possible.

**start Y1** An element of the array of Y values to plot on graph 1. **DUAL\_MOVE** plots the first point of graph 1 using the value in this element as the Y coordinate.

You can make this argument a numeric expression if the number argument is 1.

If start Y1 is an array element, the dimension of the array must be at least as large as the number argument.

**start Y2** An element of the array of Y values to plot on graph 2. DUAL\_MOVE plots the first point of graph 2 using the value in this element as the Y coordinate.

As with start Y1, this argument can be a numeric expression if one point is being plotted and must be an array element if many points are being plotted. If an array is used, its dimension must be at least as large as the number argument.

**increment** The array increment to be used by DUAL\_MOVE. If you leave this argument out of a DUAL\_MOVE statement, an increment of 1 is used by default. An increment of 1 means that each element of the arrays, beginning with the starting elements you name in the statement, will be plotted on the screen. An increment of 2 means that DUAL\_MOVE will plot from the starting elements and will then plot every other element. Similarly, you can skip two elements at a time by specifying an increment of 3. The increment cannot be less than 1 or greater than 32767.

1. DUAL\_MOVE is very similar in operation to the GRAPH routine with the MOVE option. However, GRAPH allows only one graph at a time in strip-chart mode. DUAL\_MOVE also differs from the GRAPH routine in its interpretation of the start X argument, and in general DUAL\_MOVE has considerably fewer options available.
2. When you want to display individual points in strip-chart mode, it may be more convenient to use the POINT routine, which has a MOVE option like that of GRAPH. However, as with GRAPH, you can use only one graph number at a time with POINT.
3. Although DUAL\_MOVE provides no options in its own string except DISTANCE, you can use the ERASE\_GRAPH, HLINE, SHADE, and VIEW routines to modify DUAL\_MOVE displays.
4. Because DUAL\_MOVE lacks the autoscaling feature of GRAPH, you must use the WINDOW routine to set the windows for both graphs.

## Related Routines

## DUAL\_MOVE

### Restrictions

1. You cannot use DUAL\_MOVE after a BARGRAPH, GRAPH, or POINT statement that includes the MOVE option unless you first use GRAPH\_INIT or DISPLAY\_CLEAR.
2. You cannot use the FIND\_POINT routine in conjunction with DUAL\_MOVE.
3. You cannot plot vertical lines (as with the VLINE routine) on the screen during a DUAL\_MOVE operation.
4. To add new points to a display created by DUAL\_MOVE, you should use DUAL\_MOVE again, not POINT or GRAPH.

### Error Conditions

1. Error message: ARRAY LENGTH IS TOO SMALL

Array Y1, Y2, or X array has too few elements to plot the requested number of points. Specifically, the number of elements from the starting element to the end of the array must be equal to or greater than the number argument times the increment argument. Repeat the statement with a smaller number argument or recreate the array(s) with a larger dimension. See the description of the DIM statement in Book 2 or Book 3.

2. Error message: "DISTANCE" MUST BE SPECIFIED WHEN X IS OMITTED

You have illegally omitted the start X argument when specifying the -DISTANCE option. This may mean that you have omitted the option word altogether, since the default option is -DISTANCE.

3. Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have specified an option word that is illegal for DUAL\_MOVE. The only legal options are DISTANCE and -DISTANCE.

4. Error message: VALUE OF ARGUMENT NO. 2 IS OUT OF BOUNDS

The number argument in your statement is less than 0 or greater than 32767.

5. Error message: VALUE OF ARGUMENT NO. 6 IS OUT OF BOUNDS

The increment argument in your statement is less than 1 or greater than 32767.

6. Error message: SINGLE STRIP-CHART MODE CANNOT BE ON WHEN DUAL\_MOVE IS CALLED

You used DUAL\_MOVE when the terminal was already in single strip-chart mode, for example, after a POINT or GRAPH statement with the MOVE option. First use GRAPH\_INIT to eliminate single strip-chart mode, and then repeat the DUAL\_MOVE statement.

7. Error messages:

X VALUE MUST BE GREATER THAN 0 WHEN "DISTANCE" IS SELECTED

X VALUE MUST BE LESS THAN 513 WHEN "DISTANCE" IS SELECTED

You have used DUAL\_MOVE with the DISTANCE option but with an X value that is not in the range 1-512. Repeat the statement with a legal X value.

# ERASE\_GRAPH

## Operation

ERASE\_GRAPH allows you to selectively erase any or all graphic material that you have displayed on the screen. ERASE\_GRAPH erases the material from the graphic memory as well as from the screen.

## Examples

ERASE\_GRAPH("ALL",,,0)

erases all features of graphs 1 and 2 from the screen and from the graphic memory.

The following examples show the effects of other ERASE\_GRAPH options on the original graph shown in Figure 52.

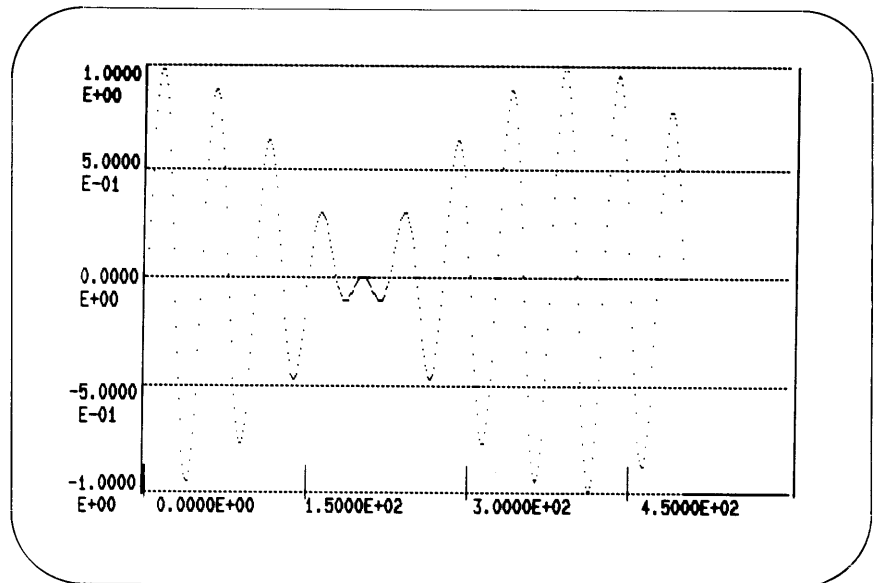


Figure 52. Original Graph

ERASE\_GRAPH("-ALL,POINTS")

erases the points from graph 1, leaving the grid features (Figure 53).

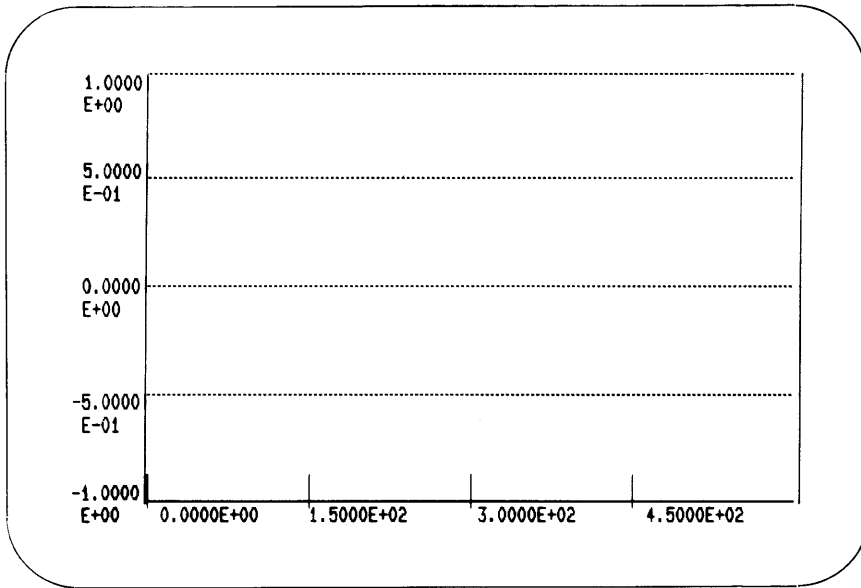


Figure 53. Result of ERASE\_GRAPH("-ALL,POINTS")

ERASE\_GRAPH("-ALL,GRID")

erases the grid features from the screen, leaving the points (Figure 54).

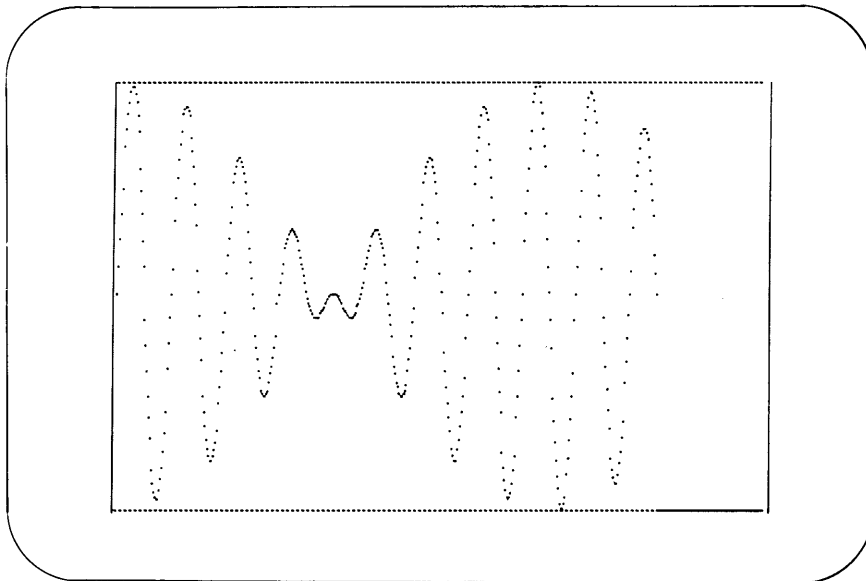


Figure 54. Result of ERASE\_GRAPH("-ALL,GRID")

## ERASE\_GRAPH

HLINE(1.00)

ERASE\_GRAPH("-ALL,HLINE",1.00)

displays, then erases a horizontal line located at Y = 1.00

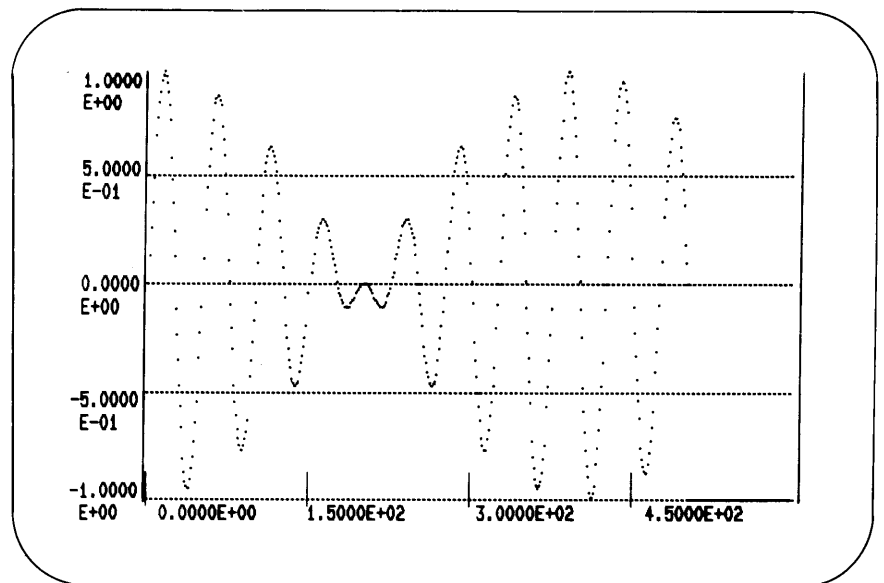


Figure 55. Result of ERASE\_GRAPH("-ALL,HLINE",1.00)

### Statement Form

ERASE\_GRAPH(options,X coordinate,Y coordinate,graph number)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]A,P,H,V,G	A
graph number	numeric expression	0, 1, or 2	1
X coordinate, Y coordinate	numeric expression	integer in range -32768 to 32767, or any real number	all lines

### Argument Descriptions

**no arguments** Using ERASE\_GRAPH with no arguments erases graph 1 in its entirety.

**options** A list of items to be erased from the screen, separated by commas and enclosed in quotation marks. The following list describes all the legal options.

1. POINTS erases the entire series of points (and brands) from the graph number you supply. If you include a graph number of 0, the points are erased from both graphs. If you



omit the graph number, only graph 1 is affected. The X and Y coordinates are not required by this option.

2. HLINE and VLINE erase horizontal and vertical lines (respectively) from the graph number you supply. HLINE can erase a single line at the Y coordinate you supply, and VLINE can erase a single line at the X coordinate you supply. If you do not supply the appropriate coordinate, HLINE and VLINE will erase all the horizontal and vertical lines on the graph number you supply.

HLINE operates on both graphs if you include a 0 as the graph number, and on graph 1 alone if you omit the graph number. (See also, "Restrictions" for ERASE\_GRAPH.)

VLINE always operates on both graphs and does not require a graph number argument.

3. GRID erases the grid (the horizontal lines, tick marks, and axis units) from the graph whose number you supply, or from both graphs if you include a graph number of 0. If you omit the graph number, the grid is erased only from graph 1. The X and Y arguments are not required by this option.
4. ALL completely erases graphs. ALL is the default option used if you include no option words in an ERASE\_GRAPH statement. Because ALL includes all the other ERASE\_GRAPH options, you must include its negative form (-ALL) in statements that are intended to erase only a single feature.

**X coordinate** The X coordinate of a vertical line to be erased. The coordinate must be within the current window to have any effect. This argument is only used by the VLINE option.

**Y coordinate** The Y coordinate of a horizontal line to be erased. The coordinate must be within the current window to have any effect. This argument is only used by the HLINE option.

**graph number** The number of the graph on which ERASE\_GRAPH operates. A value of 0 indicates both graphs, and a value of 1 or 2 indicates a single graph. If you omit the graph number, graph 1 is used by default.

## ERASE\_GRAPH

### Related Routines

1. Unlike `DISPLAY_CLEAR` and `GRAPH_INIT`, `ERASE_GRAPH` has no effect on previous `WINDOW`, `REGION`, `VIEW`, or `SHADE` statements.
2. The only text displays affected by `ERASE_GRAPH` are the units displayed by `GRID`, `POINT`, `GRAPH`, and `BARGRAPH`.
3. If you want to eliminate a graph temporarily rather than permanently, do not use `ERASE_GRAPH`; use `VIEW` instead.

### Restrictions

1. If supplied, the X and Y coordinates must be within the current window. If a coordinate is outside the window, it has no effect, although no error message is produced.
2. The horizontal grid lines displayed by `GRAPH` and `BARGRAPH` are not placed at exactly the Y position labeled on the Y axis. Consequently, you should use the `HLINE` option of `ERASE_GRAPH` only to erase those lines displayed by the `HLINE` routine.
3. Once you have erased a feature with `ERASE_GRAPH`, the feature can be recovered only by repeating the statements that produced the original display.
4. The options `ALL`, `POINTS`, and `VLINE` are illegal when applied to graphs in strip-chart mode.

### Error Conditions

1. Error message: `ILLEGAL GRAPH NUMBER`

The graph number you have supplied is not 0, 1, or 2. Repeat the `ERASE_GRAPH` statement with one of these graph numbers. (If you intend to erase graph 1, you can simply omit the graph number.)

2. Error message: `ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED`

The option word you specified is illegal for `ERASE_GRAPH`.

3. Error message: `CANNOT ERASE A CURRENTLY UNUSED GRAPH`

You have attempted to erase a graph that does not exist.

Note that if you omit the graph number from **ERASE\_GRAPH**, the statement applies to graph 1 by default.

4. **Error message: SPECIFIED OPTION(S) INCOMPATIBLE WITH MOVING GRAPHS**

You have attempted to use either the **POINTS**, **VLINE**, or **ALL** option to erase features from a graph that is in strip-chart mode. The only **ERASE\_GRAPH** options that are legal on moving graphs are **GRID** and **HLINE**. Repeat the **ERASE\_GRAPH** statement with the appropriate option, and be sure to include the option **-ALL** in the same statement.

# ERASE\_TEXT

## Operation

ERASE\_TEXT has a function very similar to that of ERASE\_GRAPH: selective erasure of the screen. The two routines differ in that ERASE\_TEXT operates only on text displays.

## Examples

ERASE\_TEXT("TEXT")

erases all text characters from the screen.

ERASE\_TEXT("TEXT,ROW",2,25,20)

erases a horizontal row of 20 characters beginning at row 2, column 25.

ERASE\_TEXT("-TEXT,COLUMN",2,25,20)

erases a vertical column of 20 characters beginning at row 2, column 25.

## Statement Form

ERASE\_TEXT(option,row number,column number,number to erase)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]T,C,R	T
row number	numeric expression	integer in range 1-24	with T option: all rows with others: row containing cursor
column number	numeric expression	integer in range 1-132	with T option: all columns with others: column containing cursor
number to erase	numeric expression	integer in range 1-32767	1

## Argument Descriptions

**no arguments** Using ERASE\_TEXT with no arguments erases all text from the screen (equivalent to the TEXT option with no other arguments).

**option** One or more option words enclosed in quotation marks.

1. If you omit the option word, the TEXT option is selected by default.
2. TEXT erases a single character at the row and column numbers you specify. If you do not give row and column numbers, all text is erased. If you give a column number without a row number, all text in that column is erased. If you give a row number without a column number, all text in that row is erased.

The number argument is not used by the TEXT option.

3. ROW erases characters in the row number you supply. Beginning with the column number you supply and working toward the right, this option erases the number of characters given by the number argument. The erasure always stops at the end of the row. If you select the ROW option and omit either the row number or column number, the current position of the cursor is used by default. If you select ROW and omit the number argument, a single character is erased.

ROW should be used only in combination with -TEXT.

4. COLUMN erases characters in the column number you supply. Beginning with the row number you supply and working downward, this option erases the number of characters given by the number argument. The erasure always stops at the end of the column. If you select the COLUMN option and omit either the column number or row number, the current position of the cursor is used by default. If you select COLUMN and omit the number argument, a single character is erased.

COLUMN should be used only in combination with -TEXT.

**row number** A number in the range 1-24. Indicates the row of text on which ERASE\_TEXT operates. If you omit the row number, the default condition depends on the option you have selected. For the TEXT option, the default is the entire column indicated by the column number. For the ROW and COLUMN options, the default is the row in which the cursor is located.

**column number** A number in the range 1-80 (or 1-132). Indicates the column of text on which ERASE\_TEXT operates. If you omit the column number, the default condition depends on the option you have selected. For the TEXT option, the default is

## ERASE\_TEXT

the entire row indicated by the row number. For the ROW and COLUMN options, the default is the column in which the cursor is located.

**number to erase** For the ROW and COLUMN options, the number of characters to erase. This argument can be any integer from 0 to 32767. A value of 0 causes no erasure.

If you omit the number to erase, a value of 1 is used by default. This argument is ignored by the TEXT option.

### Related Routines

1. Unlike TEXT\_INIT, ERASE\_TEXT does not change the effects of prior statements of DISPLAY\_MODE, CHAR\_MODE, ROLL\_AREA, or LIGHTS.
2. If you use the ROW option of ERASE\_TEXT and store the arguments for ROLL\_AREA in your program, it is straightforward to erase only the scrolling area, without affecting the labels on graphs.
3. The range of legal column numbers in an ERASE\_TEXT statement is affected by previous DISPLAY\_MODE statements. The LONG option of DISPLAY\_MODE makes the range 1-132. The -LONG option makes the range 1-80.
4. The figures produced by BOX, TEXT\_LINE, and PUT\_SYMBOL are erased by ERASE\_TEXT, since they are produced with special TEXT characters.
5. The ERASE\_GRAPH routine can erase the units displayed by GRID, GRAPH, and BARGRAPH. This is the only case in which ERASE\_GRAPH operates on text characters.

### Restrictions

1. Although the legal range of the number to erase argument is very large, the practical range is either 0-80 or 0-132, since the erasure always stops at the end of a row or column.
2. Because ERASE\_TEXT can erase the units displayed by GRID, GRAPH, and BARGRAPH, you should avoid using the TEXT option in its most general form when you have graphics on the screen.

### Error Conditions

1. Error message: VALUE OF ARGUMENT NO. 2 IS OUT OF BOUNDS

You have entered a row number that is outside the range 1-24.

2. Error message: VALUE OF ARGUMENT NO. 3 IS OUT OF BOUNDS

You have entered a column number that is outside the range 1-80 or 1-132.

3. Error message: VALUE OF ARGUMENT NO. 4 IS OUT OF BOUNDS

The number to erase is less than 0 or greater than 32767.

4. Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have specified an illegal option. The only legal ERASE\_TEXT options are TEXT, ROW, and COLUMN.

5. Error message: THE TEXT OPTION MUST BE SPECIFIED BY ITSELF

You have included other options in an ERASE\_TEXT statement besides TEXT. When you specify TEXT, neither ROW nor COLUMN may be specified in the same statement.

# FIND\_POINT

## Operation

After you have displayed a graph on the screen, FIND\_POINT allows you to find and store the coordinates of an interesting-looking point on a graph.

FIND\_POINT allows you to select a particular point interactively, by means of the following procedure.

1. As soon as FIND\_POINT is used, either in a program or in immediate mode, a flashing brand appears on the screen.
2. Move the flashing brand so that it covers the point of interest. You can move it with the four arrow keys at the top right edge of the keyboard.
3. When the brand is covering the point of interest, strike the **(RET)** key on the keyboard. The flashing brand disappears, and the coordinates of the point are stored in the variables X and Y.

If, prior to using FIND\_POINT, you set X and Y to a coordinate position that is within the current window, the flashing brand will begin at that position. If the preset position is outside the current window, it is ignored, and the flashing brand appears within the graph region, as close as possible to the coordinate position you specified in X and Y.

If you use FIND\_POINT in a program, the program will pause until you strike the **(RET)** key to store the coordinates. The program execution then resumes at the first statement following the FIND\_POINT statement.

## Examples

```
FIND_POINT(X1,Y5)
```

displays a flashing brand that you can move to cover an interesting point on a graph. When you press the **(RET)** key, FIND\_POINT stops and the flashing brand disappears. The X coordinate of the point is now stored in the variable X1, and the Y coordinate, in the variable Y5.



**FIND\_POINT(X, Y)**

**Statement Form**

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
X, Y	name of numeric variable	any legal name for real or integer variable	not allowed

**Argument  
Descriptions**

**X** The name of a variable that will hold the the X coordinate of the point. Before you use FIND\_POINT, you may set this variable to the X coordinate at which the flashing brand will first appear.

X is a required argument.

**Y** The name of a variable that will hold the Y coordinate of the point. Before you use FIND\_POINT, you may set this variable to the Y coordinate at which the flashing brand will first appear.

Y is a required argument.

**Related Routines**

1. FIND\_POINT is similar in operation to the INDEX option of GRAPH and BARGRAPH. However, FIND\_POINT actually returns coordinates that are within the current window of the graph, while the INDEX option returns an index to the arrays of input data used by GRAPH and BARGRAPH.
2. Because you can specify the exact starting position of the flashing brand, the brand is not restricted to the graphed points, as it is with the INDEX option of GRAPH and BARGRAPH.

**Restrictions**

1. The accuracy of Y coordinates found by FIND\_POINT is limited to the vertical length of the flashing brand.
2. You cannot use FIND\_POINT on a DUAL\_MOVE display, or on any other display that is in strip-chart mode.
3. When you use FIND\_POINT, only one graph may be present, either in visible or invisible form.

**Error Conditions**

1. Error message: ARGUMENT NO. n MUST BE A VARIABLE

You have entered an X or Y argument that is not a legal variable name. The value n=1 refers to the X argument; n will

## **FIND\_POINT**

also be 1 if you have entered nonvariable values for both arguments.

2. **Error message: MOVE MODES MUST BE OFF UPON INVOCATION**

You cannot use **FIND\_POINT** on graphs or bargraphs that were created in strip-chart mode, that is, by **DUAL\_MOVE** or by **GRAPH**, **POINT**, or **BARGRAPH** with the **MOVE** option.

# GET\_CURSOR and MOVE\_CURSOR

## Operation

GET\_CURSOR and MOVE\_CURSOR allow you to find out the exact row and column at which the cursor is located and to move the cursor to any screen position by supplying a new row and column.

GET\_CURSOR determines the cursor's current position.

MOVE\_CURSOR moves the cursor to any new position you specify.

## Examples

GET\_CURSOR(R1,C1)

stores the row number of the cursor's current position in the variable R1; stores the column number of the cursor's current position in the variable C1.

MOVE\_CURSOR(10,10)

moves the cursor to row 10, column 10. The next text to be displayed will start at this new position.

## Statement Form

GET\_CURSOR(row number,column number)

MOVE\_CURSOR(row number,column number)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Values</i>
row number	numeric expression or variable	integer in range 1-24	MOVE_CURSOR: top row of scrolling area GET_CURSOR: not allowed
column number	numeric expression or variable	integer in range 1-132	MOVE_CURSOR: column 1 GET_CURSOR: not allowed

**row number** For MOVE\_CURSOR, the row number, in the range 1 to 24, to which to move the cursor.

## Argument Descriptions

For GET\_CURSOR, the name of a variable that will receive the row number of the cursor.

## GET\_CURSOR and MOVE\_CURSOR

You may not omit the row number from a GET\_CURSOR statement.

If you omit the row number from a MOVE\_CURSOR statement, the top line of the current scrolling is used by default.

**column number** For MOVE\_CURSOR, the column number, in the range 1-80 or 1-132, to which to move the cursor.

For GET\_CURSOR, the name of a variable that will receive the column number of the cursor.

You may not omit the column number argument in a GET\_CURSOR statement.

If you omit the column number in a MOVE\_CURSOR statement, column 1 is used by default.

### Related Routines

1. The default position for MOVE\_CURSOR is always the first position of the current scrolling area. Therefore, any routine that changes the scrolling area also changes this default position. Routines that change the scrolling area are GRAPH, POINT, BARGRAPH, DUAL\_MOVE, REGION, ROLL\_AREA, GRAPH\_INIT, and TEXT\_INIT.
2. The legal range of column numbers in a MOVE\_CURSOR statement depends on previous DISPLAY\_MODE statements. The LONG option of DISPLAY\_MODE makes the columns range from 1 to 132. The -LONG option specifies columns ranging from 1 to 80.
3. MOVE\_CURSOR and GET\_CURSOR can address any location on the screen; they are not limited by previous REGION or ROLL\_AREA statements.
4. The PRINT and PRINT USING statements always begin their output at the current cursor position. You can use MOVE\_CURSOR to make such output appear outside the scrolling area. Figure 43 in Part 1 illustrates a graph that has a point marked with this technique.
5. In general, when you move the cursor outside the scrolling area and perform some operation, you should follow immediately with a MOVE\_CURSOR statement with no arguments, which will return the cursor to the first position in the scrolling area.

**Restrictions**

1. You cannot supply any sort of constant or arithmetic expression as a GET\_CURSOR argument. The GET\_CURSOR arguments must be variable names.

**Error Conditions**

1. Error messages: BAD COLUMN SPECIFICATION or BAD ROW SPECIFICATION

You have attempted to move the cursor to a row that is outside the range 1-24, or to a column that is outside the range currently in effect on your terminal (either 1-132 or 1-80).

2. Error message: ARGUMENT NO. n MUST BE A VARIABLE

You have used GET\_CURSOR with an argument or arguments that are not variables. The value n=1 refers to the row number; n will also equal 1 if you enter bad values for both GET\_CURSOR arguments.

3. Error message: CANNOT GET CURSOR POSITION FROM TERMINAL

The GET\_CURSOR routine was unable to return the current cursor position. Enter setup mode to ensure that the terminal is set for ANSI escape sequences. You may also need to press the NO SCROLL key.

# GRAPH and BARGRAPH

## Operation

GRAPH creates a complete graphic display using arrays of pre-computed X and Y data. GRAPH allows you to choose among several formats for displaying the data, and it allows you to index the coordinates of up to 10 points from the graph.

Like GRAPH, BARGRAPH creates a picture from arrays of precomputed data. Instead of creating a graph from points or line segments, however, BARGRAPH creates a figure in which the X and Y coordinates of a datum are represented by the position and height of a bar.

Both routines use a feature called *autoscaling*. The purpose of autoscaling is to make the MINC display screen conform to the numerical limits of your data. For instance, when you use GRAPH to display an array of Y coordinate data, you will commonly want the largest Y value in the array to appear near the top of the Y axis. The autoscaling feature will ensure this type of display, regardless of the numerical range of the input data.

Autoscaling is always used by GRAPH and BARGRAPH unless you have previously used the WINDOW routine.

## Examples

```
GRAPH("SHADE",500,,Y1(0),2,.75)
```

creates a 500-point graph from the Y coordinates stored in array Y1. The first point will be graphed with the Y coordinate in Y1(0). The increment is 2. Therefore, the Y coordinates will be taken from Y1(0), Y1(2), Y1(4), and so on until 500 points have been plotted. Note that Y1 must therefore have at least 999 elements. In the display created by this example, the units on the X axis will be the ordinal numbers of the displayed points.

The graph is shaded about the line  $Y = .75$ .

```
GRAPH("LINES,BRANDS",50,X1(0),Y1(0),,,2)
```

creates a 50-point graph with a graph number of 2. The X and Y coordinates of the points on the graph are taken from corresponding elements of arrays X1 and Y1, respectively. The 50 points are connected with line segments, and brands are placed at each point. The graph is not shaded.

BARGRAPH("EXACT",20,X1(0),Y1(0),2)

displays a bargraph of 20 bars. The positions of the bars on the X axis are determined by 20 elements from array X1. The heights of the bars are determined by 20 elements from array Y1. In both cases, the first 20 even-numbered elements are used, because of the increment of 2. The presence of the EXACT option sets the axis limits to equal the largest and smallest values in the 20 selected elements. Unless you have used SET\_BAR previously, the widths of the 20 bars are adjusted automatically to provide a pleasing separation between bars.

GRAPH(option,number,start X,start Y,increment,shade line, graph number,start index)

**Statement Form**

BARGRAPH(option,number,start X,start Y,increment,shade line,graph number,start index)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	-P, -H, -U, -T, G, [-]G, [-]M, [-]E, V, I, S, L, B	P
number	numeric expression	integer in range 0-32767	smaller no. of array elements between start X/start Y and end of X/Y array
start X, start Y, start index	name of numeric array element	name of any element in any legally dimensioned array	start X: ordinal number of plotted point start Y, start index: not allowed
increment	numeric expression	integer in range 0-32767	1
shade line	numeric expression	any legal Y coordinate, real or integer	lowest Y coordinate in window
graph number	numeric expression	1 or 2	1

**options** A list of the following option words, enclosed in quotation marks and separated by commas.

**Argument Descriptions**

1. If you omit the option list, GRID and POINTS are selected by default.
2. Three of the options control the type of symbol used to make the graph: POINTS, LINES, and BRANDS. POINTS creates a graph from a series of dots. LINES creates the graph by displaying dots and then connecting them with line segments. BRANDS creates the graph with short vertical line segments that are displayed where the points would otherwise appear.

## GRAPH and BARGRAPH

If you do not include any of the three options, POINTS is chosen by default.

### NOTE

The options POINTS and LINES have no meaning in the BARGRAPH routine; use them only with GRAPH. Note also that BRANDS cannot be used together with the MOVE option.

3. GRID places a pattern of horizontal lines across the graph region, evenly spaced in the Y direction. GRID also places short vertical line segments, called *tick marks*, on the X axis. Finally, GRID places numerical values called *units* near the points where the tick marks meet the X axis and the horizontal lines meet the Y axis. Thus, the horizontal lines, tick marks, and units all work together to make the graph or bargraph easier to interpret numerically.

GRID is selected by default.

4. If you include the GRID option in a statement, H LINES, TICKS, and UNITS are implied, and you do not have to request them separately. However, if you include the GRID option in a statement, you can subtract the grid features separately with the options -H LINES, -TICKS, and -UNITS. If you include the -GRID option in a statement, you cannot display horizontal or vertical lines, tick marks, or axis units.

The TICKS option is ignored after a previous statement that includes the MOVE option, until you use GRAPH\_INIT or DISPLAY\_CLEAR.

5. V LINES places vertical lines at the same points at which the tick marks appear. These lines are a further aid to interpreting a graph, but each vertical line extends the full length of the screen. Therefore, the V LINES option is useful only if you are using the full graph region.

The V LINES option cannot be used after a previous statement that includes the MOVE option, until you use GRAPH\_INIT or DISPLAY\_CLEAR.

6. INDEX allows you to store the indices of up to 10 points of a graph.

The indices that are stored tell your program where to find



the point's coordinates in your X and Y arrays. The index of a point is its displacement from the point at which you started the graph. If you entered X1(5) as the start X argument, an index of 0 means that the point in question was plotted from the value at X1(5). Similarly, an index of 1 refers to X1(6), an index of 2 to X1(7), and so on.

When you include the INDEX option in a BARGRAPH or GRAPH statement, MINC pauses and inspects the array element named in the start index argument. You must previously have set this argument to equal the number of allowable indices. Each of the allowable brands can be used to index a single point from the graph. If the start index element contains a 0, no indices are allowed, so MINC continues its normal operation. If the start index element contains an integer in the range 1-10, a single flashing brand appears on the screen. You can use the left- and right-arrow keys to move this brand until it covers an interesting point on the graph. When you have located the point, you can save its index by pressing the S key (for "Save"). This operation will save the point's index in the first element following the start index element.

To save the index of another point, you can press one of the numeric keys, either on the keypad or in the row above the top of the typewriter keyboard. This numeric key must be in the range 0 to 9, and it must also be in the proper numerical order as defined by your previous saving operations. For instance, if you have saved only one index so far, the numeric key you press must be either 1 or 2. The 1 key would signify that you are replacing the first index you saved with a new index. The 2 key would signify that you wanted to save a new index in the third element of the index array. The 0 key represents the tenth index.

After you have pressed S, pressing any valid numeric key makes a new flashing brand appear on the screen. This new brand can be positioned as before with the left- and right-arrow keys, and the new index can be saved with the S key.

You can return to normal operation at any time by pressing the **RET** key. This key restores the normal program flow that was interrupted by the indexing operations. After such a return, the start index element contains the number of indices actually stored, which may be different from the previous number of allowable indices. This feature is a convenience to ensure you that all indices you requested were actually stored.

## GRAPH and BARGRAPH

If, during a saving operation, you press an invalid numeric key, the terminal's buzzer will sound. You can continue normally by pressing a valid numeric key or the **RET** key.

You cannot use INDEX together with the MOVE option.

7. SHADE draws vertical lines between the points of your graph and a line called the *shade line*. Shading of this kind makes any graph more visually appealing, and it also has interpretive value.

If you display a graph containing a small number of points (relative to the maximum of 512), the SHADE option produces a result such as is shown in Figure 56.

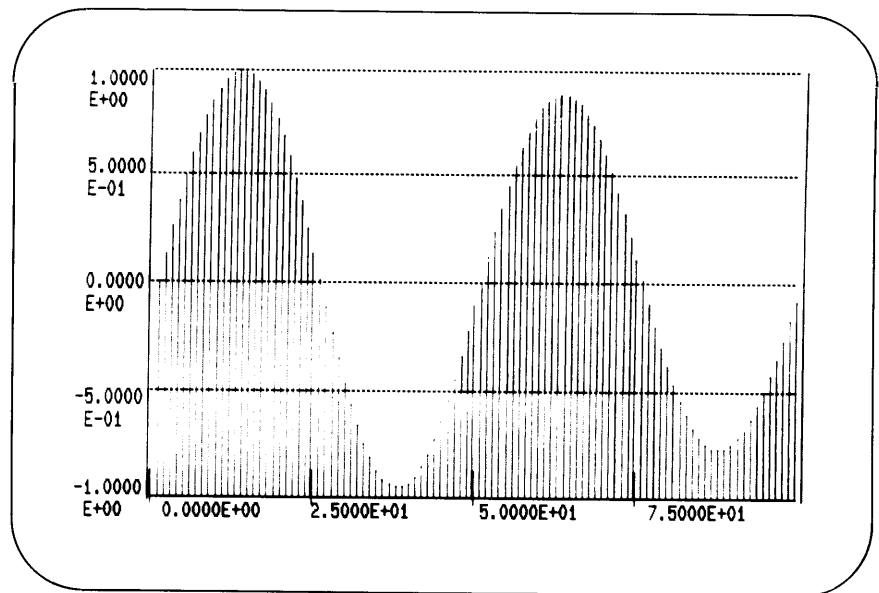


Figure 56. SHADE Option with a 100-Point Graph

To produce a solid shade, use the SHADE option together with the LINES option. LINES uses all displayable points to interpolate line segments between your actual data. The resulting graph resembles Figure 57.

8. MOVE enters strip-chart mode. After strip-chart mode is entered, subsequent statements (if they also include the MOVE option) push the entire graph or bargraph to the left to allow larger X values to be entered.

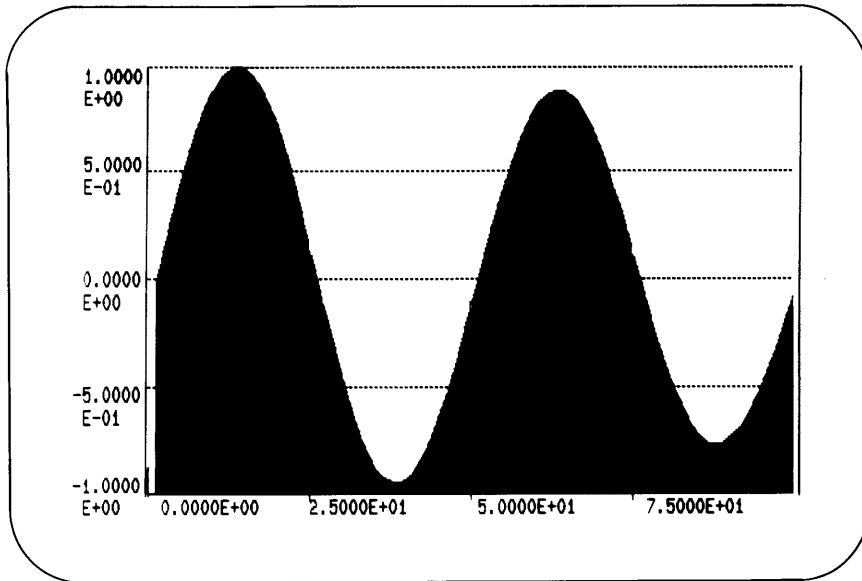


Figure 57. SHADE and LINES Options with a 100-Point Graph

If you include the MOVE option in a statement, you cannot include the option BRANDS or INDEX in the same statement. Following a statement with the MOVE option, you cannot include the VLINES option in subsequent statements, until you exit from strip-chart mode. You exit from strip-chart mode with the DISPLAY\_CLEAR routine or the GRAPH\_INIT routine.

9. EXACT produces axis units of which the largest and smallest values are exactly equal to the corresponding values of the input arrays. The default option, -EXACT, extends ranges of the axis units slightly, so that the largest and smallest values lie inside the maximum and minimum axis units. The -EXACT option has the desirable side-effect of producing reasonably "round" numbers for the axis units. Examples of graphs with and without the EXACT option are shown in Part 1.

**number** The number of values to plot from the X and Y arrays. This number must be equal to or less than the dimension of the smallest array you include in the statement, times the increment argument.

If you do not start the figure from element 0 of the arrays, the number must be correspondingly smaller. For instance, if you

## GRAPH and BARGRAPH

start plotting from Y1(3) when Y1 has 61 elements, the number to plot must be less than or equal to  $61 - 3 + 1$ , or 59. If you do not name an X array in the statement, this number of Y values will still be displayed from the Y array, and the X coordinates will be spaced evenly on the screen.

If you omit the number argument, the routine plots all values from your X and Y arrays, beginning with the array elements you supply in the start X and start Y arguments, and continuing until the end of the shorter array is reached.

**start X** An element of the optional array of X values to use in the graph. The X value in this element will be the X coordinate of the first point to be plotted. This array must have a dimension at least as large as the number of points you specify in the number argument.

If you omit the start X argument, the routine spaces the points (or bars) evenly on the screen (in the X direction).

**start Y** An element of the array of Y values to use in the graph. The value in this element will be the Y coordinate of the first point to be plotted. The array must have a dimension at least as large as the number of points you specify in the number argument.

The start Y argument is the only required argument in a BARGRAPH or GRAPH statement.

**increment** An optional integer that specifies the “step size” that the routine takes through your X and Y arrays. When you omit this argument from a statement, an increment of 1 is used by default. An increment of 1 means that after displaying the value in Y1(3), the routine displays the value in Y1(4). You can select any other increment from 1 to 32767 for the routine to use.

**shade line** The Y coordinate of the shade line. The graph is “shaded” by drawing vertical lines between all the displayed points and the shade line. The shade line has no significance unless you have included the SHADE option in your statement. If you include the SHADE option but do not supply a shade line argument, the shading begins at the bottom of the graph region by default.

**graph number** The graph number is always either 1 or 2. That is, your terminal can contain up to two figures (graphs or

bargraphs) at a time. Each of these figures has a graph number. If you specify graph 1 in a statement and there is already a graph 1 on the screen, the old graph 1 will be erased and the new one will replace it, unless the terminal is in strip-chart mode.

If you omit the graph number from the statement, graph 1 is used by default.

**start index** An element of the array used to store indices. Before you use GRAPH or BARGRAPH, you must set this element to the number of allowable indices. After an indexing operation, indices will be stored in this array, beginning with the element following the start index element. Additionally, the start index element will then contain a count of the number of indices actually stored. See the discussion of the INDEX option for more details.

1. The GRID, HLINE, VLINE, SHADE, and VIEW routines can be used to change the characteristics of a display produced by GRAPH or BARGRAPH.
2. The POINT routine can add single points to an existing display produced by GRAPH. However, POINT has no autoscaling feature.
3. Before you use GRAPH or BARGRAPH, you can use the REGION routine to place the display in the upper half or lower half of the screen. If you have not used REGION previously, the display will cover the full region by default.
4. The EXACT option of GRAPH and BARGRAPH is identical to the EXACT option of WINDOW. For more details about the EXACT option, see the description of WINDOW in Part 2.
5. You can use the ERASE\_GRAPH routine to erase some or all of a display created by GRAPH or BARGRAPH.
6. You can use the GRAPH\_INIT routine to eliminate the display produced by GRAPH or BARGRAPH. GRAPH\_INIT erases all parts of graphs 1 and 2 except the vertical text lines used to make the right and left borders of the graph and the axis units. GRAPH\_INIT also eliminates the effects of previous REGION and WINDOW statements.

GRAPH\_INIT and DISPLAY\_CLEAR cancel strip-chart mode.

### Related Routines

## GRAPH and BARGRAPH

7. `DISPLAY_CLEAR` has the same effect on `GRAPH` and `BARGRAPH` displays as does `GRAPH_INIT` except that, because it includes an `ERASE_TEXT` statement, `DISPLAY_CLEAR` also removes the vertical text lines and axis units from the graph.
8. In some circumstances, you can use the `FIND_POINT` routine as you would use the indexing feature of `GRAPH` or `BARGRAPH`. The major difference between the indexing feature and `FIND_POINT` is that `FIND_POINT` gives you the actual coordinates of a point as it appears on the screen, while the `INDEX` option supplies an index into the `X` and `Y` arrays.

### Restrictions

1. If you display graphs 1 and 2 in the same graph region (for example, both in the upper region), the axis units displayed will be those corresponding to the last figure you display. This property can lead to confusion, and it is recommended that you display different graphs on different parts of the screen.
2. Remember that the indices stored in an `INDEX` operation are offsets from the starting array element. Therefore, indices may not have the same numerical values as the array subscripts to which they refer. The only time the correspondence is exact is when you specify element 0 as the starting element.
3. When you use the `MOVE` option, you cannot use the `INDEX`, `BRANDS`, or `VLINES` option in the same `GRAPH` or `BARGRAPH` statement.

### Error Conditions

1. Error message: `ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED`

The option string has words beginning with illegal characters. The only legal characters for `BARGRAPH` and `GRAPH` are B, I, G, H, L, M, P, T, U, and V.

2. Error message: `ARRAY LENGTH IS TOO SMALL`

There are not enough array elements provided by your statement. Specifically, this message appears if the following formula is not satisfied:

where the function FLOOR computes the integer that is just less than the result of the division.

You can usually correct the error condition by repeating the routine statement with a smaller number argument. You can also increase the size of the array dimensions, although you must then recompute the data that go into the arrays (see the discussion of arrays in Book 2 and/or the description of the DIM statement in Book 3).

3. Error messages:

GRAPH NUMBER OF ZERO NOT ALLOWED HERE  
 ILLEGAL GRAPH NUMBER

The graph number argument must be either 1 or 2. The value 0, which specifies both graphs for some other routines, is illegal in a GRAPH or BARGRAPH statement.

4. Error message: VALUE OF ARGUMENT NO. 2 IS OUT OF BOUNDS

The number argument is less than 0 or greater than 32767.

5. Error message: VALUE OF ARGUMENT NO. 5 IS OUT OF BOUNDS

The increment argument is less than 1 or greater than 32767.

6. Error message: ARGUMENT NO. 4 CANNOT BE DEFAULTED

The start Y argument may be missing from the statement. This argument is the only required argument.

7. Error message: CAN'T DISPLAY VERTICAL LINES IN STRIP-CHART MODE

You have attempted to use the VLINE and MOVE options in the same statement, which is illegal.

8. Error messages:

SPECIFIED OPTION(S) INCOMPATIBLE WITH STRIP-CHART MODE

### INDEXING NOT ALLOWED IN STRIP-CHART MODE

You have included illegal options with the MOVE option. When you use GRAPH or BARGRAPH with the MOVE option, the options BRANDS and INDEX cannot be included in the statement.

#### 9. Error message: INDEX ARRAY IS NOT PRESENT

You have used GRAPH or BARGRAPH with the INDEX option, but you have illegally omitted the start index argument. To use the INDEX option, you must first create an index array, then assign to the first element of this array the number of indices you want to store, and, finally, repeat the GRAPH or BARGRAPH statement. In this statement, include both the option word INDEX and, as the start index argument, the name of the first element of the index array (for instance, I()).

#### 10. Error message: NUMBER OF INDICES IS OUTSIDE RANGE 0-10

Prior to using GRAPH or BARGRAPH, you have assigned an illegal number to the array element named in the start index argument. First, use an assignment statement to set this element to the correct number of indices. Then repeat the GRAPH or BARGRAPH statement, taking care to include the option word INDEX as well as the correct element name in the start index argument.

#### 11. Error message: PREVIOUSLY PLOTTED POINT NOT FOUND

The indexing operation in a GRAPH or BARGRAPH statement failed to locate an indexed point in your original data arrays.

This can happen when your array of coordinates is being changed dynamically by one of the MINC lab module routines (see Book 6).

#### 12. Error message: CAN'T BE IN DOUBLE STRIP-CHART MODE WHEN USING THIS ROUTINE

You have included the MOVE option in a GRAPH or BARGRAPH statement after the terminal was put in double strip-chart mode by the DUAL\_MOVE routine. First



## **GRAPH and BARGRAPH**

use `GRAPH_INIT` or `DISPLAY_CLEAR` to eliminate double strip-chart mode, then repeat the `GRAPH` or `BARGRAPH` statement.

# GRAPH\_INIT

<b>Operation</b>	<p>GRAPH_INIT eliminates the effects of all previous graphic routines, including GRAPH, BARGRAPH, SET_BAR, DUAL_MOVE, REGION, and ROLL_AREA.</p> <p>Text features that are displayed by the GRAPH, BARGRAPH, and POINT routines (the axis units and the right and left borders) are not erased by GRAPH_INIT.</p> <p>GRAPH_INIT erases all graphic features of graphs 1 and 2.</p>
<b>Examples</b>	None included.
<b>Statement Form</b>	GRAPH_INIT
<b>Argument Descriptions</b>	No arguments allowed.
<b>Related Routines</b>	<ol style="list-style-type: none"><li>1. GRAPH_INIT has no effect on the modes set by the CHAR_MODE, WIDE_LINE, and DISPLAY_MODE routines, nor does it eliminate features that are strictly textual, such as the labels displayed with LABEL, VTEXT, and HTEXT, the axis units displayed by GRAPH and BARGRAPH, and the displays created with TEXT_LINE, LIGHTS, BOX, and PUT_SYMBOL.  These textual displays are initialized by the TEXT_INIT routine.</li><li>2. Because it includes an ERASE_TEXT statement, the DISPLAY_CLEAR routine erases all features from the screen, including the axis units and borders displayed by the GRAPH, BARGRAPH, and POINT routines. Therefore, DISPLAY_CLEAR is more useful than GRAPH_INIT when you want to erase both text and graphics from the screen.</li><li>3. Because they require that a graph number be in use, the MAP_TO_GRAPH and MAP_TO_TEXT routines are illegal immediately after a GRAPH_INIT statement. The same applies to the graphic routine FIND_POINT.</li></ol>

## **GRAPH\_INIT**

None included.

The only error condition is a syntax error that will occur if you misspell `GRAPH_INIT` or if you include an argument string in the statement.

### **Restrictions**

### **Error Conditions**

# GRID

## Operation

GRID places a pattern of horizontal lines on the screen and small vertical lines (called *tick marks*) on the X axis. Numbers then appear where the tick marks and horizontal lines intersect the X and Y axes, respectively.

Figure 58 shows a typical graph that is overlaid by a grid.

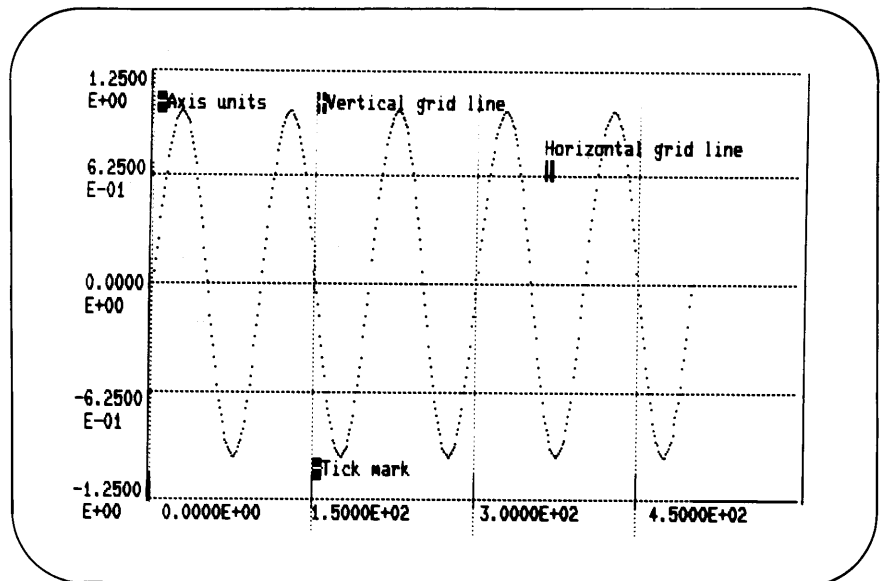


Figure 58. Sine Wave with GRID Display

## Examples

None included.

## Statement Form

GRID(option,graph number)

Argument	Type of Argument	Valid Values	Default Value
option	string expression	-H, -T, -U, V	H, T, U
graph number	numeric expression	1 or 2	1

## Argument Descriptions

**no arguments** Using GRID with no arguments displays a grid on graph 1. The grid does not include vertical lines.

**option** One or more of the option words -HLINES, -TICKS, -UNITS, and VLINES. If you omit the option word, horizontal

lines, tick marks, and units are displayed by default. For an explanation of these features, see the definition of GRAPH and BARGRAPH in Part 2.

**graph number** Graph number (1 or 2) on which the grid is displayed.

1. The numerical units displayed by GRID are a function of your previous WINDOW statements.
2. If you have used GRAPH or BARGRAPH with the -GRID option, you can add the grid later with the GRID routine and the correct graph number.

**Related Routines**

Avoid placing grids on two graphs if they are within the same graph region but have different windows.

**Restrictions**

1. Error messages: ILLEGAL GRAPH NUMBER or GRAPH NUMBER OF ZERO NOT ALLOWED HERE

**Error Conditions**

You have entered an illegal graph number. Note that a value of 0 is illegal for GRID.

# HLINE and VLINE

## Operation

HLINE and VLINE allow you to place horizontal and vertical lines on a graph. These lines help give the display an obvious scale, as do the lines on a piece of graph paper. Both routines allow you to specify the placement of the first line, the number of lines to display, the space between the lines, and whether the lines are visible.

## Examples

HLINE(,0,5,.25,1)

displays five horizontal lines. The first line is a Y=0. The lines are separated by .25 Y units. All five lines are part of graph 1.

VLINE("INVISIBLE",1.,,2)

locates the vertical line at X=1 on graph 2 and makes it invisible.

## Statement Form

HLINE(option,first Y,number,increment,graph number)

VLINE(option,first X,number,increment,graph number)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]INVISIBLE	-INVISIBLE
first X, first Y	numeric expression	any legal X or Y coordinate, real or integer	lowest X or Y coordinate in window
number	numeric expression	HLINE: integer in range 0-89 or 189 VLINE: integer in range 0-512	1 (both routines)
increment	numeric expression	any legal X or Y coordinate, real or integer	lines spaced evenly in remainder of region (both routines)
graph number	numeric expression	1 or 2	1

## Argument Descriptions

**no arguments** Using HLINE and VLINE with no arguments displays two lines on graph 1: a horizontal line at the bottom of the region and a vertical line at the left edge of the region (i.e., the X and Y axes, respectively).

**option** A single option word, enclosed in quotation marks. The option "INVISIBLE" makes the specified lines for the given

graph number temporarily invisible. The lines can be made visible again by a subsequent statement with the option “-INVISIBLE”.

**first X (Y)** The X or Y coordinate of the first vertical or horizontal line, respectively.

If you omit this argument, the lowest X or Y coordinate in the window is used by default.

**number** The number of lines to display. Must be an integer in the range 0-512 for VLINE or 0-89 or 189 for HLINE (a 0 value displays no lines). If you omit this argument, one line is displayed.

**increment** The amount of space between adjacent lines, expressed in X or Y coordinate units (for VLINE or HLINE, respectively). If you omit this argument, the requested lines are displayed, spaced evenly in the X or Y direction.

**graph number** The number of the graph (1 or 2) with which the lines are associated.

1. If you omit the first X or first Y argument, the numerical position of the first line is the lower X or lower Y as specified in a previous WINDOW use. If you did not use WINDOW, recall that, by default, lower X = 0.0 and lower Y = 0.0.
2. If you omit the increment argument, the number of lines displayed depends on the window for the graph number you specify. For instance, the space between vertical lines in such a case is given by

$$(\text{upper X} - \text{first X}) / \text{number of lines},$$

where upper X was specified by a previous WINDOW statement. If you did not use WINDOW previously, recall that upper X = 1.0 by default.

3. If the first X or first Y argument falls outside the current window, the VLINE or HLINE routine operates normally (that is, without an error message), and any lines that end up within the current window are displayed.
1. Vertical lines, whether displayed with the VLINE routine or with the VLINE option of GRAPH and BARGRAPH, always extend across the full height of the screen, regardless

## Related Routines

## Restrictions

## **HLINE and VLINE**

of previous REGION statements. There is no way to limit a vertical line to just the upper or lower graph region.

2. If a graph number is used by GRAPH or BARGRAPH with the MOVE option, VLINE cannot be used on the same graph number without an intervening GRAPH\_INIT or DISPLAY\_CLEAR statement.

### **Error Conditions**

1. Error message: VALUE OF ARGUMENT NO. 3 IS OUT OF BOUNDS

You have entered a number argument that is less than 0 or greater than 32767.

2. Error messages: ILLEGAL GRAPH NUMBER or GRAPH NUMBER OF ZERO NOT ALLOWED HERE

You have entered an illegal graph number. Note that the value 0 is illegal for HLINE and VLINE.



## HTEXT and VTEXT

### Operation

HTEXT and VTEXT allow you to display text strings either horizontally or vertically at any point on the screen.

After displaying a string, both HTEXT and VTEXT return the cursor to its previous position.

Figure 59 shows some of the kinds of displays possible with these two routines.

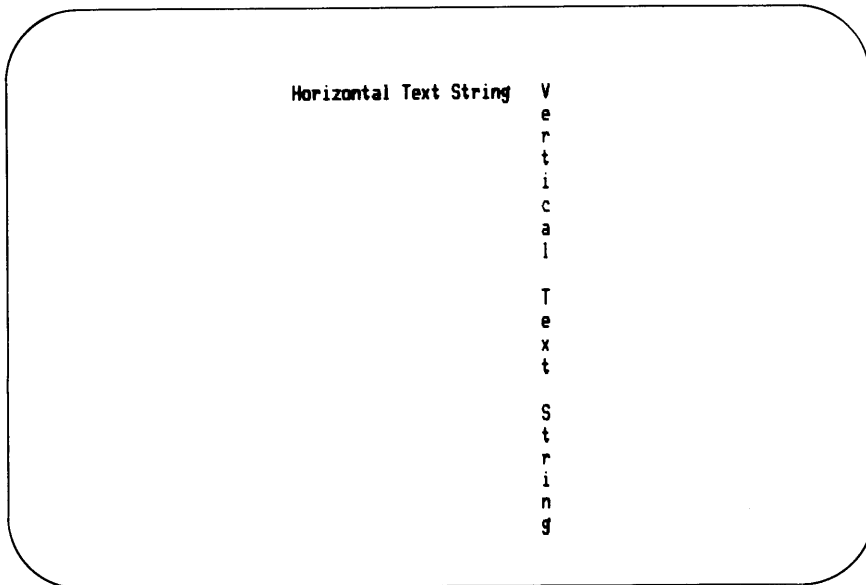


Figure 59. Typical HTEXT and VTEXT Displays

Notice that the characters are in different modes; both HTEXT and VTEXT allow you to specify a character mode that is unique to the string displayed by the statement.

```
HTEXT("BOLD",1,1,"MINC-11")
```

### Examples

displays the letters MINC-11 horizontally, beginning at row 1, column 1. The letters are displayed in boldface, but subsequent text is displayed in the current character mode.

```
VTEXT("BOLD,FLASH",10,132,"MINC-11")
```

## HTEXT and VTEXT

displays the letters MINC-11 vertically, beginning at column 132 of row 10. The letters are displayed in flashing boldface.

### Statement Form

HTEXT(option,row number,column number,text string)

VTEXT(option,row number,column number,text string)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	B, R, U, F	normal
row number	numeric expression	integer in range 1-24	row containing cursor
column number	numeric expression	integer in range 1-132	column containing cursor
text string	string expression	any legal string expression	empty (null) string

### Argument Descriptions

**no arguments** Using HTEXT or VTEXT with no arguments is legal but has no effect.

**option** One or more of the following option words, enclosed in quotation marks and separated by commas.

1. **BOLD** displays the text string in boldface.
2. **FLASH** displays the text string in flashing characters.
3. **REVERSE** displays the text string in a small area that is opposite in color from the screen background.
4. **UNDERLINE** underlines the text string.
5. If you omit the option, normal character mode is used.

**row number** The row number (1-24) on which the text string will start.

If you omit the row number from an HTEXT statement, the row on which the cursor is currently located is used by default.

If you omit the row number from a VTEXT statement, the text string is centered on the column.

**column number** The column number (1-80 or 1-132) on which the text string will start.

If you omit the column number from an HTEXT statement, the text string is centered on the row.

If you omit the column number from a VTEXT statement, the column in which the cursor is currently located is used by default.

**text string** The string to be displayed, expressed either as a string variable or as a literal (with quotation marks).

If you omit the text string, nothing is displayed.

If the text string is too long to fit in the given space, characters will be dropped, or “clipped,” from the right end of the string to make it fit.

1. Previous CHAR\_MODE statements do not affect the result of an HTEXT or VTEXT statement, because these statements use only the character mode defined in their own option strings.
2. Previous DISPLAY\_MODE statements can alter the appearance of a VTEXT or HTEXT display. The BRIGHT option of DISPLAY\_MODE, since it creates a white screen background, causes the REVERSE option of HTEXT and VTEXT to display white characters on a small black area. The LONG option of DISPLAY\_MODE makes the range of column numbers in HTEXT and VTEXT 1-132 instead of 1-80 and reduces the size of each character accordingly.

### Related Routines

1. Although options can be entered in any combination, underlining may reduce the readability of reverse video characters.
2. Do not enter such characters as carriage returns, line feeds, or form feeds in HTEXT or VTEXT strings.

### Restrictions

1. Error messages: BAD ROW SPECIFICATION or BAD COLUMN SPECIFICATION

### Error Conditions

Either the row number or column number in your statement is outside its legal range.

2. Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have attempted to enter an illegal option word. The only legal entries for HTEXT and VTEXT are BOLD, FLASH, UNDERLINE, and REVERSE.

# LABEL

## Operation

LABEL puts axis labels on graphic displays. It is particularly useful with displays created by the GRAPH and BARGRAPH routines.

Figure 60 shows a bargraph that has been labeled with the LABEL routine.

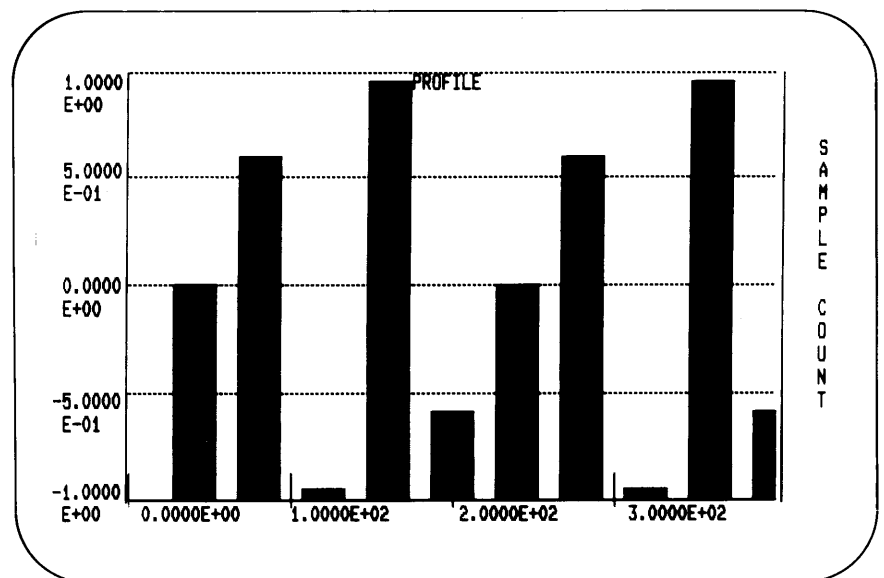


Figure 60. Result of the LABEL Routine

## Examples

LABEL ("BOLD,REVERSE","NUMBER OF STUDENTS",  
"AVERAGE IQ")

displays the title NUMBER OF STUDENTS at the top of graph 1; displays the title AVERAGE IQ vertically to the right of the graph region. Both titles are displayed in boldface reversed characters.

## Statement Form

LABEL(option,X label,Y label,graph number)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	B, R, U, F	normal
X label	string expression	any string of 0-132 characters	empty (null) string; current labels are erased if Y label is also omitted from statement
Y label	string expression	any string of 0-24 characters	empty (null) string; current labels are erased if X label is also omitted from statement
graph number	numeric expression	1 or 2	1

**no arguments** If you use LABEL with no arguments, it will erase the current X and Y labels.

**Argument  
Descriptions**

**option** One or more option words to describe the character mode of the labels. Legal words are FLASHING, BOLD, REVERSE, and UNDERLINE. If you omit the option word, normal characters are displayed.

**X label** A label that will be centered at the top of the graph region. This label can either be a literal (that is, a string of characters enclosed in quotation marks) or the name of a string variable (such as X1\$).

If you omit the X label from the statement, any X label that is already on the screen is erased.

**Y label** A label that will be displayed vertically on the right edge of the graph region. The Y label can either be a literal or a string variable.

If you omit the Y label, any Y label that is already on the screen is erased.

**graph number** The graph number (1 or 2) to which the labels apply. The LABEL routine will recognize the graph region being used for this graph and will place the labels in the correct position.

**Related Routines**

The character mode options of LABEL are identical to the options of the CHAR\_MODE routine. However, the BOLD, REVERSE, FLASHING, and UNDERLINE options of LABEL apply only to the X and Y labels supplied in the same statement, not to any other text on the screen.

## **LABEL**

### **Restrictions**

1. The X label cannot exceed the screen width (80 or 132 columns) defined by the DISPLAY\_MODE routine. Excess characters are ignored.
2. The Y label cannot exceed 24 characters for graphs using the full region, or 10 characters for graphs using the upper or lower region. Excess characters are ignored.
3. Do not include carriage returns, line feeds, or form feeds in a LABEL string.

### **Error Conditions**

Error messages: ILLEGAL GRAPH NUMBER or GRAPH NUMBER OF ZERO NOT ALLOWED HERE

The graph number you included in the LABEL statement was neither 1 nor 2.

# LIGHTS

There are seven small red light-emitting diodes, or LEDs, above the keyboard on the MINC terminal. With the LIGHTS routine, you can turn four of these lights (the ones labeled L1, L2, L3, and L4) on or off.

## Operation

LIGHTS("A,B,C,D")

## Examples

turns on all the LEDs.

LIGHTS("-B,-C")

turns off the two LEDs in the middle.

LIGHTS(option)

## Statement Form

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]A, [-]B, [-]C, [-]D	no change

**option** One or more of the letters A, B, C, or D, each of which stands for a single LED. The letters are separated by commas and the list is enclosed in quotation marks.

## Argument Descriptions

An option such as "A" turns on the LED represented by A. To turn off the same LED, use an option such as "-A".

Any letters that are omitted from a LIGHTS statement represent unchanged LEDs. Therefore, using LIGHTS with no arguments has no effect.

Figure 61 shows the correspondence between the letters A, B, C, and D and the four LEDs on the terminal.

## LIGHTS

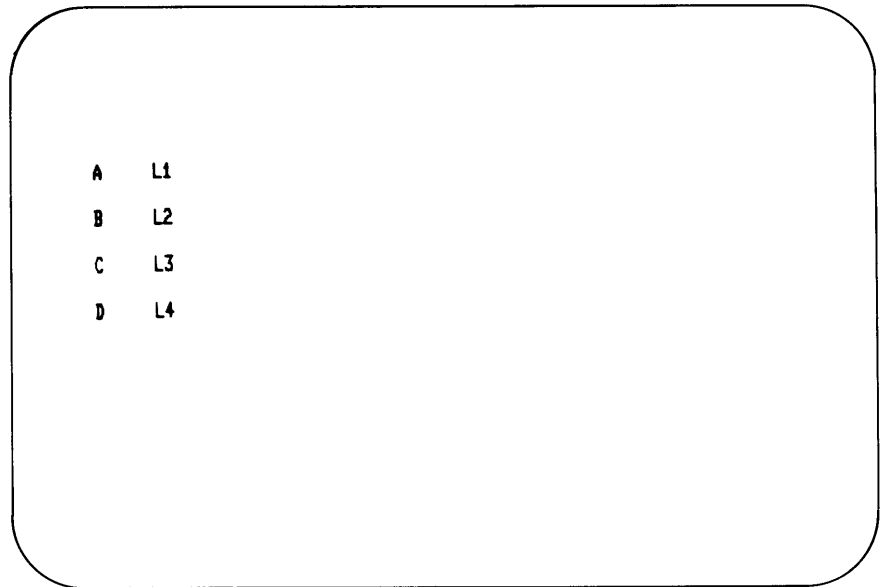


Figure 61. Programmable LED Options

### Related Routines

The `TEXT_INIT` routine turns off all the LEDs. No other routine affects the result of `LIGHTS`.

### Restrictions

Only the four LEDs named in Figure 61 are programmable with `LIGHTS` or any other routine.

### Error Conditions

Error message: `ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED`

You have attempted to enter an illegal option. The only legal entries for `LIGHTS` are the letters A, B, C, and D.



# MAP\_TO\_GRAPH

MAP\_TO\_GRAPH transforms a screen position from row-column coordinates to X-Y coordinates.

**Operation**

MAP\_TO\_GRAPH(5,10,X1,Y1,2)

**Examples**

transforms row 5 into a Y coordinate in the window of graph 2 and puts the Y coordinate in variable Y1; transforms column 10 into an X coordinate in the window and puts the X coordinate in variable X1.

MAP\_TO\_GRAPH(row number,column number,X,Y,graph number)

**Statement Form**

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
row number	numeric expression	integer in range 1-19	not allowed
column number	numeric expression	integer in range 1-132	not allowed
X, Y	name of numeric variable	legal name of real variable	not allowed
graph number	numeric expression	1 or 2	1

**row number** A row number in the range 1-19 (see also Restrictions, below). This is a required argument.

**Argument Descriptions**

**column number** A column number in the appropriate range (1-80 or 1-132). This is a required argument.

**X** A variable name. MAP\_TO\_GRAPH sets this variable to the value of the X coordinate corresponding to the row number.

The possible range of this coordinate is a property of the window and placement on the screen of the graph indicated by the graph number argument.

The X argument is required in all MAP\_TO\_GRAPH statements.

**Y** A variable name. MAP\_TO\_GRAPH sets this variable to the value of the Y coordinate corresponding to the column number.

The possible range of this coordinate is a property of the window

## MAP\_TO\_GRAPH

and region of the graph indicated by the graph number argument.

The Y argument is required in all MAP\_TO\_GRAPH statements.

**graph number** The number of the graph from which to extract window and screen placement information. This argument must be either 1 or 2 if specified; a 0 value is illegal.

If you omit the graph number, graph 1 is used by default.

### Related Routines

1. Previous REGION and WINDOW statements specify the correspondence between a row-column position and an X-Y position.
2. The MAP\_TO\_TEXT routine performs a function that is the reverse of MAP\_TO\_GRAPH. That is, MAP\_TO\_TEXT transforms X-Y coordinates to row-column coordinates.

### Restrictions

1. Do NOT use integer variables (such as X% or Y%) for the X or Y argument if the window is such that points are not represented adequately by integers. Using integer variables will not cause an error message, but the fractional parts of the returned coordinates will be lost.
2. Do NOT use row numbers that are outside the graph region. For graphs in the upper graph region, the row number must be in the range 1-9. For graphs in the lower region, the range is 11-19. For graphs in the full graph region, the range is 1-19. (Rows 10 and 20 are used for axis units.)
3. The X and Y coordinates returned by MAP\_TO\_GRAPH correspond to the approximate center of a capital H displayed at the given row-column position. Except by visual estimate, you cannot determine exact coordinates of arbitrary text positions such as the beginning of a letter or the point at which the period character (.) appears.

### Error Conditions

1. Error message: ARGUMENT NO. n MUST BE A VARIABLE

You have X and/or Y arguments that are not variable names. The value n=3 refers to the X argument; n will also equal 3 if you enter nonvariable values for both the X and Y arguments.

2. Error message: SPECIFIED ROW OR COLUMN NOT IN GRAPH REGION

You have entered a row-column position that is outside the region of the graph indicated by the graph number argument.

3. Error messages: ILLEGAL GRAPH NUMBER or GRAPH NUMBER OF ZERO NOT ALLOWED HERE

You have entered an illegal graph number. The only legal entries for MAP\_TO\_GRAPH are 1 and 2.

# MAP\_TO\_TEXT

## Operation

MAP\_TO\_TEXT transforms a screen position from the X-Y coordinates used by graphic routines to the row-column coordinates used by text routines.

## Examples

MAP\_TO\_TEXT(.5,-10.17,R1,C1)

transforms the X coordinate .5 to a column number and puts the column number in variable C1; transforms the Y coordinate -10.17 to a row number and puts the row number in variable R1.

## Statement Form

MAP\_TO\_TEXT(X coordinate,Y coordinate,R,C,graph number)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
X coordinate, Y coordinate	numeric expression	any legal X or Y coordinate	not allowed
R	name of numeric variable	any name of any integer variable	not allowed
C	name of numeric variable	any name of any integer variable	not allowed
graph number	numeric expression	1 or 2	1

## Argument Descriptions

**X coordinate** The X coordinate of a screen position. This coordinate must be within the current window of the specified graph number; otherwise, a negative column number is returned. This is a required argument.

**Y coordinate** The Y coordinate of a screen position. This coordinate must be within the current window of the specified graph number; otherwise, a negative row number is returned. This is a required argument.

**R** A variable name. This variable receives the computed row number. This is a required argument.

**C** A variable name. This variable receives the computed column number. This is a required argument.

**graph number** The number of the graph in which the X-Y point is located. This number must be either 1 or 2.

If you omit the graph number, MAP\_TO\_TEXT uses graph 1 by default.

## Related Routines

1. Prior WINDOW statements, which set the window for a graph number, govern the useful range of X and Y coordinates that you include in a MAP\_TO\_TEXT statement. If you have not used WINDOW previously, or if you have used WINDOW with no arguments, the window for both graphs is from the X-Y point (0,0) to the point (1,1).
2. The MAP\_TO\_GRAPH routine reverses the function of MAP\_TO\_TEXT, computing an X-Y position from a row-column position.

An important difference between the two routines is that MAP\_TO\_TEXT does not consider X-Y coordinates outside the window to be an error. Instead, MAP\_TO\_TEXT returns negative values for the row and column. Note that the returned values are negative whether the X-Y coordinates are larger or smaller than the window. Therefore, nothing can be deduced from these variables except that there is no row-column position corresponding to your X-Y position.

3. The R and C arguments in a MAP\_TO\_TEXT statement can be integer variables such as R1% and C1%, because row and column numbers are always integers.

## Restrictions

1. You can supply the graph numbers 1 and 2 in a MAP\_TO\_TEXT statement, or you can omit the graph number, implying graph 1. However, you cannot use the value 0 to represent both graphs, even when both graphs occupy the full screen.
2. You cannot use numeric expressions for R or C. Both must be legal variable names. Either single variables or array elements can be supplied for R and C.

## Error Conditions

1. Error messages: ILLEGAL GRAPH NUMBER or GRAPH NUMBER OF ZERO NOT ALLOWED HERE

You have attempted to supply an illegal graph number to MAP\_TO\_TEXT. You must either supply the value 1, the value 2, or omit the graph number.

2. Error message: ARGUMENT NO. n MUST BE A VARIABLE

## MAP\_TO\_TEXT

The items you supplied for R and C are not variables. The value n=3 refers to the R argument; n will also equal 3 if you enter bad values for both the R and C arguments.

3. Error message: SPECIFIED REAL NUMBER NOT CONVERTIBLE TO TEXT COORDINATE

The X and/or Y coordinate you supplied refers to a point that is “off the screen.” Therefore, it cannot be converted to a row/column number.

## **MOVE\_CURSOR**

See “GET\_CURSOR and MOVE\_CURSOR”

# POINT

## Operation

POINT displays a single point in the graph region you specify. You can associate this point with either graph 1 or graph 2, and you have the options of putting a brand on the point or entering strip-chart mode.

The X and Y coordinates in a POINT statement can be references to single elements of a virtual array file. Therefore, POINT with the MOVE option is useful for displaying large data sets that are stored as virtual array files on a MINC diskette.

## Examples

```
POINT(,5,1.)
```

displays a single point on graph 1 at X=.5, Y=1.

```
FOR I=0 TO 1200 \ POINT("MOVE",,Y(I)) \ NEXT I
```

displays 1201 points from array Y in strip-chart mode on graph 1. The first point appears at the extreme right edge of the graph region, and the entire set of points moves to the left as new points are displayed at the right edge.

## Statement Form

```
POINT(option,X coordinate,Y coordinate,graph number)
```

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]I, M, B, U	-I
X coordinate	numeric expression	any real or integer expression	with M option: right edge of region with others: not allowed
Y coordinate	numeric expression	any real or integer expression	not allowed
graph number	numeric expression	1 or 2	1

## Argument Descriptions

**option** One or more of the following option words, separated by commas and enclosed in quotation marks.

1. **INVISIBLE** makes the point temporarily invisible. The point can be made visible again by a subsequent POINT statement that includes the default option **-INVISIBLE**, with the same X and Y coordinates.



2. **BRAND** places a brand at the point. You can change a brand back to a regular point with another **POINT** statement having the same X and Y coordinates together with the default option **-BRAND**. You cannot include **BRAND** and **MOVE** in the same **POINT** statement.
3. **MOVE** displays the point in strip-chart mode. In strip-chart mode, subsequent **POINT** statements with the **MOVE** option that have no X coordinate plot the new point at the right edge of the graph region and shift existing points one raster unit to the left.

If the subsequent statement includes an X coordinate AND the **MOVE** option, the shift will occur only when the X coordinate exceeds the current window.

The **MOVE** option (and thus, strip-chart mode) is negated by the **DISPLAY\_CLEAR** and **GRAPH\_INIT** routines.

4. If you omit the option word, the options **-BRAND**, **-MOVE**, and **-INVISIBLE** are selected by default.

**X coordinate** The X coordinate of the point.

If you do not select the **MOVE** option in the **POINT** statement, the X coordinate is a required argument.

If you select the **MOVE** option and omit the X coordinate, the Y coordinate is still used, and the point is displayed at the right edge of the screen. All previous points are shifted one raster unit to the left.

**Y coordinate** The Y coordinate of the point. This is a required argument in all cases.

**graph number** The graph number (1 or 2) with which the point will be associated. If you omit the graph number, graph 1 is used by default.

1. Strip-chart mode can be terminated only with the **GRAPH\_INIT** or **DISPLAY\_CLEAR** routines.
2. In strip-chart mode, the point at which the leftward shift occurs is dictated by previous **WINDOW** statements. The shift occurs only when the X coordinate is greater than the current maximum X value of the window. If the X coordinate falls below the X range, the point is not displayed. (If the

## Related Routines

## POINT

point's Y coordinate is above or below the Y range, the point is forced to the top or bottom of the region, respectively.)

3. It is not an error to give coordinates to POINT that fall outside the ranges set by WINDOW. Therefore, although the point is not displayed, no error message is produced.
4. When a leftward shift occurs in strip-chart mode, the X range of the window is shifted by an amount equal to  $(\text{upper X} - \text{lower X})/511$ .

### Restrictions

1. Unlike GRAPH and BARGRAPH, POINT has no autoscaling feature. Therefore, although you can use POINT to add new elements to a GRAPH or BARGRAPH display, for a new point to be displayed, its coordinates must be within the same window as the display to which the point is added.
2. Because brands have a noticeable length, you cannot interpret Y coordinates accurately when brands are used on displayed points.

### Error Conditions

1. Error message: MOVE OPTION MUST BE SELECTED IF X ARGUMENT OMITTED

You have omitted the X coordinate from a POINT statement without selecting the MOVE option.

2. Error messages: ILLEGAL GRAPH NUMBER or GRAPH NUMBER OF ZERO NOT ALLOWED HERE

You have entered an illegal graph number. Note that 0 is illegal in POINT statements.

3. Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have entered an illegal option word.

4. Error message: CAN'T BE IN DOUBLE STRIP-CHART MODE WHEN USING THIS ROUTINE

You have followed a DUAL\_MOVE statement with a POINT statement including the MOVE option, without an intervening GRAPH\_INIT or DISPLAY\_CLEAR statement.

5. Error message: MOVE AND BRAND OPTIONS CANNOT BE SPECIFIED TOGETHER

You have included BRAND and MOVE in the same POINT statement.

6. Error message: MOVE OPTION NOT ALLOWED IF OTHER GRAPH IS IN USE

If you use graph 1 in strip-chart mode, you must first erase graph 2, and vice versa. If you must display both graphs in strip-chart mode, use DUAL\_MOVE.

7. Error message: MOVE OPTION MUST BE SPECIFIED IF MOVE SPECIFIED PREVIOUSLY

Following a statement with the MOVE option, you used POINT again without the MOVE option. If you want to stay in strip-chart mode, all POINT statements must contain the MOVE option. If you want to leave strip-chart mode, you must include the option -MOVE explicitly.

# PUT\_SYMBOL

## Operation

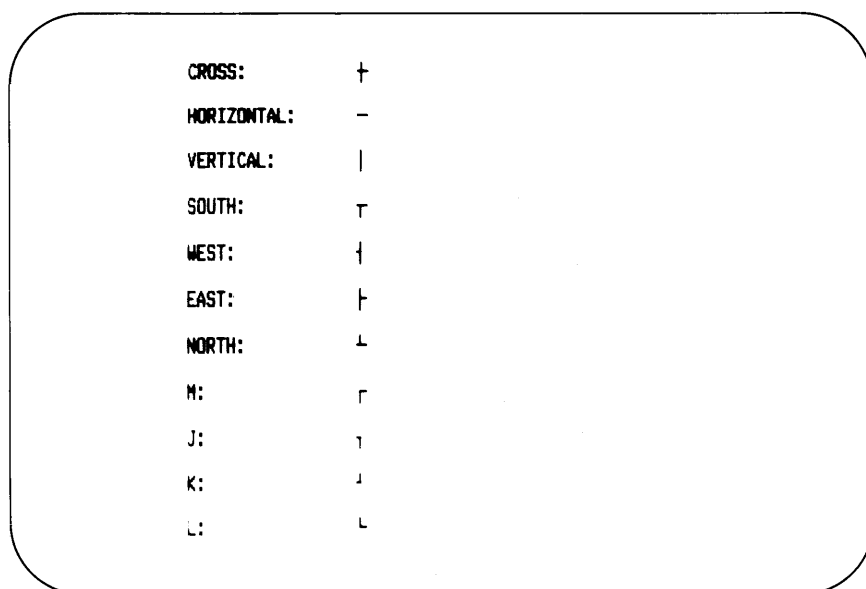
PUT\_SYMBOL displays one of a set of special symbols at the position you specify. PUT\_SYMBOL is useful for putting symbols on graphs and for connecting boxes and lines drawn by other routines.

## Examples

PUT\_SYMBOL("BOLD,FLASHING,CROSS",10,20)

displays a boldface flashing cross at column 20 of row 10.

Figure 62 shows the set of symbols that you can display with PUT\_SYMBOL.



CROSS:	+
HORIZONTAL:	-
VERTICAL:	
SOUTH:	⌞
WEST:	⌟
EAST:	⌞
NORTH:	⌟
M:	┌
J:	┐
K:	└
L:	┘

Figure 62. PUT\_SYMBOL Characters

## Statement Form

PUT\_SYMBOL(option,row number,column number)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	1 of: C, H, V, N, S, E, W, M, J, K, L + 1 or more of: B, R, U, F, [-]I	-I
row number	numeric expression	integer in range 1-24	row containing cursor
column number	numeric expression	integer in range 1-132	column containing cursor

**no arguments** Using PUT\_SYMBOL with no arguments displays a space at the current cursor position.

**option** One of the following option words, enclosed in quotation marks.

1. CROSS displays a square cross.
2. HORIZONTAL LINE displays a short horizontal line segment.
3. VERTICAL LINE displays a short vertical line segment.
4. SOUTH displays a T-bar with the foot pointing down.
5. WEST displays a T-bar with the foot pointing to the left.
6. EAST displays a T-bar with the foot pointing to the right.
7. NORTH displays a T-bar with the foot pointing up.
8. M displays a “northwest” corner character.
9. J displays a “northeast” corner character.
10. K displays a “southeast” corner character.
11. L displays a “southwest” corner character (roughly the shape of the letter L).
12. INVISIBLE displays a space.
13. One or more of the character mode options REVERSE, FLASHING, UNDERLINE, and BOLD can also be included.

If you omit the option word, the option “-INVISIBLE” is selected by default.

**row number** The number (1-24) of the row on which the symbol will appear. If you omit the row number, the row containing the cursor is used by default.

**column number** The number (1-80 or 1-132) of the column in which the symbol will appear. If you omit the column number, the column containing the cursor is used by default.

## PUT\_SYMBOL

### Related Routines

1. The range of legal column numbers, which is normally 1-80, can be extended to 1-132 by the "LONG" option of DISPLAY\_MODE. DO NOT use setup mode to set the 132-column feature.
2. The range of screen positions that can be addressed by PUT\_SYMBOL is unaffected by previous ROLL\_AREA statements or by other routines that limit the size of the scrolling area.

### Restrictions

1. The GRAPH, BARGRAPH, DUAL\_MOVE, POINT, and REGION routines begin their operations by clearing all text from their graph regions. Therefore, if you want to put a special symbol on such a graph, you must use PUT\_SYMBOL *after* using GRAPH or BARGRAPH.
2. The symbols displayed by PUT\_SYMBOL are special text characters. Therefore, any symbols displayed in your scrolling area will scroll along with the other text in the scrolling area.

### Error Conditions

1. Error messages: BAD ROW SPECIFICATION or BAD COLUMN SPECIFICATION

You have entered row and/or column numbers that are outside their legal ranges. If your column number is in the range 81-132, you must previously have selected the "LONG" option of DISPLAY\_MODE.

2. Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have entered an illegal option word.

3. Error message: ONLY ONE SYMBOL MAY BE DISPLAYED AT A TIME.

You have attempted to display more than one symbol with a single PUT\_SYMBOL statement.

# REGION

## Operation

REGION lets you decide which regions of the screen are used for graphic displays. You can select the full region, which corresponds to text rows 1-20; the lower region, corresponding to rows 11-20; or the upper region, corresponding to rows 1-10.

A REGION statement erases all text and graphics within the selected region.

A REGION statement also resets the scrolling area to the part of the screen that lies below the selected region. That is, selecting the upper region resets the scrolling area to rows 11-24. Selecting the lower region or the full region resets the scrolling area to rows 21-24.

REGION("LOWER")

specifies the lower region for use by graph 1; also erases all material in rows 11-20 and resets the scrolling area to rows 21-24.

REGION("FULL",2)

specifies the full region for use by graph 2, erases all material in rows 1-20, and resets the scrolling area to rows 21-24.

REGION("UPPER"),0

specifies the upper region for use by both graphs, erases all material in rows 1-10, and resets the scrolling area to rows 11-24.

REGION(option,graph number)

## Statement Form

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	U, L, F	F
graph number	numeric expression	0, 1, or 2	1

## Argument Descriptions

**no arguments** Using REGION with no arguments selects the full region for graph 1. The region for graph 2 is not changed.

**option** Your choice of one of three option words, enclosed in quotation marks. UPPER selects the upper region, LOWER selects the lower region, and FULL selects the full region. If you omit the option word, FULL is used by default.

## REGION

**graph number** The number of the graph (1 or 2) that will appear in the selected region. A value of 0 specifies both graphs in the same region. If you do not include a graph number in the statement, only graph 1 is associated with the selected region.

### Related Routines

1. When the GRAPH, BARGRAPH, and POINT routines begin a new display, they erase all previous graphics and text in the selected graph region.
2. The GRAPH\_INIT and DISPLAY\_CLEAR routines reset both graph numbers to the full region, canceling the effects of previous REGION statements.

### Restrictions

1. Do not use REGION when the terminal is in strip-chart mode. You can exit from strip-chart mode by using the DISPLAY\_CLEAR or GRAPH\_INIT routine.
2. REGION affects subsequent statements of such routines as GRAPH and BARGRAPH; you cannot use it to change the placement of existing displays.
3. You cannot use REGION or any other routine to display graphics in rows 21-24, which form the default scrolling area when the full or lower region is in use.
4. If you are using both graph numbers and the two graphs have different windows, you should avoid displaying them in the same graph region. The reason is that the units displayed by GRAPH, GRID, and BARGRAPH will apply only to the figure that was displayed most recently.

### Error Conditions

1. Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have specified an option that is illegal for REGION. The only legal options are UPPER, LOWER, and FULL.

2. Error message: SPECIFY ONLY ONE REGION, PLEASE

You have attempted to specify more than one option in the REGION statement.

3. Error message: ILLEGAL GRAPH NUMBER

You have attempted to specify an illegal graph number. If you specify the graph number, the only legal values are 0, 1, and 2.



# ROLL\_AREA

ROLL\_AREA defines the size of the scrolling area.

**Operation**

The *scrolling area* is the area where messages produced by your communication with MINC appear. Examples are the READY message, the output of a LIST command, the lines of a new program you are writing, and error messages.

ROLL\_AREA(13,24)

**Examples**

designates the lower half of the screen as the scrolling area.

ROLL\_AREA(21,24)

sets the scrolling area to rows 21-24 (the bottom four rows on the screen).

ROLL\_AREA(*top row*,*bottom row*)

**Statement Form**

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
top row	numeric expression	integer in range 1-23	1
bottom row	numeric expression	integer in range 2-24	24

**no arguments** Using ROLL\_AREA with no arguments resets the top row of the scrolling area to 1 and the bottom row to 24.

**Argument Descriptions**

**top row** A number from 1 to 23 that is the top row of the scrolling area.

If you omit this argument, the value 1 is used by default.

**bottom row** A number from 2 to 24 that is the bottom row of the scrolling area.

If you omit this argument, the value 24 is used by default.

1. Use ROLL\_AREA carefully after a REGION statement. In particular, avoid designating a scrolling area that overlaps a graph region, because the graph-plotting routines begin their operation by erasing all old material in their region, including text.

**Related Routines**

## ROLLAREA

2. The GRAPH and BARGRAPH routines always reserve the last four rows on the screen (rows 21-24) as a scrolling area. If, for example, you use GRAPH without a prior REGION statement, the default conditions call for a full-region graph. This graph will not intrude on the default scrolling area in rows 21-24.
3. The use of ROLL\_AREA has no effect on the MOVE\_CURSOR, GET\_CURSOR, BOX, TEXT\_LINE, HTEXT, VTEXT, MAP\_TO\_GRAPH, MAP\_TO\_TEXT, and PUT\_SYMBOL routines. All of these routines can address the full range of rows and columns at any time, regardless of the setting of the scrolling area.
4. CHAR\_MODE and DISPLAY\_MODE affect text in the scrolling area as well as text in the graph region.

### Restrictions

1. You cannot eliminate the scrolling area from the screen. As mentioned previously, the graphic routines provide a small default scrolling area at the bottom of the screen even when you select the FULL option of REGION. By the same token, you cannot specify the same row as the top and bottom row in a ROLL\_AREA statement, since doing so would effectively eliminate the scrolling area.
2. The scrolling area you designate is used only for ordinary text, that is, text not produced by the MINC graphic routines.

### Error Conditions

1. Error messages:

ZERO IS ILLEGAL AS ROLL\_AREA ARGUMENT

ARGUMENT OUTSIDE TERMINAL SCROLLING LIMITS

Either the top row or bottom row argument in your statement is outside the range 1-24.

2. Error message: FIRST ARGUMENT MUST BE SMALLER THAN SECOND

You have used ROLL\_AREA with a bottom row number that is less than or equal to the top row number.

# SET\_BAR

SET\_BAR is a special routine used together with BARGRAPH. Its only purpose is to control the width of bars in a bargraph.

**Operation**

SET\_BAR(40,2)

**Examples**

defines the width of the bars in bargraphs using graph 2. Subsequent BARGRAPH statements will display bars that are 40 raster units wide.

## NOTE

The screen measures 512 raster units in the X direction and 240 raster units in the Y direction. These units are "absolute" measurements that are unaffected by the autoscaling features of such routines as GRAPH and BARGRAPH. Absolute units are used for SET\_BAR arguments so that the bars will always have the width you want, regardless of changes in the scaling. Thus, a bar that is 51 raster units wide is always 10% of the width of the total screen.

SET\_BAR(*width,graph number*)

**Statement Form**

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
width	numeric expression	integer in range 1-512	automatic settings are restarted
graph number	numeric expression	0, 1, or 2	1

**no arguments** A SET\_BAR statement with no arguments causes BARGRAPH to revert to automatic setting of the bar widths for graph 1.

**Argument Descriptions**

**width** The width of the bar, measured in raster units. The width must be an integer in the range 0-511. If you omit the width argument, the widths of bars in subsequent bargraphs are set automatically by the BARGRAPH routine.

**graph number** The graph number to which the bar widths apply, either 0, 1, or 2. A value of 0 makes the SET\_BAR statement apply to both graphs.

## **SET\_BAR**

### **Related Routines**

1. Using SET\_BAR with a width argument disables the automatic bar setting feature of BARGRAPH.
2. Using SET\_BAR with a width argument disables autoscaling in the X direction. You can reinstitute autoscaling by using WINDOW again with no arguments, or by using DISPLAY\_CLEAR or GRAPH\_INIT.

### **Restrictions**

1. The width argument may not exceed 511.
2. Bars that are less than 2-3 raster units wide may be difficult to see.

### **Error Conditions**

1. Error message: NEGATIVE NUMBERS NOT ALLOWED

One of the arguments in the SET\_BAR statement was negative.

2. Error message: WIDTH MUST BE LESS THAN 512.

The width argument is greater than 511.

# SHADE

SHADE lets you add shading to existing graphs or to declare that future graphs with a given graph number will be shaded. Shading is especially useful for distinguishing two waveforms of similar shape.

## Operation

SHADE(.,5,2)

## Examples

shades graph 2 by connecting all points on the graph with the line  $Y=5$ .

SHADE("INVISIBLE")

removes shading from graph 1.

SHADE(option,shade line,graph number)

## Statement Form

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]INVISIBLE	-INVISIBLE
shade line	numeric expression	any legal Y coordinate	lowest value in window
graph number	numeric expression	0, 1, or 2	1

**no arguments** Using SHADE with no arguments shades graph 1, with the shade line located at the bottom of graph 1's region.

## Argument Descriptions

**option** One of two option words, enclosed in quotation marks. INVISIBLE makes the shading temporarily invisible. The shading can be made visible by a subsequent SHADE statement that includes the option word -INVISIBLE.

**shade line** The Y coordinate of a horizontal line. The shading consists of vertical lines drawn between this line and the displayed points. See "Related Routines," below.

**graph number** Either 1 or 2, designating the graph to be shaded. If you supply a value of 0 instead, both graphs are shaded. If you omit the graph number, graph 1 is shaded.

## SHADE

### Related Routines

1. The coordinate of the shade line is automatically adjusted to compensate for changes in the window and for the autoscaling feature of GRAPH and BARGRAPH. Thus, once you have placed the shade line at a particular Y coordinate, it will remain associated with that Y value until you change it with another statement.
2. If the coordinate of the shade line is above or below the Y range of the current window, the immediate effect is identical to setting the line to the top or bottom of the graph region, respectively. The desired position is remembered, however, so that if the window is subsequently changed and that Y coordinate appears on the screen, the shade line will be placed in the proper position.

### Restrictions

None.

### Error Conditions

1. Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have entered an illegal option word. The only legal entries are "INVISIBLE" and "-INVISIBLE".

2. Error message: ILLEGAL GRAPH NUMBER

You have entered an illegal graph number. Legal entries are 0, 1, and 2.

# TEXT\_INIT

## Operation

TEXT\_INIT erases all text from the screen (including the features of TEXT\_LINE, PUT\_SYMBOL, and BOX), eliminates any character modes you have set, and returns the terminal to the setup parameters saved during setup mode.

TEXT\_INIT also performs the function of the GRAPH\_INIT routine, erasing all graphic material from the screen.

None included.

## Examples

TEXT\_INIT

## Statement Form

No arguments allowed.

## Argument Descriptions

1. TEXT\_INIT eliminates all text and graphics displayed by the GRAPH, POINT, DUAL\_MOVE, and BARGRAPH routines, and it turns off any of the lights that were turned on by previous LIGHTS statements.
2. TEXT\_INIT eliminates the effects of all previous REGION, WINDOW, CHAR\_MODE, and DISPLAY\_MODE statements. After a TEXT\_INIT statement, the only mode settings will be those that you saved during setup mode.

## Related Routines

TEXT\_INIT returns the MINC terminal to the state prescribed in setup mode. The graphic routines work on the assumption that certain setup mode parameters are set to the standard initial state.

## Restrictions

The only error condition is a syntax error, created if you misspell TEXT\_INIT or if you use TEXT\_INIT with an argument string.

## Error Conditions

# TEXT\_LINE

## Operation

TEXT\_LINE draws lines of any length and direction on the screen. The routine uses a special set of text characters to draw the lines.

## Examples

None included.

## Statement Form

TEXT\_LINE(option,row 1,column 1,row 2,column 2)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	B, R, U, F, [-]I	-I
row 1, row 2	numeric expression	integer in range 1-24	row containing cursor
column 1, column 2	numeric expression	integer in range 1-132	column containing cursor

## Argument Descriptions

**no arguments** Although all arguments have default values, you cannot use TEXT\_LINE without arguments. See "Restrictions," below.

**option** Your choice of the options INVISIBLE or -INVISIBLE. INVISIBLE displays spaces instead of lines, erasing characters previously drawn in the path of the line. -INVISIBLE draws a visible line with small horizontal and vertical characters.

You can also include one or more of the character mode options REVERSE, BOLD, FLASHING, and UNDERLINE.

If you omit the option word, -INVISIBLE is used by default.

**row 1** The row number of one endpoint of the line; must be in the range 1-24.

If you omit row 1, the row in which the cursor is currently located is used by default.

**column 1** The column number of one endpoint of the line; must be in the range 1-80 or 1-132, depending on previous statements.



If you omit column 1, the column in which the cursor is currently located is used by default.

**row 2** The row number of one endpoint of the line; must be in the range 1-24.

If you omit row 2, the row in which the cursor is currently located is used by default.

**column 2** The column number of one endpoint of the line; must be in the range 1-80 or 1-132, depending on previous statements.

If you omit column 2, the column in which the cursor is currently located is used by default.

1. Your previous use of the `DISPLAY_MODE` routine governs the range of legal column numbers. The `LONG` option of `DISPLAY_MODE` sets the screen width to 132 columns. **DO NOT** use setup mode for this purpose.
2. The extent of lines drawn by `TEXT_LINE` is not limited by `ROLL_AREA`, nor by any other routine that restricts the scrolling area.

#### Related Routines

1. `TEXT_LINE` cannot allow both endpoints of the line to be located at the same row-column position. Since the current cursor position is the default for all the row-column arguments, it is therefore illegal to use `TEXT_LINE` with no arguments, or with only the option word.
2. `TEXT_LINE` uses special text characters to draw lines. Therefore, any part of the line that extends into your scrolling area will scroll along with the other text in the scrolling area.
3. `TEXT_LINE` always uses vertical or horizontal line segment characters to draw lines; one character is vertical and the other is horizontal. Therefore, lines that are neither horizontal nor vertical will exhibit a “stair-step” effect, which becomes more apparent as the angle approaches 45 degrees.

#### Restrictions

Figure 63 shows this stair-step effect on a 45-degree line.

## TEXT\_INIT

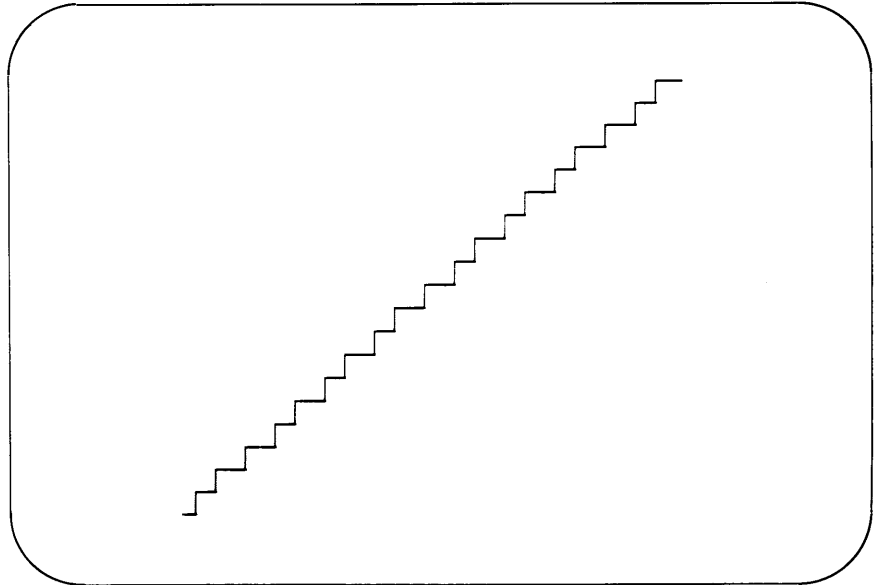


Figure 63. TEXT\_LINE Display with Stair-Step Effect

### Error Conditions

1. Error message: ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED

You have entered an illegal option word. "INVISIBLE" and "-INVISIBLE" are the only legal values.

2. Error message: LINE MUST HAVE DIFFERENT ENDPOINTS

You have entered arguments such that row 1 = row 2 AND column 1 = column 2. This condition is also caused implicitly by using TEXT\_LINE without any row or column arguments.

3. Error messages: BAD ROW SPECIFICATION or BAD COLUMN SPECIFICATION

You have entered row and/or column numbers that are outside the legal range. If you have entered column numbers in the range 81-132, be sure that you have selected the LONG option of DISPLAY\_MODE.

# VIEW

With the option **INVISIBLE**, **VIEW** makes graphs temporarily invisible without erasing the graphic memory. With the option **-INVISIBLE**, **VIEW** also makes an invisible graph visible again.

None included.

**VIEW**(option,graph number)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-] <b>INVISIBLE</b>	<b>-INVISIBLE</b>
graph number	numeric expression	0, 1, or 2	1

**no arguments** Using **VIEW** with no arguments makes graph 1 visible.

**option** A single option word, enclosed in quotation marks. **INVISIBLE** makes the specified graph temporarily invisible. **-INVISIBLE** makes the graph visible again.

**graph number** Either 1 or 2 if you want to affect a single graph. If you omit the graph number, only graph 1 is affected.

1. **VIEW** operates only on the points and brands displayed by **GRAPH**, **POINT**, **BARGRAPH**, and **DUAL\_MOVE**. It has no effect on previous or subsequent displays created by **SHADE**, **GRID**, **HLINE**, or **VLINE**.
2. **VIEW** has no effect on text displays created by **LABEL** or by any text-display routine.

None.

1. **Error messages: ILLEGAL GRAPH NUMBER**  
You have entered an illegal graph number.
2. **Error message: CAN'T TURN ON GRAPH THAT'S NOT IN USE**  
You have asked to view a graph that has not yet been displayed.

## Operation

## Examples

## Statement Form

## Argument Descriptions

## Related Routines

## Restrictions

## Error Conditions

## **VLINE**

See "HLINE and VLINE."

## **VTEXT**

See “HTEXT and VTEXT.”

# WIDE\_LINE

## Operation

**WIDE\_LINE** controls the width of characters on a single row. Figure 64 shows the two character widths available on a MINC terminal.

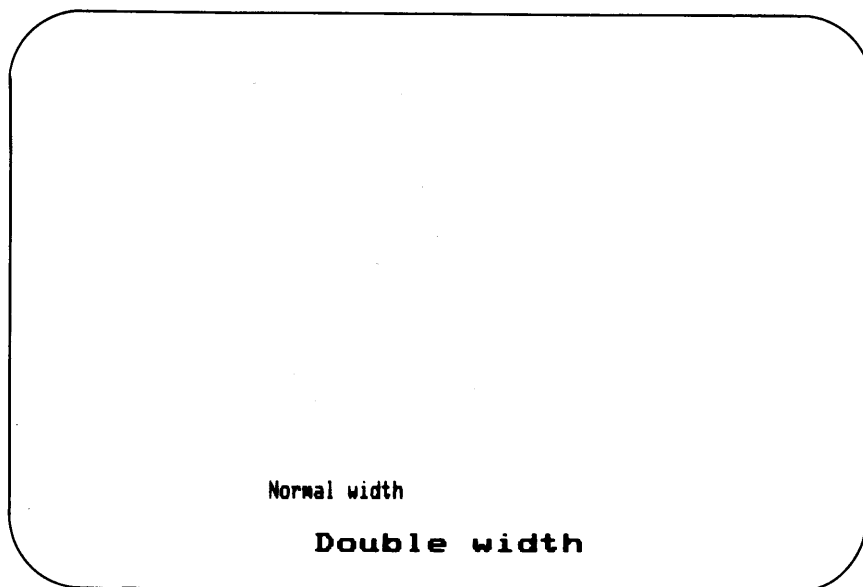


Figure 64. Regular and Double-Width Characters

When you use **WIDE\_LINE** on a line of characters, the double-width attribute moves with the line of characters if they scroll. Subsequent lines of text on the same row will be in regular width.

## Examples

None included.

## Statement Form

**WIDE\_LINE**(option,row number)

<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]WIDE	no change
row number	numeric expression	integer in range 1-24	row containing cursor

## Argument Description

**no arguments** Using **WIDE\_LINE** with no arguments is legal, but has no effect.

**option** A single option word enclosed in quotation marks.

**WIDE** sets the characters in the specified row to double width.  
**-WIDE** returns the characters on a row to normal width.

If you omit the option word, **WIDE\_LINE** has no effect on any row.

**row number** Number of the row on which **WIDE\_LINE** operates (range: 1-24). If you omit the row number, **WIDE\_LINE** operates on the row in which the cursor is currently located.

1. **WIDE\_LINE** controls the width of characters on a given row, either before or after the characters have actually been displayed by a text routine.
2. The **TEXT\_INIT** routine eliminates the effect of all previous **WIDE\_LINE** statements. Following a **TEXT\_INIT** statement, all lines will display characters of normal width.
3. Erasing a line of characters with **ERASE\_TEXT** does not change the effect of a previous **WIDE\_LINE** statement; the row will continue to display characters of the width that you specified prior to using **ERASE\_TEXT**.
4. **WIDE\_LINE** can operate on any row of the screen, whether it is part of the scrolling area or part of the graph region.

**Related Routines**

1. **WIDE\_LINE** affects all characters in a given row, regardless of the current position of the cursor. Consequently, there is no way to display a line that contains both normal and double-width characters.
2. Note that a line of more than 40 normal characters is truncated if you switch it to double width characters. If you have previously selected the **LONG** option of **DISPLAY\_MODE**, lines longer than 66 characters are truncated.

**Restrictions**

1. Error message: **BAD ROW SPECIFICATION**

**Error Conditions**

You have entered an illegal row number.

2. Error message: **ILLEGAL OR CONFLICTING OPTION(S) SPECIFIED**

You have either entered an illegal option word, or you have entered more than one option word in a single statement.

# WINDOW

## Operation

WINDOW sets the numerical limits of values that can appear on a graph. WINDOW can set a legal range for graph values in the Y direction and in the X direction. After these ranges have been set by WINDOW, only data that fall within the ranges will be displayed on the graph(s). Subsequent graphic routines will still accept data that are outside the window, but the data will not be displayed.

## Examples

WINDOW("EXACT",0.,1.,100.,511.,0)

sets the window for both graphs such that the X range is 0.-100. and the Y range is 1.-511. Autoscaling is disabled for both graphs. The lowest and highest units on the graph axes will have exactly those values specified in the WINDOW statement.

WINDOW("-EXACT",0.,1.,100.,511.,0)

sets the same window for both graphs as the previous example and disables autoscaling for both graphs. However, the highest units on the X and Y axes will exceed 100. and 511. slightly, to produce more readable units throughout the graph.

WINDOW

sets the window of graph 1 such that the X and Y ranges are 0.-1. Since no X or Y values are specified, reenables autoscaling for graph 1. (Note that the -EXACT option is also selected by default.)

WINDOW(,,,,,0)

sets the window for both graphs such that the X and Y ranges are 0.-1. Reenables autoscaling for both graphs. (Note that the -EXACT option is also selected by default.)

## Statement Form

WINDOW(option,lower x,lower y,upper x,upper y,graph number)



<i>Argument</i>	<i>Type of Argument</i>	<i>Valid Values</i>	<i>Default Value</i>
option	string expression	[-]EXACT	-EXACT
lower X, lower Y	numeric expression	any real-valued X or Y coordinate	0.0
upper X, upper Y	numeric expression	any real-valued X or Y coordinate	1.0
graph number	numeric expression	0, 1, or 2	1

**no arguments** Using WINDOW with no arguments resets the window of graph 1 to the default values and selects the -EXACT option. Such a statement also reenables autoscaling for graph 1. Graph 2 is unaffected.

If you include the option and/or graph number but omit all the X and Y arguments, the X and Y ranges are set to their default values and autoscaling is reenabled for the graph number you specify.

**option** Either EXACT or -EXACT. EXACT makes the lowest and highest numbers on the graph axes exactly equal to the lower and upper values in the WINDOW statement. -EXACT makes the lowest and highest numbers on the graph axes fall slightly outside the ranges set by the WINDOW statement. The effect of -EXACT is to make the units on your graph more readable.

If you do not include an option word in the WINDOW statement, -EXACT is selected by default.

**lower x** The lowest value of the X range. (Default value: 0.0)

**lower y** The lowest value of the Y range. (Default value: 0.0)

**upper x** The highest value of the X range. (Default value: 1.0)

**upper y** The highest value of the Y range. (Default value: 1.0)

**graph number** The optional number (1 or 2) of the graph to which this window applies. If you enter 0 for the graph number, the window applies to both graphs. If you omit the graph number, the window applies only to graph 1.

## Argument Descriptions

## WINDOW

### Related Routines

1. In general, all graphic and text routines that accept or return X and Y coordinates will produce visible results only when coordinates fall within the current window. No routine will fail because coordinates are outside the current window; the usual result is that a given point or object is not displayed on the screen. See the description of individual routines for exact details.
2. When you use `DUAL_MOVE`, or when you use the `GRAPH`, `BARGRAPH`, and `POINT` routines with the `MOVE` option, the X range set by a `WINDOW` statement may be shifted to the left. In other words, each time a new point exceeds the maximum value of X in the window, the window's X range is shifted by the amount required to place that point at the high end of the X axis. The Y range remains the same in these cases.

The `DUAL_MOVE` routine always uses the X window of graph 1 for *both* graphs 1 and 2.

If a point is plotted in strip-chart mode and its X coordinate is less than the minimum value of X in the window, the point is simply not displayed, and the X range of the window is unchanged.

3. If the starting coordinates in a `FIND_POINT` statement are outside the current window, they are ignored, and the `FIND_POINT` operation begins at the upper end of the current window.
4. If the X and/or Y coordinate in a `MAP_TO_TEXT` statement falls outside the current window, a negative value is returned for the corresponding row or column.
5. If you include any of the X or Y arguments in a `WINDOW` statement, the autoscaling feature of `GRAPH` and `BARGRAPH` is disabled for that graph. See the descriptions of `GRAPH` and `BARGRAPH` for more details about autoscaling.
6. The `GRAPH_INIT`, `TEXT_INIT`, and `DISPLAY_CLEAR` routines not only erase both graphs but also reset the windows of both graphs to the default values, reenables autoscaling for both graphs, and (effectively) select the `-EXACT` option for both graphs.

## WINDOW

### Restrictions

1. You cannot change the window of an existing graph. WINDOW operates only on subsequently displayed graphs.
2. You cannot set a window such that the relative difference between upper and lower limits is less than .0001. That is, the following two conditions will cause fatal error in a WINDOW statement:

$$|(\text{upper } X - \text{lower } x) / \text{lower } X| < .0001$$

$$|(\text{upper } Y - \text{lower } y) / \text{lower } Y| < .0001$$

1. Error message: DIFFERENCE IN WINDOW VALUES MUST BE GREATER THAN 1.0E-04

### Error Conditions

You have entered WINDOW coordinates that violate the relative difference restriction. See "Restrictions," above.

2. Error message: ILLEGAL GRAPH NUMBER

You have entered an illegal graph number. Legal values are 0, 1, and 2.



# INDEX

- Arrays
  - displaying large, 35
  - related to, 72, 73
- Autoscaling
  - definition of, 26
  - related to, 167, 182
- BARGRAPH routine
  - definition of, 122
  - related to, 167
- BASIC language, 19
- Boxes, 90
- BOX routine
  - definition of, 90
- Brands
  - illustration of, 24
- Character modes
  - changing, 93
  - definition of, 12
  - local, 13
- Characters
  - boldface, 12
  - controlling appearance of, 12
  - double-width, 178
  - flashing, 12
  - reverse video, 12
  - underlined, 12
  - width of, 178
- CHAR\_MODE routine
  - definition of, 93
  - example of, 12, 13
- Coordinates of point, 6
  - indexing, 126
  - related to, 152
  - storing, 116
- Cursor
  - moving, 119
  - reading position of, 119
- Data sets
  - displaying large, 77, 78
- DISPLAY\_CLEAR routine
  - definition of, 96
- DISPLAY\_MODE routine
  - definition of, 98
  - related to, 94
- Display modes
  - definition of, 13
- DUAL\_MOVE routine
  - definition of, 102
- ERASE\_GRAPH routine
  - definition of, 106
  - example of, 26
  - introduction, 14
- ERASE\_TEXT routine
  - definition of, 112
- Erasing graphs, 14, 106
- Erasing text, 112
- FIND\_POINT routine
  - definition of, 116
- GET\_CURSOR routine
  - definition of, 119
- Graph numbers
  - definition of, 14

## INDEX

- Graph regions
  - definition of, 7
  - related to, 163
- GRAPH routine
  - definition of, 122
  - examples of, 21
- Graphic memory
  - definition of, 14
- Graphics
  - clearing, 134
- GRAPH\_INIT routine
  - definition of, 134
  - example of, 23
- Graphs
  - coordinates for, 6
  - erasing, 14, 106
  - erasing axis units from, 129, 134
  - erasing borders of, 129, 134
  - hiding, 175
  - invisible, 14, 27, 175
  - labeling, 141, 144
  - long-format, 98
  - marking with symbols, 160
  - number per screen, 14
  - placement of, 9
  - shaded, 169
  - illustration of, 25
  - storing information about, 126
  - type-ahead during, 22
  - visibility of, 14
  - visible, 175
  - windows of, 7
- Grid features, 136
- GRID routine
  - definition of, 136
- HLINE routine
  - definition of, 138
- Horizontal lines
  - graphic, 138
  - text, 172
- HTEXT routine
  - definition of, 141
- Index array
  - definition of, 41
- Index option, 126
- Initial state of terminal, 15
- Interlacing, 94
- Labeling graphs, 141, 144
- LABEL routine
  - definition of, 144
- LEDs, 147
- LENGTH command, 77
- Light-emitting diodes, 147
- LIGHTS routine
  - definition of, 147
- Long-format graph, 98
- MAP\_TO\_GRAPH routine
  - definition of, 149
- MAP\_TO\_TEXT routine
  - definition of, 152
- Memory
  - graphic, 14
- MOVE\_CURSOR routine
  - definition of, 119
- MOVE option, 126
- Option strings
  - examples of, 22
  - negative forms of, 27
- Photographing screen, 94
- POINT routine
  - definition of, 156
- Points
  - coordinates of, 6
  - displaying single, 156
  - finding coordinates of, 116
  - indexing coordinates of, 126
- Programming language, 19
- PUT\_SYMBOL routine
  - definition of, 160
- Raster units
  - definition of, 167
- REGION routine
  - definition of, 163
  - example of, 9
  - examples of, 23
  - related to, 96
- ROLL\_AREA routine
  - definition of, 165
  - related to, 96
  - example of, 7
- Screen
  - changing background color of, 98
  - changing width of, 98
  - clearing, 96
  - columns per, 98
  - photographing, 94
  - resetting, 96, 134
  - subdivision of, 7
- Scrolling area
  - definition of, 8
  - erasing, 114
  - related to, 165

- Scrolling modes
  - definition of, 9
  - related to, 98
- SET\_BAR routine
  - definition of, 167
- Setup mode
  - related to, 15, 99
- Setup parameters
  - related to, 171
- SHADE routine
  - definition of, 169
- Standard initial state, 15, 100, 171
- Start index argument, 42
- Start X argument
  - related to, 72, 73
- Start Y argument
  - related to, 72, 73
- Strip-chart mode
  - definition of, 35
  - dual, 102
  - exiting from, 38, 129
  - related to, 126
- Symbols
  - special, 160
- Terminal
  - initial state of, 19
- Text displays
  - definition of, 12
  - erasing, 112
  - positioning, 6
  - vertical, 141
- TEXT\_INIT routine
  - definition of, 171
- TEXT\_LINE routine
  - definition of, 172
- Text parameters
  - resetting, 171
- Text positions
  - related to, 149
- Type-ahead during graph-drawing, 21
- Vertical lines
  - graphic, 138
  - text, 172
- VIEW routine
  - definition of, 175
  - illustration of, 27
  - introduction, 14
- Virtual array files, 77, 156
- VLINE routine
  - definition of, 138
- VTEXT routine
  - definition of, 141
- WIDE\_LINE routine
  - definition of, 178
- WINDOW routine
  - canceling effects of, 39
  - definition of, 180
- Windows
  - definition of, 7
  - resetting, 180
- Words of memory, 77
- Workspace length, 77





READER'S COMMENTS

**NOTE:** This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

---

---

---

---

---

---

---

---

---

---

Did you find errors in this manual? If so, specify the error and the page number.

---

---

---

---

---

---

---

---

---

---

Please indicate the type of user/reader that you most nearly represent.

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) \_\_\_\_\_

Name \_\_\_\_\_ Date \_\_\_\_\_

Organization \_\_\_\_\_

Street \_\_\_\_\_ Telephone \_\_\_\_\_

City \_\_\_\_\_ State \_\_\_\_\_ Zip Code \_\_\_\_\_  
or  
Country

Do Not Tear - Fold Here and Tape

**digital**



No Postage  
Necessary  
if Mailed in the  
United States



**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

**SOFTWARE PUBLICATIONS**  
200 FOREST STREET MR1-2/E37  
MARLBOROUGH, MASSACHUSETTS 01752

Do Not Tear - Fold Here and Tape

Cut Along Dotted Line