AT&T

305-506
Issue 1

AT&T 3B2 Computer
UNIX™ System V Release 2.0

Utilities – Volume 2

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

Replace this

page with the

GRAPHICS

tab separator.

AT&T

AT&T 3B2 Computer
UNIX™ System V Release 2.0
Graphics Utilities Guide

CONTENTS

Chapter 1

INTRODUCTION

Chapter 1

INTRODUCTION

GENERAL

This guide describes command formats (syntax) and use of the Graphics Utilities available with your AT&T 3B2 Computer. The numerical and graphical commands described in this guide are used to build and edit numerical data plots and hierarchical charts. This guide is designed for individuals experienced in using the **UNIX*** Operating System. Although these individuals are not expected to know UNIX System Shell Programming Language to use this guide, it would be helpful in understanding the examples at the end of Chapter 3.

---

* Trademark of AT&T

GUIDE ORGANIZATION

This guide is structured so you can easily find information without having to read the entire text. The remainder of this guide is organized as follows:

- Chapter 2, "OVERVIEW," gives a general description of the basic concepts of the Graphics Utilities and how to get started using the Graphics Utilities.
- Chapter 3, "STAT- A TOOL FOR ANALYZING DATA," describes routines that can be interconnected using the UNIX System shell to form numerical processing networks.
- Chapter 4, "COMMAND DESCRIPTIONS," describes the formats (syntax) for each command in the Graphics Utilities. The descriptions include the purpose of the command, a discussion of the command syntax and options, and examples of using each command.
- Chapter 5, "GRAPHICS EDITOR(**ged**)," describes an interactive editor used to display, edit, and build drawings on a TEKTRONIX 4014* display terminal.

* Registered Trademark of Tektronix, Inc.

Chapter 2

OVERVIEW

Chapter 2

OVERVIEW

INTRODUCTION

The **Graphics Utilities** is a collection of numerical and graphical commands used to build and edit numerical data plots and hierarchical charts. This chapter will help a user get started when using the **Graphics Utilities**. The best way to learn about graphics is to log onto the 3B2 Computer and use it. The examples below assume that the user is familiar with the UNIX System.

INTERFACING THE 5620 DMD TO THE 3B2 COMPUTER

To display drawings from the **Graphics Utilities**, you must use a graphics display terminal. The recommended graphics display terminal for the 3B2 Computer is the **TELETYPE*** 5620 Dot-Mapped Display (DMD) terminal. This terminal can emulate a TEKTRONIX 4014, a **HP†** 2621, and an **APS‡** 5. Unless otherwise noted, capabilities in this guide pertains to a 5620 DMD connected to a 3B2 Computer.

To interface the 5620 DMD to the 3B2 Computer, the following steps should be completed.

1. Interconnect the 5620 DMD to the 3B2 Computer.
2. Install the 5620 DMD Core Utilities. Instructions on how to install this utilities can be found in the *5620 Dot-Mapped Display Administrator Guide.*
3. Log on the system and create a layer. Instructions on how to do this can be found in the *5620 Dot-Mapped Display User Guide.*
4. Load the TEKTRONIX 4014 program in the layer you just created. The strap options *GINcount -g -u* must be entered so that the graphics editor (**ged**) will operate on a four-stage position instead of a two-stage position. Instructions on how to do this can be found in the *5620 Dot-Mapped Display User Guide.*

---

* Trademark of AT&T
† Trademark of Hewlett-Packard, Inc
‡ Trademark of Autologic, Inc

HOW COMMANDS ARE DESCRIBED

A common format is used to describe each of the commands. This format is as follows:

- **General:** The purpose of the command is defined. Any uncommon or special information about the command is also provided.
- **Command Format:** The basic command line format (syntax) is defined and the various arguments and options discussed.
- **Sample Command Use:** Example command line entries and system responses are provided to show you how to use the command.

In the command format discussions, the following symbology and conventions are used to define the command syntax.

- The basic command is shown in bold type. For example: **command** is in bold type.
- Arguments that you must supply to the command are shown in a special type. For example: **command** *argument*
- Command options and arguments that do not have to be supplied are enclosed in brackets ([ ]). For example: **command** [*optional arguments*]
- The pipe symbol (¦) is used to separate arguments when one of several forms of an argument can be used for a given argument field. The pipe symbol can be thought of as an exclusive OR function in this context. For example: **command** [*argument1* ¦ *argument2*]

In the sample command discussions, user inputs and 3B2 Computer response examples are shown as follows:

```
This style of type is used to show system generated
responses displayed on your screen.
```

This style of bold type is used to show inputs entered from your keyboard that are displayed on your screen.

These bracket symbols, < > identify inputs from the keyboard that are not displayed on your screen, such as: *<CR>* carriage return, *<CTRL d>* control d, *<ESC g>* escape g, passwords, and tabs.

This style of italic type is used for notes that provide you with additional information.

Refer to the *AT&T 3B2 Computer User Reference Manual* for UNIX System V manual pages supporting the commands described in this guide.

ACCESSING THE GRAPHICS UTILITIES

To access the **graphics** commands when logged in on the 3B2 Computer, type **graphics**. The shell variable *PATH* will be altered to include the **graphics** commands, and the shell primary prompt will be changed to^.

```
$graphics<CR>
^
```

Any command accessible before typing **graphics** will still be accessible; **graphics** only adds commands, it does not take any away. Once in **graphics**, a user can find out about any of the **graphics** commands using **whatis**. Typing **whatis** by itself on a command line will generate a list of all the commands in **graphics** along with instructions on how to find out more about any of them.

All the **graphics** commands accept the same command line format:

- A *command* is a *command-name* followed by *argument*(s).
- A *command-name* is the name of any of the **graphics** commands.
- An *argument* is a *file-name* or an *option-string*.
- A *file-name* is any file name not beginning with –, or a – by itself to reference the standard input.
- An *option-string* is a – followed by *option*(s).
- An *option* is a letter(s) followed by an optional value. Options may be separated by commas.

The **graphics** commands can be removed from the user's *PATH* by typing an end-of-file indication (<*CTRL-d*> control-d on most terminals) or by typing exit. This will put you in the UNIX System shell.

```
^exit<CR>
$
```

BASIC CONCEPTS

> ***Note:*** Many of the basic concepts of the Graphics Utilities will be explained in this chapter by using some of the **graphics** commands. It is not necessary now to fully understand these commands. However, if you need a more detailed explanation of a command, refer to Chapter 4, " Command Descriptions."

The basic approach taken with **graphics** is to generate a drawing by describing it rather than by drafting it. Any drawing is seen as having two fundamental attributes: **its underlying logic** and **its visual layout**. The layout contains one representation of the logic. For example, consider the $y=x^2$ for the value of x being between 0 and 10:

- The logic of the plot is the description as just given, namely $y=x^2$, for the value of x being between 0 and 10.
- The layout consists of an x-y grid, axis labeled perhaps 0 to 10 and 0 to 100, and lines drawn connecting the x-y pairs 0,0 to 1,1 to 2,4 etc.

The way to generate a drawing in **graphics** is:

1. Gather Data

2. Transform the Data

3. Generate a Layout

4. Display the Layout.

The following is an example of generating a drawing of $y=x^2$, for the value of x being between 0 and 10 and displaying it on a TEKTRONIX 4014 display terminal.

- The **gas** command is used to *gather the data*. The **gas** command generates a sequence of numbers, in this case **s**tart at 0 and **t**erminating at 10.

- The **af** command is used to *transform the data*. The **af** command performs general arithmetic transformations.

- The **plot** command is used to *generate a layout*. The **plot** command builds x-y plots.

- The **td** command is used to *display the layout*. The **td** command displays drawings on TEKTRONIX 4014 display terminal.

The command line format to generate the drawing for $y=x^2$ would be:

```
^gas -s0,t10 | af " x^2" | plot | td<CR>
```

The results of the drawing is shown in Figure 2-1.

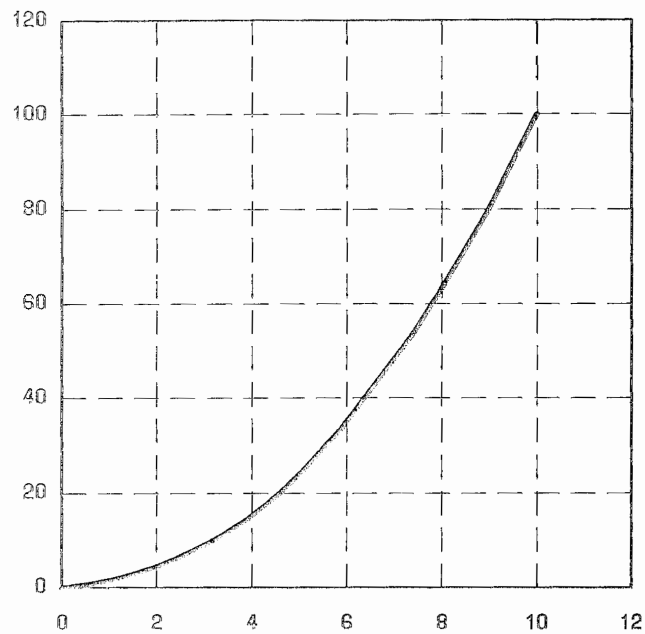

Figure 2-1. **Plot of gas -s0,t10 | af "x^2" | plot | td**

To clear the screen after displaying a drawing use the **erase** command.

```
^erase<CR>
```

The layout generated by a **graphics** program may not always be precisely what is wanted. There are two ways to influence the layout. The first way to influence the layout is to use the **plot** command options. For instance, in the previous example, it may be desired to have the x-axis labels show each of the numbers plotted and not have any y-axis labels at all. To achieve this the **plot** command would be changed to:

```
^gas -s0,t10 | af " x *^2" | plot -xi1,ya | td<CR>
```

The results of the drawing is shown in Figure 2-2.

The second way to influence a layout is by editing it directly at a display terminal using the graphical editor, **ged**. To edit a drawing really means to edit the computer representation of the drawing. For **graphics**, the representation is called a graphical primitive string or GPS. All the drawing commands (for example **plot**) write GPS, and all the device filters (for example **td**) read GPS. **Ged** allows manipulation of GPS at a display terminal by interacting with the drawing that the GPS describes (see Chapter 5.)

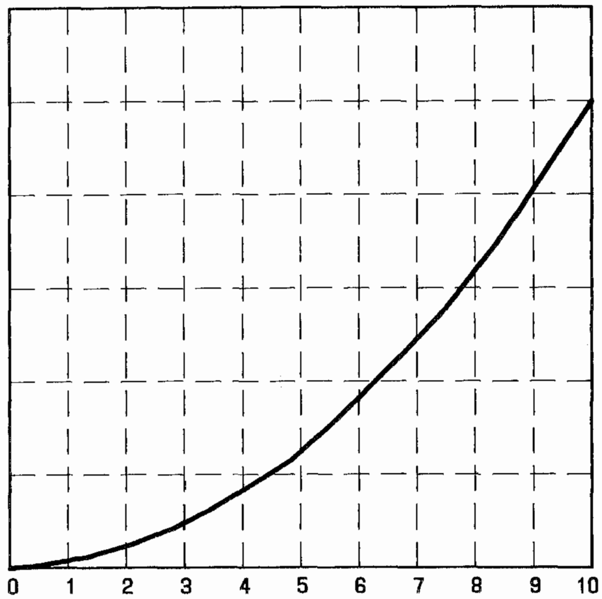

Figure 2-2. Plot of gas -s0,t10 | af " x^2" | plot -xi1,ya | td

The GPS describes graphical objects drawn within a Cartesian plane (Figure 2-3). The Cartesian plane has 65,534 units on each x and y axis. The plane, known as the *universe*, is partitioned into 25 equal-sized square regions. Multi-drawing displays can be produced by placing drawings into adjacent regions and then displaying each region. This will be shown in Chapter 5 (graphics editor).

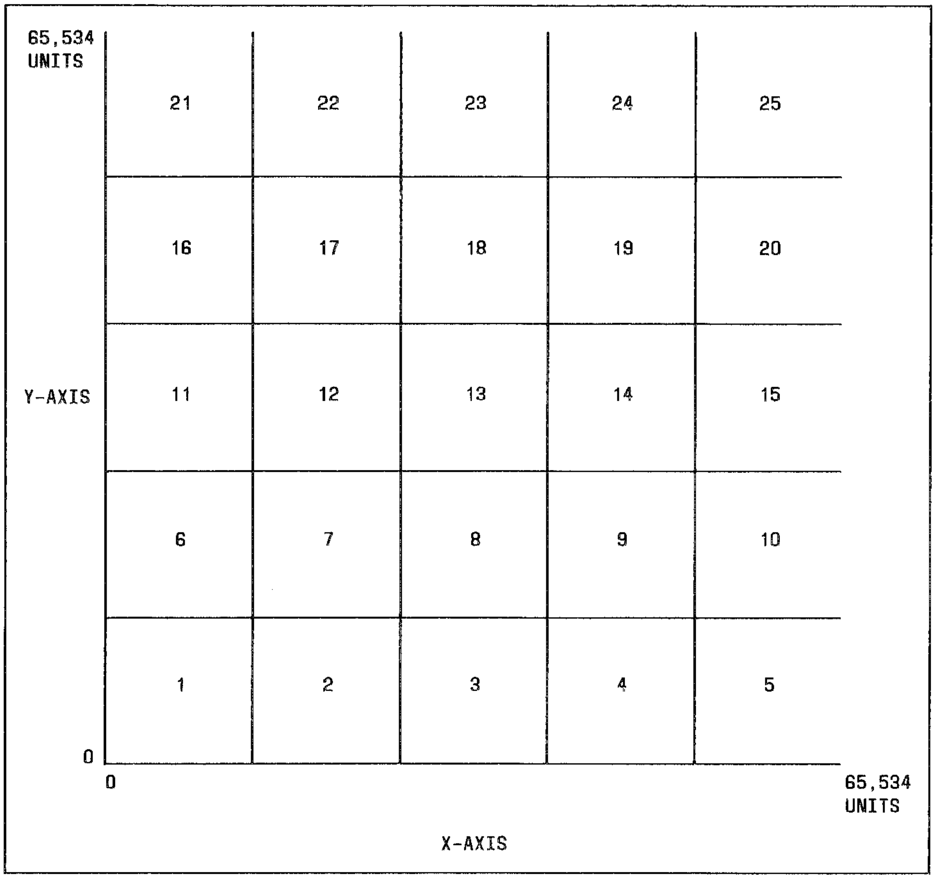

Figure 2-3. Cartesian plane

EXAMPLES OF WHAT YOU CAN DO

Numerical Manipulation and Plotting

Stat is a collection of numerical and plotting commands. All these commands operate on vectors. A vector is a text file that contains a sequence of numbers separated by a delimiter and where a delimiter is anything that is not a number.

For example:

1 2 3 4 5, and
hhh tty47 July 19 09:52

are both vectors. The vectors are:

First Vector = 1 2 3 4 5
Second Vector = 47 19 09 52

Here is an easy way to generate a Celsius-Fahrenheit conversion table using **gas** to generate the vector of Celsius values:

```
^gas –s0,t100,i10 | af " C,9/5*C+32" <CR>
```

The results are:

```
0               32
10              50
20              68
30              86
40              104
50              122
60              140
70              158
80              176
90              194
100             212
```

where:

- **gas –s0,t100,i10** generates a sequence that **s**tarts at 0, **t**erminates at 100, and the **i**ncrement between successive elements is 10.

- **af " C,9/5*C+32"** generates the table. Arguments to **af** are expressions. Operands in an expression are either constants or file names. If a file name is given that does not exist in the current directory, it is taken as the name for the standard input. In this example, **C** references the standard input.

Here is an example that illustrates the use of vector titles and makes a multi-line plot:

```
^gas | title –v" first ten integers" >N<CR>
^root N >RN<CR>
^root –r3 N >R3N<CR>
^root –r1.5 N >R1.5N<CR>
^plot –FN,g N R1.5N RN R3N | td<CR>
```

where:

- **title –v"** *name*" associates a *name* with a vector. Here, the **first ten integers** are associated with the vector output by **gas**. The vector is stored in file **N**.

- **root –r***n* outputs the *n*th root of each element on the input. If **–r***n* is not given, then the square root is output. Also, if the input is a titled vector, the title will be transformed to reflect the **root** function.

- **plot –F***X,g Y(s)* generates a multi-line plot with *Y(s)* plotted versus *X(s)*. The **g** option causes tick marks to appear instead of grid lines.

The results of the plot is shown in Figure 2-4.

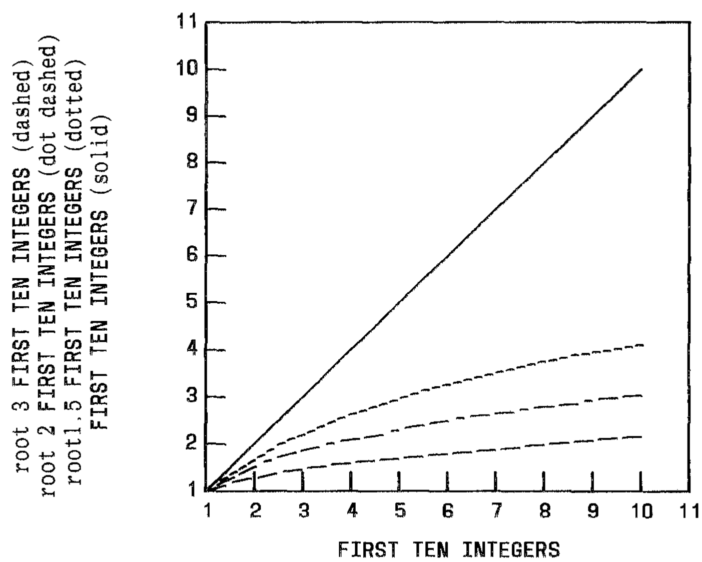

Figure 2-4. Some Roots of the First Ten Integers

The next example generates a histogram of random numbers:

```
^rand –n100 | title –v" 100 random numbers" | qsort |
    bucket | hist | td<CR>
```

where:

- **rand –n100** outputs random numbers using **rand**(3C). Here, 100 numbers are output in the range 0 to 1.
- **title -v" name"** associates a name with a vector. In this case, **100 random numbers** is associated with the vector output by gas.
- **qsort** sorts the elements of a vector in ascending order.
- **bucket** breaks the range of the elements in a vector into intervals and counts how many elements from the vector fall into each interval. The output is a vector with odd elements being the interval boundaries and even elements being the counts.
- **hist** builds a histogram based on interval boundaries and counts.
- **td** command displays drawings on TEKTRONIX 4014 display terminal.

The output is shown in Figure 2-5.

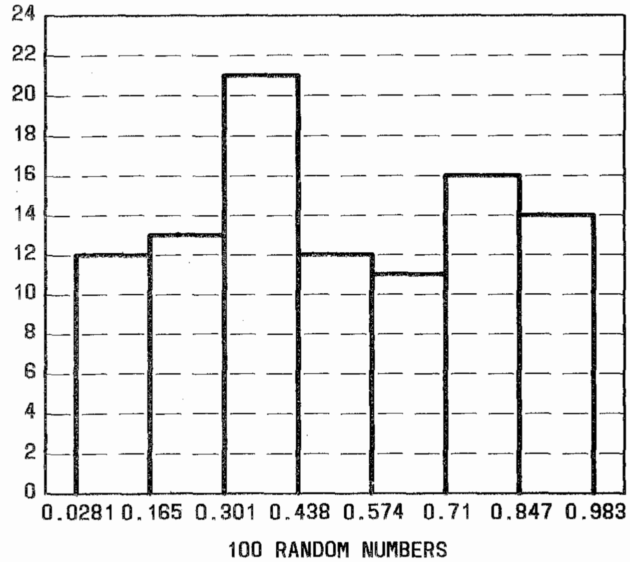

Figure 2-5. Histogram of 100 Random Numbers

Drawings Built From Boxes

There is a large class of drawings composed from boxes and text. Examples are structure charts, configuration drawings, and flow diagrams. In **graphics**, the general procedure to build such box drawings is the same as that for numerical plotting; namely, gather and transform the data, build and display the layout.

As an example, for hierarchical charts, the command line

```
^dtoc | vtoc | td<CR>
```

outputs drawings representing directory structures.

- The **dtoc** command outputs a table of contents that describes a directory structure (Figure 2-6). The fields from left to right are the level number, the directory name, and the number of ordinary readable files contained in the directory.
- The **vtoc** command reads a (textual) table of contents and outputs a visual table of contents, or hierarchical chart (Figure 2-7). Input to **vtoc** consists of a sequence of entries, each describing a box to be drawn. An entry consists of a level number, an optional style field, a text string to be placed in the box, and a mark field to appear above the top right-hand corner of the box.
- The **td** command displays the drawing on a TEKTRONIX terminal.

```
0.      "source"     2
1.      "glib.d"     1
1.1.    "gpl.d"      12
1.2.    "gsl.d"      14
2.      "gutil.d"    6
2.1.    "cvrtopt.d"  7
2.2.    "gtop.d"     8
2.3.    "ptog.d"     5
3.      "stat.d"     54
4.      "tek4000.d"  5
4.1     "ged.d"      37
4.4.    "td.d"       8
5.      "toc.d"      3
5.1.    "ttoc.d"     3
5.2.    "vtoc.d"     22
6.      "whatis.d"   108
```

Figure 2-6. Output of dtoc Command

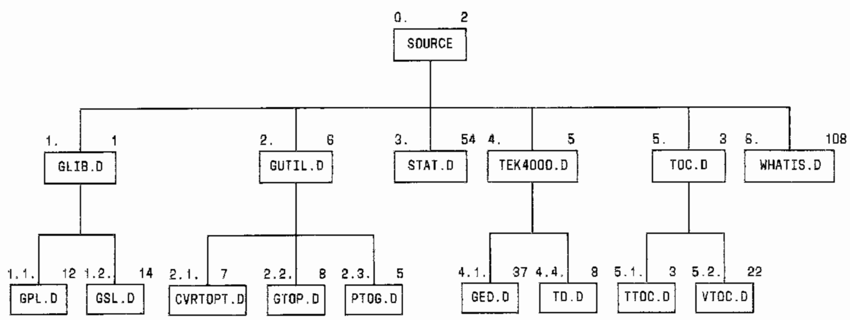

Figure 2-7. Output of vtoc Command

Chapter 3

STAT - A TOOL FOR ANALYZING DATA

Chapter 3

STAT - A TOOL FOR ANALYZING DATA

INTRODUCTION

Stat is a collection of command level functions (nodes) that can be interconnected using the UNIX System shell to form processing networks. Included within **stat** are programs to generate simple statistics and pictorial output.

This chapter introduces **stat** concepts by using a few commands through a collection of examples. A complete definition of all **stat** commands with examples is discussed in Chapter 4.

Much of the power for manipulating text in the UNIX System comes from well-defined, text-processing programs such as the DOCUMENTER'S WORKBENCH* software text processing utilities. These processing programs can be readily interfaced to one another. The general interface

---

* Trademark of AT&T

is an unformatted text string, and the interconnection mechanism is usually the UNIX System shell. The programs are independent of one another, so new functions can easily be added and old ones changed. Because the text editor operates on unformatted text, arbitrary text manipulation can always be performed even when the more specialized routines are insufficient.

Stat uses the same mechanisms to bring similar power to the manipulation of numbers. It consists of a collection of numerical processing routines that read and write unformatted text strings. It includes programs to build graphical files that can be manipulated using a graphical editor. And since **stat** programs process unformatted text, they can readily be connected with other UNIX System command-level (that is, callable from shell) routines.

It is useful to think of the shell as a tool to build processing networks in the sense of data-flow programming. Command-level routines are the nodes of the network, and pipes and tees are the links. Data flows from node-to-node in the network via data links.

NODE DESCRIPTIONS

Stat nodes are divided into four classes. These classes are:

- Transformer
- Summarizer
- Translator
- Generator.

All these nodes accept the same command-line format:

- A *command* is a *command-name* followed by zero or more *arguments*.
- A *command-name* is the name of any **stat** node.
- An *argument* is a *file-name* or an *option-string*.
- An *option-string* is a — followed by one or more *options*.
- An *option* is one or more letters followed by an optional value. Options may be separated by commas.
- A *file-name* is any name not beginning with a –, or a – by itself (to reference the standard input).

Each file argument to a node is taken as input to one occurrence of the node. That is, the node is executed from its initial state once per file. If no files are given, the standard input is used. All nodes, except *generators*, accept files as input.

Vector(s)

All numerical data in **stat** are stored in text files. These text files are vectors, where a vector is a sequence of numbers separated by delimiter and a delimiter is anything that is not a number. These vectors are processed by command-level routines called nodes.

Transformer Node

A *transformer* is a node that reads an input element, operates on it, and outputs the resulting value. For example, suppose vector **A** contains

```
1 2 3 4 5
```

then the command:

```
^root A<CR>
```

produces the square root of each input elements.

```
1      1.41421      1.73205      2      2.23607
```

Af, for arithmetic function, is a particularly versatile *transformer*. Its argument is an expression that is evaluated once for each complete set of input values. A simple example is:

```
^af " 2*A^2" <CR>
```

that produces

```
2          8          18          32          50
```

twice the square of each element from **A**. Expression arguments to **af** are surrounded by quotes since some of the operator symbols have special meaning to the shell.

The following is a list of all *transformer* commands that are discussed in detail in Chapter 4:

- **abs**, absolute value
- **af**, arithmetic function
- **ceil**, ceiling function
- **cusum**, cumulative sum function
- **exp**, exponential function
- **floor**, floor function
- **gamma**, gamma function
- **list**, list vector
- **log**, logarithm function
- **mod**, modulus function
- **pair**, pair element group
- **power**, power function
- **root**, root function
- **round**, rounded value
- **siline**, generate a line given slope and intercept
- **sin**, sin function
- **subset**, generate a subset.

Parameters

Most nodes accept parameters to direct their operation. Parameters are specified as command-line options. **Root**, for example, is more general than just square root, any root may be specified using the **r** option. For example:

```
^root -r3 A<CR>
```

produces

```
1      1.25992      1.44225      1.5874      1.70998
```

the cube root of each element from **A**.

Summarizers Node

A *summarizer* is a node that calculates a statistic for a vector. Typically, *summarizers* read in all the input values; then, calculates and outputs the statistic. For example, using the vector **A** from the previous example,

A = 1 2 3 4 5

```
^mean A<CR>
```

produces

```
3
```

The following is a list of all *summarizer* commands that are discussed in detail in Chapter 4:

- **bucket**, generates buckets and counts
- **cor**, ordinary correlation coefficient
- **hilo**, finds high and low values
- **lreg**, linear regression
- **mean**, mean function
- **point**, empirical cumulative density function point
- **prod**, product function
- **qsort**, quick sort
- **rank**, rank of vectors
- **total**, sum total
- **var**, variance function.

Building Networks

Nodes are interconnected using the standard UNIX System shell concepts and syntax. A pipe is a linear connector that attaches the output of one node to the input of another. As an example, to find the mean of the cube roots of vector **A** is:

```
^root –r3 A | mean<CR>
```

that produces

```
1.39991
```

Often the required network is not so simple. Tees and sequence can be used to build nonlinear networks. The tee is a pipe fitting that transcribes the standard input to the standard output and makes a copy in a file. To find the mean and median of the transformed vector **A** is:

```
^root –r3 A | tee B | mean; total B<CR>
```

that produces

```
1.39991
6.99955
```

Beware of the distinction between the sequence operator (;), and the linear connector, the pipe (¦). The pipe (¦) takes the output from one command and inputs it to the other command. Each command is run as a separate process; the shell waits for the last command to end. The sequence operator semicoln (;) allows you to put more than one command on a command line. The output is not directed unless otherwise specified.

Command Substitution

There is a special case of nonlinear networks where the result of one node is used as command-line input for another. Command substitution makes this easy. For example, to generate residuals from the mean of **A** is simply

A = 1 2 3 4 5
mean A = 3

```
^af " A–■nean A■ <CR>
```

that results in

```
–2      –1      0      1      2
```

This example shows that command substitution does the operation in grave accents (") first, then substitutes that value for the expression in the grave accents. Here it takes the mean of **A** that is 3. It then substitutes the value 3 for **mean A** in the grave accents. Then, the arithmetic expression in quotation marks is completed.

Generator Node

Thus far, vectors have been used but not created. One way to create a vector is by using a *generator*. A *generator* is a node that accepts no input, and outputs a vector based on definable parameters. **Gas** is a *generator* that produces additive sequences. One parameter to **gas** is the **n**umber of elements in the generated vector. As an example, to create the vector **A** that we have been using is:

```
^gas -n5<CR>
```

that produces

```
1        2        3        4        5
```

To name the vector **A**, you can direct the output to **A**.

```
^gas -n5 > A<CR>
```

Vectors are, however, merely text files. Hence, the text editor can be used to create and change the same vector.

```
$vi A<CR>
<a> 1<CR>
2<CR>
3<CR>
4<CR>
5<CR>
<ESC>
<ZZ>
```

A useful property of vectors is that they consist of a sequence of numbers surrounded by delimiters, where a delimiter is anything that is not a number. Numbers are constructed in the usual way

[sign](digits)(.digits)[**e**[sign]digits]

where fields are surrounded by brackets and parentheses. All fields are optional, but at least one field surrounded by parentheses must be present.

An example of entering the number $2.7x10^6$ in a text file **T** would be:

```
^vi T<CR>
<a> +2.7e+06<CR>
<ESC>
<ZZ>
```

Thus, vector **B** could also be created by building the file **B** in the text editor as

```
$vi B<CR>
<a> 1partridge,2tdoves,3frhens,4cbirds,5gldnrings<CR>
<ESC>
<ZZ>
```

Note: Remember that a vector is separated by a delimiter. A delimiter is anything that is not a number.

that, when read by

```
^list B<CR>
```

produces

```
1        2        3        4        5
```

The following is a list of all *generator* commands that are discussed in detail in Chapter 4.

- **gas**, generate additive sequence
- **prime**, generate prime numbers

- **rand**, generate random sequence.

A Simple Example: Interacting with a Data Base

When used with the UNIX System tools for manipulating text, **stat** provides an effective means for exploring a numerical data base. Suppose, for example, there is a subdirectory called **data** containing data files that include the lines:

path length = *nn*   (*nn* is any number)
node count = *nn*

To access the value for **node count** from each file, sort the values into ascending order, store the resulting vector in file **C**, and get a copy on the terminal by typing

```
^grep " node count" data/* | qsort | tee C<CR>
17    19    22    32    39
50    68    78    125   139
```

The slash (/) in the above example was used because we were in our home directory when this command was entered. This will scan all files in the subdirectory **data**.

If some of the data files have numbers in their name, we must protect those numbers from being considered data. Using **cat**, this is easy:

```
^cat data/* | grep " node count" | qsort | tee C<CR>
```

To get a feel for the distribution of node counts, shell iteration can be used to an advantage. In this example, we will generate the lower hinge, the median, and the upper hinge of the sorted vector **A**.

```
for i in .25 .5 .75<CR>
do point -p$i A<CR>
done<CR>
24.5
44.5
75.5
```

Translators Node

Translators are used to view data pictorially. A *translator* is a node that produces a stream of a different structure from what it consumes. Graphical *translators* consume vectors and produce pictures in a language called GPS, for graphical primitive string. A GPS is a format for storing a picture. A picture is defined in a Cartesian plane of 64K points on each axis. The plane, or universe, is divided into 25 square regions numbered 1 to 25 from the lower left to the upper right (see Figure 2-3.) Various commands exist that can display and edit a GPS.

The following is a list of all *translator* commands that are discussed in detail in Chapter 4.

- **bar**, build a bar chart
- **hist**, build a histogram
- **pie**, build a pie chart
- **plot**, plots a x-y plot.

For example:

Hist is a *translator* that produces a GPS that describes a histogram from input consisting of interval limits and counts. The *summarizer* **bucket** produces limits and counts, thus:

```
^bucket A | hist | tdCR>
```

generates a histogram of the data of vector **A** and displays it on a display terminal (Figure 3-1). **Td** translates the GPS into machine code for TEKTRONIX 4014 display terminals.

A wide range of X-Y plots can be constructed using the *translator* **plot**. For example, to build a scatter plot of **path length** with **node count** (Figure 3-2) is:

```
^grep " path length"  data/* | title -v" path length"  >A<CR>
^grep " node count"  data/* | title -v" node count"
    | plot -FA,dg | td<CR>
```

A vector may be given a title using **title**. When a titled vector is plotted, the appropriate axis is labeled with the vector title. The **plot -FA,dg** uses the **A** vector for the x-axis and standard input for the y-axis.

When a titled vector is passed through a *transformer*, the title is altered to reflect the transformation. Thus, in a graph of log **node count** versus the cube root of **path length**, such as

```
^grep " node count" | title -v" node count" | log >B<CR>
^root -r3 A | plot -F-,dg B | td<CR>
```

the axis labels automatically agree with the vectors plotted (Figure 3-3). The **plot -F-,dg B** uses the **B** vector as the y-axis and standard input for the x-axis.

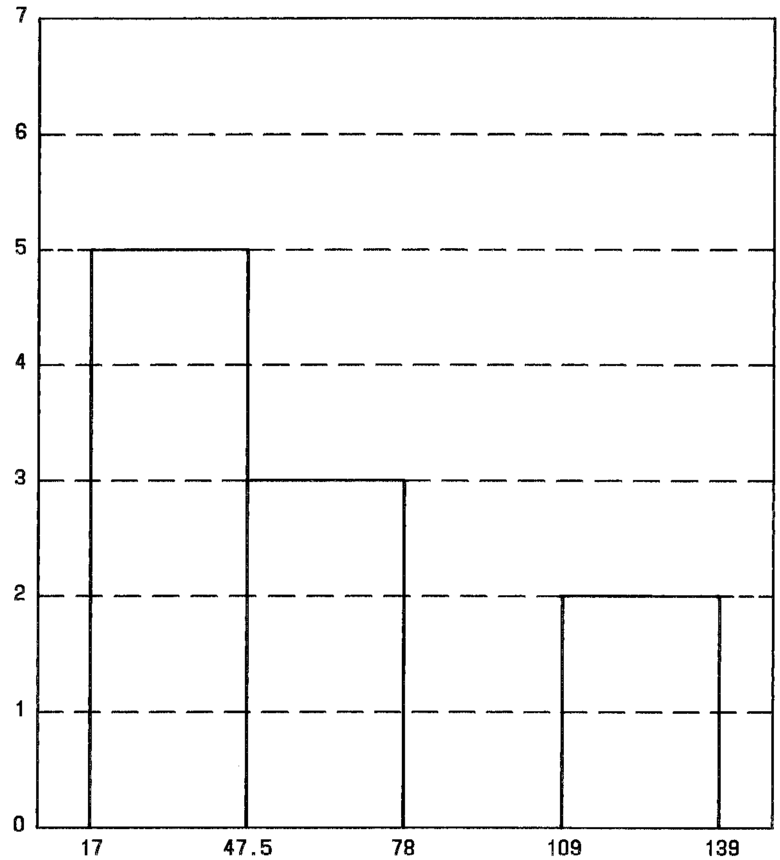

Figure 3-1. bucket A | hist | td

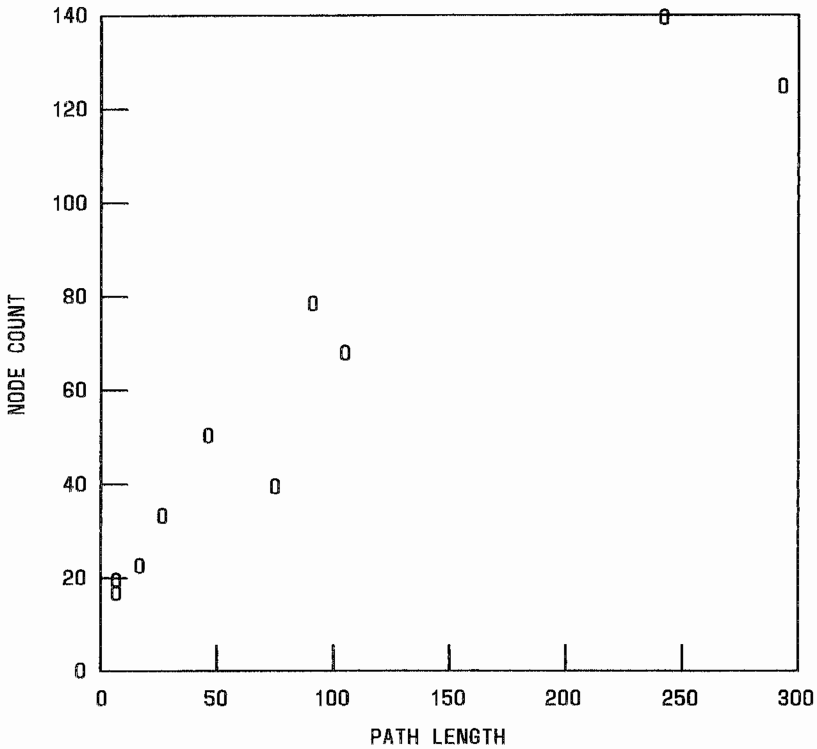

Figure 3-2. Scatter Plot

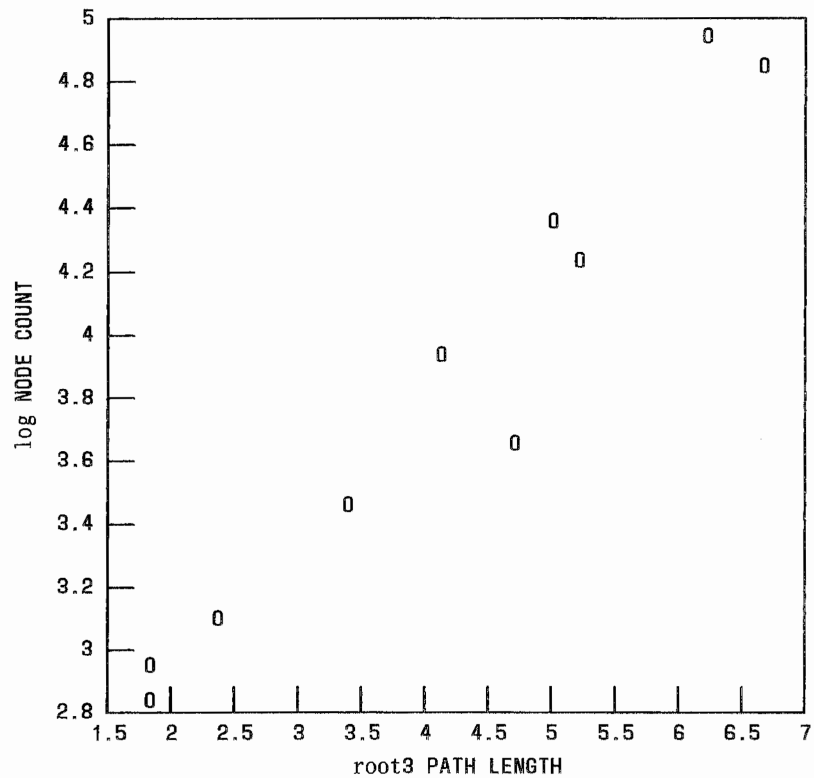

Figure 3-3. Transformed Scatter Plot

EXAMPLES

Example 1

Calculate the total value of an investment held for many years at an interest rate compounded annually.

SOLUTION

```
^Principal=1000<CR>
^echo Total return on $Principal units compounded annually<CR>
^echo " rates:\t\t\c" ; gas -s.05,t.15,i.03 | tee rate<CR>
^for Years in 1 3 5 8<CR>
^do echo " $Years year(s):\t\c" ; af " $Principal*(1+rate)^$Years" <CR>
^done<CR>

Total return on 1000 units compounded annually rates:
          0.05       0.08       0.11       0.14

1 year(s):  1050       1080       1110       1140
3 year(s):  1157.62    1259.71    1367.63    1481.54
5 year(s):  1276.28    1469.33    1685.06    1925.41
8 year(s):  1477.46    1850.93    2304.54    2852.59
```

Note:

Notice the distinction between vectors and constants as operands in the expression to **af**. The shell variables *$Principal* and *$Years* are constants to **af**, while the file *rate* is a vector. **Af** executes the expression once per element in *rate*.

Example 2

PROBLEM

Three ordered vectors (**A**, **B**, and **C**) of scores from many tests are given. Each vector is from one test-taker, each element in a vector is the score on one test. There are missing scores in each vector shown by the value −1. Generate three new vectors containing scores only for those tests where no data is missing.

SOLUTION

```
^echo Before:<CR>
^gas –n■ank A■| tee N | af " label,A,B,C" <CR>

^for i in N B C A<CR>
^do subset –FA,l–1 $i >s$i; done<CR>
^for i in N A C B<CR>
^do subset –FsB,l–1 s$i | yoo s$i; done<CR>
^for i in N A B C<CR>
^do subset –FsC,l–1 s$i | yoo s$i; done<CR>
^echo " \nAfter:" <CR>
^af " sN,sA,sB,sC" <CR>

Before:
1       5       6       –1
2       7       10      10
3       –1      10      9
4       10      –1      8
5       6       5       –1
6       5       7       5
7       –1      7       8
8       –1      –1      8
9       3       –1      8
10      6       10      10
11      7       5       7

After:
2       7       10      10
6       5       7       5
10      6       10      10
11      7       5       7
```

Notes:

1. The approach is to eliminate those elements in all vectors that correspond to –1 in the base vector. Each of the three vectors takes a turn at being the base. It is important to subset the base last. The command **yoo** (see **gutil** in the *AT&T 3B2 Computer User Reference Manual*) takes the output of a pipeline and copies it into a file used in the pipeline. This cannot be done by redirecting the output of the pipeline as this would cause a concurrent read and write on the same file.

2. The printing of the "Before" matrix illustrates a useful property of **af**. The first name in an expression that does not match any name in the present working directory is a reference to the standard input. In this example, **label** references the input coming through the pipe.

Example 3

PROBLEM

Generate a bar chart of the percent of execution time consumed by each routine in a program.

SOLUTION

```
^prof | cut -c1-15 | sed -e 1d -e " / 0.0/d" -e "s/^ *//" >P<CR>
^echo These are the execution percentages; cat P<CR>
^title P -v" execution time in percent" | bar -xa -yl0,
  yh100 | label -br-45,FP | td<CR>

These are the execution percentage:

_fork  32.9
_creat  14.3
_sbrk  14.3
_read  14.3
_open  14.3
_prime  9.9
```

Note:

Prof is a UNIX System command that generates a listing of execution times for a program (see **prof**(1)). **Cut** and **sed** are used to eliminate extraneous text from the output of **prof**. (Because verbiage can get in the way, **stat** nodes say little.) Notice that **P** is a vector to **title**, while it is a text file to **cat** and **label**.

Figure 3-4 shows the output of these commands.

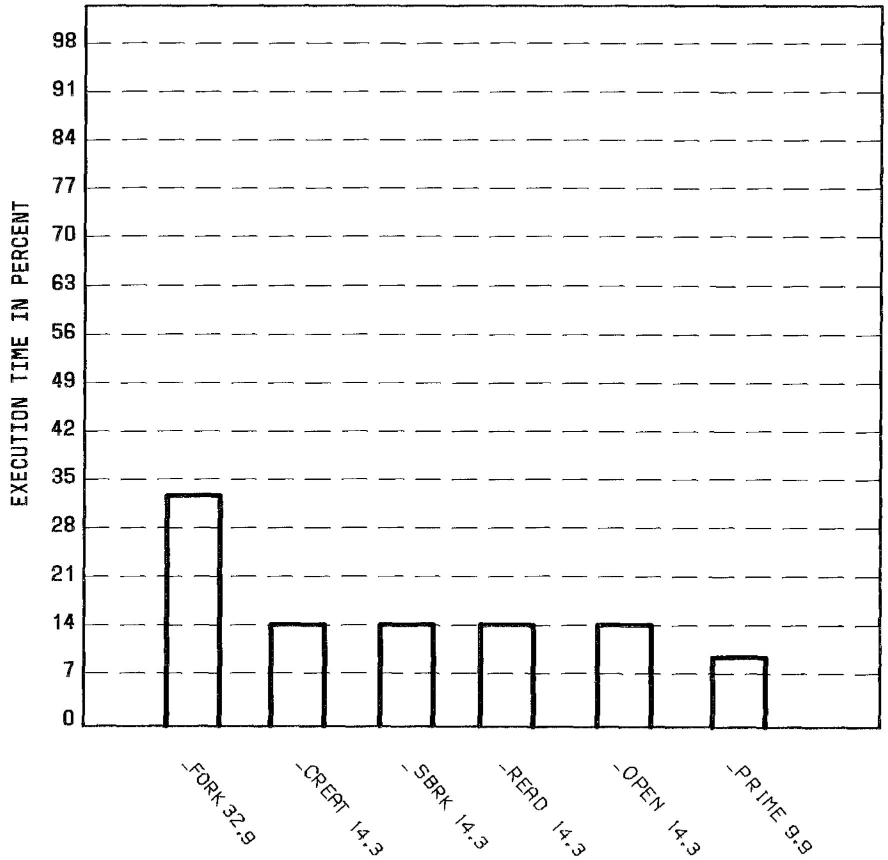

Figure 3-4. Bar Chart Showing Execution Profile

Example 4

PROBLEM

Plot the relationship between the execution time of a program and the number of processes in the process table.

SOLUTION

The first program generates the performance data

```
^for i in [redacted]gas –n12[redacted]<CR>
^do<CR>
^ps –ae | wc –l >>Procs&<CR>
 time prime –n1000 >/dev/null 2>>Times<CR>
 sleep 300<CR>
^done<CR>
```

The second program analyzes and plots the data

```
^for i in real user sys<CR>
^do<CR>
^grep $i Times | sed " s/$i//" |
    awk -F: " { if(NF==2) print \$1*60+\$2; else
   print }" | title -v" $i time in seconds" >$i<CR>
^siline -[illegible]reg -o,FProcs $i [illegible]Procs >$i.fit<CR>
^done<CR>
^title -v" number of processes" Procs | yoo Procs<CR>
^plot -dg,FProcs real -r12 >R12<CR>
^plot -ag,FProcs real.fit -r12 >>R12<CR>
^plot -dg,FProcs sys -r13 >R13<CR>
^plot -ag,FProcs sys.fit -r13 >>R13<CR>
^plot -dg,FProcs user -r8 >R8<CR>
^plot -ag,FProcs user.fit -r8 >>R8<CR>
^ged R12 R13 R8<CR>
```

Notes:

1. The performance data is the execution time, as reported by the UNIX System **time** command, to generate the first 1000 prime numbers. **Times** outputs three times for each run:

   - The time in system routines
   - The time in user routines
   - Total real time.

2. The output of the **time** command is saved in the file **Times**. Each of these types of time is treated separately by the analysis program.

3. In the file **Procs** are the number of processes running on the system during each execution of **prime**. The short **awk** program converts "minutes:seconds" format to "seconds." **Lreg** does a linear regression of the time vectors on the size of the process table. **Siline** generates a line based on the parameters from the regression. One plot is generated for each type of time. Each plot is put into a different region so that they can be displayed and manipulated simultaneously in **ged**.

4. Figure 3-5 shows the output of these commands.

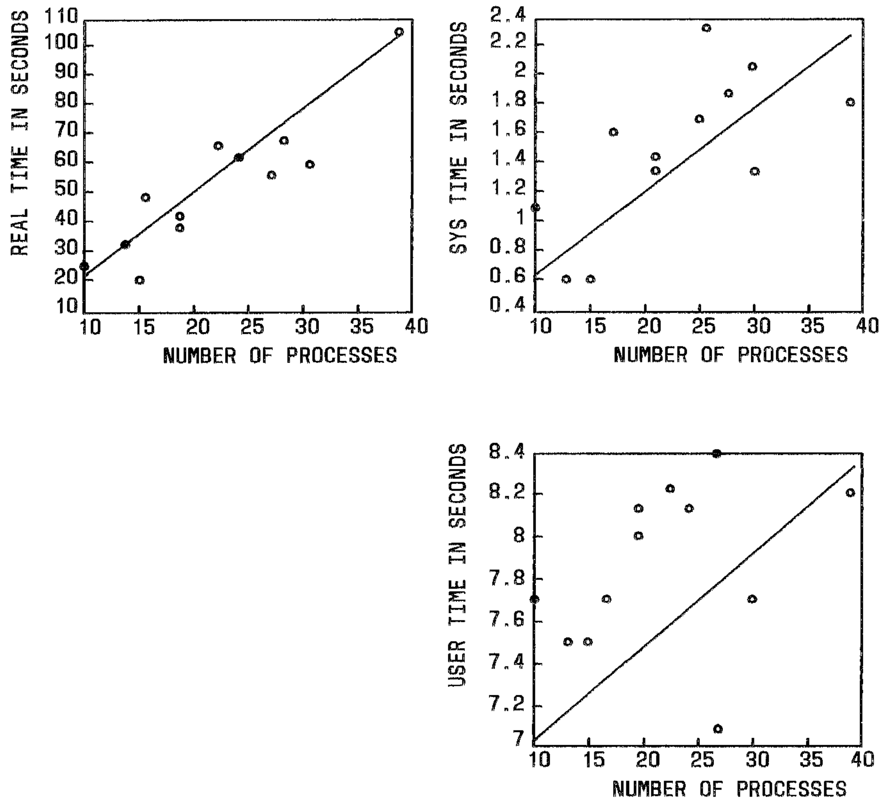

Figure 3-5. Relationship Between Execution Time and Number of Processes

Chapter 4

COMMAND DESCRIPTIONS

Chapter 4

COMMAND DESCRIPTIONS

COMMAND SUMMARY

The Graphics Utilities provide 60 UNIX System commands. A summary of these commands are provided in Figure 4-1.

COMMAND	DESCRIPTION
abs	**Absolute value**, its output is the absolute value for each element of the input vector(s).
af	**Arithmetic function**, its argument is an expression that is evaluated once for each complete set of input values.

Figure 4-1. Graphics Utilities—COMMAND SUMMARY (Sheet 1 of 8)

COMMAND	DESCRIPTION
bar	**Bar chart**, its output is a GPS that describes a bar chart display.
bel	**Bel**, causes most terminals to sound an audible tone, a useful nonvisual signal.
bucket	**Bucket**, breaks the range of a vector into intervals and counts how many elements from the vector fall into each interval. The output is a vector with odd elements being the interval boundaries and even elements being the counts.
ceil	**Ceiling function**, its output is a vector with each element being the smallest integer greater than the corresponding element from the input vector(s) (rounds up to the next integer).
cor	**Ordinary correlation coefficient**, its output is the ordinary correlation coefficient between a base vector and another vector.
cusum	**Cumulative sum**, its output is a vector that calculates the sum of all the elements found in the input vector.

Figure 4-1. Graphics Utilities—COMMAND SUMMARY (Sheet 2 of 8)

COMMAND	DESCRIPTION
cvrtopt	**Cvrtopt**, it reformats the arguments (usually the command line arguments of a calling shell procedure) to improve processing by shell procedures.
dtoc	**Directory table of contents**, its output is a list of all readable subdirectories beginning at a specified directory or the current directory.
erase	**Erases the screen of the TEKTRONIX 4014 display terminal.**
exp	**Exponential function**, its output is a vector with elements e raised to the *x* power, where e is about 2.71828, and *x* are the elements from the input vector(s).
floor	**Floor function**, its output is a vector with each element being the largest integer less than the corresponding element from the input vector(s) (rounds down to next integer).
gamma	**Gamma function**, its output is the gamma value for each element of the input vector(s).
gas	**Generate additive sequence**, its output is a vector of number elements determined by the parameters **s** (start), **t** (terminate), and **i** (interval).
gd	**Gd**, prints a human readable listing of GPS.

Figure 4-1. Graphics Utilities—COMMAND SUMMARY (Sheet 3 of 8)

COMMAND	DESCRIPTION
ged	**Graphical editor**, allows displaying and editing of GPS.
graph	**Graph**, it takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Then, it draws a straight line connecting successive points.
graphics	**Graphics**, puts the 3B2 Computer into the graphics mode.
gtop	**Gtop**, it transforms a GPS into plot(5) format displayable by the **tplot** command.
hardcopy	When issued from a TEKTRONIX 4014 display terminal with a hard copy unit (printer), it generates a screen copy on the unit.
hilo	**Hilo**, its output is the high and low values found across all the input vector(s).
hist	**Hist**, its output is a GPS that describes a histogram display.
hpd	Display GPS on a HP 7221A Graphics Plotter, its output is scope coded for a HP 7221A Plotter.
label	**Label**, it appends the axis labels from a label file to a GPS of a data plot (like that produced by hist, bar, and plot).
list	**List**, its output is a listing of the elements of the input vector(s).

Figure 4-1. Graphics Utilities—COMMAND SUMMARY (Sheet 4 of 8)

COMMAND	DESCRIPTION
log	**Logarithm**, its output is the logarithm for each element of the input vector(s).
lreg	**Linear regression**, its output is the slope and intercept from the least squares linear regression of each vector on a base vector.
mean	**Mean**, its output is the mean of the elements in the input vector(s).
mod	**Modulo function**, its output is a vector with each element being the remainder of dividing the corresponding element from the input vector(s).
pair	**Pair element groups**, its output is a vector with elements taken alternately from a base vector and from a vector.
pd	**Pd**, prints a human readable listing of plot(5) format.
pie	**Pie**, its output is a GPS that describes a pie chart.
plot	**Plot**, its output is a GPS that describes an x-y graph.
point	**Empirical cumulative density function point**, its output is a linearly interpolated value from the empirical cumulative density function (e.c.d.f) for the input vector.

Figure 4-1. Graphics Utilities—COMMAND SUMMARY (Sheet 5 of 8)

COMMAND	DESCRIPTION
power	**Power function**, its output is a vector with each element being a power of the corresponding element from the input vector(s).
prime	**Generate prime numbers**, its output is a vector of number elements determined by the parameters low and high.
prod	**Product**, its output is the product of the elements in the input vector(s).
ptog	**Ptog**, transforms plot(5) format into a GPS.
qsort	**Quick sort**, its output is a vector of the elements from the input vector in ascending order.
quit	**Quit**, ends the current terminal session.
rand	**Generate random sequence**, its output is a vector of number elements determined by the parameters low, high, multiplier, and seed.
rank	**Rank**, its output is the number of elements in each input vector.
remcom	**Remove comments**, the input is copied to its output, with the comments removed.

Figure 4-1. Graphics Utilities—COMMAND SUMMARY (Sheet 6 of 8)

COMMAND	DESCRIPTION
root	**Root function**, its output is a vector with each element being the root of the corresponding element from the input vector(s).
round	**Rounded value**, its output is the rounded value for each element of the input vector(s).
siline	**Siline**, it generates a line given slope and intercept.
sin	**Sine**, its output is the sine for each element of the input vector(s).
spline	**Interpolate smooth curve**, it takes pairs of numbers from the standard input as abscissas and ordinates of a function.
subset	**Generates a subset**, its output is elements selected from the input based on a key and option(s).
td	**Td**, it displays a GPS on a TEKTRONIX 4014, its output is scope coded for a TEKTRONIX 4014 terminal.
tekset	**Tekset**, clears the display screen, sets the display mode to alpha, and the characters to the smallest font.

Figure 4-1. Graphics Utilities—COMMAND SUMMARY (Sheet 7 of 8)

COMMAND	DESCRIPTION
title	**Title**, it appends a title to a vector or it appends a title to a GPS.
total	**Total**, its output is the sum total of the elements in the input vector(s).
tplot	**Tplot**, it reads plotting instructions from the standard input for a particular terminal.
ttoc	**Ttoc**, its output is the textual table of contents generated by the .H macro of the nroff or troff raw data of the Document Workbench Utilities.
var	**Var**, it finds the difference between the slope point and the outer point.
vtoc	**Visual table of contents**, its output is a GPS that describes a Visual table of contents (vtoc or hierarchical chart) of the Textual Table of Contents (TTOC) entries from the input.
whatis	Brief online documentation, prints a brief description of each command given. If no command is given, then the current list of description commands are printed.
yoo	**Yoo**, is a piping primitive that deposits the output of a pipeline into a file used in the pipeline.

Figure 4-1. Graphics Utilities—COMMAND SUMMARY (Sheet 8 of 8)

COMMAND DESCRIPTIONS

abs — Absolute Value

General

The absolute value (**abs**) is a transformer node that is used to find the absolute value of each element of the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

abs [-option] [vector(s)]

Option:

c*n*	*n* is the number of output elements per line. By default, c = 5.

Command Example

The following example will output the absolute value of each element of the vector **A**, two per line.

A = 23 -44 55 -66 70

```
^abs -c2 A<CR>

23  44
55  66
70
```

af — Arithmetic Function

General

The arithmetic function (**af**) is a particularly versatile transformer node. Its argument is an expression that is evaluated once for each complete set of input values. The input values comes from vectors specified by an expression. The expression consists of an operand and an operator.

An operand is either a vector, function, or constant.

Expression operands are:

Vectors:	Filenames with the restriction that they must begin with a letter and be composed only from letters, digits, '_', and '.'. The first unknown filename (one not in the current directory) references the standard input. A warning will appear if a file cannot be read.
Functions:	The name of a command, followed by the command arguments in parentheses. Arguments are written in command-line format.
Constants:	Floating point or integer (but not E notation) number.

The operators are listed below in order of their decreasing precedence.

The x_i (y_i) represents the start element from X (Y) for the expression.

- *'Y* – reference y_{i+1}. y_{i+1} is consumed; the next value from *Y* is y_{i+2}. *Y* is a vector.
- *X^–Y* – x_i raised to the y_i power, negation of y_i. Association is left to right. *X* and *Y* are expressions.

- $X*Y \quad X/Y \quad X\%Y$ – x_i multiplied by, divided by, modulo y_i. Association is left to right. *X* and *Y* are expressions.

- $X+Y \quad X-Y$ – x_i plus, minus y_i. Association is left to right. *X* and *Y* are expressions.

- X,Y – yields x_i, y_i. Association is left to right. *X* and *Y* are expressions.

Note: Parentheses may be used to alter precedence. Because many of the operator characters are special to the shell, it is good practice to surround *expressions* in quotes.

Command Format

af [-option(s)] *expression(s)*

Options:

c*n*	*n* elements per line in the output
t	output is titled from the vector on the standard input
v	verbose mode, function expansions are echoed.

Command Example

The following example will solve the expression $y=3(A)^2$ for each element of vector **A**, two per line.

A = 1 2 3 4 5

```
^af -c2 " 3*A^2" <CR>

3   12
27  48
75
```

bar — Build a Bar Chart

General

The **bar** command is a translator node thats output is a GPS that describes a bar chart. The input is a vector of counts that describes the y-axis. By default, x-axis will be labeled with integers beginning at 1; for other labels, see label. If no vector is given, then the standard input is assumed.

Command Format

bar [-option(s)] [vector(s)]

Options:

a	Suppress axis
b	Plot bar chart with bold weight lines; otherwise, use medium.
f	Do not build a frame around plot area.
g	Suppress background grid.
r*n*	Put the bar chart in GPS region *n*, where *n* is between 1 and 25, inclusively. The default is 13.
w*n*	*n* is the ratio of the bar width to center-to-center spacing expressed as a percentage. Default is 50, giving equal bar width and bar space.
x*n* (**y***n*)	Position the bar chart in the GPS universe with x-origin (y-origin) at *n*.

xa (ya)	Do not label x-axis (y-axis).
yl*n*	*n* is the y-axis low-tick value.
yh*n*	*n* is the y-axis high-tick value.

Command Example

The following example will show how to create a bar chart of the vector **C**.

C = 1 2 3 6 7 8 9

```
^bar C | td<CR>
```

where

- **bar C** output is a GPS describing a bar chart.
- **td** command displays the drawing on a TEKTRONIX 4014 display terminal.

See Figure 4-2 for a result of the drawing.

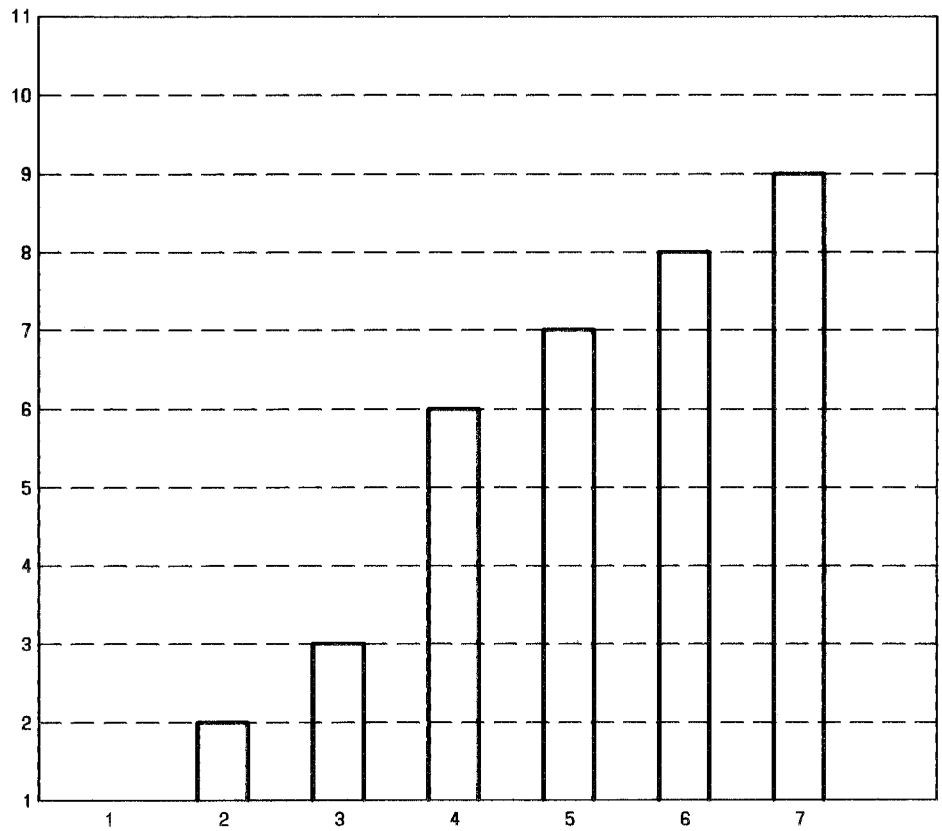

Figure 4-2. Plot of bar C ¦ td

bel — Bell Character

General

The **bel** command causes most terminals to sound an audible tone, which is a useful nonvisual signal.

bucket — Generate Buckets and Counts

General

The **bucket** command is a summarizer node that breaks the range of a vector into intervals and counts. The output is a vector with odd values being bucket limits (in parentheses) and even values being the number of elements from the input within the limits. Input is assumed to be sorted. If no input vector(s) are given, the standard input is assumed.

Command Format

bucket [-option(s)] [sorted vector(s)]

Options:

a*n*	Choose limits such that *n* is the average count per bucket.
c*n*	*n* elements per line in the output.
F*vector*	Take limit values from *vector*.
h*n*	*n* is the highest limit.
i*n*	*n* is the interval between limits.
l*n*	*n* is the lowest limit.
n*n*	*n* is the number of buckets.

Command Example

The following example will determine the intervals and counts of the vector **D**, by using the **bucket** command.

D = 10 20 30 40 50

```
^bucket D<CR>

(10) 2 (23.333) 1 (36.6667) 2 (50)
```

where

- The first bucket interval is between 10 and 23.3. The bucket count is 2, composed of elements 10 and 20.
- The second bucket interval is between 23.3 and 36.6. The bucket count is 1, composed of element 30.
- The third bucket interval is between 36.6 and 50. The bucket count is 2, composed of elements 40 and 50.

ceil — Ceiling Function

General

The ceiling function (**ceil**) is a transformer node. The output is a vector with each element being the smallest integer greater than or equal to the corresponding element from the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

ceil [-option] [vector(s)]

Options:

c*n*	*n* is the number of output elements per line.

Command Example

The following example will find the ceiling function of each element of the input vector **E**.

E = 10.5 -20.5 30 40.6 50

```
^ceil E<CR>

11 -20 30 41 50
```

cor — Ordinary Correlation Coefficient

General

The ordinary correlation coefficient (**cor**) is a summarizer node that determines if a base vector and another vector are related. The base vector is specified by using the **F** option. If the base or vector is not given, it is assumed to come from the standard input. Each vector is compared to the base. Both base and vector must be of the same length.

Command Format

```
cor [-option] [vector(s)]
```

Option:

F*vector* *vector* is the base.

Command Example

The following is an example of finding the correlation coefficient between the base vector **A** and another vector **B**.

A = 10 20 30 40 50
B = 20 30 40 50 60

```
^cor -FA B<CR>

1
```

where

- 0 = vector(s) are independent.
- 1 = vector(s) are related.
- -1 = vector(s) are inverse related.

Note: The ordinary correlation coefficient does not have to be only 0,1,-1. It could be .9, .3, ..etc.

where

.9	shows a strong relationship between vectors.
.3	shows a weak relationship between vectors.

cusum — Cumulative Sum

General

The cumulative sum (**cusum**) is a transformer node that calculates the running sum of all the elements found in the input vector. If no input vector is given, then the **cusum** implements a running accumulator. The data is then entered by the keyboard until an end-of-file command is given. On most terminals, the end-of-file command is control-d (<*CTRL-d*>).

Command Format

cusum [-option] [vector(s)]

Option:

c*n*	*n* is the number of output elements per line.

Command Example

The following example finds the cumulative sum of all the elements of vector **F**, two per line.

F = 20 30 40 50 60

```
^cusum -c2 F<CR>

20  50
90  140
200
```

The following example uses a running accumulator for the standard input in finding the cumulative sum of the values.

```
^cusum<CR>
20<CR>
20  30<CR>
50  40<CR>
90  50<CR>
140 60<CR>
200
<control-d>
```

cvrtopt — Options Converter

General

The **cvrtopt** command reformats arguments (usually the command line arguments of a calling shell procedure) to improve processing by shell procedures. An argument is either a filename (a string not beginning with a '-', or a '-' by itself) or an option string (a string of options beginning with a '-'). Output is of the form:

-option -option . . . filename(s)

All options appear singularly and preceding any filenames. Option names that take values (e.g., -r1.1) or are two letters long must be described through options to **cvrtopt**. Output is to the standard output.

Cvrtopt is usually used with set, in the following way, as the first line of a shell procedure:

set - 'cvrtopt [-option(s)] $@'

Set will reset the command argument string ($1,$2,...) to the output of **cvrtopt**. The minus option to set turns off all flags so that the options produced by **cvrtopt** are not interpreted as options to set.

Command Format

```
cvrtopt [-option(s)] arg(s)
```

s String accepts string values.

f String accepts floating point numbers as values.

i String accepts integers as values.

t String is a two letter option name that takes no value.

Note: String is a one or two letter option name.

Command Example

The following example shows how the **cvrtopt** command breaks up the option string (-lds).

```
^cvrtopt -lds<CR>
-l -d -s
```

dtoc — Directory Table of Contents

General

The **dtoc** command outputs a table of contents that describes a directory structure. It lists all readable subdirectories beginning at directory. If no directory is given, the list begins at the current directory. The output is as a textual table of contents (TTOC) readable by **vtoc**. The number of nondirectory files in each directory is shown in the marked field of the table of contents. The fields from left to right are level number, directory name, and the number of ordinary readable files contained in the directory.

Command Format

dtoc [directory]

Command Example

The following is an example of using the **dtoc** command to describe the directory **mtp**.

```
^dtoc<CR>

0. "mtp"  13
```

where

- **0.** is the level number.
- **mtp** is the directory name.
- **13** is the number of files in that directory.

erase — Erase Character

General

The **erase** command erases the screen of the TEKTRONIX 4014 display terminal.

exp — Exponential Function

General

The exponential function (**exp**) is a transformer node. The output is a vector with elements e raised to the _x power, where e is about 2.71828, and _x are the elements from the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

```
exp [-option] [vector(s)]
```

Option:

c*n*	*n* is the number of output elements per line.

Command Example

The following is an example of finding the exponential function of each element of vector **G**, the values are printed out two per line.

G = 10 20 30 40 50

```
^exp -c2 G<CR>

22026.5      4.85165e+08
1.06865e+13  2.35385e+17
5.18471e+21
```

floor — Floor Function

General

The floor function (**floor**) is a transformer node. The output is a vector with each element being the largest integer less than the corresponding element from the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

floor [-option] [vector(s)]

Option:

c*n*	*n* is the number of output elements per line.

Command Example

The following example will find the floor function of each element of the input vector **E**.

E = 10.5 -20.5 30 40.6 50

^**floor E**<*CR*>

10 -21 30 40 50

gamma — Gamma Function

General

The gamma function (**gamma**) is a transformer node. The output is the gamma value for each element of the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

```
gamma [-option] [vector(s)]
```

Option:

c*n*	*n* is the number of output elements per line.

Command Example

The following example will find the gamma function of each element of the input vector **A**. The output will be placed three per line.

A = 10 20 30 40 50

```
^gamma -c3 A<CR>

367880      1.21645e+17      8.84176e+30
2.o3979e+46 6.08282e+62
```

gas — Generate Additive Sequence

General

The **Gas** command is a generator node that produces additive sequences. A generator is a node that accepts no input, and outputs a vector based on definable parameters. These parameters are **s** (start), **t** (terminate), and **i** (interval). These parameters are set by command options.

Command Format

```
gas [-option(s)]
```

Options:

c*n*	*n* elements per output line.
i*n*	*n* defines interval. If not given, interval = 1.
n*n*	*n* = number. If not given, number = 10, unless terminate is given, then number = (terminate - start) divided by the interval.
s*n*	*n* = start. If not given, start = 1.
t*n*	*n* = terminate. If not given, terminate = positive infinity. The default value of number usually terminates generation before positive infinity is reached.

Command Example

The following example will generate a vector that has values starting at 0, incremented by 10, and terminated at 100.

```
^gas -s0,t100,i10<CR>

10 20 30 40 50 60 70 80 90 100
```

gd — GPS Dump

General

The **Gd** command prints a human readable listing of GPS. If no file is given, the standard input is used.

Command Format

gd [GPS file(s)]

Command Example

The following is an example of creating a GPS of the expression $y=x^2$ and printing the GPS in readable form.

```
^gas | af "x^2" | plot >A<CR>
^gd A<CR>

comment   37777700002   37777771037   37777771037
lines     col0 wt1 st0   -3553 -3553   5554 -3553
text      col 0 fon 16 tsz200 tro 0 -3553 -3833 0
text      col 0 fon 16 tsz200 tro 0 -2642 -3833 1
text      col 0 fon 16 tsz200 tro 0 -1732 -3833 2
text      col 0 fon 16 tsz200 tro 0 -821 -3833 3
text      col 0 fon 16 tsz200 tro 0 90 -3833 4
text      col 0 fon 16 tsz200 tro 0 1001 -3833 5
text      col 0 fon 16 tsz200 tro 0 1911 -3833 6

(All data not shown.)
```

ged — Graphical Editor

General

The graphical editor (**ged**) allows displaying and editing of the GPS. The graphics editor will not be discussed in detail at this point. Chapter 5 has been devoted to the graphics editor.

R	Invoke the editor in a restricted shell environment.
e	Do not erase screen before initial display.
r*n*	Window on GPS region *n*, *n* between 1 and 25, inclusively.
u	Window on the entire GPS universe.

Command Example

The following is an example of how to enter the graphics editor.

```
^ged<CR>
*
```

graph — Draw a Graph

General

The **graph** command, with no options, takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the **tplot** command.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes ("), in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

Command Format

graph [-options]

Options:

a	Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by -x).
b	Break (disconnect) the graph after each label in the input.
c	Character string given by next argument is default label for each point.

g	Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
l	Next argument is labeled for graph.
m	Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (such as the TEKTRONIX 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).
s	Save screen, do not erase before plotting.
x [l]	If l is resent, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.
y [l]	Similarly for y.
h	Next argument is fraction of space for height.
w	Similarly for width.
r	Next argument is fraction of space to move right before plotting.
u	Similarly to move up before plotting.
t	Transpose horizontal and vertical axis. (Option -x now applies to the vertical axis.)

Note: A legend indicating grid range is produced with a grid unless the -s option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

Command Example

The following is an example of plotting an x-y graph using the input vector **A**.

```
A = 0 0
    1 2
    3 4
    5 6
    7 9
   10 11
```

```
^graph A | tplot<CR>
```

> ***Note:*** The output of **graph A** is a plot(5) format that requires the **tplot** command to draw on a display terminal.

The results of the drawing is shown in Figure 4-3.

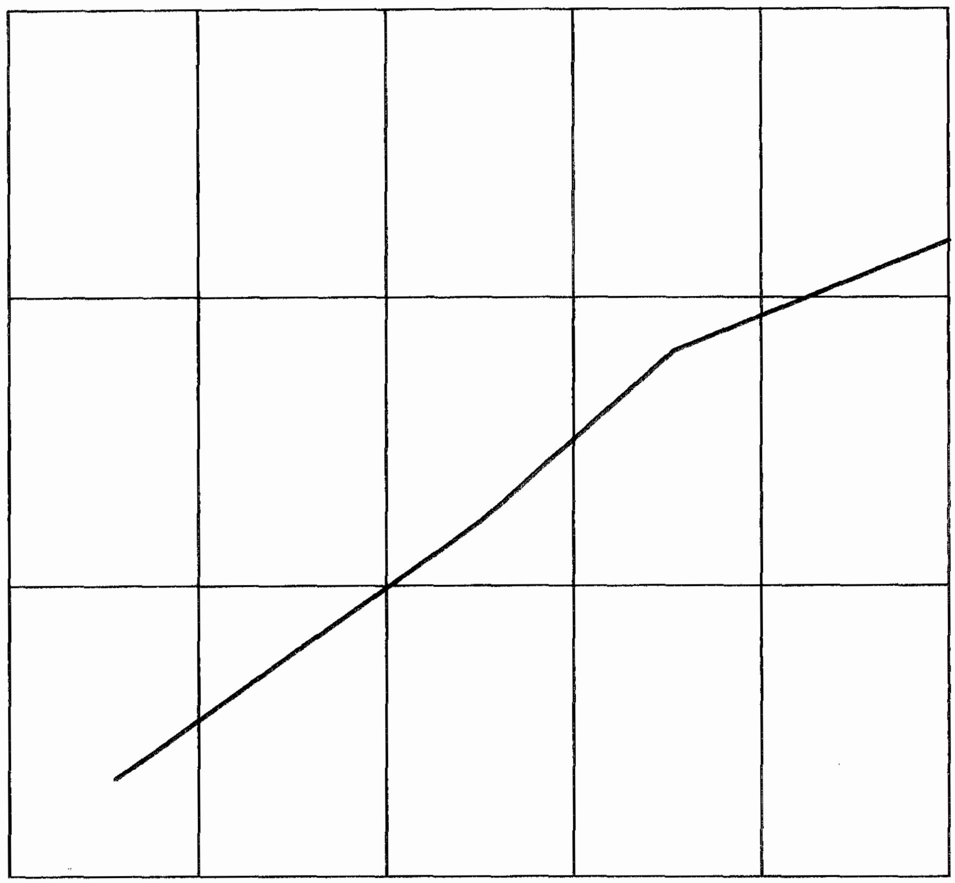

Figure 4-3. Plot of graph A | tplot

graphics — Access Graphical and Numerical Commands

General

The **graphics** command prefixes the path name /usr/bin/graf to the current $PATH value, changes the primary shell prompt to ^, and executes a new shell. The directory /usr/bin/graf contains all the graphics subsystem commands. To restore the environment that existed before issuing the graphics command, type EOT (*<CTRL-d>* control-d on most terminals). To log off from the graphics environment, type **quit**. If the -r option is given, access to the graphical commands is created in a restricted environment; that is, $PATH is set to :/usr/bin/graf:/rbin:/usr/rbin and the restricted shell, rsh, is invoked.

Command Format

```
$graphics<CR>
```

gtop — GPS to Plot(5) Format

General

The **gtop** command transforms a GPS into plot(5) format. The plot(5) format can be displayed on the 5620 DMD display terminal by using the **tplot** command. Input is taken from a file, if given; otherwise from the standard input. GPS objects are translated if they fall within the window that circumscribes the first file unless an option is given. Output is to the standard output.

Command Format

gtop [-option(s)] [GPS file(s)]

Options:

r*n*	Translate objects in GPS region *n*.
u	Translate all objects in the GPS universe.

Command Example

The following is an example showing the differences between displaying a drawing with a GPS and displaying a drawing with a plot(5) format.

With a GPS the command line is:

```
^gas | af " x^2" | plot | td<CR>
```

With a plot(5) format the command line is:

```
^gas | af " x^2" | plot | gtop | tplot<CR>
```

where

- The output of **plot** is a GPS.
- **td** command displays a GPS on a TEKTRONIX 4014.
- **gtop** transforms the GPS into plot(5) format.
- **tplot** command can display plot(5) format on such devices as a DASI 300, DASI 300s, DASI 450, TEKTRONIX 4014, Versatec* D12200A.

* Registered Trademark of Tektronix, Inc.

hardcopy — Sends Make Copy Character

General

When issued from a TEKTRONIX 4014 display terminal with a hard copy unit (printer), hardcopy generates a screen copy on the printer.

Command Format

```
^hardcopy
```

hilo — High and Low Values

General

The **hilo** command is a summarizer node. The output is the high and low values across all the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

hilo [-option(s)] [vector(s)]

Options:

h	Only output high value.
l	Only output low value.
o	Output high, low values in " option" form (see plot).
ox	Output high, low values with x prepended.
oy	Output high, low values with y prepended.

Command Example

The following is an example of finding the high and low values of vector **A**.

A = 10 20 30 40 50

^**hilo A**<*CR*>

low = 10   high = 50

hist — Build a Histogram

General

The **hist** command is a translator node that generates a GPS that describes a histogram. The input vector for the **hist** command must be made up of intervals and counts. If the input is not as intervals and counts, you must use the **bucket** command to put the input vector in that form. If no vector is given, then the standard input is assumed.

Command Format

hist [-option(s)] [vector(s)]

Options:

a	Suppress axes.
b	Plot histogram with bold weight lines, otherwise use medium.
f	Do not build a frame around the plot area.
g	Suppress background grid.
r*n*	Put the histogram in GPS region *n*, where *n* is between 1 and 25, inclusively.
x*n***(y***n***)**	Position the histogram in the GPS universe with x-orgin (y-orgin) at *n*.
xa (ya)	Do not label x-axis (y-axis).

yl*n*	*n* is the y-axis low-tick value.
yh*n*	*n* is the y-axis high-tick value.

Command Example

The following example will produce a histogram of the input vector **F**.

F = 1 1.2 1.3 2 3 7 9 22 64 70 70.4

```
^qsort F | bucket | hist | td<CR>
```

- **qsort F**, the vector F must be sorted for the bucket command.
- **bucket**, breaks the vector F into intervals and counts.
- The output of **hist** is a GPS that describes a histogram.
- **td** command displays drawings on TEKTRONIX 4014 terminal.

The results of the drawing is shown in Figure 4-4.

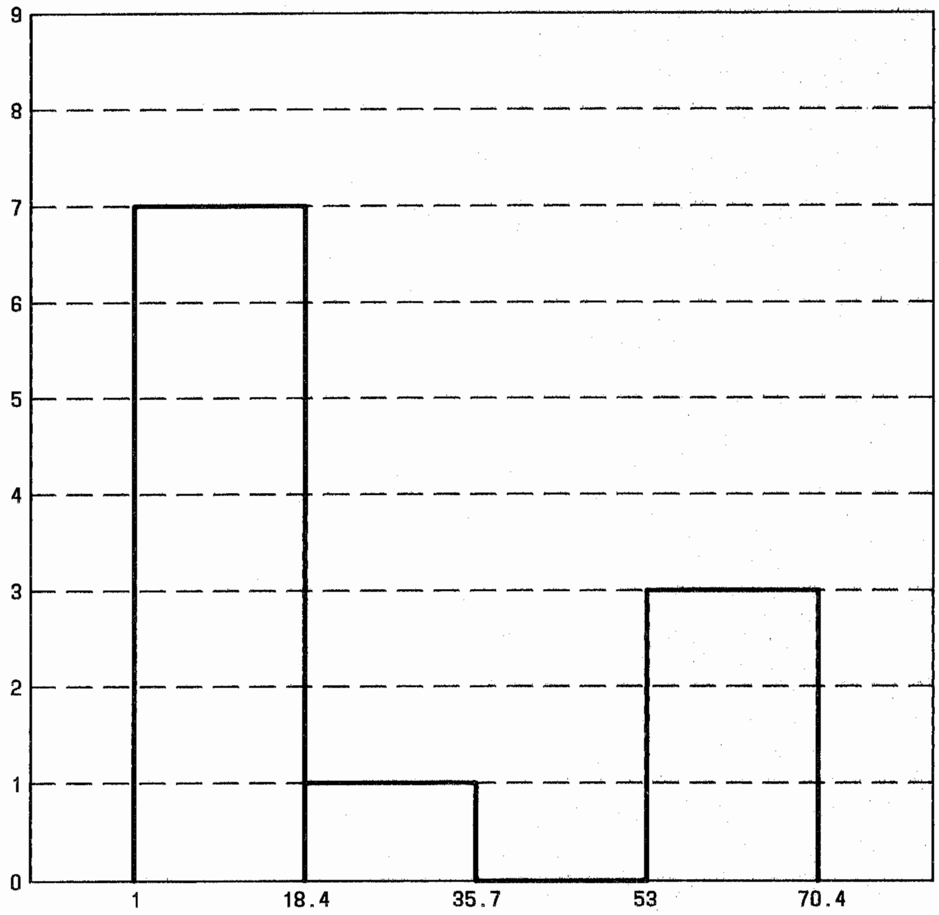

Figure 4-4. Plot of qsort F ¦ bucket ¦ hist ¦ td

hpd — Display GPS on a HP 7221A Graphics Plotter

General

The output is scope coded for a HEWLETT PACKARD 7221A Plotter. A viewing window is computed from the maximum and minimum points in the GPS file(s) unless the *r* or *u* option is provided. If no file is given, then the standard input is assumed.

Command Format

hpd [-option(s)] [GPSfile(s)]

Options:

c*n*	Select character set *n*, *n* between 0 and 5 (see the *HEWLETT PACKARD 7221A Plotter Operating and Programming Manual* and the *AT&T 3B2 Computer User Reference Manual*).
p*n*	Select pen numbered*n*, *n* between 1 and 4, inclusively.
r*n*	Window on GPS region *n*, *n* between 1 and 25, inclusively.
s*n*	Slant characters *n* degrees counterclockwise from the vertical.
u	Window on the entire GPS universe.
xd*n*	Set x displacement of the viewports lower left corner to *n* inches.

xv*n*	Set width of viewport to *n* inches.
yd*n*	Set y displacement of the viewports lower left corner to *n* inches.
yv*n*	Set height of viewport to *n* inches.

Command Example

The following is an example of displaying a drawing on a HEWLETT PACKARD 7221A Graphics Plotter.

```
^gas | af "x^2" | plot | hpd<CR>
```

where the output of the **plot** command is a GPS that the **hpd** command displays that GPS on a HP 7221A Graphics Plotter.

label — Label the Axis of a Data Plot

General

The **label** command is a translator node that labels the axis of a data plot. The **label** command attaches the axis with a label by appending a label file to a GPS of a data plot (like that produced by **hist**, **bar**, and **plot**). Each line of the label file is taken as one label. Once the label has been appended to the GPS drawing, the label may not be in the appropriate position. The graphics editor (ged) can be used to position the label correctly. For plot labels, be sure to include *x i1* on the **plot** command line (see **plot**). Blank lines yield null labels. Either the GPS or the label file, but not both, may come from the standard input.

Command Format

label [-option(s)] *GPS file(s)*

Options:

b	Assume the input is a bar chart.
c	Retain lower case letters in labels; otherwise, all letters are upper case.
F*file*	*file* is the label file.
h	Assume the input is a histogram.
p	Assume the input is an x-y plot. This is the default.
r*n*	Labels are rotated *n* degrees. The pivot point is the first character.

x	Label the x-axis. This is the default.
xu	Label the upper x-axis (example, the top of the plot).
y	Label the y-axis.
yr	Label the right y-axis (example, the right side of the plot).

Command Example

The following is an example of appending a label to the y-axis of a GPS drawing:

1. Create your label file by using a text editor.

```
^vi lab<CR>
<a>Frequency of Random Numbers
<ESC>
<ZZ>
```

2. Create a GPS drawing and direct it to a file.

```
^rand -n1000 | qsort | bucket | hist >Randplot<CR>
```

3. Use the **label** command to append the label file to the GPS and display it on a TEKTRONIX 4014.

^**label -Flab,h,r90,y Randplot ¦ td**<*CR*>

See Figure 4-5 for a result of the drawing.

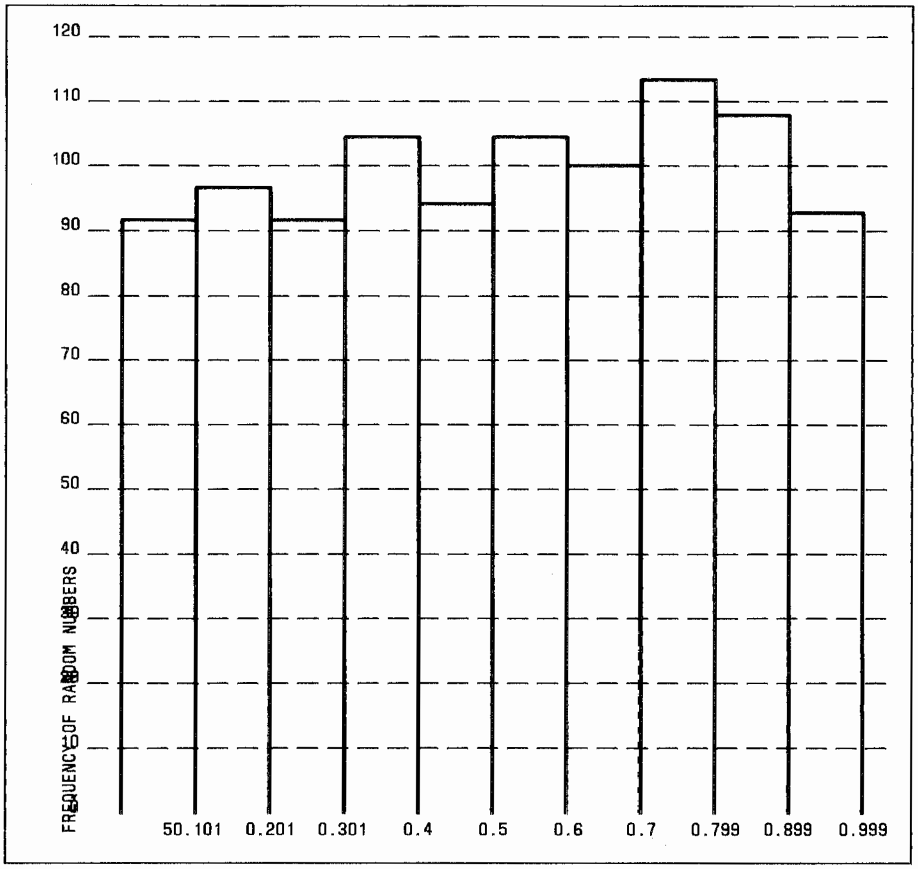

Figure 4-5. Plot of label -Flab,h,r90,y Randplot ¦ td

List — List Vector

General

The **list** command is a transformer node that list vectors. The output is a listing of the elements of the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

list [-option(s)] [vector(s)]

Options:

c*n*	*n* is the number of output elements per line. Five is the default value.
d*string*	The characters in *string* serve as delimiters. Only elements that are delimited by these characters will be listed. The white space, characters space, tab, and newline are always delimiters.

Note: If **d** is not specified, then any character that is not part of a number is a delimiter. If **d** is specified, then the white space characters (space, tab, and new-line) plus the character(s) of string are delimiters. Only numbers surrounded by delimiters are listed.

Command Example

The following is an example of using the **list** command to display the **A** vector.

A = 10 20 30 40 50

```
^list A<CR>

10 20 30 40 50
```

log — Logarithm

General

The **log** command is a transformer node that takes the logarithm for each element of the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

```
log [-option(s)] [vector(s)]
```

Options:

b*n*	*n* is the logarithm base. If not given, 2.71828... is used.
c*n*	*n* is the number of output elements per line.

Command Example

An example of finding the logarithm function of the vector **A** follows:

A = 10 20 30 40 50

```
^log A<CR>
2.302  2.99  3.4  3.688  3.912
```

lreg — Linear Regression

General

The linear regression (**lreg**) is a summarizer node. The output is the slope and intercept from a least squares linear regression of each vector on a base vector. The base vector is specified using the **F** option. If the base is not given, it is assumed to be ascending positive integers from zero.

Command Format

```
lreg [-option(s)] [vector(s)]
```

F*vector*	*vector* is the base.
i	Only output the intercept.
o	Output the slope and intercept in " option" form (see siline).
s	Only output the slope.

Command Example

The following is an example of finding the linear regression of the vectors **A** and **C**; **A** is the base vector (x-axis) and **C** is the y-axis.

A = 10 20 30 40 50
C = 30 40 50 60 70

```
^lreg -FA,C<CR>

intercept=20  slope=1
```

mean — Mean

General

The **mean** command is a summarizer node. The output is the mean of the elements in the input vector(s). The input may optionally be trimmed. If no vector is given, then the standard input is assumed.

Command Format

mean [-option(s)] [vector(s)]

Options:

f*n*	Trim (1/n)*r elements from each end, where r is the rank of the input vector.
p*n*	Trim n*r elements from each end, n is between 0 and 0.5.
n*n*	Trim *n* elements from each end.

Command Example

The following is an example of finding the mean of the vector **A**.

A = 10 20 30 40 50

```
^mean A<CR>

30
```

mod — Modulo Function

General

The modulo function (**mod**)is a transformer node. The output is a vector with each element being the remainder of dividing the corresponding element from the input vector(s) by the modulus. If no vector is given, then the standard input is assumed.

Command Format

mod [-option(s)] [vector(s)]

Options:

c*n*	*n* is the number of output elements per line.
m*n*	*n* is the modulus. If not given, 2 is used.

Command Example

The following is an example of finding the modulo function of vector **G**.

G = 1 2 3 4 5 6 7 8 9

```
^mod -m3 G<CR>

1 2 0 1 2 0 1 2 0
```

pair — Pair Element Group

General

The **pair** command is a transformer node. The output is a vector with elements taken alternately from a base vector and a vector. The base vector is specified either with the **F** option, or else it comes from the standard input. Vector(s) are specified either on the command line or else one may come from the standard input. If both the base and vector come from the standard input, base precedes vector.

Command Format

pair [-option(s)] [vector(s)]

Options:

c*n*	*n* is the number of output elements per line.
F*vector*	*vector* is the base.
x*n*	*n* is the number of elements taken from the base for each one element taken from vector.

Command Example

The following is an example of finding the pair element group of vectors **A** and **C**.

A = 10 20 30 40 50
C = 30 40 50 60 70

```
^pair -FA,C<CR>

10 30 20 40 30
50 40 60 50 70
```

pd — Plot(5) Format Dump

General

The **pd** command prints a human readable listing of plot(5) format. If no file is given, then the standard input is assumed.

Command Format

```
pd [plot(5) file(s)]
```

Command Example

The following is an example of printing a human readable listing of a plot(5) format.

1. Create a plot(5) format and direct it to a file.

```
^gas | af " x^2" | plot | gtop >Y
```

2. Print a readable listing of plot(5) format.

```
^pd Y<CR>
erase
space   13461   13351   19765   19655
jump    13461   13351
text                    ;
jump    14607   14607
cont    19160   14607
cont    19160   18660
cont    14607   18660
cont    14607   14607
jump    14607   14467
text                    0
jump    15062   14467
text                    1
jump    15517   14467

(All data not shown.)
```

pie — Build a Pie Chart

General

The **pie** command is a translator node. Its output is a GPS that describes a pie chart. The input is a text file that has the data form:

    [<control>] value [label]

The control field specifies the way that slice should be handled. Legal values are:

i	The slice will not be drawn, although a space will be left for it.
e	The slice is " exploded," or removed from the pie.
f	The slice is filled. The angle of fill lines depends on the color of the slice.
c*color*	The slice is drawn in *color* rather than the default black. Legal values for *color* are 'b' for black, 'r' for red, 'g' for green, and 'u' for blue.

The pie is drawn with the value of each slice printed inside and the label printed outside. If no file is specified, the standard input is assumed.

Command Format

```
pie [-option(s)] [file(s)]
```

Options:

b	Draw pie chart in bold weight lines; otherwise, use medium
p	Output value as a percentage of the total pie
pp*n*	Only draw *n* percent of a pie
pn*n*	Output value as percentage, but total of percentages equals *n* rather than 100. pn100 is equivalent to p
v	Do not output the values
o	Output values around the outside of the pie
r*n*	Put the pie chart in region *n*, where *n* is between 1 and 25, inclusively
x*n* **(y***n***)**	Position the pie chart in the GPS universe with x-origin (y-origin) at *n*.

Command Example

The following is an example of creating a pie chart.

1. Create the input text file.

   Note: The following sample commands show the *i*, *e*, and *f* enclosed in brackets. These brackets and the corresponding control field must be entered by the user. This is an exception to the rule on "HOW COMMANDS ARE DESCRIBED"in Chapter 2.

```
^vi piedata<CR>
<a><i> 1 a<CR>
<e> 2 b<CR>
<f> 3 c<CR>
4<CR>
<ESC>
<ZZ>
```

2. Create the pie drawing and display it on a TEKTRONIX 4014.

```
^pie -p piedata | td<CR>
```

See Figure 4-6 for a result of the drawing.

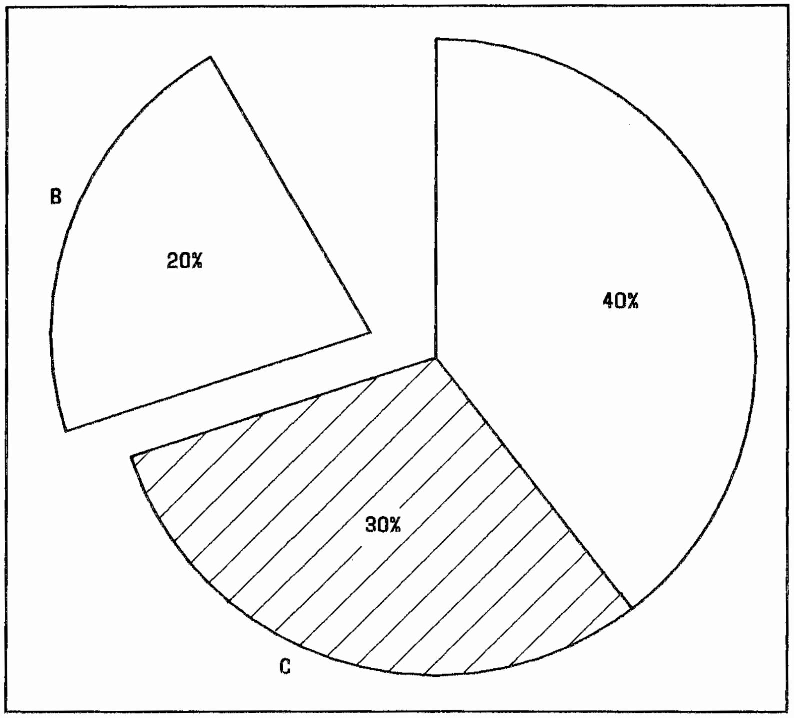

Figure 4-6. Plot of pie -p piedata | td

plot — Plot an X-Y Graph

General

The **plot** command is a translator node that describes an x-y graph. Input is one or more vector(s). Y-axis values come from vector(s), x-axis values from the **F** option. Axis scales are determined from the first vector(s) plotted. If no vector is given, then the standard input is assumed.

Command Format

plot [-option(s)] [vector(s)]

Options:

a	Suppress axis.
b	Plot graph with bold weight lines; otherwise, use medium.
c*char(s)*	Use *char(s)* for plotting characters, implies option m. The first character of *char(s)* is used to mark the first graph, the second is used to mark the second graph, etc.
d	Do not connect plotted points, implies option m.
F*vector*	Use *vector* for x-values, otherwise the positive integers are used.
g	Suppress background grid.

m	Mark the plotted points.
r*n*	Put the graph in GPS region *n*, where *n* is between 1 and 25, inclusively.
x*n* **(y***n***)**	Position the graph in the GPS universe with x-origin (y-origin) at *n*.
xa (ya)	Omit x-axis (y-axis) labels.
xi*n* **(yi***n***)**	*n* is the x-axis (y-axis) tick increment.
xl*n* **(yl***n***)**	*n* is the x-axis (y-axis) low-tick value.
xh*n* **(yh***n***)**	*n* is the x-axis (y-axis) high-tick value.
xn*n* **(yn***n***)**	*n* is the approximate ticks on the x-axis (y-axis).
xt (yt)	Omit x-axis (y-axis) title.

Command Example

The following is an example of creating an x-y graph by using the **plot** command.

B = 1 2 3 4 5 6 7 8 9 10

```
^gas ! af " x^2" ! plot -F-,dg B ! td<CR>
```

Note: The **plot -F-,dg B** uses the **B** vector as the y-axis and uses the standard input for the x-axis.

See Figure 4-7 for a result of the drawing.

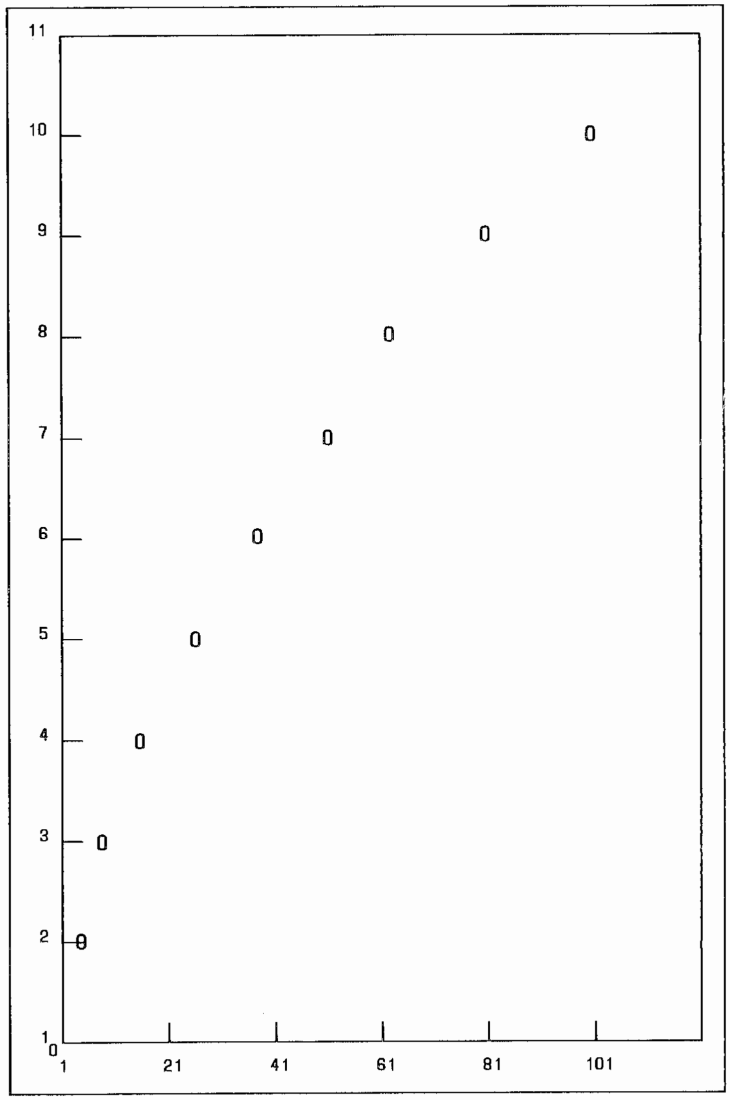

Figure 4-7. Plot of gas ¦ af " x^2" ¦ plot -F-,dg B ¦ td

point — Empirical Cumulative Density Function Point

General

The **point** command is a summarizer node. The output is a linearly interpolated value from the empirical cumulative density function (e.c.d.f) for the input vector. By default, point returns the median (50 percent(%) point). If no vector is given, the standard input is assumed.

Command Format

point [-option(s)] [vector(s)]

Options:

f*n*	Return the (1/*n*)*100 percent point from the e.c.d.f.
p*n*	Return the *n* *100 percent point
n*n*	Return the *n*th element
s	The input is assumed to be sorted.

Command Example

The following is an example of finding the empirical cumulative density function point of vector **A**.

A = 10 20 30 40 50

```
^point -p.25 A<CR>

20
^point -p.50 A<CR>

30
^point -p.75 A<CR>

40
```

power — Power Function

General

The **power** function is a transformer node. The output is a vector with each element being a power of the corresponding element from the input vector(s). If no vector is given, the standard input is assumed.

Command Format

power [-option(s)] [vector(s)]

Options:

c*n*	*n* is the number of output elements per line
p*n*	Input elements are raised to the *n*th power. If not given, 2 is used.

Command Example

The following is an example of finding the 4th power of the input vector **A**.

A = 10 20 30 40 50

```
^power -p4 A<CR>

10e+03  160e+03  810e+03 2.56e+06 6.25e+06
```

prime — Generate Prime Numbers

General

The **prime** command is a generator node that generates prime numbers. The output is a vector of number elements determined by the parameters low and high. The parameters are set by command options.

Command Format

prime [-option(s)]

Options:

c*n*	*n* elements per output line
h*n*	*n* = high
l*n*	*n* = low. If not given, low = 2.

Command Example

The following is an example of generating prime numbers from 5 through 20.

^**prime -l5,h20**<*CR*>

5 7 11 13 17 19

prod — Product

General

The **prod** command is a summarizer node that finds the product of each element in the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

prod [vector(s)]

Command Example

The following is an example of finding the product of the input vector **A**.

A = 10 20 30 40 50

```
^prod A<CR>
1.2e+07
```

ptog — Plot(5) Format to GPS Format

General

The **ptog** command transforms plot(5) format into a GPS. Input is taken from the file, if given; otherwise, it is taken from the standard input. Output is to the standard output.

Command Format

```
ptog [plot(5) file(s)]
```

Command Example

The following is an example of transforming a plot(5) format into a GPS.

1. Create a plot(5) format and direct it to a file.

```
^gas | af " x^2"  | plot | gtop > B<CR>
```

2. Use **ptog** command to transform the plot(5) format to a GPS and display it on a TEKTRONIX 4014.

```
^ptog B | td<CR>
```

See Figure 4-8 for a result of the drawing.

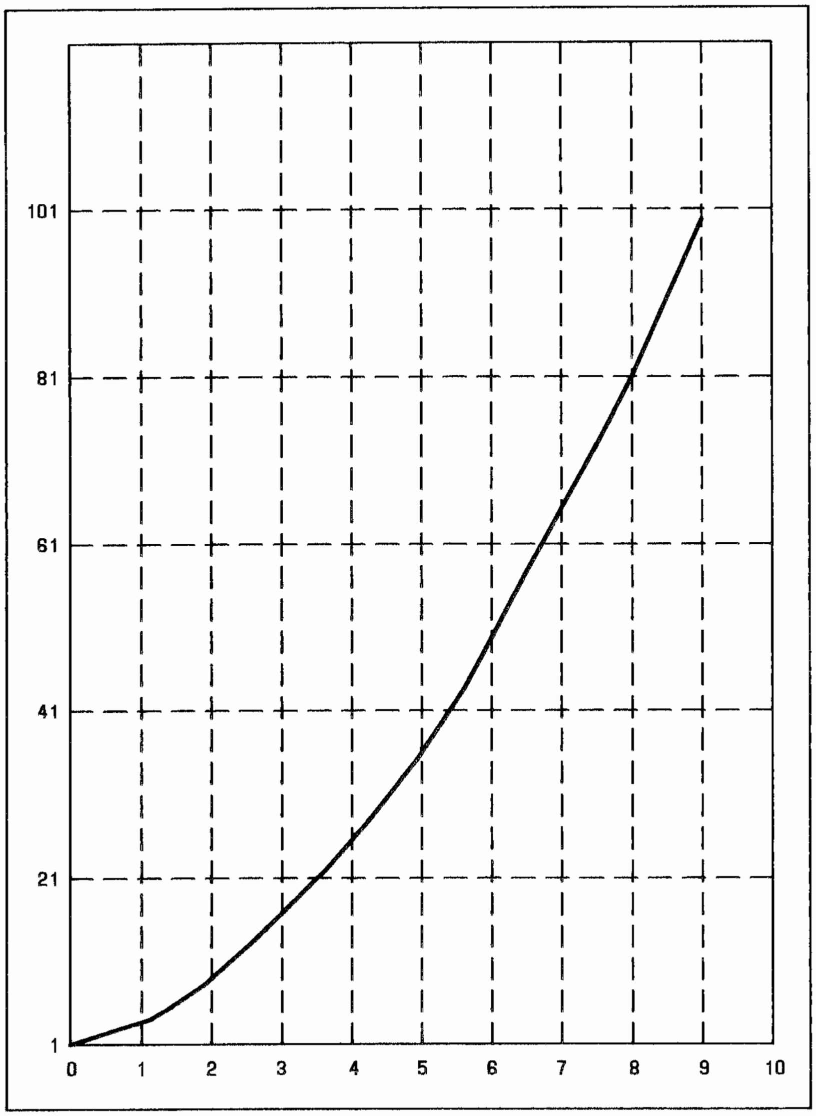

Figure 4-8. Plot of ptog B ¦ td

qsort — Quick Sort

General

The **qsort** command is a summarizer command that sorts the input vector in ascending order. If no vector is given, then the standard input is assumed.

Command Format

```
qsort [-option] [vector(s)]
```

Option:

c*n* — *n* is the number of output elements per line.

Command Example

The following is an example of sorting the vector **P**.

P = 60 50 40 30 20 10

```
^qsort P<CR>
10 20 30 40 50 60
```

quit — Terminate Session

General

The **quit** command terminates the current terminal session.

rand — Generate Random Sequence

General

The **rand** command is a generator node that generates random numbers. The output is a vector of number elements determined by the parameters low, high, multiplier, and seed. Random numbers are first generated in the range 0 to 1, initialized by the seed. Then if a multiplier is given, each number is multiplied accordingly. The parameters are set by command options.

Command Format

rand [-option(s)]

Options:

c*n*	*n* elements per output line
h*n*	*n* = high. If not given, high = 1
l*n*	*n* = low. If not given, low = 0
m*n*	*n* = multiplier. If not given, multiplier is determined from high and low
n*n*	*n* = number. If not given, number = 10
s*n*	*n* = seed. If not given, seed = 1.

Command Example

The following is an example of generating a vector of random numbers with the **s**eed = 10 and high = 20.

```
^rand -s10,h20<CR>

2.77291 17.221  6.86361 5.41398 10.3073
1.36357 12.3063 16.8413 12.363  18.7445
```

rank — Rank of Vector

General

The **rank** command is a summarizer node that gives the number of elements in each input vector. If no vector is given, then the standard input is assumed.

Command Format

```
rank [vector(s)]
```

Command Example

The following is an example of finding the rank of vector **A**.

A = 10 20 30 40 50

```
^rank A<CR>

5
```

remcom — Remove Comments

General

The **remcom** command copies its input to its output with comments removed. Comments are as defined in C (such as, /* comment */). Input is from file(s), if given, otherwise it is from the standard input.

Command Format

```
remcom [file(s)]
```

Command Example

The following is an example of removing the comments from vector **G**.

```
G = 23  /*comments*/
   45  /*comments*/
```

```
^remcom G<CR>

23
45
```

root — Root Function

General

The **root** function is a transformer node. The output is a vector with each element being the root of the corresponding element from the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

```
root [-option(s)] [vector(s)]
```

Options:

c*n*	*n* is the number of output elements per line
r*n*	*n* = root. If not given, root = 2.

Command Example

The following is an example of finding the square root of vector **A**.

A = 10 20 30 40 50

```
^root A<CR>
3.16778 4.47214 5.47723 6.32456 7.07107
```

round — Rounded Value

General

The **round** command is a transformer node. The output is the rounded value for each element of the input vector(s). If no vector is given, the standard input is assumed.

Command Format

```
round [-option(s)] [vector(s)]
```

Options:

c*n*	*n* is the number of output elements per line
p*n*	*n* is the number of places following the decimal point rounded to the next number where *n* is in the range 0 to 9, 0 by default
s*n*	*n* is the number of significant digits rounded to the next number where *n* is in the range 0 to 9, 9 by default.

Command Example

An example of rounding each elements of the input vector **X**, follows:

X = 2.4 3.6 8.2

```
^round X<CR>

2 4 8
```

siline — Generate a Line Given Slope and Intercept

General

The **siline** command is a transformer node that generates a line from a given slope and intercept. The output is a vector of values slope times x plus intercept, where x takes on values from vector(s). If the *n* option is given, vector is the ascending positive integers. If neither the *n* option nor a vector is given, vector comes from the standard input.

Command Format

siline [-option(s)] [vector(s)]

Options:

c*n*	*n* is the number of output elements per line
i*n*	*n* is the intercept, 0 if not given
n*n*	*n* is the number of positive integers to be used for x
s*n*	*n* is the slope, 1 if not given.

Command Example

The following is an example of generating a line that has the slope of 2 and intercept of 1.

```
^siline -n10,s2,i1 | plot | td
```

- **siline -n10,s2,i1** generates the following data.

1	3	5	7	9
11	13	15	17	19

- **plot** generates a GPS of an x-y graph.
- **td** command displays a drawing on TEKTRONIX 4014 terminal.

See Figure 4-9 for a result of the drawing.

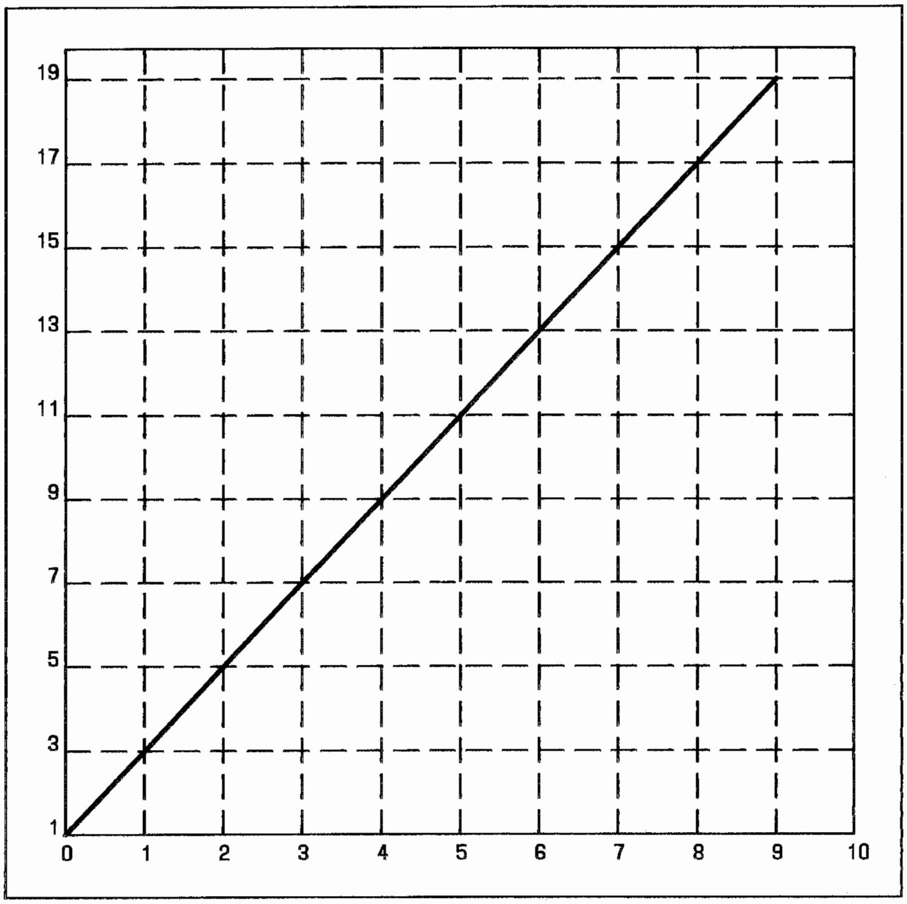

Figure 4-9. Plot of siline -n10,s2,i1 ¦ plot ¦td

sin — Sine Function

General

The **sin** command is a transformer node that takes the sine for each element of the input vector(s). Input is assumed to be in radians. If no vector is given, then the standard input is assumed.

Command Format

```
sin [-option] [vector(s)]
```

Option:

c*n* *n* is the number of output elements per line.

Command Example

The following is an example of finding the sine of each element of the input vector **A**.

A = 10 20 30 40 50

```
^sin A<CR>
-.544071 .912945 -.988032 .745113 -.262375
```

spline — Interpolate Smooth Curve

General

The **spline** command takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is almost equally spaced and includes the input set, on the standard output. The cubic spline output has two continuous derivatives, and has enough points to look smooth when plotted (for example, by **graph**).

Command Format

```
spline [options]
```

Options:

-a Supplies abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.

-k The constant k used in the boundary value computation: $y_0=ky_1, y_n=ky_{n-1}$ is set by the next argument (default k = 0).

-n Space output points so that n intervals occur almost between the lower and upper x limits (default n = 1000).

-p Makes output periodic, such as, matching derivatives at the ends. First and last input values should normally agree.

-x	Next 1 (or 2) argument is lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at the lower limit (default 0).

Command Example

The following is an example of using the **spline** command to produce an x-y graph from the **Z** vector.

```
Z = 1 2
   3 4
   5 6
   7 9
  10 11
```

```
^spline <Z | graph | ptog | td<CR>
```

Note: The **spline** command take the pairs of numbers from the input and generates points that will create a smooth curve when used with the **graph** command. Remember the **graph** command must use the **ptog** command to change the format from plot(5) to a GPS.

See Figure 4-10 for a result of the drawing.

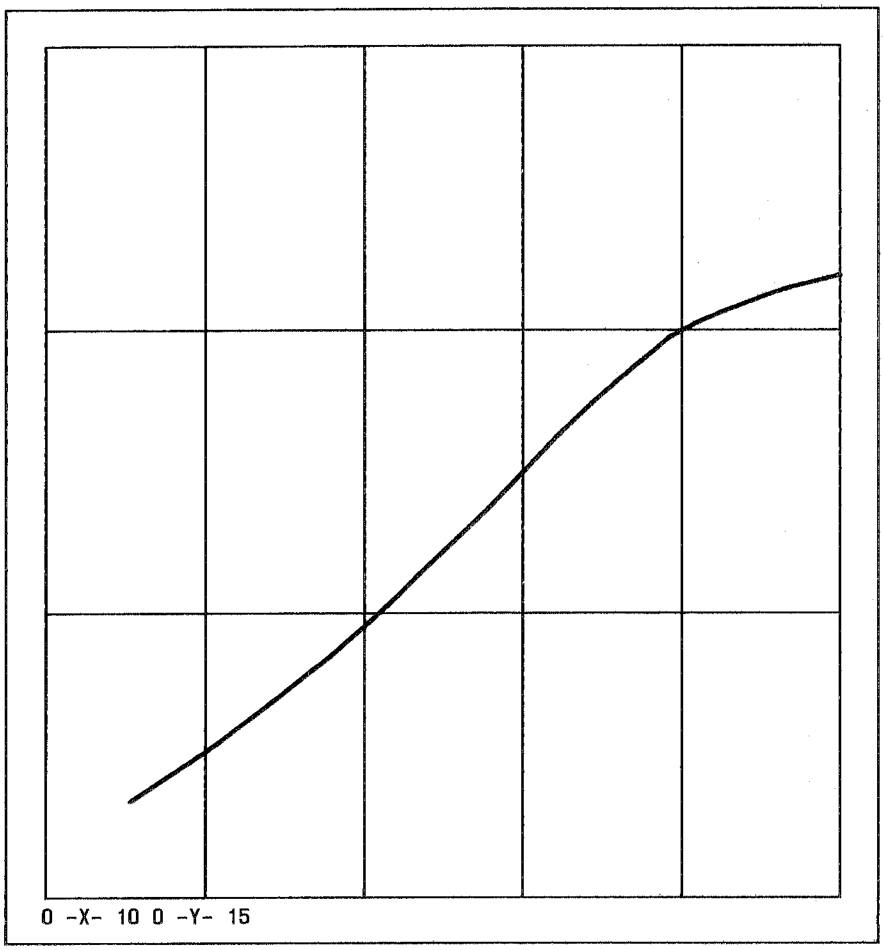

Figure 4-10. Plot of spline <Z ¦ graph ¦ ptog ¦ td

subset — Generate a Subset

General

The **subset** command is a transformer node that generates a subset. The output is elements selected from the input based on a key and option(s). If no vector is given, then the standard input is assumed.

Selection

If a master vector is given, then the key for the *i*th element of the input is the *i*th element of master, otherwise the key is the input element itself. In either case, *i* goes from start to terminate.

The input element is selected if the key is either above, below, or equal to pick, and not equal to leave. If neither above, below, nor pick is given, then the element is selected if it is not equal to leave.

Command Format

subset [-option(s)] [vector(s)]

Options:

a*n*	*n* = above
b*n*	*n* = below
c*n*	*n* elements per output line
F*vector*	*vector* is the master

i*n*	*n* = interval, default is 1
l*n*	*n* = leave
nl	Leave elements whose index is given in master
np	Pick elements whose index is given in master
p*n*	*n* = pick
s*n*	*n* = start, default is 1
t*n*	*n* = terminate, default is 32767.

Command Example

The following is an example of generating a **subset** from the master vector and another vector.

(master vector) xvector = 1 2 3 4 5 6 7 8 9
yvector = 11 22 33 44 55 66 77 88 99

```
^subset -Fxvector,a5<CR>

66 77 88 99
```

td — Display GPS on a TEKTRONIX 4014

General

The **td** command displays the GPS on a TEKTRONIX 4014. The output is scope coded for a TEKTRONIX 4014 terminal. A viewing window is computed from the maximum and minimum points in the first file unless options are provided. If no file is given, then the standard input is assumed.

Command Format

td [-option(s)] [GPS file(s)]

Options:

r*n*	Window on GPS region *n*, *n* between 1 and 25, inclusively
u	Window on the entire GPS universe
e	Do not erase screen before initiating display.

tekset — Send Reset Character for TEKTRONIX 4014 Display Terminal

General

The **tekset** command clears the display screen, sets the display mode to alpha, and the characters to the smallest font.

Command Format

```
^tekset<CR>
```

title — Title a Vector or GPS

General

The **title** command is a translator node used to title a vector or a GPS. Input is taken from the file(s), if given, otherwise it is from the standard input.

Command Format

title [-option(s)] [file(s)]

Options:

b	Make the GPS title bold.
c	Retain lower case letters in title; otherwise, all letters are upper case.
l*string*	For a GPS, generate a lower title = string.
u*string*	For a GPS, generate an upper title = string.
v*string*	For a vector, title = string.

Command Example

The following is an example of titling a GPS.

1. Create a file containing a GPS.

```
^rand -n1000 ¦ qsort ¦ bucket ¦ hist >Randplot<CR>
```

2. Title the GPS file (**Randplot**).

```
^title -l" lower title" ,u" upper title"  Randplot ¦ td<CR>
```

See Figure 4-11 for a result of the drawing.

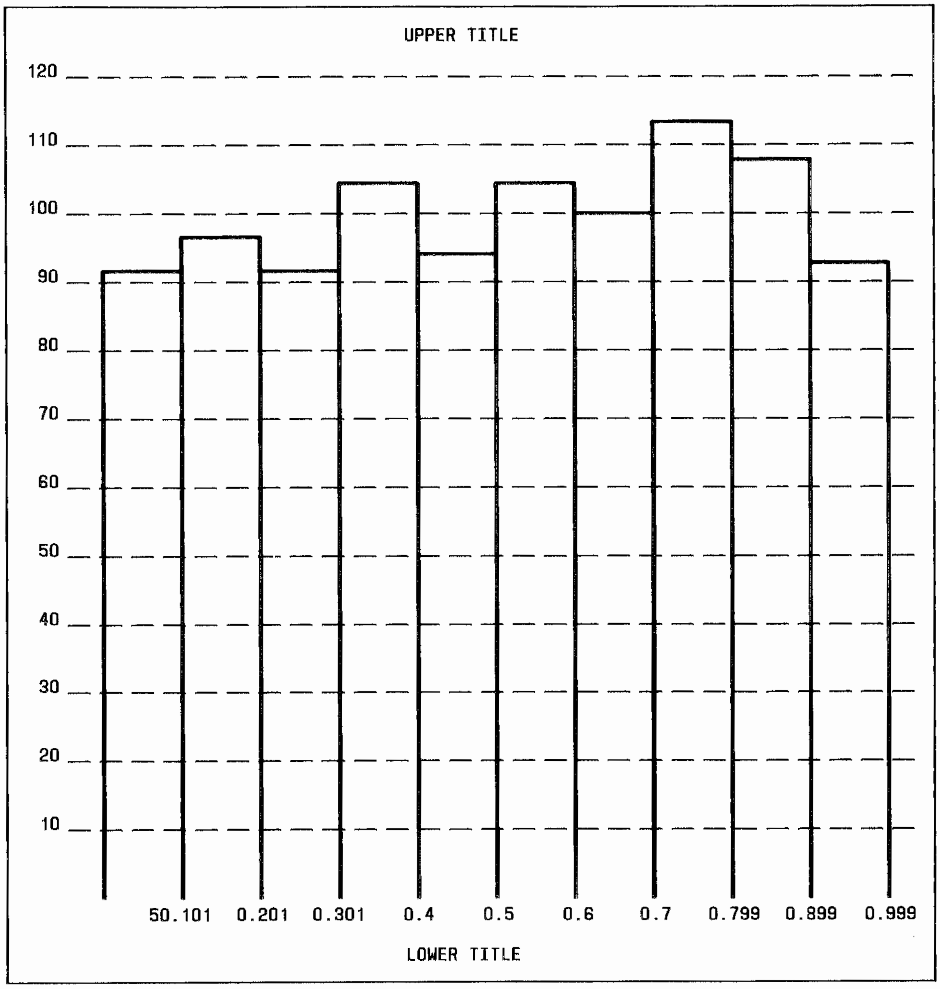

Figure 4-11. Plot of title -l" lower title" ,u" upper title" Randplot | td

total — Sum Total

General

The **total** command is a summarizer node. The output is the sum total of the elements in the input vector(s). If no vector is given, the standard input is assumed.

Command Format

```
total [vector(s)]
```

Command Example

The following is an example of finding the sum total of all the elements in vector **A**.

A = 10 20 30 40 50

```
^total A<CR>

150
```

tplot — Graphics Filter

General

The **tplot** command reads plotting instructions, plot(5), from the standard input; and in general, produce plotting instructions suitable for a particular terminal on the standard output. If no terminal is specified, the environment parameter **$TERM** is used. Known terminals are:

300 DASI 300
300S DASI 300s
450 DASI 450.
4014 TEKTRONIX 4014
ver Versatec D1200A. (This version of the plot places a scan-converted image in /usr/tmp/raster and sends the result directly to the plotter device, rather than to the standard output. The -e option causes a previously scan-converted file raster to be sent to the plotter.)

Command Format

tplot [-T terminal [-e raster]]

Command Example

(See the **graph** example.)

ttoc — Make Textual Table of Contents

General

The output is the Textual Table of Contents (TTOC) generated by the .H macro of the nroff or troff raw data file of Document Workbench. If no file is given, then the standard input is assumed.

Command Format

ttoc [mm(1) file]

Command Example

The following is an example of outputting a Textual Table of Contents (**TTOC**).

1. Create a heading with text.

```
^vi txt<CR>
^.H 1 " example one" <CR>
 A line of text<CR>
.H 2 " example two" <CR>
 A second line of text<CR>
.H 3 " example three" <CR>
A third line of text<CR>
.H 3 " example four" <CR>
A fourth line of text<CR>
.H 2 " example five<CR>
A fifth line of text<CR>
.H 3 " example six<CR>
A sixth line of text<CR>
.H 1 " example seven<CR>
end
```

2. List a Textual Table of Contents (**TTOC**).

```
^ttoc txt<CR>

0.   " Table of Contents"
1.   " example one"  0
1.1  " example two"  0
1.1.1" example three"  0
1.1.2" example four"  0
1.2  " example five"  0
1.2.1" example six"  0
2.   " example seven"  0
```

3. Display the Textual Table of Contents (**TTOC**) command on the TEKTRONIX 4014.

```
^ttoc txt ¦ vtoc ¦ td<CR>
```

See Figure 4-12 for a result of the drawing.

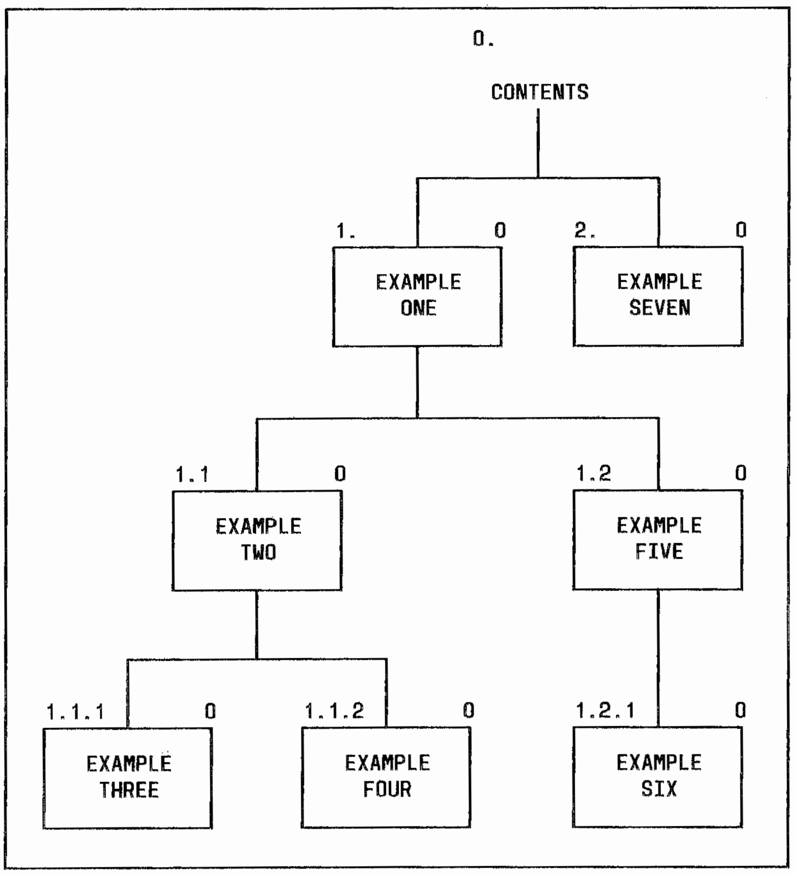

Figure 4-12. Plot of ttoc txt ¦ vtoc ¦ td

var — Variance

General

The **var** command is a summarizer node that finds the difference between the slope point and outer point. The output is the variance of the elements in the input vector(s). If no vector is given, then the standard input is assumed.

Command Format

```
var [vector(s)]
```

Command Example

The following is an example of finding the variance of the input vector **A**.

A = 10 20 30 40 50

```
^var A<CR>
250
```

vtoc — Visual Table of Contents

General

The output is a GPS that describes a Visual Table of Contents (**vtoc** or hierarchical chart) of the Textual Table of Contents (**TTOC**) entries from the input. If no file is given, then the standard input is assumed. **TTOC** entries have the form:

id [line weight,line style] " text" [mark]

where

id	Is an alternating sequence of numbers and dots.
line weight	Is either **n** for narrow, **m** for medium, or **b** for bold.
line style	Is either **so** for solid, **do** for dotted, **dd** for dot-dashed, or **ld** for long-dashed.
text	Is a string of characters surrounded by quotes.
mark	Is a string of characters (surrounded by quotes if it contains spaces), with included dots being escaped.

Command Format

vtoc [-option(s)] [TTOC file]

Options:

c	Take the ext as entered (default is all upper case).
d	Connect the boxes with diagonal lines.
h*n*	Horizontal interbox space is *n*% of box width.
i	Suppress the box id.
m	Suppress the box mark.
s	Do not compact boxes horizontally.
v*n*	Vertical interbox space is *n*% of box height.

Command Example

(See the **TTOC** example.)

whatis — Brief Online Documentation

General

The **whatis** command prints a brief description of each command given. If no command is given, then the current list of the description command is printed. The **whatis** command prints out every description.

Command Format

whatis [-option] [name(s)]

Option:

o Just print command options.

Command Example

The following is an example of using the **whatis** command.

^**whatis bel**<*CR*>

bel; send bel character to terminal

_B_e_l causes most terminals to sound an audible
tone, a useful nonvisual signal.

yoo — Pipe Fitting

General

The **yoo** command is a piping primitive that deposits the output of a pipeline into a file used in the same pipeline. Note that without **yoo**, this is not usually successful as it causes a read and write on the same file, simultaneously.

Command Format

```
yoo file
```

Command Example

The following is an example of using the **yoo** command.

```
^af " x^2"  | plot | yoo x<CR>
```

Chapter 5

GRAPHICS EDITOR

PAGE

Chapter 5

GRAPHICS EDITOR

INTRODUCTION

The graphics editor (**ged**), is an interactive graphical editor used to display, edit, and build drawings on a TEKTRONIX 4014 display terminal. The drawings are represented as a sequence of objects in a token language known as a GPS (graphical primitive string). A GPS is produced by the drawing commands in the UNIX System Graphics such as **vtoc** and **plot**, as well as by **ged** itself.

Drawings are built from objects consisting of lines, arcs, and text. Using the editor, the objects can be viewed at various magnifications and from various locations. Objects can be created, deleted, moved, copied, rotated, scaled, and modified.

The examples in this chapter will illustrate how to build and edit simple drawings. Try them to become familiar with how the editor works, but keep in mind that **ged** is intended primarily to edit the output of other programs rather than to build drawings from scratch.

GETTING STARTED

To enter the graphics editor (**ged**), enter the following command while in the graphics shell:

```
^ged<CR>
```

After a moment the screen should be clearing except for the **ged** prompt, *, in the upper left corner. The * shows that **ged** is ready to accept a command.

```
*
```

Each command passes through a sequence of stages during which you describe what the command is to do. All commands pass through a subset of these stages:

1. Command line
2. Text
3. Points
4. Pivot
5. Destination.

As a rule, each stage is stopped by typing <*CR*>. The <*CR*> for the last stage of a command triggers execution.

COMMAND FORMAT

The simplest commands consist only of a *command line*. The command line is modeled after a conventional command line in the shell.

command name [–option(s)] *filename*

GRAPHICS EDITOR COMMAND DESCRIPTION

The graphics editor will echo the full name of all commands and wait for the rest of the command line. For example, **e** references the **e**rase command. As **e**rase consists only of stage1, typing *<CR>* causes the **e**rase command to clear the display screen,

erase<CR>*

bringing the editor back to the **ged** prompt, *.

Following the command name, *options* may be entered. Options control such things as the width and style of lines to be drawn or the size and orientation of the text. Most options have a default value that applies if a value for the option is not specified on the command line. The **s**et command allows examination and change of the default values. To see the current default values, type:

set<CR>*

The option value is one of three types: integer, character, or Boolean. Boolean values are represented by a + (for true) and a – (for false). A default value is modified by providing it as an option to the **set** command. For example, to change the default text height to 300 units, type:

```
*set –h300<CR>
```

The following list will name each of the options default values. A complete description of each default option is discussed in the *Command Summary* section located at the rear of this chapter.

- **a** - angle
- **f** - factor
- **h** - height
- **s** - styletype
- **w** - withtype
- **e** - echo
- **k** - kopy
- **m** - midpoint
- **o** - out
- **p** - points
- **r** - rightpoint

- **t** - text
- **x** - x.

A question mark (**?**) is a command used to list the commands and options understood by **ged**. To generate the list, type the following:

```
*?<CR>
```

The delete key (**del**) on the 5620 DMD is used to abort a command. This is done by depressing the key after the command and before the carriage return *<CR>*. The following is an example of using this command:

```
*?<del>
```

Arguments on the command line, but not the command name, may be edited using the erase and kill characters from the shell. This applies whenever text is being entered.

CONSTRUCTING GRAPHICAL OBJECTS

Drawings are stored as a GPS in a display buffer internal to the editor. Typically, a drawing in **ged** is composed of instances of three graphical primitives: *arcs*, *lines*, and *text*.

GENERATING TEXT

To put a line of text on the display screen, use the **T**ext command.

First enter the *command line* (stage 1):

***Text**<*CR*>

Next enter the text (stage 2):

a line of text<*CR*>

Next place the graphics cursor at the desired position on the screen. The graphics cursor is the point at which the lines intercept on the screen. It can be moved by using the mouse on the DMD.

<**position cursor**><*CR*>

Positioning of the graphic cursor is done either with the thumbwheel knobs on the TEKTRONIX 4014 display terminal keyboard or with the mouse on the 5620 DMD. The <*CR*> establishes the location of the cursor to be the starting point for the text string. The **T**ext command ends at stage 3, so this <*CR*> shows the drawing of the text string.

The **T**ext command accepts options to vary the angle, height, and line width of the characters, and to either center or right justify the text object. The text string may span more than one line by escaping the *<CR>* (i.e., *<CR>*) to show continuation. To illustrate some of these capabilities, try the following:

```
*Text -r<CR>     (right justify text)
top\<CR>
right<CR>
<position cursor><CR>
*Text -a90<CR>   (rotate text 90 degrees)
lower\<CR>
left<CR>
<position cursor><CR>       (pick a point below and left of
                            the previous point)
```

Results of these commands are shown in Figure 5-1.

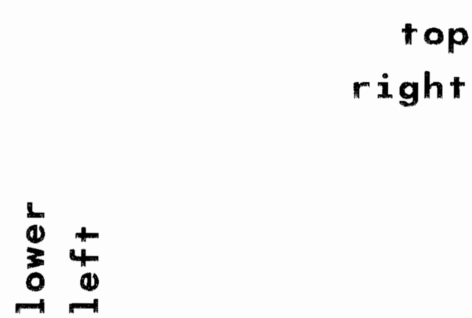

Figure 5-1. Generating Text Objects

DRAWING LINES

The **L**ines command is used to make objects built from a sequence of straight lines. It consists of stages 1 and 3. Stage 1 is straightforward:

```
*Lines [options]<CR>
```

The **L**ines command accepts options to specify line style and line width.

Stage 3, the entering of *points*, is more interesting. *Points* are referenced either with the graphic cursor or by name. We have already entered a point with the cursor for the **T**ext command. For the **L**ines command, it is more of the same. As an example, to build a triangle, type:

```
*Lines<CR>
<position cursor><SP>   (locate the first point)
<position cursor><SP>   (the second point)
<position cursor><SP>   (the third point)
<position cursor><SP>   (back to the first point)
<CR>                    (end points, draw triangle)
```

Results of these commands are shown in Figure 5-2.

Typing <**SP**> enters the location of the crosshairs as a point. **Ged** identifies the point with an integer and adds the location to the current *point set*. The last point entered can be erased by typing #. The current point set can be cleared by typing @. On receiving the final <*CR*>, the points are connected in numerical order.

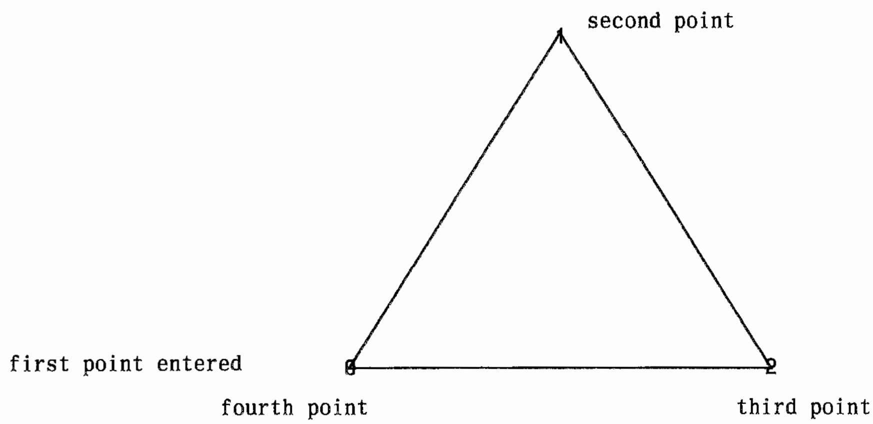

Figure 5-2. Building a Triangle

ACCESSING POINTS BY NAME

The points in the current point set may be referenced by name using the **$** operator. For instance, **$***n* references the point numbered *n*. By using **$**, the triangle above can be redrawn by entering:

```
*Lines<CR>
<position cursor><SP>
<position cursor><SP>
<position cursor><SP>
$0<CR>          (reference point 0)
<CR>
```

At the start of each command that includes stage 3, *points*, the current point set is empty. The point set from the previous command is saved and is accessible using the . operator. The . swaps the points in the previous point set with those in the current set. The = operator can be used to identify the current points. To illustrate, use the triangle just entered as the basis for drawing a quadrilateral:

```
*Lines<CR>
.       (access the previous set)
=          (identify the current points)
#        (erase the last point)
<position cursor><SP>  (add a new point)
$0<CR>       (close the figure)
<CR>
```

Results of these commands are shown in Figure 5-3.

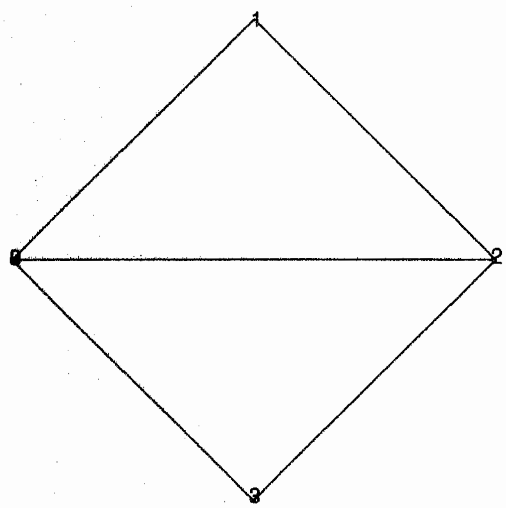

Figure 5-3. Accessing the Previous Point Set

Individual points from the previous point set can be referenced by using the **.** operator with **$**. The following example builds a triangle that shares an edge with the quadrilateral:

```
*Lines<CR>
$.1<CR>   (reference point 1 from the previous point set)
$.2<CR>   (reference point 2)
<position cursor><SP> (enter a new point)
$0<CR>    (or $.1, to close the figure)
<CR>
```

Results of these points are shown in Figure 5-4.

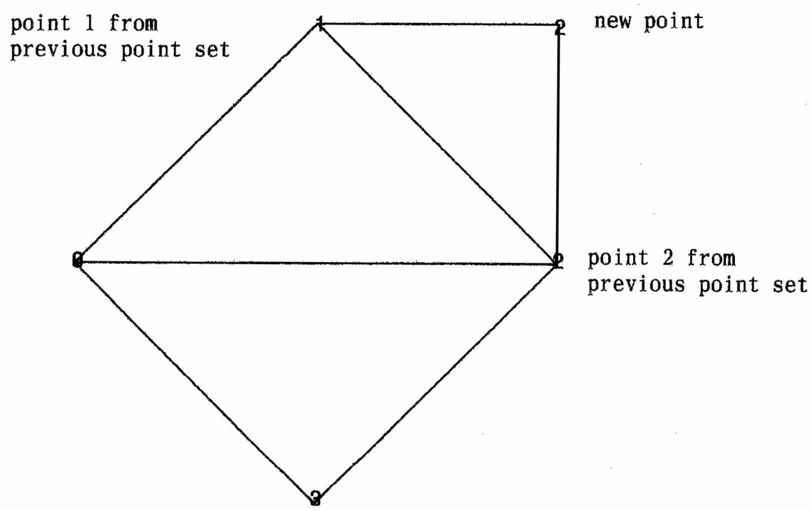

Figure 5-4. Referencing Points from Previous Point Set

A point can also be given a name. The > operator permits an upper case letter to be associated with a point just entered. A simple example is:

```
*Lines<CR>
<position cursor><SP>    (enter a point)
>A<CR>       (name the point A)
<position cursor><SP>
<CR>
```

In commands that follow, point **A** can be referenced using the **$** operator, as in:

```
*Lines<CR>
$A<CR>
<position cursor><SP>
<CR>
```

DRAWING CURVES

Curves are interpolated from a sequence of three or more points. The **A**rc command generates a circular arc given three points on a circle. The arc is drawn starting at the first point, through the second point, and ending at the third point. A circle is an arc with the first and third points coincidentally touching. Thus, one way to draw a circle is:

```
*Arc<CR>
<position cursor><SP>
<position cursor><SP>
$0<CR>
<CR>
```

Also, a circle can be generated by using the **C**ircle command. A simple example is:

```
*Circle<CR>
<position cursor><SP>    (specify the center)
<position cursor><CR>    (specify a point on the circle)
```

EDITING OBJECTS

Addressing Objects

An object is addressed by pointing to one of its *handles*. All objects have an *object-handle*. Usually the object-handle is the first point entered when the object was created. The **o**bject command marks the location of each object-handle with an **O**. For example, to see the handles of all the objects on the screen, type:

```
*objects -v<CR>
```

Some objects, **L**ines for example, also have *point-handles*. Typically, each of the points entered when an object is constructed becomes a point-handle. (An object-handle is also a point-handle.) The **p**oints command marks each of the point-handles.

A handle is pointed to by including it within a *defined area*. A defined area is generated either with a command line option or interactively using the graphic cursor. As an example, to delete one object that was created on the screen, type:

```
*Delete<CR>
<position cursor><SP>  (above and to the left of some
                       object-handle)
<position cursor><SP>  (below and to the right of the
                       object-handle)
<CR>                   (the defined area should include the
                       object-handle)
<CR>                   (if all is well, delete the object)
```

The defined area is outlined with dotted lines. The reason for the seemingly extra *<CR>* at the end of the **D**elete command is to provide an opportunity to stop the command (using *<del>* key) if the defined area is not right. Every command that accepts a defined area will wait for a confirming *<CR>*. The **n**ew command can be used to get a fresh copy of the remaining objects.

Defined areas are entered as **p**oints in the same way that objects are created. Actually, a defined area may be generated by giving anywhere from 0 to 30 points. Inputting zero points is particularly useful to point to a single handle. It creates a small defined area which is about the location of the terminating *<CR>*. Using a zero point defined area, the **D**elete command would be:

```
*Delete<CR>
<position cursor>    (center crosshairs on the object-handle)
<CR>                 (end the defined area)
<CR>                 (delete the object)
```

A defined area can also be given as a command line option. For example, to delete everything in the display buffer gives the **u**niverse option (**u**) to the **D**elete command. Note the difference between the commands **D**elete **–u**niverse and **e**rase. The **u**niverse option deletes all points in the buffer. The **e**rase command clears the screen.

Changing the Location of an Object

Objects are moved, using the **M**ove command. Create a circle using **A**rc, then move it as follows:

```
*Move<CR>
<position cursor><CR> (centered on the object-handle)
<CR>                  (this establishes a pivot, marked with
                       an asterisk)
<position cursor><CR> (this establishes a destination)
```

The basic move operation relocates every point in each object within the defined area by the distance from the *pivot* to the *destination*. Here, the pivot was chosen to be the object-handle. So effectively, the object-handle was moved to the destination point.

Changing the Shape of an Object

The **B**ox command is a special case of generating lines. Given two points, it creates a rectangle such that the two points are at opposite corners. The sides of the rectangle lie parallel to the edges of the screen. To draw a box, type:

```
*Box<CR>
<position cursor><SP>
<position cursor><CR>
```

The **B**ox command generates point-handles at each vertex of the rectangle. Use the **p**oints command to mark the point-handles. The shape of an object can be altered by moving point-handles. The next example illustrates one way to double the height of a box (see Figure 5-5.)

```
*Move -p+<CR>
<position cursor><SP>    (left of the box, between the
                         top and bottom edges)
<position cursor><CR>    (right of the box, below the
                         bottom edge)
<position cursor><CR>    (on the top edge)
<position cursor><CR>    (directly below on the bottom
                         edge)
```

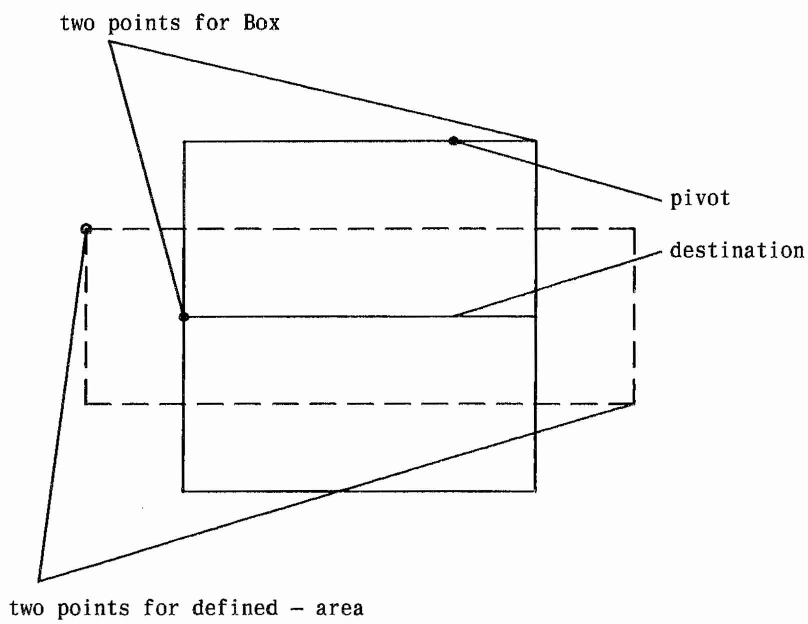

Figure 5-5. Growing a Box

When the **p**oints flag (**p**) is true, operations are applied to each point-handle addressed. In this example, the **p**oints flag was set to true using the command-line option **–p**+ causing each point-handle within the defined area to be moved the distance from the pivot to the destination. If **p** was false, only the object-handle would have been addressed.

Changing the Size of an Object

The size of an object can be changed using the **S**cale command. The **S**cale command scales objects by changing the distance from each handle of the object to the pivot by a factor. Put a line of text on the screen and try the following **S**cale commands (Figure 5-6):

```
*Scale -f200<CR>        (factor is in percent)
<position cursor><CR>  (point to object-handle)
<position cursor><CR>  (set pivot to rightmost character)
<CR>

*Scale -f50<CR>
.<CR>                  (reference the previous defined area)
<position cursor><CR>  (set pivot above a character
                        near the middle)
<CR>
```

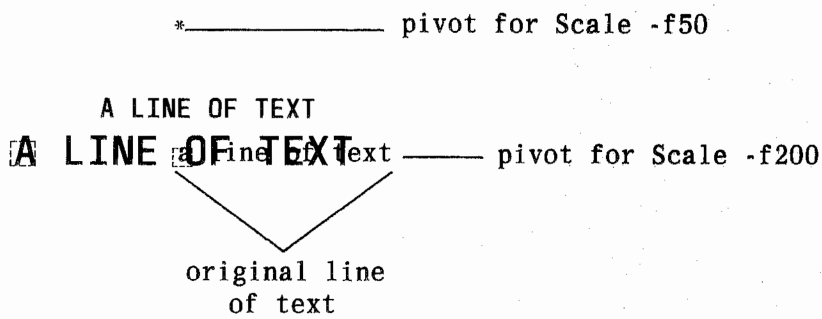

Figure 5-6. Scaling Text

A useful insight into the behavior of scaling is to note that the position of the pivot does not change. Also observe that the defined area is scaled to preserve its relationship to the graphical objects.

The size of objects can also be changed by moving point-handles. Generate a circle, this time using the **C**ircle command:

```
*Circle<CR>
<position cursor><SP>   (specify the center)
<position cursor><CR>   (specify a point on the circle)
```

The **C**ircle command generates an arc with the first and third point at the point specified on the circle. The second point of the arc is located 180 degrees around the circle. One way to change the size of the circle is to move one point-handle (using **M**ove **–p+**).

The following is an example of using **M**ove **–p**+:

1. Create a circle.

```
*Circle<CR>
<position cursor><SP>   (Point A Figure 5-7.)
<position cursor><CR>   (Point B Figure 5-7.)
```

2. Use the **p**oint command to mark the point-handles.

```
*point<CR>
<position cursor><CR>   (Point B Figure 5-7.)
<CR>
```

3. Move the circle by using the -p+ option.

```
*Move -p+<CR>
<position cursor><CR>   (point-handle, Point B)
<CR>   (pivot point B, Figure 5-7)
<position cursor><CR>   (Point C Figure 5-7.)
```

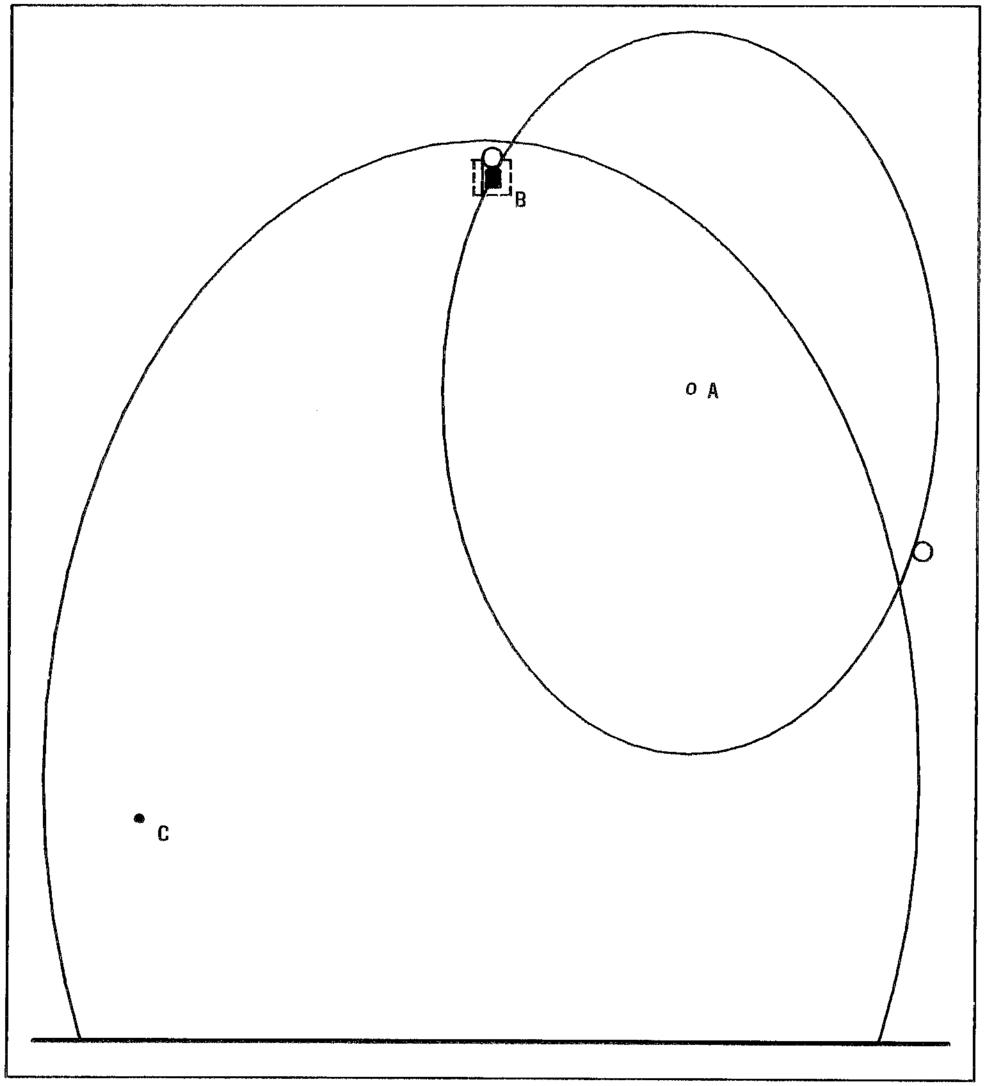

Figure 5-7. Example of Moving a Circle Using the Move −p+

The size of text characters can be changed via a third mechanism. Character height is the property of a line of text. The **Edit** command allows you to change the character height, shown in the following example:

1. Create a line of text.

   *Text<*CR*>
   A line of text.
   <position cursor><*CR*>

   See Figure 5-8 (small print) for a result of the drawing.

2. Use the **Edit** command to enlarge the text.

   *__Edit –h1000__<*CR*> (increase height to 1000.)
   <position cursor><*CR*> (point to the object-handle point A)
   <CR>

See Figure 5-8 (large print) for a result of the drawing.

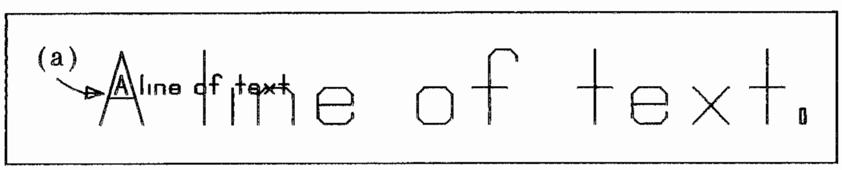

Figure 5-8. Example of Edit -h1000

Changing the Orientation of an Object

The orientation of an object can be altered using the **R**otate command. The **R**otate command rotates each point of an object about a pivot by an angle. Try the following rotations on a line of text (Figure 5-9).

```
*Rotate -a90<CR>  (angle is in degrees)
<position cursor><CR>   (point to object-handle)
<position cursor><CR>   (set pivot to rightmost
                  character)
<CR>

*Rotate -a-90<CR>
.<CR>    (reference previous defined area)
<position cursor><CR>    (set pivot to a character near
                  the middle)
<CR>
```

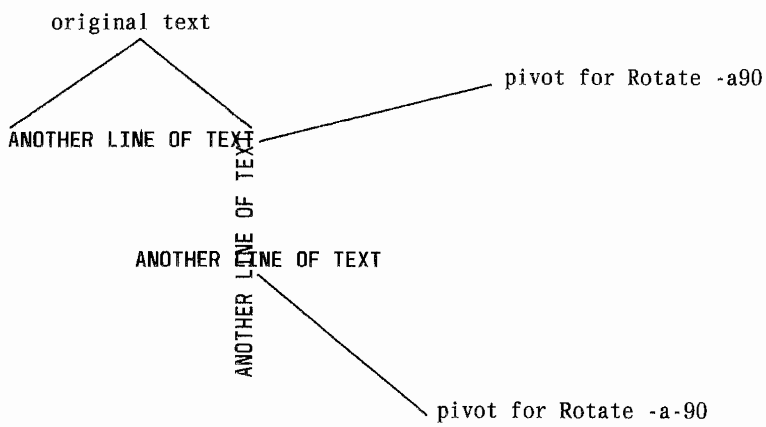

Figure 5-9. Rotating Text

Changing the Style and Width of Lines

In the current editor, objects can be drawn from lines in any of five styles: solid (**so**), dashed (**da**), dot-dashed (**dd**), long-dashed (**ld**), and three widths -- narrow (**n**), medium (**m**), and bold (**b**). Style is controlled by the **s** option and width by the **w**. The next example creates a narrow-dotted line:

```
*Lines -wn,sdo<CR>
<position cursor><SP>
<position cursor><SP>
<CR>
```

Using the **E**dit command, the line can be changed to bold, dot-dashed:

```
*Edit -wb, sdd<CR>
$.0<CR>     (reference the object-handle of the previous line)
<CR>        (complete the defined area)
<CR>
```

VIEW COMMANDS

All the objects drawn lie within a Cartesian plane, 65,534 units on each axis, known as the *universe*. Thus far, only a small portion of the universe has been displayed on the screen. The command:

```
*view -u<CR>
```

displays the entire universe.

WINDOWING

A mapping of a portion of the universe onto the display screen is called a *window.* The extent or magnification of a window is altered using the **zoom** command. To build a window that includes all the objects drawn, type:

```
*zoom<CR>
<position cursor><SP>     (above and to the left of all
                           the object)
<position cursor><CR>     (below and to the right, also
                           end points)
<CR>                       (verify)
```

Zooming can be either *in* or *out.* Zooming in, as with a camera lens, increases the magnification of the window. The area outlined by *points* is expanded to fill the screen. Zooming out decreases magnification. The current window is shrunk so that it fits within the defined area. The direction of the zoom is controlled by the sense of the **out** flag; **o** true means zoom out.

The location of a window is altered using the **view** command. **View** moves the window so that a given point in the universe lies at a given location on the screen.

```
*view<CR>
<position cursor><CR>     (locate a point in the universe)
<position cursor><CR>     (locate a point on the screen)
```

View also provides access to several predefined windows. As seen earlier, **view –u** displays the entire universe. The **view –h** command displays the *home-window.* The home-window is the window that encircles all the objects in the universe. The result is similar to that of the example using **zoom** that was given earlier.

Lastly, the **view** command permits selection of a window on a particular *region*. The universe is partitioned into 25 equal-sized regions. Regions are numbered from 1 through 25, beginning at the lower left and proceeding toward the upper right. Region 13, the center of the universe, is used as the default region by drawing commands such as **plot**(1) and **vtoc**(1).

OTHER COMMANDS

Interacting with Files

The **write** command saves the contents of the display buffer by copying it to a file:

```
*write filename<CR>
```

The contents of *filename* will be a GPS. Thus, it can be displayed using any of the device filters (such as, **td** (1)) or read back into **ged**.

A GPS is read into the editor using the **read** command:

```
*read filename<CR>
```

The GPS from *filename* is appended to the display buffer and then displayed. Because **read** does not change the current window, only some (or none) of the objects read may be visible.

A useful command sequence to view everything read is:

```
*read -e- filename<CR>
*view -h<CR>
```

The display function of **r**ead is inhibited by setting the **e**cho flag to false; **v**iew **–h** windows on and displays the full display buffer.

The **r**ead command may also be used to input text files. The form is:

```
read [-option(s)] filename<CR>
```

Followed by a single point to locate the first line of text. A text object is created for each line of text from *filename*. Options to the **r**ead command are the same as those for the **T**ext command.

EXAMPLE OF EDITING A GPS IN THE GRAPHICS EDITOR

Note: From the label command example in Chapter 4, you will observe that the label for the y-axis was not in the correct position. In this example, we will position the label in the correct position by using the graphics editor (**ged**).

1. Instead of displaying the drawing after appending the label file to the histogram, you must direct the output to a file so that it can be read into the graphics editor (**ged**).

```
^label -Flab,h,r90,y Randplot >Q<CR>
```

2. Enter the graphics editor (**ged**) and read the file **Q** into the graphics editor (**ged**). Then, observe the drawing by using the **view** command.

```
^ged<CR>
*read -e- Q<CR>
*view -h<CR>
```

The same drawing that is shown in Figure 4-5 will be seen.

3. Now, you can use the **m**ove command to position the label in the correct position.

```
*Move<CR>
<position cursor><CR>     (point A, Figure 5-10(a))
<CR>     (point A, Figure 5-10(a))
<position cursor><CR>     (point B, Figure 5-10(a))
```

4. Use the **n**ew command and observe that the label is in the correct position.

```
*new<CR>
```

The results of the drawing is shown in Figure 5-10(b)).

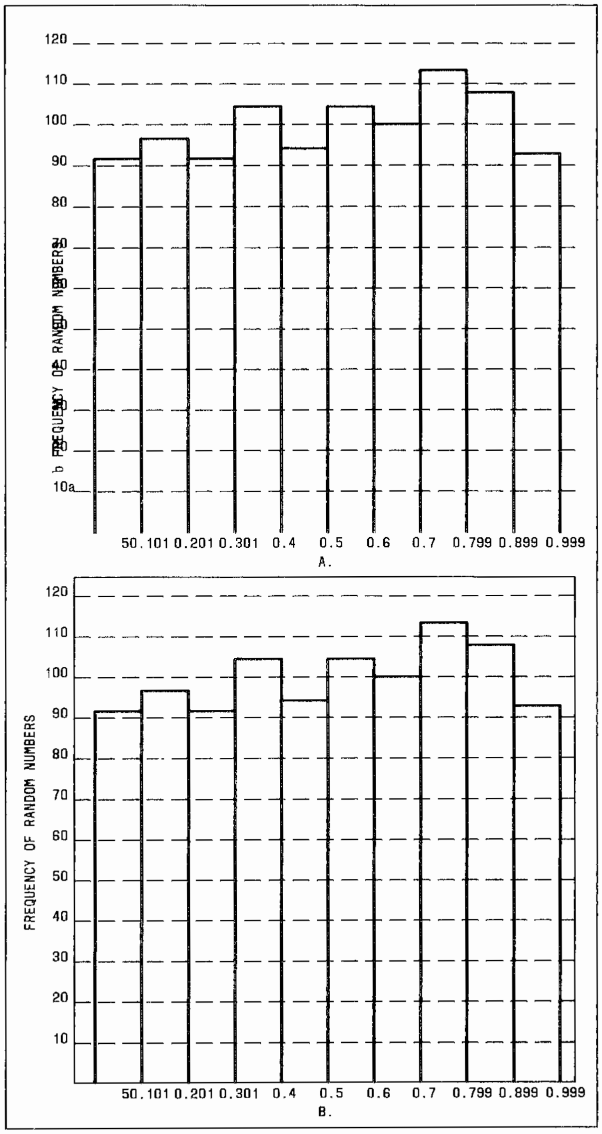

Figure 5-10. Example of Editing a GPS in the Graphics Editor

EXAMPLE OF CREATING MULTIDRAWINGS IN THE SAME UNIVERSE

The following is an example of creating an x-y graph and a histogram in the graphics shell; then placing both drawings in the same universe by using the graphics editor (**ged**).

1. Create an x-y graph and direct it to file **A**; then, display the drawing on a TEKTRONIX 4014 display terminal.

```
^gas -s0,t10 | af " x^2" | plot >A<CR>
^td A<CR>
```

   The results of the drawing is shown in Figure 5-11(a).

2. Create a file of 100 random numbers, then break that file of 100 random numbers into intervals and counts by using the **bucket** command, and direct it to file **C**.

```
^rand -n100 | title -v" 100 random numbers" | qsort | bucket >C<CR>
```

3. Create a histogram of file **C** and display it on a TEKTRONIX 4014 display terminal.

```
^hist C | td<CR>
```

The results of the drawing is shown in Figure 5-11(b).

4. Now, direct the histogram to region 14.

```
^hist -r14 C >D<CR>
```

5. Enter the graphics editor and read in the two files containing the GPS for the x-y graph and histogram. Then, use the view command to observe the results.

```
^ged<CR>
*read -e- D<CR>
*read -e- A<CR>
*view -u<CR>
```

The results are shown in Figure 5-11(c).

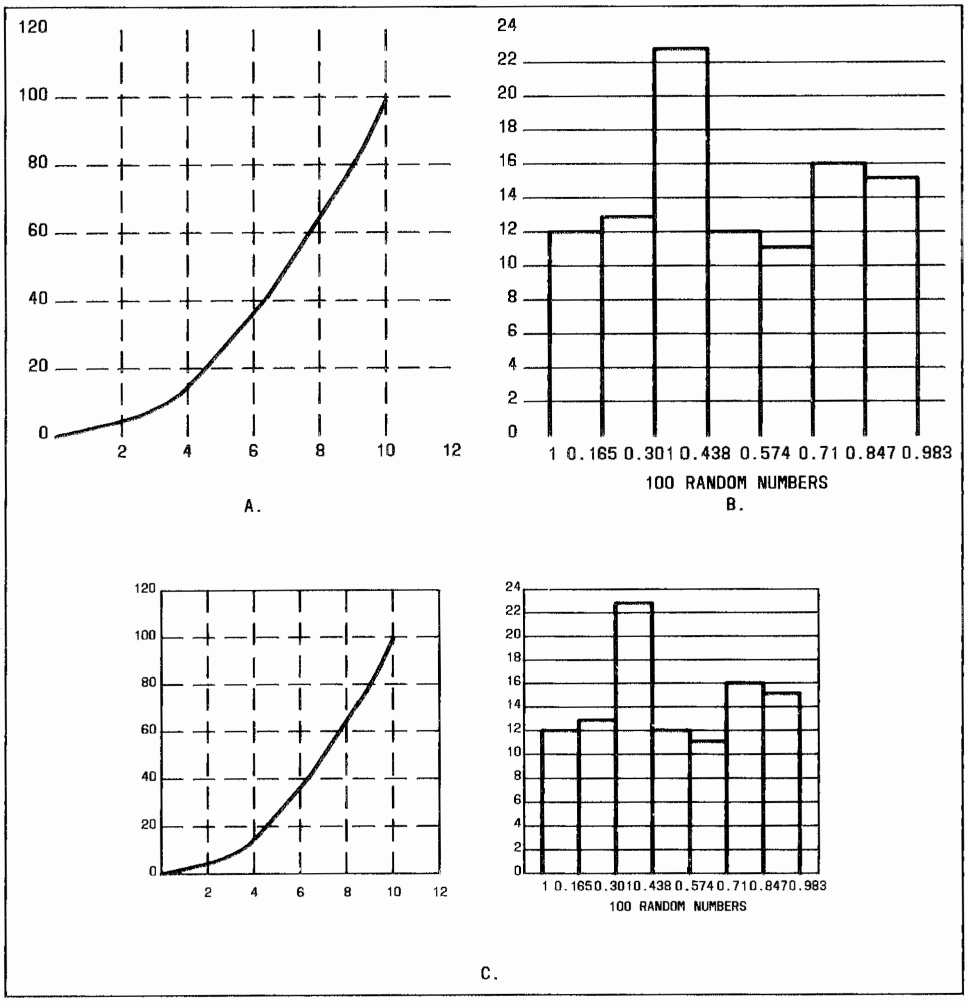

Figure 5-11. Creating a Multidrawing in the Same Screen

LEAVING THE GRAPHICS EDITOR

The **quit** command is used to end an editing session. As with the text editor **ed**, **quit** responds with **?** if the internal buffer has been modified since the last **write** command. A second **quit** command forces exit.

OTHER USEFUL INFORMATION

One-Line UNIX System Escape

As in **ed**, the **!** provides a temporary escape to the shell.

Typing Ahead

Most programs under the UNIX System allow input to be typed before the program is ready to receive it. In general, this is not the case with **ged**; characters typed before the appropriate prompt are lost.

Speeding up Things

Displaying the contents of the display buffer can be time consuming, particularly if much text is involved. The use of two flags to control what gets displayed can make life more pleasant:

- The **echo** flag controls echoing of new additions to the display buffer.

- The **text** flag controls whether text will be outlined or drawn.

COMMAND SUMMARY

In the summary, characters actually typed are printed in boldface. Command stages are printed in italics. Arguments surrounded by brackets (e.g., [...]) are optional. Parentheses surrounding arguments, separated by "or," means that exactly one argument must be given.

For example, the **D**elete command accepts the arguments **–u**niverse, **–v**iew, and *points.*

Construct Commands

Arc	[**–e**cho,**s**tyle,**w**idth] *points*
Box	[**–e**cho,**s**tyle,**w**idth] *points*
Circle	[**–e**cho,**s**tyle,**w**idth] *points*
Hardware	[**–e**cho] *text points*
Lines	[**–e**cho,**s**tyle,**w**idth] *points*
Text	[**–a**ngle,**e**cho,**h**eight,**m**idpoint,**r**ightpoint, **t**ext,**w**idth] *text points*

Edit Commands

Delete	(**–** (**u**niverse or **v**iew) or *points*)
Edit	[**–a**ngle,**e**cho,**h**eight,**s**tyle,**w**idth] (**–** (**u**niverse or **v**iew) or *points*)
Kopy	[**–e**cho,**p**oints,**x**] *points pivot destination*
Move	[**–e**cho,**p**oints,**x**] *points pivot destination*

Rotate	[–**a**ngle,**e**cho,**k**opy,**x**] *points pivot destination*
Scale	[–**e**cho,**f**actor,**k**opy,**x**] *points pivot destination*

View Commands

coordinates *points*	
erase	
new	
objects	(– (**u**niverse or **v**iew) or *points*)
points	(– (**l**abelled-points or **u**niverse or **v**iew) or *points*)
view	(– (**h**ome or **u**niverse or **r**egion) or [–**x**] *pivot destination*)
x	[–**v**iew] *points*
zoom	[–**o**ut] *points*

OTHER COMMANDS

quit	
read	[–**a**ngle,**e**cho,**h**eight,**m**idpoint,**r**ightpoint,**t**ext, **w**idth] *filename* [*destination*]
set	[–**a**ngle,**e**cho,**f**actor,**h**eight,**k**opy,**mi**dpoint, **p**oints,**r**ightpoint,**s**tyle,**t**ext,**w**idth,**x**]
write	*filename*

!*command*

?

OPTIONS

Options specify parameters used to build, edit, and view graphical objects. If a parameter, used by a command, is not specified as an *option*, the default value for the parameter will be used. The format of command *options* is:

–*option* [,*option*]

where *option* is *keyletter*[*value*]. Flags take on the values of true or false, shown by + and –, respectively. If no value is given with a flag, true is assumed. Object options are:

angle*n*	Specify an angle of *n* degrees.
echo	When true, changes to the display buffer will be echoed on the screen.
factor*n*	Specify a scale factor of *n* percent.
height*n*	Specify height of text to be *n* universe-units (*n* greater than or equal to 0 and less than 1280).
kopy	The commands **S**cale and **R**otate can be used to either create new objects or to alter old ones. When the **k**opy flag is true, new objects are created.
midpoint	When true, use the midpoint of a text string to locate the string.
out	When true, reduce magnification during **z**oom.

points	When true, operate on points; otherwise, operate on objects.
rightpoint	When **true**, use the rightmost point of a text string to locate the string.
style*type*	Specify line style to be of the following *types*: **so** solid **da** dashed **dd** dot-dashed **do** dotted **ld** long-dashed
text	Most text is drawn as a sequence of lines. This can sometimes be painfully slow. When the **t**ext flag (**t**) is false, strings are outlined rather than drawn.
width*type*	Specify line width to be of the following *types*: **n** narrow **m** medium **b** bold
x	One way to find the center of a rectangular area is to draw the diagonals of the rectangle. When the **x** flag is true, defined areas are drawn with their diagonals.

Area options are:

home	References the home-window
region*n*	References the region *n*
universe	Reference the universe-window
view	Reference those objects currently in view.

SOME EXAMPLES OF USING THE ged

The following examples are used to illustrate use of the **ged**.

Example 1--Text Centered Within a Circle

```
*Circle<CR>
<position cursor><SP>      (establish center)
<position cursor><CR>      (establish radius)
*Text -m<CR>       (text is to be centered)
some text<CR>
$.0<CR>            (first point from previous set,
                   i.e., circle center)
<CR>
```

Figure 5-12 shows the output of these commands.

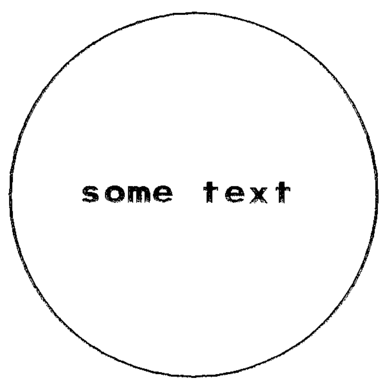

Figure 5-12. Text Centered Within a Circle

Example 2--Making Notes on a Plot

```
*! gas | plot -g >A<CR>  (generate a plot, put it in file A)
*read -e- A<CR>          (input the plot, but do not display it)
*view -h<CR>             (window on the plot)
*Lines -sdo<CR>          (draw dotted lines)
<position cursor><SP>  (0,6.5 y-axis)
<position cursor><SP>  (6.5,5.5)
<position cursor><SP>  (5.5,0 x-axis)
<CR>                   (end of Lines)
*set -h150,wn<CR>        (set text height to 150, line width to
                         narrow)
*Text -r<CR>   (right justify text)
threshold beyond that nothing matters<CR>
<position cursor><CR>  (set right point of text)
*Text -a-90<CR>  (rotate text negative 90 degrees)
threshold beyond that nothing matters<CR>
<position cursor><CR>  (set top end of text)
*x<CR>  (find center of plot)
<position cursor><SP>  (top left corner of plot)
<position cursor><CR>  (bottom right corner of plot)
*Text -h300,wm,m<CR>  (build title: height 300, weight
                      medium, centered)
SOME KIND OF PLOT<CR>
<position cursor><CR>  (set title centered above plot)
*view -h<CR>  (window on the resultant drawing)
```

Figure 5-13 shows the output of these commands.

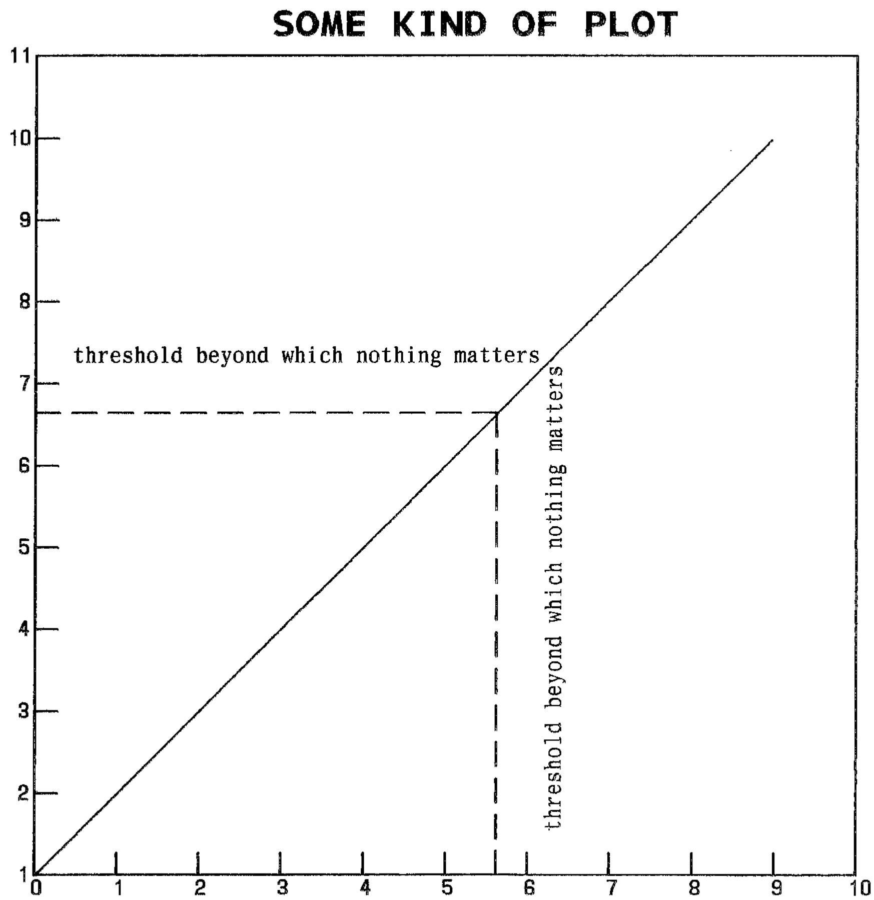

Figure 5-13. Making Notes on a Plot

Example 3--A Page Layout with Drawings and Text

```
*! rand –s1,n100 | title –v" seed 1" | qsort | bucket |
      hist –r12 >A<CR>        (put a histogram, region
            12, of 100 random numbers in file A)
*! rand –s2,n100 | title –v" seed 2" | qsort | bucket |
      hist –r13 >B<CR>        (put another histogram,
                  region 13, into file B)
*! ed<CR>  (create a file of text using the text editor)
a<CR>
On this page are two histograms<CR>
from a series of 40<CR>
designed to illustrate the weakness<CR>
of multiplicative congruential random number
generators.<CR>
.pl 3<CR>     (mark end of page)
.<CR>
w C<CR>     (put the text into file C)
151
q<CR>
*! nroff C | yoo C<CR>     (format C, leave the output
                  in C)
*view –u<CR>        (window on the universe)
*read –e– A<CR
*read –e– B<CR>
*view –h<CR>        (view the two histograms)
*read –h300,wn,m C<CR>   (text height 300, line weight
                  narrow, text centered)
<position cursor><CR>      (center text over two plots)
*view –h<CR>        (window on the resultant drawing)
```

Figure 5-14 shows the output of these commands.

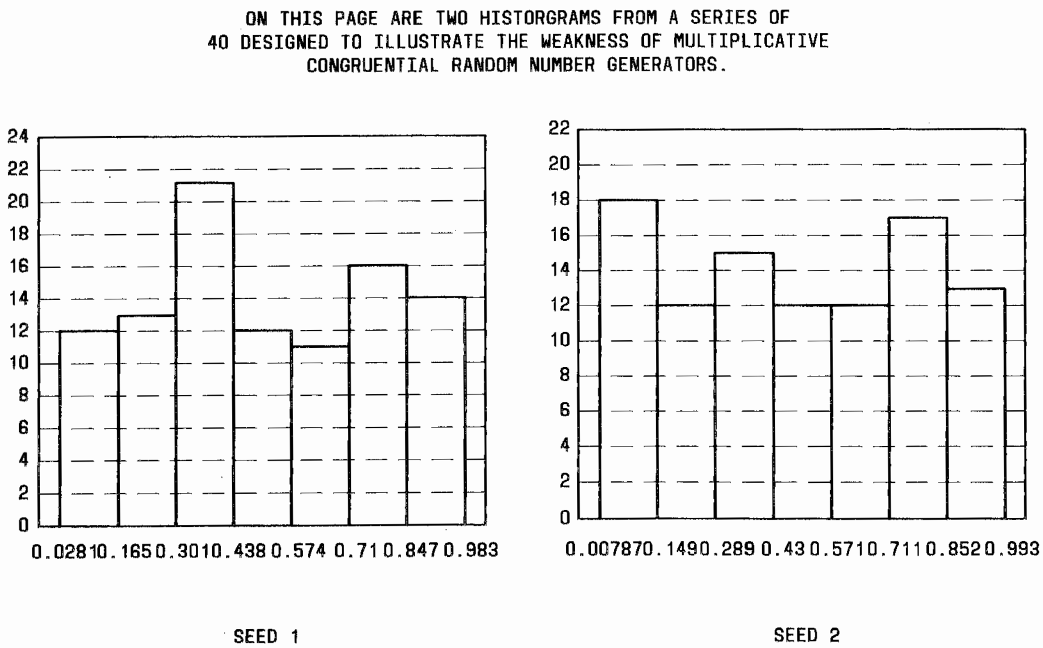

Figure 5-14. Page Layout with Drawings and Text