STAT—A Tool for Analyzing Data

A. R. Feuer
A. Guyton

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Stat is a collection of numerical programs under the UNIX† operating system that can be interconnected using the shell [1] to form processing networks. Included within *stat* are programs to generate simple statistics and pictorial output.

This paper introduces *stat* concepts and commands through a collection of examples. A complete definition of each command is given.

1. INTRODUCTION

Much of the power for manipulating text under UNIX comes from the numerous well defined text processing programs that can be readily interfaced to one another. The general interface is an unformatted text string and the interconnection mechanism is usually the shell [1]. Because the programs are independent from one another, new functions can easily be added and old ones changed. And because the text editor also 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 a 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 command-level (i.e., callable from shell) routines.

It is useful to think of the shell as a tool for constructing 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 links.

Section 2 of this paper is a introduction to the concepts of *stat*. Section 3 contains a description of each of the nodes. A few examples of *stat* usage are given in an appendix.

2. BASIC CONCEPTS

All numerical data in *stat* is of type *vector*. A *vector* is a sequence of numbers separated by delimiters. Vectors are processed by command-level routines called *nodes*.

2.1 Transformers

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

```
1 2 3 4 5
```

then the command

```
root A (typed input is bold)
```

produces

---

† UNIX is a trademark of Bell Laboratories.

```
1    1.41421    1.73205    2    2.23607
```

the square root of each input element. Analogously,

```
log A
```

produces

```
0    0.693147    1.09861    1.38629    1.60944
```

the natural logarithm of each element of vector *A*.

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"
```

which produces

```
2    8    18    32    50
```

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

2.2 Summarizers

A *summarizer* is a node that calculates a statistic for a vector. Typically, summarizers read in all of the input values, then calculate and output the statistic. For example, using the vector *A* from above,

```
mean A
```

produces

```
3
```

and

```
total A
```

produces

```
15
```

2.3 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. Thus

```
root -r3 A
```

produces

```
1    1.25992    1.44225    1.5874    1.70998
```

the cube root of each element from *A*.

2.4 Building Networks

Nodes are interconnected using standard shell concepts and syntax. Pipes are the linear connector attaching 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 simply

```
root -r3 A | mean
1.39991
```

Often the required network is not so simple. Tees and sequence can be used to build nonlinear networks. To find the mean and median of the transformed vector *A* is

```
root −r3 A | tee B | mean; point B
1.399
1.442
```

Beware of the distinction between the sequence operator, ";", and the linear connector, the pipe. Because processes in a pipeline run concurrently, each file name in the pipeline must be unique. Sequence implies run to completion (so long as "&" isn't used) hence names may be duplicated, and often are.

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

```
af "A−`mean A`"
−2    −1    0     1     2
```

2.5 Vectors, a Closer Look

Thus far we have used vectors, but not created them. 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 upon definable parameters. *Gas* is a generator that produces additive sequences. One of the parameters to *gas* is the number of elements in the generated vector. As an example, to create the vector *A* that we have been using is

```
gas −n5
1     2     3     4     5
```

Vectors are, however, merely text files. Hence we could use the text editor to create and modify the same vector.

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 of the fields surrounded by parentheses must be present.) Thus vector *A* could also be created by building the file **B** in the text editor as

```
1partridge,2tdoves,3frhens,4cbirds,5gldnrings,
```

which when read yields

```
list B
1     2     3     4     5
```

A note should be made as to the size of a vector: vectors are as long as they are. That is, a vector is a stream containing numbers terminated by an EOF (EOT from the keyboard). A good illustration of this is to use the keyboard as the source of the input vector, as in

```
cusum -c1
2   <return>
2
16.3  <return>
18.3
25.4  <return>
43.7
14  <return>
57.7
<cntrl d>
```

which implements a running accumulator. Since no vector was given to **cusum**, the input is taken from the standard input until an EOT.

2.6 A Simple Example: Interacting with a Data Base

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

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

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

```
grep "node count" data/* | qsort | tee A
17      19      22      32      39
50      68      78      125     139
```

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

```
cat data/* | grep "node count" | qsort | tee A
```

To get a feel for the distribution of node counts shell iteration can be used to advantage.

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

generates the lower hinge, the median, and the upper hinge of the sorted vector *A*.

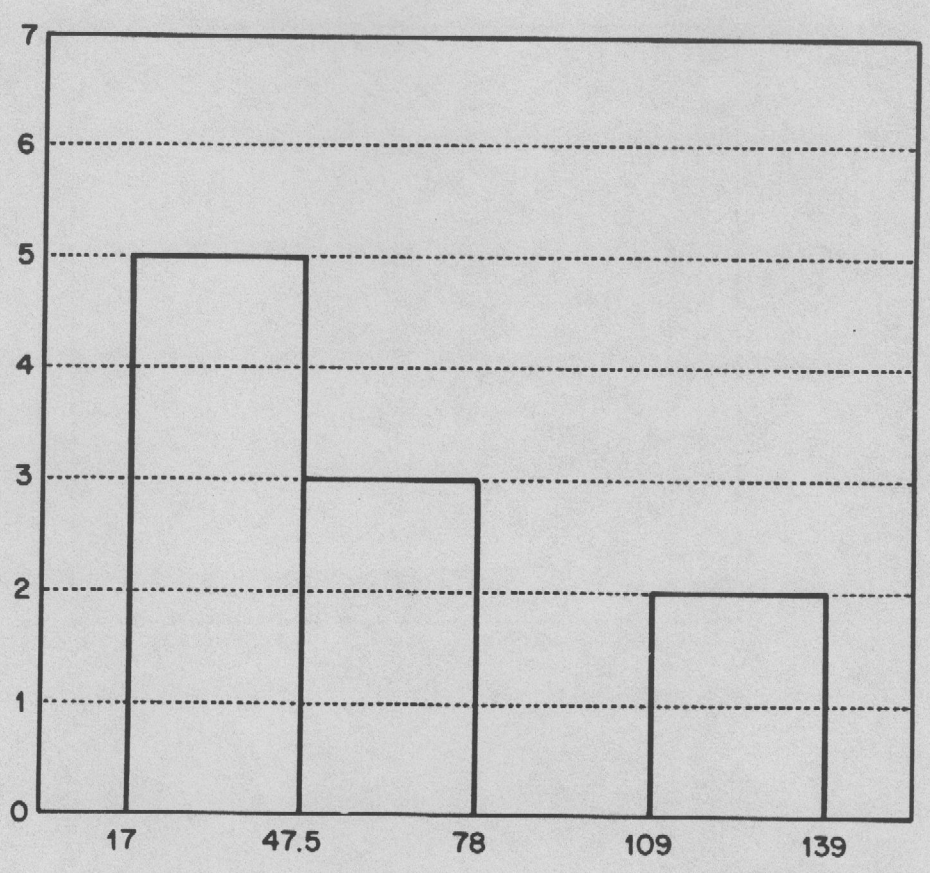
2.7 Translators

Translators are used to view data pictorially. A *translator* is a node that produce a stream of a different structure from that which it consumes. Graphical translators consume vectors and produce pictures in a language called GPS, for graphical primitive string. (Among the programs that understand GPS is *ged*, the graphical editor [2], which means that the graphical output of any translator can be directly edited at a display terminal.) **Hist** is an example of a translator; it 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 | td
```

generates a histogram of the data of vector *A* and displays it on a display terminal (Fig. 1). **Td** translates the GPS into machine code for Tektronix 4010 series display terminals.

Figure 1. bucket A | hist | td



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** (Fig. 2) is

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

A vector may be given a title using **title**. When a titled vector is plotted the appropriate axis is labeled with the vector title. 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**, i.e.,

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

the axis labels automatically agree with the vectors plotted (Fig. 3).

3. NODE DESCRIPTIONS

The *stat* nodes are divided into four classes: *transformers*, *summarizers*, *translators*, and *generators*. In this section a description of each node is given. The descriptions are organized by node class.

Figure 2. Scatter plot

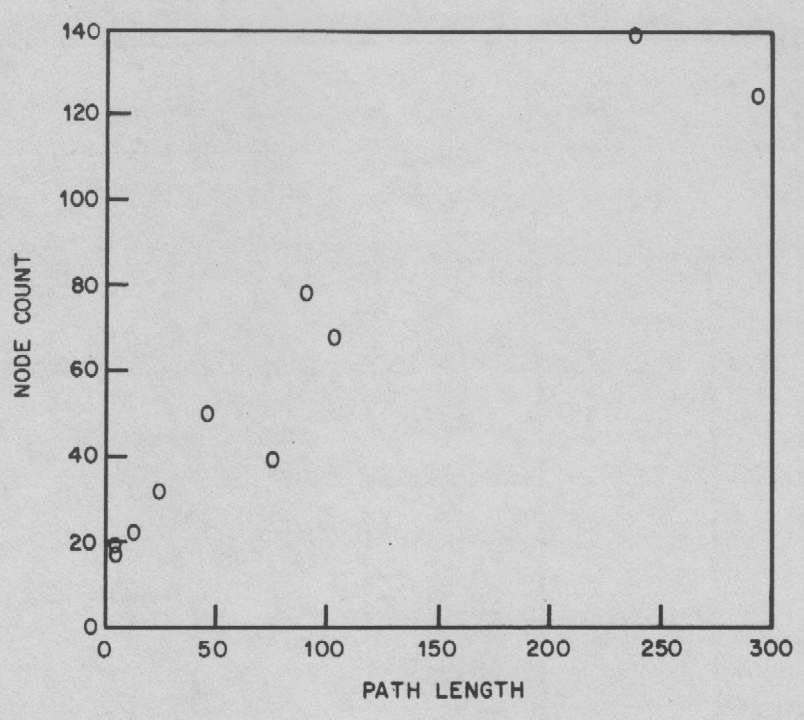
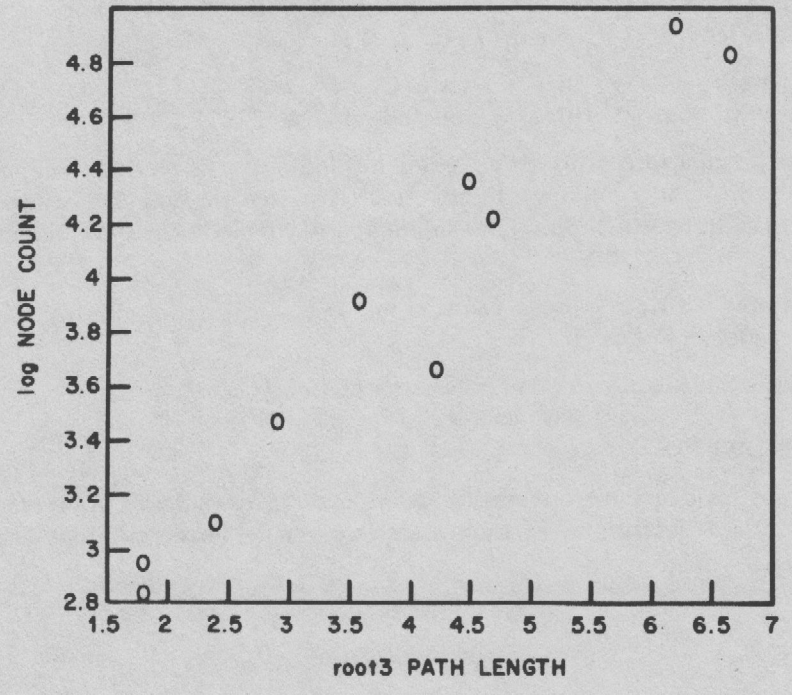


Figure 3. Transformed scatter plot



All of the 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 −, 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, hence it is not made explicit in the synopses that follow.

Most nodes accept command-line *options* to direct the execution of the node. Some *options* take values. In the following synopses, to indicate the type of value associated with an *option*, the *option* key-letter is followed by:

i	to indicate integer,
f	to indicate floating point or integer,
string	to indicate a character string, or
file	to indicate a *file-name*

Thus the *option* **c***i*, implies **c** expects an integer value (**c** := *integer*).

3.1 Transformers

Transformers have the form

V_{in} *transform* V_{out}

where, by convention, V_{in} is a vector Y, with elements y_1 through y_k ($y_{1:k}$) and V_{out} is a vector Z, $z_{1:m}$. All transformers have a **c***i* option, where **c** specifies the number of columns per line in the output. By default, **c** := 5.

abs − absolute value

$z_i := |y_i|$

af [−**t** **v**] − arithmetic function

The command-line format of **af** is an extension of the command-line description given above, with *expression* replacing *file-name*; an *expression* consists of *operands* and *operators*.

An *operand* is either a *vector*, *function*, *constant*, or *expression*:

A *vector* is	a file name with the restriction that file names begin with a letter, and are composed only of letters, digits, ".", and "_". The first unknown file name (one not in the current directory) references the standard input.
A *function* is	the name of a command followed by its arguments in parentheses. Arguments are written in command-line format.
A *constant* is	an integer or floating point (but not "E" notation) number.

The *operators* are listed below in order of decreasing precedence. Parentheses may be used to alter precedence. 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.
$X \hat{} Y \quad -Y$	x_i raised to the y_i power, negation of y_i. Association is right to left. 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.

Options:

t	causes the output to be titled from the vector on the standard input.
v	causes function expansions to be echoed.

ceil — ceiling

$z_i := $ smallest integer greater than y_i

cusum — cumulative sum

$$z_i := \sum_{j=1}^{i} y_j$$

exp — exponential function

$z_i := e^{y_i}$

floor — floor

$z_i := $ largest integer less than y_i

gamma — gamma function

$z_i := \Gamma(y_i)$

list [−**d***string*] — list vector elements

$z_i := y_i$

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.

log [−**b***f*] — logarithmic function

$z_i := \log_{\mathbf{b}} y_i$

By default, **b** $:= e$ $\quad (e \approx 2.71828\ldots)$

mod [−**m***f*] − modulus

$z_i := y_i$ modulo **m**

By default, **m** := 2

pair [−**F***file* **x***i*] − pair elements

F is a vector X, $x_{1:j}$, and **x** is the number of elements per group from X.
Let % denote modulo and / denote integer division, then

$$z_i := \begin{cases} y_{(i/(\mathbf{x}+1))} & \text{if } i\%(\mathbf{x}+1) = 0 \\ x_{(i-i/(\mathbf{x}+1))} & \text{if } i\%(\mathbf{x}+1) \neq 0 \end{cases}$$

$rank(Z) = (\mathbf{x}+1)minimum(k, j/\mathbf{x})$

If **F** is not specified, then X comes from the standard input.
If both X and Y come from the standard input, X precedes Y.

By default, **x** := 1

power [−**p***f*] − raise to a power

$z_i := y_i^{\mathbf{p}}$

By default, **p** := 2

root [−**r***f*] − extract a root

$z_i := \sqrt[\mathbf{r}]{y_i}$

By default, **r** := 2

round [−**p***i* **s***i*] − round off values

if **s** is specified, then

$z_i := y_i$ rounded up to **s** significant digits,

else if **p** is specified, then

$z_i := y_i$ rounded up to **p** digits beyond the decimal point.

By default, **p** := 0

siline [−**i***f* **n***i* **s***f*] − generate a line, given a slope and intercept

$z_i = \mathbf{s}\, y_i + \mathbf{i}$

if **n** is specified, then

$Y = 0, 1, 2, 3, \cdots, \mathbf{n}$.

By default, **i** := 0, **s** := 1

sin − sine function

$z_i := \sin(y_i)$

spline [−options] − interpolate smooth curve

Y and Z are sequences of X,Y coordinates (like that produced by **pair**).

For more information about **spline**, see *spline*(1) in the *UNIX User's Manual* [4].

subset [−**a***f* **b***f* **F***file* **i***i* **l***f* **nl np p***f* **s***i* **t***i*] − generate a subset

Z consists of elements selected from Y. Selection occurs as follows:

Let C(w) be true if

($w > \mathbf{a}$ or $w < \mathbf{b}$ or $w = \mathbf{p}$) and $w \neq \mathbf{l}$

is true. If neither **a**, **b**, nor **p** are specified, C(w) is true if $w \neq \mathbf{l}$ is true.

CASE 1 − **nl** or **np** not specified.

If **F** is specified, then $key_i = x_i$
else $key_i = y_i$.

For $r = \mathbf{s}, \mathbf{s}+\mathbf{i}, \mathbf{s}+2\mathbf{i}, \cdots$ with $r \leq \mathbf{t}$,
y_r becomes an element of Z if C(key_r) is true.

By default, **i** := 1, **s** := 1, **t** := 32767.

CASE 2 − **np** is specified.

F is a vector X, $x_{1:j}$.
For $r = x_1, x_2, \cdots, x_j$,
y_r becomes an element of Z if C(y_r) is true.

CASE 3 − **nl** is specified.

F is a vector X, $x_{1:j}$.
For $r \neq x_1, x_2, \cdots, x_j$,
y_r becomes an element of Z if C(y_r) is true.

For cases 2 and 3, if **F** is not specified then the standard input is used for X. Either X or Y may come from the standard input, but not both.

3.2 Summarizers

Summarizers have the form

V_{in} *summarize* V_{out}

where, again, V_{in} is a vector Y, $y_{1:k}$, and V_{out} is a vector Z, $z_{1:m}$. For many summarizers, $rank(Z) = 1$.

bucket [−**a***i* **c***i* **F***file* **h***f* **i***i* **l***f* **n***i*] − break into buckets

Y must be a sorted vector.
Z consists of odd elements (parenthesized) which are bucket limits and even elements which are bucket counts.

The count is the number of elements from Y greater than the lower limit (greater than or equal to for the lowest limit), and less than or equal to the higher limit. If specified, the limit values are taken from **F**. Otherwise the limits are evenly spaced between **l** and **h** with a total of **n** buckets. If **n** is not specified, the number of buckets is determined as follows:

$$\mathbf{n} := \begin{cases} \dfrac{\mathbf{h} - \mathbf{l}}{\mathbf{i}} & \text{if } \mathbf{i} \text{ is specified} \\ \dfrac{k}{\mathbf{a} + 1} & \text{if } \mathbf{a} \text{ is specified} \\ 1 + \log_2 k & \text{if neither } \mathbf{a} \text{ nor } \mathbf{i} \text{ are specified.} \end{cases}$$

c specifies the number of columns in the output.

By default:

c := 5
h := largest element of Y
l := smallest element of Y

cor [−**F***file*] − correlation coefficient

If **F** is a vector X, $x_{1:k}$, let $\bar{x} = \frac{\sum_{i=1}^{k} x_i}{k}$ and $\bar{y} = \frac{\sum_{i=1}^{k} y_i}{k}$, then

$$z_1 := \frac{\sum_{i=1}^{k} (x_i - \bar{x})(y_i - \bar{y})}{\left[\sum_{i=1}^{k} (x_i - \bar{x})^2\right]^{1/2} \left[\sum_{i=1}^{k} (y_i - \bar{y})^2\right]^{1/2}}$$

X and Y must have the same rank. If **F** is not specified, the standard input is used for X. If both X and Y come from the standard input, X precedes Y.

hilo [−**h l o ox oy**] − high and low values

z_1 := lowest value across all input vectors

z_2 := highest value across all input vectors

Options to control output:

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

lreg [−**F***file* **i o s**] − linear regression

If **F** is a vector X, $x_{1:k}$, let $\bar{x} = \frac{\sum_{i=1}^{k} x_i}{k}$ and $\bar{y} = \frac{\sum_{i=1}^{k} y_i}{k}$, then

$$z_1 := \bar{y} - z_2\bar{x} \qquad \text{(intercept)}$$

and

$$z_2 := \frac{\frac{\sum_{i=1}^{k} x_i y_i}{k} - \bar{x}\bar{y}}{\frac{\sum_{i=1}^{k} x_i^2}{k} - \bar{x}^2} \qquad \text{(slope)}$$

X and Y must have the same rank.
If **F** is not specified, then

$$X = 0, 1, 2, \cdots, k$$

Options to control output:

i	Only output the intercept.
o	Output the slope and intercept in option form (suitable for **siline**).
s	Only output the slope.

mean [−**f***f* **n***i* **p***f*] − (trimmed) mean

$$z_1 := \frac{\sum_{i=1}^{k} y_i}{k}$$

Y may be trimmed by

(1/**f**) k	elements from each end,
p *k*	elements from each end, or
n	elements from each end.

By default, **n** := 0

point [−**f***f* **n***i* **p***f* **s**] − empirical cumulative density function point

z_1 := linearly interpolated Y value corresponding to the

100 (1/**f**)	percent point, the
100 **p**	percent point, or the
nth	element.

Negative option values are taken from the high end of Y. Option **s** implies Y is sorted.

By default, **p** := .5 (median)

prod − product

$$z_1 := \prod_{i=1}^{k} y_i$$

qsort [−**c***i*] − quicksort

z_i := *i*th smallest element of Y.

By default, **c** := 5

rank − rank

z_1 := number of elements in Y.

total − sum

$$z_1 := \sum_{i=1}^{k} y_i$$

var − variance

$$z_1 := \frac{\sum_{i=1}^{k} (y_i - \bar{y})^2}{k - 1}$$

3.3 Translators

Translators have the form

F_{in} *translate* F_{out}

where F_{in} may be a vector or a GPS depending upon the translator. F_{out} is a GPS. A *GPS* (Graphical Primitive String) 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. Various commands exist that can display and edit a GPS. For more information, see *graphics*(1) in the *UNIX User's Manual* [4] and *UNIX Graphics Overview* [3].

bar [**—a b f g r***i* **w***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] — build a bar chart

F_{in} is a vector, each element of which defines the height of a bar. By default, the x-axis will be labeled with positive integers beginning at 1; for other labels, see **label**.

Options:

a	Suppress axes.
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*i*	Put the bar chart in GPS region *i*, where *i* is between 1 and 25 inclusive. The default is 13.
w*i*	*i* 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*f* (**y***f*)	Position the bar chart in the GPS universe with x-origin (y-origin) at *f*.
xa (**ya**)	Do not label x-axis (y-axis).
yl*f*	*f* is the y-axis low tick value.
yh*f*	*f* is the y-axis high tick value.

hist [**—a b f g r***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] — build a histogram

F_{in} is a vector (of the type produced by **bucket**) of odd rank, with odd elements being limits and even elements being bucket counts.

Options:

a	Suppress axes.
b	Plot histogram with bold weight lines, otherwise use medium.
f	Do not build a frame around plot area.
g	Suppress background grid.
r*i*	Put the histogram in GPS region *i*, where *i* is between 1 and 25 inclusive. The default is 13.
x*f* (**y***f*)	Position the histogram in the GPS universe with x-origin (y-origin) at *f*.
xa (**ya**)	Do not label x-axis (y-axis).
yl*f*	*f* is the y-axis low tick value.
yh*f*	*f* is the y-axis high tick value.

label [**—b c F***file* **h p r***i* **x xu y yr**] — label the axis of a GPS file

F_{in} is 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. Blank lines yield null labels. Either the GPS or the *label file*, but not both, may come from the standard input.

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*i*	Labels are rotated *i* degrees. The pivot point is the first character.
x	Label the x-axis. This is the default.
xu	Label the upper x-axis, i.e., the top of the plot.
y	Label the y-axis.
yr	Label the right y-axis, i.e., the right side of the plot.

pie [**−b o p pn***i* **pp***i* **r***i* **v x***i* **y***i*] − build a pie chart

F_{in} is a vector with a restricted format. Each input line represents a slice of pie and is of the form:

[< **i e f c***color* >] value [label]

with brackets indicating optional fields. The control field options have the following effect:

i	The slice will not be drawn, though a space will be left for it.
e	The slice is "exploded," or moved away 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.

Options:

b	Draw pie chart in bold weight lines, otherwise use medium.
o	Output values around the outside of the pie.
p	Output *value* as a percentage of the total pie.
pn*i*	Output *value* as a percentage, but total of percentages equals *i* rather than 100. **pn**100 is equivalent to **p**.
pp*i*	Only draw *i* percent of a pie.
r*i*	Put the pie chart in region *i*, where *i* is between 1 and 25 inclusive. The default is 13.
v	Do not output values.
x*i* (**y***i*)	Position the pie chart in the GPS universe with x-origin (y-origin) at *i*.

plot [**−a b c***string* **d f F***file* **g m r***i* **x***f* **xa xh***f* **xi***f* **xl***f* **xn***i* **xt y***f* **ya yh***f* **yi***f* **yl***f* **yn***i* **yt**] − plot a graph

F_{in} is a vector(s) which contains the y values of an x-y graph. Values for the x-axis come from **F**. Axis scales are determined from the first vector plotted.

Options:

a	Suppress axes.
b	Plot graph with bold weight lines, otherwise use medium.
c*string*	The character(s) of *string* are used to mark points. Characters from *string* are used, in order, for each separately plotted graph included in the plot. If the number of characters in *string* is less than the number of plots, the last character will be used for all remaining plots. The **m** option is implied.
d	Do not connect plotted points, implies option **m**.
f	Do not build a frame around plot area.
F*file*	Use *file* for x-values, otherwise the positive integers are used. This *option* may be used more than once, causing a different set of x-values to be paired with each input vector. If there are more input vectors than sets of x-values, the last set applies to the remaining vectors.
g	Suppress the background grid.
m	Mark the plotted points.
r*i*	Put the graph in GPS region *i*, where *i* is between 1 and 25 inclusive. The default is 13.
x*f* (**y***f*)	Position the graph in the GPS universe with x-origin (y-origin) at *f*.
xa (**ya**)	Omit x-axis (y-axis) labels.
xh*f* (**yh***f*)	*f* is the x-axis (y-axis) high tick value.
xi*f* (**yi***f*)	*f* is the x-axis (y-axis) tick increment.
xl*f* (**yl***f*)	*f* is the x-axis (y-axis) low tick value.
xn*i* (**yn***i*)	*i* is the approximate number of ticks on the x-axis (y-axis).

xt (yt) Omit x-axis (y-axis) title.

title [−**b c l***string* **v***string* **u***string*] − title a vector or GPS

F_{in} can be either a GPS or a vector with F_{out} being of the same type as F_{in}. **Title** prefixes a *title* to a vector or appends a *title* to a GPS.

Options apply as indicated:

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*.

3.4 Generators

Generators have the form

generate V_{out}

where V_{out} is a vector Z, $z_{1:k}$. All generators have a **c***i* option where **c** specifies the number of columns per line in the output. By default, **c** := 5.

gas [−**i***f* **n***i* **s***f* **t***f*] − generate additive sequence

Z is constructed as follows:

$$z_1 := \mathbf{s}$$

$$z_{i+1} := \begin{cases} z_i + \mathbf{i} & \text{if } |z_i| \leq \mathbf{t} \\ z_1 & \text{otherwise} \end{cases}$$

$rank(Z) = \mathbf{n}$.

By default, **i** := 1, **n** := 10, **s** := 1, **t** := ∞

prime [−**h***i* **l***i* **n***i*] − generate prime numbers

The elements of Z are consecutive prime numbers with

$$\mathbf{l} \leq z_i \leq \mathbf{h}$$

$rank(Z) \leq \mathbf{n}$.

By default, **n** := 10, **l** := 2, **h** := ∞

rand [−**h***f* **l***f* **m***f* **n***i* **s***i*] − generate random sequence

The elements of Z are random numbers generated by a multiplicative congruential generator with **s** acting as a seed, such that

$$\mathbf{l} \leq z_i \leq \mathbf{h}$$

If **m** is specified, then

$$\mathbf{h} = \mathbf{m} + \mathbf{l}$$

$rank(Z) = \mathbf{n}$.

By default, **h** := 1, **l** := 0, **n** := 10, **s** := 1

REFERENCES

[1] S. R. Bourne. *An Introduction to the UNIX Shell*, Bell Laboratories.

[2] A. R. Feuer. *A Tutorial Introduction to the Graphical Editor*, Bell Laboratories.

[3] A. R. Feuer. *UNIX Graphics Overview*, Bell Laboratories.

[4] T. A. Dolotta, S. B. Olsson, and A. G. Petruccelli (eds.). *UNIX User's Manual*—Release 3.0, Bell Laboratories (June 1980).

APPENDIX

- Example 1:

PROBLEM

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

SOLUTION

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

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

NOTES

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

Given are three ordered vectors (*A*, *B*, and *C*) of scores from a number of tests. 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 indicated by the value −1. Generate three new vectors containing scores only for those tests where no data is missing.

SOLUTION

```
echo Before:
gas -n`rank A` | tee N | af "label,A,B,C"

for i in N B C A
do subset -FA,l-1 $i >s$i; done
for i in N A C B
do subset -FsB,l-1 s$i | yoo s$i; done
for i in N A B C
do subset -FsC,l-1 s$i | yoo s$i; done

echo "\nAfter:"
af "sN,sA,sB,sC"
```

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

The approach is to eliminate those elements in all vectors that correspond to −1 in the base vector. Each of the three vectors takes turn at being the base. It is important that the base be subsetted last. The command **yoo** (see *gutil*(1) [4]) takes the output of a pipeline and copies it into one of the files 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.

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

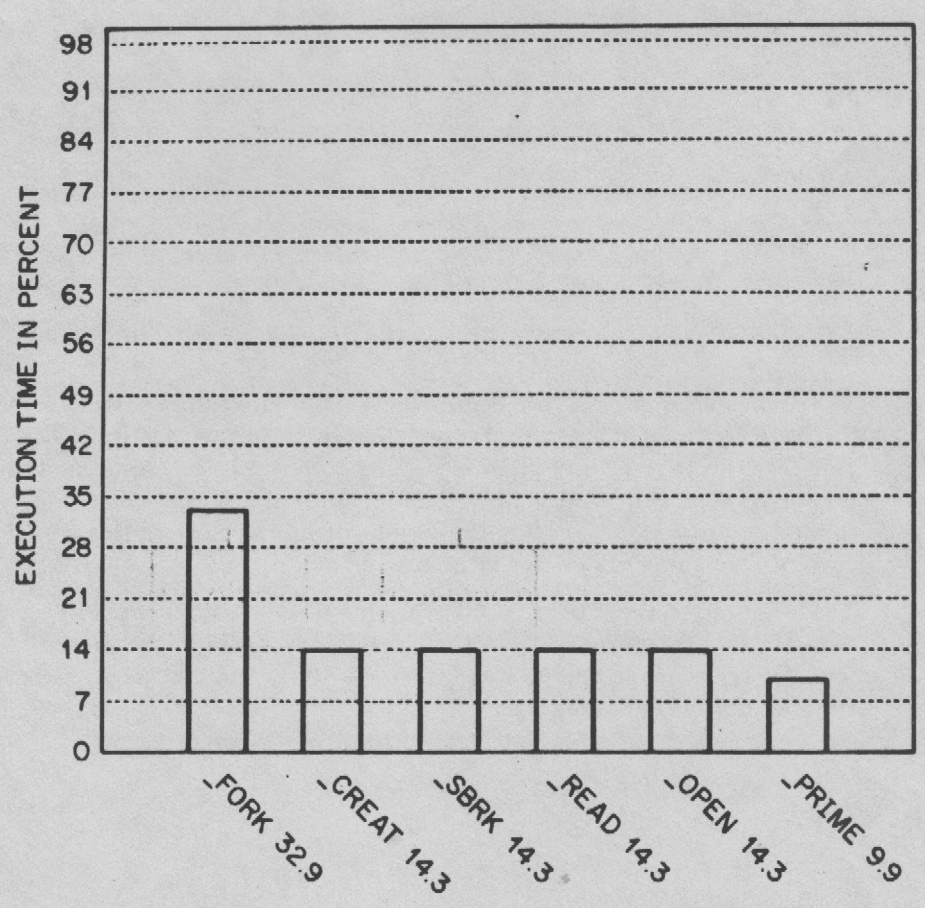
```
prof | cut −c1 −15 | sed −e 1d −e "/ 0.0/d" −e "s/^  *//" >P
echo These are the execution percentages; cat P
title P −v"execution time in percent" | bar −xa −yl0,yh100 |
        label −br−45,FP | td
```

These are the execution percentages
_fork 32.9
_creat 14.3
_sbrk 14.3
_read 14.3
_open 14.3
_prime 9.9

NOTES

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

Figure a1



- Example 4:

PROBLEM

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

SOLUTION

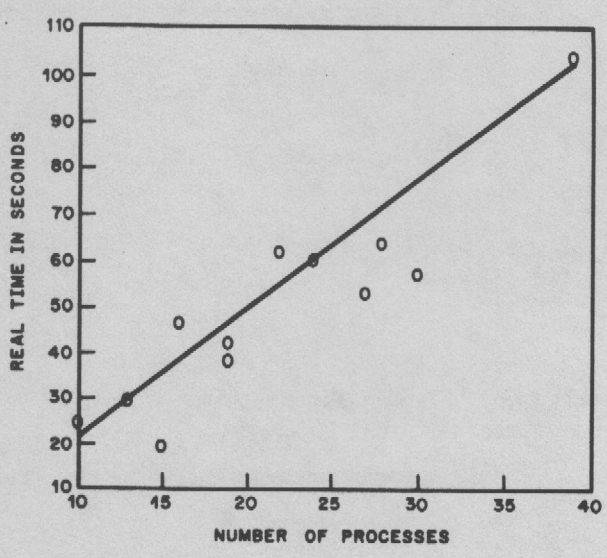
```
# The first program generates the performance data

for i in `gas -n12`
do
        ps -ae | wc -l >>Procs&
        time prime -n1000 >/dev/null 2>>Times
        sleep 300
done

# The second program analyzes and plots the data

for i in real user sys
do
        grep $i Times | sed "s/$i//" |
                awk -F: "{ if(NF==2) print \$1*60+\$2; else print }" |
                        title -v"$i time in seconds" >$i
        siline -`lreg -o,FProcs $i` Procs >$i.fit
done
title -v"number of processes" Procs | yoo Procs

plot -dg,FProcs real -r12 >R12
plot -ag,FProcs real.fit -r12 >>R12
plot -dg,FProcs sys -r13 >R13
plot -ag,FProcs sys.fit -r13 >>R13
plot -dg,FProcs user -r8 >R8
plot -ag,FProcs user.fit -r8 >>R8
ged R12 R13 R8
```

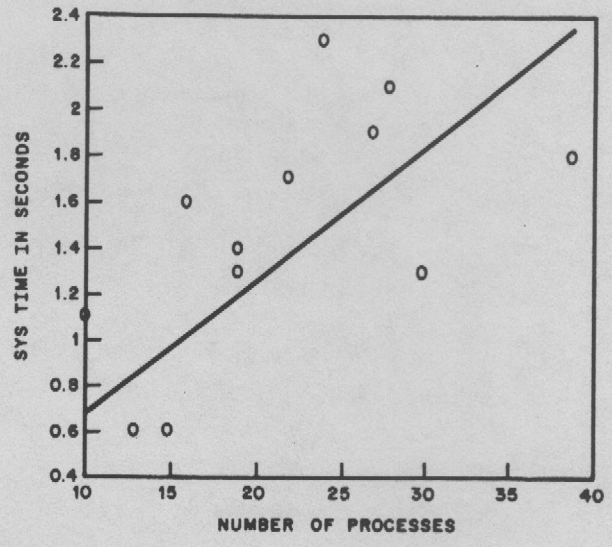
NOTES

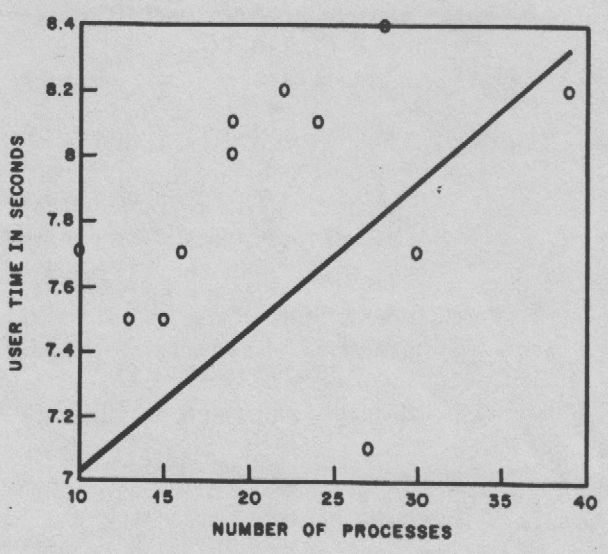
The performance data is the execution time, as reported by the UNIX **time** command, to generate the first 1000 prime numbers. **Time** outputs three times for each run: the time in system routines, the time in user routines, and the total real time. Each of these types of time is treated separately by the analysis program.

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 the graphical editor.

Figure a2


REAL TIME IN SECONDS
NUMBER OF PROCESSES


SYS TIME IN SECONDS
NUMBER OF PROCESSES


USER TIME IN SECONDS
NUMBER OF PROCESSES