UNIX|STAT: Data Analysis Programs for UNIX

Gary Perlman

ABSTRACT

I describe some programs for data analysis for use on the UNIX operating system. The programs fall into four classes: validation, transformations, description, and inference. These programs are compact and versions run on micro-computers as small as a PDP 11/23. The programs are designed to be used in pipelines and require little user interaction because the programs often are able to infer the design of the data from data file formats. An ordinary command involves a pipeline of transformations terminated by an analysis command. Most users familiar with statistics learn to do simple analyses in a few seconds, and complex analyses inside an hour.

1. INTRODUCTION

In this paper I demonstrate some programs available on UNIX* for analyzing data. The programs were written by me while I was at the University of California at San Diego between 1979 and 1982. Collectively, they are called UNIX|STAT†, so named because of their heavy use of the "|" symbol for the UNIX "pipe" operation. The programs have been designed to be easy to use, and small enough to fit on mini computers such as DEC PDP†† 11/45's, 11/34's, and 11/23's. I will first describe the format prescribed for the programs, and how people use the programs in the UNIX environment. Although this document might be a sufficient introduction to the facilities for analyzing data, it is not complete and should not be used as a substitute for the reference manuals for selected programs (DM and CALC), nor for the manual entries on all the programs (which can be printed by typing "man program" in the shell). In general, the on-line documentation is more up to date than the printed documentation.

The programs are of several different types:

Data Transformation. These programs are useful for changing the format of data files, for transforming data, and for filtering unwanted data. In addition, one program is useful for monitoring the progress of the data transformations. Programs described: IO, ABUT, DM, TRANSPOSE, SERIES, REPEAT, PERM, MAKETRIX.

Data Validation. These include programs for checking the number of columns in data files and their types (e.g., alphanumeric, integer). Programs described: VALIDATA, DM.

Descriptive Statistics. These procedures include both numerical statistics, and simple graphical displays. There are procedures for single distributions, paired data, and multivariate cases. Programs described: DESC, PAIR(BIPLOT), CORR.

Inferential Statistics. These include multivariate linear regression and multi-factor analysis of variance. Some simple inferential statistics are also incorporated into the descriptive statistics programs, but are used less often. Programs described: REGRESS(CORR), ANOVA, PAIR, POF, CRITF.

Other Programs. Some programs peculiar to psychological data analysis are included in the UNIX|STAT distribution tape, but are not described here. For mathematical psychology fans there is a d'

---

* UNIX is a trademark of Bell Laboratories.

† UNIX|STAT is not a product nor supported by Bell Laboratories.

†† PDP is a trademark of Digital Equipment Corporation.

analysis program (DPRIME) and a vincentizing program (VINCENT).

Table of Programs Described.

ABUT	abut files beside each other.
ANOVA	multi-factor anova with unequal cell sizes and repeated measures.
BIPLOT	bivariate plotting with options for summary statistics.
CALC	algebraic calculator.
CORR	multivariate linear correlation with summary statistics.
DESC	univariate statistics, frequency tables, and histograms.
DM	conditional transformations of data.
IO	control and monitor file input and output.
MAKETRIX	form a matrix from an unstructured file.
PAIR	bivariate summary statistics with options for scatterplots.
PERM	randomly permute lines
REGRESS	multivariate linear regression.
REPEAT	repeat a file or string
REVERSE	reverse lines, fields, or characters
SERIES	print a series of numbers
TRANSPOSE	transpose matrix type file (flip rows and columns).

All these programs run on the UNIX operating system, a program development and text editing facility developed at Bell Laboratories. UNIX is a highly interactive operating system, and because of this, it is an ideal environment for data analysis. Persons doing data analysis sit at a terminal and repeatedly specify program options and data on which these program act. They have immediate access to intermediate results, and can make analysis decisions based on them.

2. USING UNIX

This section describes the typical use of UNIX and is meant to give non-users an brief introduction to its use. It may also provide a useful summary for somewhat experienced users who have little experience with constructing complex commands with pipelines. UNIX users generally sit in front of a terminal at which they repeatedly specify a program, the input to that program, and to where the output from the program should be directed. They specify this program, input, and output to a program called a "shell." The shell is most users primary way of interacting with UNIX. If the user does not specify from where the input to a program is to be read, the default "standard input" is the user's terminal keyboard. (For data analysis programs, this often is a mistake.) Similarly, if unspecified, the default "standard output" is the terminal screen. To override these default standard input and outputs, UNIX shells provide simple mechanisms called "redirection" and "pipelining." To indicate that a program should read its input from a file rather than the terminal keyboard, a user can "redirect" the input from a file with the "<" symbol. (In all following examples, I will use the convention of capitalizing the names of programs and files, though in actual use, all these names would be lower case.) Thus,

```
PROGRAM < FILE
```

indicates to UNIX (really it indicates to the shell which controls input and output) that the input to the program named PROGRAM should be read from the file named FILE rather than the terminal keyboard. Analogously, the output from a program can be saved in a file be redirecting it to a file with the ">" symbol. Thus,

```
PROGRAM < INPUT > OUTPUT
```

indicates that the program PROGRAM should read its input from the file INPUT and put its output into a new file called OUTPUT. If the file INPUT does not exist, an error message will be printed. If the file OUTPUT exists, then whatever was in that file before will get destroyed. A mistake novices and experts alike should avoid is a command like:

```
PROGRAM < DATA > DATA
```

which might be used to replace the contents of data with whatever PROGRAM does to it. The effect of such a command is to destroy the contents of DATA before PROGRAM has a chance to read it. For a safer method of input and output, see the later discussion of the IO program.

The output from one program van be made the input to another program without the need for temporary files. This action is called "pipelining" or "piping" and is used to create one complex function from a series of simple ones. The vertical bar, or "pipe" symbol, "|," is placed between programs to pipe the output from one into the other. Thus,

```
PROGRAM1 < INPUT | PROGRAM2
```

tells UNIX to run the program PROGRAM1 on the file INPUT and feed the output to PROGRAM2. In this case, the final output would be printed on the terminal screen because the final output from PROGRAM2 is not redirected. Redirection could be accomplished with a command line like:

```
PROGRAM1 < INPUT | PROGRAM2 > OUTPUT
```

In general, only one input redirection is allowed, and only one output redirection is allowed, and the latter must follow the former. Any number of programs can be joined with piping.

In general, UNIX programs do not know if their input is coming from a terminal keyboard or from a file or pipeline. Nor do they generally know where their output is destined. One of the features of UNIX is that the output from one program can be used as input to another via a pipeline making it possible to make complex programs from simple ones without touching their program code. Pipelining makes it desirable to keep the outputs of programs clean of annotations so that they can be read by other programs. This has the unfortunate result that the outputs of many UNIX programs are cryptic and have to be read with a legend. The advantages of pipelining will be made clear in the examples of later sections.

3. THE MASTER DATA FILE

The key ideas of the format of the master data file are simplicity and self documentation. The reason for this is to make transformation of data easy, and to be able to use a wide variety of programs to operate on a master data file. Each line of a master data has the same number of alphanumeric fields. For readability, the fields can be separated by any amount of white space (blank spaces or tabs), and, in general, blank lines are ignored. Each line of a master data file corresponds to the data collected on one trial or series of trials of an experiment. Along with the data, a set of fields describe the conditions under which those data were obtained. Usually, a master data file contains all the data for an experiment. However, in many cases, a user would not want all the data from an experiment from this file to be input to a program. Some parts may be of particular interest for a specific statistical test, or some data may need to be transformed before input to a data analysis program.

4. PROGRAM DESCRIPTIONS

The programs described here were designed with the philosophy that data, in a simple format, can implicitly convey all or most of the information a program needs to analyze them. With data transforming utilities, the need for a special language to specify design information all but disappears. Users can implicitly specify design information by putting their data into specific formats. For the most part, the programs have few options, preferring to print more statistics than any one person might want, but removing the need for specifying options.

The strategy I will use here is to describe the programs available and give examples of how they are used. This is meant only to make the ideas of analysis with these programs familiar and should not be used as a substitute for the manual entries on individual programs. Only a few of the capabilities of

the programs are described.

4.0.1 **CALC***: An Algebraic Calculator*

CALC is a program that converts your computer and terminal into a $20.00 algebraic calculator. To use it, you just type its name, and CALC prompts you for expressions like:

```
12 + sqrt (50) / log (18)
hypotenuse = sqrt (a^2 + b^2)
```

for which it will print reasonable values. The only exception is recursive assignments which can make CALC blow up if you are not careful.

4.1 Transforming Data

In this section, I will describe programs for transforming data. The reason for describing these before statistical programs is that in most cases, user will want to transform their data before analyzing them. The general form of a command would thus be:

```
TRANSFORM < DATA | ANALYZE > OUTPUT
```

where TRANSFORM is some program to transform data from an input file DATA and the output from TRANSFORM is piped to an analysis program, ANALYZE whose output is directed to an output file, OUTPUT. One, ABUT, is used to take the data from several files, and put corresponding lines from those files on the same line of the output. Another, DM, is a data manipulator used to extract and transform columns of lines.

4.1.1 **ABUT***: Abut Files*

ABUT is a program to take several files, each with N lines, and make one file with N lines. This is useful when data from repeated measures experiments, such as paired data, are in separate files and need to be placed into one file for analysis (see PAIR and REGRESS). For example, the command:

```
ABUT FILE1 FILE2 FILE3 > FILE123
```

would create FILE123 with its first line the first lines of FILE1, FILE2, and FILE3, in order. Successive lines of FILE123 would have the data from the corresponding lines of the named files joined together.

4.1.2 **IO***: Control and Monitor Input/Output*

IO is a general program for controlling input and output of files. It can be used instead of the standard shell redirection mechanisms and is in some cases safer. It can also be used to monitor the progress of data analysis commands, some of which can take a long time.

4.1.2.1 Catenate Files.

IO has a similar function to ABUT. Instead of, in effect, placing the files named *beside* each other, IO places them one after another. A user may want to analyze the data from several files, and this can be accomplished with a command like:

```
IO FILE1 FILE2 FILE3 | PROGRAM
```

4.1.2.2 Monitoring Input and Output.

IO also can be used to monitor the flow of data between programs. When called with the **-m** flag, it acts as a meter of input and output flow, printing the percentage of its input that has been processed. (In UNIX, it is common for options to be specified to programs by preceding letters by a dash to distinguish them from file names.) The program can also be used in the middle of

pipelines, and at the end of pipelines to monitor the absolute flow of data, printing a special character for every 1000 bytes of data processed.

4.1.2.3 Input and Output Control.

Finally, it can be used as a safe form of controlling i/o to files, creating temporary files and copying rather than automatically overwriting output files. For example, IO can be used to sort a file onto itself with one pipeline using the standard UNIX SORT program:

```
IO -m FILE | SORT | IO -m FILE
```

The above command would replace FILE with a sorted version of itself. Because the monitor (**-m**) flag is used, the user would see an output like:

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

The percentages show the flow from FILE into the SORT program (percentages are possible because IO knows the length of FILE), and the equal signs indicate a flow of about a thousand bytes each coming out of the SORT program. The command:

```
SORT FILE > FILE
```

would destroy the contents of FILE before SORT had a chance to read it. The command with IO, is both safer and acts as a meter showing input and output progress.

4.1.2.4 Saving Intermediate Transformations.

In a command like:

```
IO FILE1 FILE2 | PROGRAM1 | PROGRAM2 | IO OUTPUT
```

the intermediate results before PROGRAM1 and before PROGRAM2 are lost. IO can be used to save them by diverting a copy of its input to a file before continuing a pipeline:

```
IO FILE1 FILE2 | IO IN1 | PROGRAM1 | IO IN2 | PROGRAM2 | IO OUTPUT
```

Any of these calls to IO could be made metering versions by using the **-m** flag.

4.1.3 **TRANSPOSE***: Transpose Matrix-Type Files*

TRANSPOSE is a program to transpose a matrix-like file. That is, it flips the rows and columns of the file. For example, if FILE looks like:

```
1 2 3 4
5 6 7 8
9 10 11 12
```

then the command:

```
TRANSPOSE FILE
```

will print:

```
1          5          9
2          6          10
3          7          11
4          8          12
```

TRANSPOSE is useful as a pre/post-processor for the data manipulator, DM, as it can read from the standard input stream when no files are supplied.

4.1.4 **SERIES***: Print a Series of Numbers*

SERIES is useful for creating some test data or dummy variables. You supply a starting

number and an ending number, and SERIES prints all the numbers in between. Optionally, the increments between the endpoints can be adjusted. Series can be in reverse order, so the command:

```
SERIES 20 10 | TRANSPOSE
```

produces something like:

```
20 19 18 17 16 15 14 13 12 11 10
```

4.1.5 **REPEAT***: Repeat a File or String*

REPEAT prints the named file the specified number of times, or the standard input if no file name is supplied. The number of repetitions is controlled by a numerical argument which, if negative, tells REPEAT to simply repeat the name, not the file. For example,

```
REPEAT "The rain in Spain falls mainly on the plain" -3
```

will produce:

```
The rain in Spain falls mainly on the plain
The rain in Spain falls mainly on the plain
The rain in Spain falls mainly on the plain
```

REPEAT can be used to generate random numbers when combined with DM, by providing DM with a specified number of input lines:

```
REPEAT "x" -100 | DM RAND
```

4.1.6 **REVERSE***: Reverse Lines, Fields, and Characters*

REVERSE is used to reorder lines, space separated fields, or characters in lines. This may be useful for DM if the fields you want to work with are in the final columns of a file. The options can be combined to provide interesting results.

4.1.7 **PERM***: Randomly Permute Lines*

PERM produces a random ordering of the lines in a file. PERM is usually used to randomize experimental conditions, but it can be used for Monte Carlo simulations with some work.

4.1.8 **MAKETRIX***: Create Matrix-Type File*

MAKETRIX is a program to create matrix type files for input to other UNIX|STAT programs. Its integer argument is the number of columns to form the file from. If FILE contains:

```
1 2 3 4 5 6 7 8 9 10 11 12

MAKETRIX 3 < FILE
```

will produce:

```
1          2          3
4          5          6
7          8          9
10         11         12
```

4.1.9 **DM***: A Data Manipulator*

DM is a data manipulating program that allows its user to extract columns (delimited by

white space) from a file, possibly based on conditions, and produce algebraic combinations of columns. DM is probably the most used of all the programs described in this paper. To use DM, a user writes a series of expressions, and, for each line of its input, DM reevaluates and prints the values of those expressions in order.

DM allows users to access the field of each line of its input. Numerical values of fields on a line can be accessed by the letter "x" followed by the column number. Character strings can be accessed by the letter "s" followed by the column number. Consider for example the following contents of the file EX1:

```
12          45.2        red         ***
10          42          blue        ---
8           39          green       ---
6           22          orange      ***
```

The first line of EX1 has four columns, or fields. In this line, x1 is the number 12, and s1 is the string ’12’. DM distinguishes between numbers and strings (the latter are enclosed in quotes) and only numbers can be involved in algebraic expressions.

4.1.9.1 Column extraction.

Simple column extraction can be accomplished by typing the strings in the columns desired. To print, in order, the second, third, and first columns from the file EX1, one would use the call to DM:

```
DM s2 s3 s1 < EX1
```

This would have the effect of reordering the columns and removing column 4.

4.1.9.2 Algebraic Expressions.

DM can be used to produce algebraic combinations of columns. For example, the following call to DM will print the first column, the sum of the first two columns, and the square root of the second column.

```
DM x1 x1+x2 "sqrt(x2)" < EX1
```

Note that the parentheses in the third expression requires quotes around the whole expression. This is because parentheses are special characters in the shell. If a string in either of the columns was not a number, DM would print and error message and stop.

4.1.9.3 Conditional Operations.

DM can be used to filter out lines that are not wanted. A simple example would be to print only those lines with stars in them.

```
DM "if ’*’ C INPUT then INPUT else NEXT" < EX1
```

The above call to DM has one expression in quotes to overcome problems with special characters and spaces inserted for readability. The conditional has the syntax "if-then-else." Between the "if" and "then" is a condition that is tested, in this case, testing if the one-character string ’*’ is in INPUT, a special string holding the input line. If the condition is true, the expression between the "then" and "else" parts is printed, in this example, the input line, INPUT. If the condition is not true, then the expression after the "else" part is printed, in this case, NEXT is a special control variable that is not printed and causes the next line to be read.

4.2 Data Validation

Before analysis begins, it is a good idea to make sure data are entered correctly. The programs described in this sub-section are useful for verifying the consistency of data files. Individual analysis programs do their own verification, so the programs described are, in practice, used to help find errors picked up by specific analysis programs.

4.2.1 **VALIDATA***: Check Data Validity*

A master data file is assumed to have an equal number of fields per line. VALIDATA checks its input from the standard input or argument file and complains if the number of fields per line changes. After reading all its input, VALIDATA reports the number of entries of various data types for each column. The data types VALIDATA knows about include integer and real numbers, alphabetics and alphanumeric fields, and some others. The minimum and maximum values of each column are also reported. VALIDATA can be used to spot incorrect entries such as columns expected to be numerical that are not, and accidently entered invisible control characters.

4.2.2 **DM***: Range Checking on Columns*

DM can be used to verify data as well as transform it. For example, to check that all numbers in column three of a file are greater than zero and less than 100, the following call to DM would print all lines that did not display that property.

```
DM "if !(x3>0 & x3<100) then INPUT else NEXT"
```

If non-numerical data appeared in column three, DM would report an error.

4.3 Descriptive Statistics.

4.3.1 **DESC***: Describing a Single Distribution*

DESC can be used to analyze a single distribution of data. Its input is a series of numbers, in any format, so that numbers are separated by spaces, tabs, or newlines. Like most of these programs, DESC reads from the standard input.

4.3.1.1 Summary Statistics.

DESC prints a variety of statistics, including order statistics. Under option, DESC will print a t-test for any specified null mean.

4.3.1.2 Frequency Tables.

DESC can print frequency tables, or tables of proportions, with cumulative entries if requested. These tables will be formed based on the data so they are a reasonable size, or they can be formatted by the user who can specify the minimum value of the first interval, and the interval width. For example, the following command would print a table of cumulative frequencies and proportions (cfp) in a table with a minimum interval value of zero (m0) and an interval width of ten (i10).

```
DESC i10 m0 cfp < DATA
```

4.3.1.3 Histograms.

If requested, DESC will print a histogram with the same format as would be obtained with options used to control frequency tables. For example, the following command would print a histogram of its input by choosing an appropriate interval width for bins.

```
DESC h < DATA
```

The format of the histogram, as well as tables, can be controlled by setting options. The following line sets the minimum of the first bin to zero, and the interval width to ten, an appropriate histogram for grading exams.

```
DESC h i10 m0 < GRADES
```

4.3.2 **PAIR***: Paired Data Analysis*

PAIR can be used to analyze paired data. Its input is a series of lines, two numbers per line, which it reads from the standard input. Options are available for printing a bivariate plot, which is the default when the program is called by its alias, BIPLOT. Other options control the type of output.

4.3.2.1 Summary Statistics.

From PAIR's input, minima, maxima, means, and standard deviations are printed for both columns as well as their difference. Also printed is the correlation of the two columns and the regression equation relating them. The simplest use of PAIR is with no arguments. To analyze a data file of lines of X-Y pairs, the following command will in most cases be satisfactory:

```
PAIR < DATA
```

Often the paired data to be analyzed are in two files, each variable occupying a single column. These can be joined with ABUT and input to PAIR via a pipe:

```
ABUT VAR1 VAR2 | PAIR
```

Or perhaps the two variables occupy two columns in a master data file. If the variables of interest are in columns four and six, the following command would produce the paired data analysis:

```
IO DATA | DM s4 s6 | PAIR
```

4.3.2.2 Scatterplots.

With the "p" or "b" options, a scatterplot of the two variables can be printed. Alternatively, PAIR has an alias, called BIPLOT, which lets the user get a bivariate plot of a data file of X-Y pairs:

```
BIPLOT < DATA
```

4.3.3 **CORR***: Multiple Correlation*

CORR can be used to get summary statistics for repeated measures data. Its input is a series of lines, each with an equal number of data. It prints the mean, standard deviation, minimum, and maximum for each column in its input. Then it prints a correlation matrix with all pairwise correlations. Like PAIR, columns from files can be joined with ABUT or extracted from files with DM.

4.4 Inferential Statistics

4.4.1 **ANOVA***: Multi-Factor Analysis of Variance*

ANOVA performs multi-factor analysis of variance with repeated measures factors (within subjects), and with unequal cell sizes allowed on grouping factors (between subjects). ANOVA reads in a series of lines from the standard input, each with the same number of alphanumeric fields. Each datum occupies one line and is preceded by a list of levels of independent variables describing the conditions under which that datum was obtained. The first field is some string indicating the level of the random variable in the design, and subsequent fields describe other independent variables. From this input, ANOVA infers which factors are between subjects, and which are within subjects. ANOVA prints cell sizes, means, and standard deviations for all main effects and interactions. Then ANOVA prints a summary of the design of its input, followed by a standard F-table.

Suppose you had a design in which you presented problems to subjects. These problems varied in difficulty (easy/hard), and in length (short/medium/long). The dependent measure is time

to solve the problem, with a time limit of five minutes. Your data file would have lines like this:

```
fred        easy        medium      5
ethel       hard        long        2
```

In the first column is a string that identifies the level of the random factor (here, subject name), followed by strings indicating the level of the independent factors, followed by the dependent measure (here, solution time). The data file holding lines like those above would be analyzed with:

```
ANOVA subject difficulty length time < DATA
```

Individual factors can be ignored by excluding their columns from the analysis:

```
DM s1 s2 s4 < DATA | ANOVA subject difficulty time
```

Similarly, different factors can be used as the random factor. This is common in psycho-linguistic experiments in which both subjects and items can be thought of as random variables.

4.4.2 **REGRESS***: Multivariate Linear Regression*

REGRESS reads its input of a series of lines from the standard input, each with the same number of columns of numbers. From this input, REGRESS prints minima, maxima, means, and standard deviations for each variable in each column. Also printed is the correlation matrix showing the correlations between all pairs of variables. Suppose you had a file called DATA with any number of lines and five columns, respectively called "blood pressure," "age," "height," "risk," and "salary." You could do a multiple regression with:

```
REGRESS pressure age height risk salary < DATA
```

If only a few columns were of interest, they could be extracted with DM:

```
DM x1 x3 x5 < DATA | REGRESS pressure height salary
```

4.4.3 **PAIR***: Paired Data Comparisons*

PAIR can be used to compare two distributions of paired data. Often, the two columns of interest are pulled out of a master data file with DM. The following command takes columns 5 and 3 from DATA and inputs them to PAIR:

```
DM x5 x3 < DATA | PAIR
```

For its two-column input, PAIR will print a t-test on the differences of the two columns, which is equivalent to a paired t-test. PAIR will also print a regression equation relating the two, along with a significance test on their correlation, which is equivalent to testing the slope against zero.

4.4.4 **TS***: Time Series Analysis*

TS produces a variety of statistics for data collected in repeated measures over time. The special options include auto-correlations of various lags (these compare one part of the series to another), and transformations between time series of different lengths. A few simple plots are available.

4.4.5 **POF & CRITF***: F-Ratio to Probability Conversion*

POF can be used to determine the probability of an F-ratio given the F-ratio, and degrees of freedom:

```
POF 12.2 3 48
```

CRITF can be used to determine the critical F-ratio needed to attain the given significance level

(useful for confidence intervals):

```
CRITF .01 3 48
```

5. DISTRIBUTION

Traditionally, the package has been distributed at cost to any interested parties provided they agreed not to market the package for personal profit. The most common form of distribution is a *tar* format 800 bpi tape. This tape contains a C source files directory with a Makefile, one with manual entries to be formatted by *nroff* with the **-man** macros, and some test examples.

6. ACKNOWLEDGMENTS

Jay McClelland has been instrumental in his influence in the user interfaces. He also wrote the original version of ABUT. Mark Wallen has been very helpful in being able to convey many of the intricacies of UNIX C programming. Greg Davidson was the force behind most of the error checking facilities. Don Norman wrote the original flow meter on which the IO program is based. Bob Elmasian was helpful in working on Version 6 compatibility. Jeff Miller kindly supplied the F-ratio to probability conversion functions.

The research reported here was conducted under Contract N00014-79-C-0323, NR 157-437 with the Personnel and Training Research Programs of the Office of Naval Research, and was sponsored by the Office of Naval Research and the Air Force Office of Scientific Research.

7. OTHER UNIX|STAT PAPERS

1. Data analysis programs for the UNIX operating system. *Behavior Research Methods and Instrumentation*, 1980, **12**, 554-558.
2. An example of cooperating compact data analysis programs. *Behavior Research Methods and Instrumentation*, 1981, **13**, 290-293. Presented at the National Conference on the Use of On-Line Computers in Psychology, November 12, 1980, St. Louis, Missouri.
3. Data analysis in the UNIX environment: Techniques for automated experimental design specification. Presented at the XIV Annual Conference on the Interface, July 7, 1982, Rensselaer Polytechnic Institute, Troy, New York. Proceedings to be published by Springer-Verlag.
4. UNIX|STAT: Compact data analysis programs for UNIX. Presented at the 1982 Summer USENIX Conference, July 9, 1982, Boston, Massachusetts. Proceedings to be published by USENIX.

CONTENTS