Statistics by Computer without Spreadsheets

KAWAGUCHI, Yuuichi*

(Received 30 NOVEMBER. 1998)

Abstract

This paper shows (1) to compute basic statistical data in Perl and (2) to make them graphs with GNUPLOT. Perl and GNUPLOT can run on many computing environments, such as UNIX, X window system, Microsoft Windows series, MS-DOS, Machintosh, and so on. Therefore, those method provides not only the way that does not need heavy spreadsheet applications, but also the independent way of computing environments. This is very flexible.

1 Introduction

We teachers usually process statistical data[1], *ie.*, data of our students. Most of them are the points of examinations. I use the word "statistical," however, in this case it does not means any high level statistical processings. It means a very basic statistics, such as a total, an average, a standard deviation, and so on.

For processing data of students statistically, ordinary people use a spreadsheet software, such as Microsoft EXCEL and Lotus 1－2－3. These softwares have many useful functions. Needless to say, they can calculate a total or an average, can sort points in order, and even can eliminate candidates!! They also can make graphs or make lists of those numbers. Thus, they are very useful. For a poor worker like me, however, they are too heavy. Not only the prices of them are very expensive, but also they use the very large amount of disk space and RAM, reaching up to 500 mega bytes and 64 mega bytes or above, and they need a high-performance CPU that is very expensive (especially for me).

This paper aims to show the way to process statistical data without such heavy spreadsheet applications. To put it concretely, I show the way to program such processings by a programming language. In this paper, I use a language Perl[2]. By Perl, programmers can write powerful and efficient application programs with very simple notations. Perl can run on UNIX, MS-DOS, Microsoft Windows 3.1/95/98/NT, maybe MacOS, and so on. Therefore once a programmer wite an application program by Perl, it can run on different operating systems without changing. There are many other languages that can run on some different operating systems. I chose Perl out of them since it is free, is very powerful especially in processing text, is used for a long time in the world, and is the most stable. New and evolving languages, such as JAVA[3], are updated repeatedly in a short cycle, and in every time the specifications of them changes widely. I can not wait them to be stable.

Spreadsheets can make graphs of data. Perl can not do that, however, there is a useful application program GNUPLOT[4]. It can run on UNIX with X window system, MS-DOS, Microsoft Windows series, and Machintosh series, and so on. In this paper, I show the way to make graphs by using GNUPLOT.

This paper is written on the assumption that readers have experiences in C or other procedural programming languages.

*Associate Professor: Department of Computer Engineering

2 Programming Language Perl

Perl is an abriviation of "Practical Extraction and Report Language." It provides many powerful and efficient functions to programmers. In many cases, programs in Perl are compact than those in C. Perl is an interpreter language. Programs in Perl may take more time than those in C. In the rest of this section, I explain how to program in Perl, comparing with C.

2.1 Syntax

Any procedural programming languages have three control structures, (1) seqnential execution, (2) branch, (3) iteration. Perl also hasthem. Perl can recognize the syntax for control structures like C, `if-else` and `while`. Perl has more statements, but for the sake of simplicity I use them in this paper.

2.2 Variables

When programer use variables in Perl, it is not need to declare them. This is different from C. The name of variable in Perl must begin with `$` sign. The second letter can be `_`, `a-z`, and `A-Z`. The third or later can be `_`, `a-z`, `A-Z`, and `0-9`.

These are examples of valid names for variables:

```
$a, $a_b, $pen_1
```

2.3 Functions and Arguments

Similarly to C, Perl has an ability to build and call user-defined functions. Those are called "subroutines" in Perl. To define a subroutine, programmers should use the following syntax:

```
sub subroutine {
    statements
}
```

In calling the *subroutine*, use the following syntax:

```
&subroutine
```

In passing arguments to subroutines, use the following syntax:

```
&subroutine arguments
```

Arguments are separated by commas. Subroutines receive the list of arguments in a variable `@_`. To assign each argument into differnt variables, use the syntax:

```
sub subroutine {
    ($a, $b, ···) = @_;
}
```

In this case, each arguments are assigned into `$a`, `$b`,···. These variables are local.

The value of the last statement in the subroutine is returned.

2.4 I/O

In input/output operations, data are given and taken via filehandles. Filehandles are created by `open` operator. In default, Perl provides three filehandles, `STDIN`, `STDOUT`, and `STDERR`. They correspnd to `stdin`, `stdout`, `stderr` in C respectively.

By using `open`, filehandles are linked with files. Suppose that `file` is a file, and `FH` is a filehandle, then

```
open(FH, "file")
```

links `FH` with `file`. In this case, data are read via `FH` from `file`. To write data to files, call `open` as:

```
open(FH, "> file")
```

The '>' shows data are written into `file`.

In any cases, `open` returns a false status if it errors. By using this mechanism, the following statement is usually used:

```
open(FH, "> file") ||
   die "Can't open: file\n";
```

This statement have a meaning such as "open the file, or die."

To read data via a filehandle, use the syntax:

```
$line = <FH>
```

This statement assigns the variable `$line` to one line from a file via a filehandle `FH`. This statement returns a false status when there are not unread data. By using this mechanism, to read all lines from file, the following statements are usually used:

```
open(FH, "file") || die "bye\n";
while ( $line=<FH> ) {
    statements
}
```

Alternatively, to write a data into a file via a filehandle:

```
print FH data
```

In this case, the *data* can be variables, constant numbers, or strings. When `FH` is `STDOUT` or `STDERR`, data are shown on a CRT (or LCD, and so on) display.

When date are completely read, the file opened by `open` must be closed by `close`:

```
close(FH)
```

This statement closes the file linked with the filehandle `FH`.

3 Statiscics by Perl

This section explains the way to compute statistical values by using Perl without spreadsheet applications. Some fragments of programs are shown.

3.1 Conventions

There is a file named `points.dat`. This includes the points ofstudents in a class. The format of the file is:

stud	prog	circ	netw
abe	90	65	80
aoki	70	35	40
fujita	45	85	70
jiro	60	60	20

Of cource they are not real students. They are virtual students for this paper. The number of students are unknown, however, when students and their points are completely read from the file, Perl can compute the number of lines read. It implies the number of students.

At the top of the program, there are statements shown in Fig. 1.

```
#!/usr/local/bin/perl

open(FH, "points.dat") ||
   die "Can't open:  points.dat\n";
$line = <FH>;
chop($line);
@subjs = split(/[ \t\n]/, $line);

@points = ();
while ( $line=<FH> ) {
   chop($line);
   push @points, $line;
}
close(FH);
```

Figure 1: At The Top of The Program

Data read are stored into the array named `@points`, and the variable `$#points` is assigned the number of students. The notation `$#points` stands for the last index number for the array `@points`. Since the top of the lines stored in the file is a headding, it is read and stored another array `$subjs`. The operator `split` divides its second argument into elements by its first argument, and returns tha array that includes all the elemnts. The first argument is `/[ \t\n]+/`. This notation is called a regular expression. This stands for "the sequence of a space, a horizontal tab, and a new line." Therefore, the heading line is devided by any white spaces and stored into the array `@subjs` respectively. Note that the element stored in `$subjs[0]` is the string `stud`, and this does not stand for any subject. This element should not be used.

3.2 Total

There are two kinds of total points. One is a total by a subject, and another is by a student. The total points by a student are computed by the fragmnents shown in Fig. 2.
This fragments outputs:

```
abe     235
aoki    145
fujita  200
jiro    140
```

```
@name = ();
@tot_std = ();
for ($i = 0; $i<=$#points; $i++) {
   @pt =
     split(/[ \t\n]+/,$points[$i]);
   push @name, $pt[0];
   push @tot_std, 0;
   foreach $p ( 1..$#pt ) {
      $tot_std[$i] += $pt[$p];
   }
}
foreach $i ( 0..  $#name ) {
   print STDOUT
      "$name[$i]\t$tot_std[$i]\n";
}
```

Figure 2: The Total by A Student

Suppose the name of the file storing the program in Perl is `total-std.pl`, then type the following line to the prompt from the operating system to execute it:

```
host$ perl total-std.pl
```

The line underlined stands for the line typed by a user.

The following fragments computes the total points by a subject is shown in Fig. 3.

```
@tot_sub = ();
foreach $i ( 1..$#subjs ) {
   push @tot_sub, 0;
}

for ($i = 0; $i<=$#points; $i++) {
   @pt =
     split(/[ \t\n]+/,$points[$i]);
   foreach $p ( 1..$#pt ) {
      $tot_sub[$i] += $pt[$p];
   }
}
foreach $i ( 1..$#subjs ) {
   print STDOUT
      "$subjs[$i]\t$tot_sub[$i]\n";
}

exit 0;
```

Figure 3: The Total by A Subject

The fragment outputs:

```
prog    265
circ    245
netw    210
```

3.3 Average

An average is easily computed by:

$$average = total \div number$$

In this case the value of *number* is assigned into the variable `$#tot_std` or `$#subjs - 1` in Perl. I omit the fragments of the porgram in Perl.

3.4 Standard Deviation

Suppose data are stored into x_i $(i = 1, \cdots, n)$, then thestandard deviation of them is denoted by s and is computed by:

$$\bar{x}=\frac{1}{n}\sum_{i=1}^{n}x_i, \qquad s=\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2},$$

where $\bar{x}$ stands for the average of x_i's, and it is already computed by Perl in the avobe section. In this section, I compute the standard deviation of the total points by a student. The fragment to compute them is shown in Fig. 4, where the average of `@tot_std` is assumed to be assigned into the array `$ave`. The operator `sqrt` compute the square root of its argument.

```
$s = 0;
foreach $p ( 1..$#tot_std ) {
   $a = $ave - $tot_std[$p];
   $s += $a * $a;
}
$s $#tot_std;
$s = sqrt $s;
}
print STDOUT "$s\n";
```

Figure 4: The Standard Deviation

3.5 Sorting

Perl has a useful function for sorting data. In the case that data are stored into an array, `@a`, the statement to sort them is:

```
@b = sort @a
```

where @b has the sorted array out of @a. In fact, outputs may want to be like:

```
name total
```

To do this, different arrays @name and @tot_std are sorted at the same time. In this case, Perl's sort is no use. I write a subroutine by myself. It is shown in Fig. 5.

```
sub sort_2 {
    for($i=0; $i<=$#name-1; $i++) {
        $max = $i;
        for($j=$i+1; $j<=$#name; $j++){
            if ( $tot_std[$max] <
                    $tot_std[$j] ) {
                $max = $j;
            }
        }
        $w = $tot_std[$max];
        $tot_std[$max] = $tot_std[$i];
        $tot_std[$i] = $w;
        $w = $name[$max];
        $name[$max] = $name[$i];
        $name[$i] = $w;
    }
}
```

Figure 5: Subroutine for Sorting

To call this function, do simply:

```
&sort_2
```

It is not needed in newer version of Perl to specify '&' for calling subroutines. The subroutine sort_2 uses the famous algorithm for sorting dada, "Selection Sort," which is the most simple and efficient out of many sort algorithms, when the number of data is small.

4 Make Them Graphs

This section explains the way to make data graphs by using GNUPLOT. The version string of it is"Linux version 3.5, patchlevel 3.50.1.17, 27 Aug 93."

In the subsections, I show three example of the graph of their total points and average of them. Their total points are computed in Section 3.2. The average of them is 180. Their points are placed on the vertical axis. Their names are placed on the horizontal axis.

4.1 Dots

When data are stored into a file, for example named"tot-std.dat," with the format such as:

```
0.5   235
1.0   145
1.5   200
2.0   140
2.5   180
```

GNUPLOT can read it and make them graph easily by entering the command:

```
plot [0.0:3.0] "tot-std.dat"
```

where "[0.0:3.0]" specifies the range of horizontal axis. The range of the vertical axis is automatically decided by GNUPLOT in according to given data. This plot command makes a graph shown in Fig. 6.

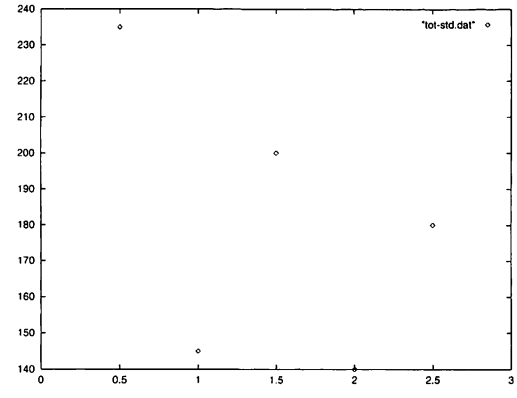

Figure 6: Plots by Dots

Horizontal axis is numerically plotted by 0.5, however, I want to place the names of the students. To do this, the following commands should be entered:

```
set xtics ("abe" 0.5, "aoki" 1.0,→
    "fujita" 1.5, "jiro" 2.0,→
    "avg" 2.5)
```

The result of the command is shown in Fig. 7.

4.2 Lines

To draw an average line in the graph rather

to plot it, (1) remove the last item in the file "tot-std.dat," renaming it "tot-std2.dat," and (2) enter the following commands:

```
plot [0.0:2.5] "tot-std2.dat"
replot 180 with lines
```

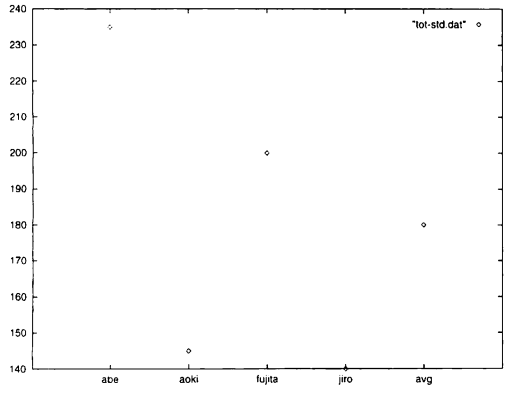

Figure 7: Names on The X Axis

This results in Fig. 8.

The second argument for `replot`, *ie.*, `180`, stands for the equation $y = 180$.

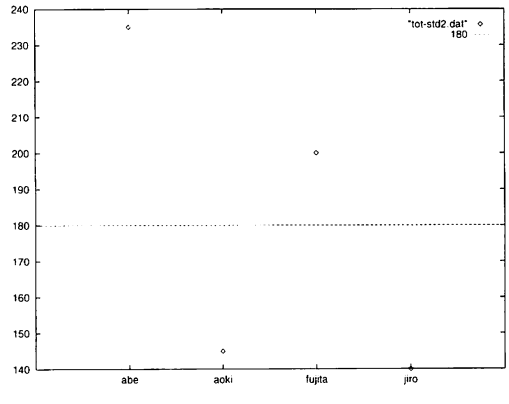

Figure 8: An Average Line

4.3 Boxes

In graphs shown in avobe sections, points of students are plotted by dots. In this sction, boxes are used. To do this, enter the command:

```
plot [0.0:2.5] "tot-std2.dat"→
  with boxes
replot 180 with lines
```

This results in Fig. 9.

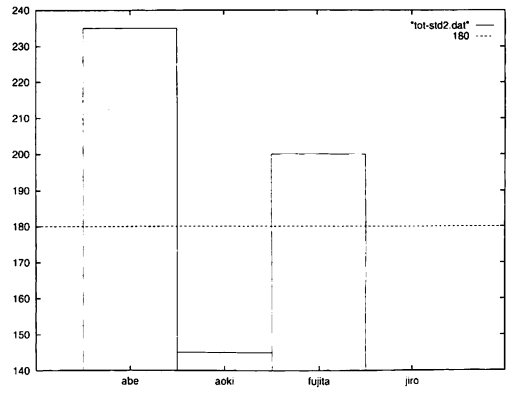

Figure 9: Jiro is not Shown

In all cases, the points of jiro is the minimun, and GNUPLOT automatically decides to place the point on the ground level. To change the range of the vertical axis, enter the command:

```
plot [0.0:2.5] [0:300]→
  "tot-std2.dat" with boxes
replot 180 with lines
```

This results in Fig. 10.

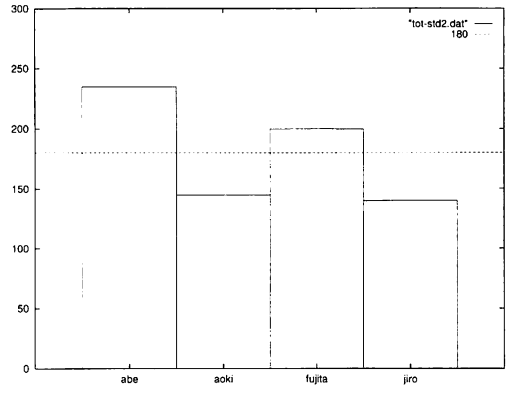

Figure 10: Jiro is Shown

5 Concluding Remarks

This paper shows (1) to compute basic statistical numbers and (2) to make them graphs. Perl and GNUPLOT can run on many computing environments, such as UNIX, X window system, Micosoft Windows series, MS-DOS, Machintosh, and so on. Therefore, those method provides not only the way that does not need heavy spreadsheet applications, but also independent way of computing environments. This

is very flexible.

Acknowledgement

When I write this paper, I use Mule for the editting software, LaTeX 2ε for the typesetting software, and shin-eiwa·waei chujiten in CD-ROM for the English－Japanese dictionary. These are very useful. Without them, I coudn't write this paper, and can't write any other papers in the future. I thank those fine softwares and the authors of them. If, however, more useful softwares appear in me, I maybe use them.

References

[1] Kichirou Takamatsu, Tashiro Yoshihiro, joho no kisosuugaku, Baihu Kan, 1988.

[2] Larry Wall, Tom Christiansen, Randal L. Schwartz, Programming Perl 2nd Edition, O'Reilly & Associates, Inc., 1996.

[3] Patrick Niemeyer, Joshua Peck, Exploring JAVA 2nd Edition, O'Reilly & Associates, Inc., 1997.

[4] Michirou Yabuki, Tsuyoshi Ootake, tsukaikonasu GNUPLOT, Techno Press, 1996.