PROGRAMMING LANGUAGES AND OPPORTUNITIS THEY OFFER TO THE STATISTICAL COMMUNITY

Naeve, Peter
University of Bielefeld

At first I would like to modify the title of my paper to "programming languages and opportunities they should offer to the statistical community". By adding this one word "should" I want to express two things:

- Firstly a somewhat critical review of the field of accessible programming languages (in contrast to the unnumbered body of proposals for programming languages burried in the literature).
- Secondly a plea for a shift in attitude of the so called user from a supply oriented to a demand oriented point of view.

The term programming languages stands for what computer scientists call high level languages or general purpose languages (e.g. FORTRAN, PASCAL), but I will let it include for the moment dedicated problem oriented languages (like GLIM).

It is claimed that high level languages free the user from the burden of machine dependencies, bit and byte manipulations etc. and allow him to concentrate on the formulation of the solution to his problem. Valuable tools are provided to ease the users task. High level programming languages usually incorporate facilities to support

- structured programming
- user supplied data types (and structures)
- recursion.

But unfortunately it turns out that the effort to avoid machine orientation resulted in a kind of splendid isolation for the high level languages. They usually do not provide reasonable interfaces to

- other high level languages
- new hardware features such as graphics devices and array processors
- the operating system.

This is an especially sad finding with respect to graphics. Either

you leave it or dig deep into escape sequences and this kind of stuff. Some may consider this a little bit unjustified but I think for most users it comes close to the true description of their situation.

To proceed with our theme let us investigate who makes up the statistical community. In alphabetic order there is the consultant, researcher and the teacher. The computer influences the work of all of them.

When looked at in detail we may conclude that although with different weights statisticians use mathematics, logic and deal with data which usually have their unique kind of structure. The language of statisticians resembles these different roots. Turning to a computer the statistician has to use a programming language to convey his ideas and thoughts. What are the requirements a programming language should fulfil to suit the statistician's needs?

1. It should allow him both:
   - not to bother about the computer (i.g. he does not want to be a computer specialist)
   - to bring as much special knowledge about computers and computer programming he has into action (i.e. with respect to the computer the language should be open).
2. It must be as close to mathematics as possible - at least to the 'useful and essential' part of mathematics. Consider the well known formula from the field of linear models

$$SS = \hat{B}'C(C'GC)^{-1}C'\hat{B}.$$

   Would it not be nice to have a programming language which allows to evaluate this formula with the statement

   $SS \leftarrow CB +.\times (\boxplus (\circledcirc C) +.\times G +.\times C) +.\times CB \leftarrow (\circledcirc C) +.\times B$.
3. It should allow to deal with logic in a condensed form. For instance: to speak about the non-negative values in a data vector X should be as easy as

   $(X \geq 0) / X$ or $X[X \geq 0]$.
4. It should contain statistical concepts, i.e. mean, variance etc., for instance in the form mean X, var X.
5. It should allow to deal with data objects in an easy way of name, attribute(s), value(s). To give an example

```
res ← reg(y,X)        result of regression of y on X
plot res$resid        plot of the residuals
regsum res            summary of regression statistics
plot y, res$resid     plot of residuals versus y
regprt res            print report of regression
```

where the result res of the regression of y on X is a structure with components such as residuals, which are accessed in an obvious way. The structure may be processed in various ways (i.e. regprt, regsum).

6. It should not only allow to speak about analysis of data but also about the form one wants the result to be presented.
7. It should support the interactive style of work a statistician is used to. This calls for a kind of protocol feature.
8. It should fit to the user's way of thinking and working and not vice versa.
9. It should support the statistician in using as much of the hardware and other software available at his site.

Think for a moment how your favourite programming language fulfils these requirements.

Here are some statements to back your findings:

"We learned several things from this experiment, one of the earliest in the Computational Probability Project. The programming for computing the functions mentioned was easy, but still took a good deal of time to do, debugging and running a program in batch mode is not very efficient in term of the user's time. It was more time consuming to program the CALCOMP and, more important, it was psychologically unattractive to run the plotter off-line." Grenander (5, p. 14).

"It is quite common that when writing a research report containing numerical tables the output from the computer cannot be used as such, but the results have to be retyped manually. This may happen even if the computer output is well designed, since the needs of the user may change during the reporting phase." Mustonen (9, p. 337).

Perhaps you will sympathize with the following two quotes from a panel discussion on "programming languages issues for the 1980's"(10)

"On the whole programming languages are overrated. ... What's really important in the programming game is not the language but the tools you have to work with." "... there's an intermediated sort of

thing which profession oriented language is the buzzword for ..."

In 1984 a software catalog (13) listed 252 statistical software packages. Many of these packages claim to offer a kind of statistical language for ease of use too. Do these packages fulfil our scheme of requirements? My answer is no. They suffer from the same kind of isolation as do programming languages. They are rather fixed in scope too.

There is still another reason to be a bit reluctant with respect to statistical packages. There are some doubts regarding their standards.

The paper by J.L. Longley: An appreisal of least square programs for the user (8) dates 19 years back but its message is not yet obsolate. The same is true with respect to the subject dealt with in papers by Ling (7) or Wilkinson and Dallal (14) on sample moments calculation to mention just two. To give some evidence to this claim I will present some citations form "Statistical Computing Software Reviews" found in the last two volumes of The American Statistician.

1. "Results of the analyses cannot be stored for analysis. For example one cannot store means and standard deviations to plot in checking for possible variance-stabilizing transformations. Similary, residuals from regressions can be printed but not stored to plot, for example against predicted values." (12, p. 165)
2. "When the median of an odd sample size was programmed incorrectly, one wonders about computations for more complicated procedures." (12, p. 166)
3. "The variance function in the command FUN, however, failed the "Anscombe test" dismally; Anscombe (1967, p. 3) discussed what could go wrong in the computation of the three readings 9,000, 9,001, and 9,003. The function VAR in FUN produced 1.33 for the variance of these values (should be 2.33) and a negative variance for 9,997, 9,998, and 9,999, suggesting the inappropriate use of the "desk calculator" formula." (11, p. 218f.)

I intendently did not mention the name of the packages reviewed. I do not want to turn those packages down, for I strongly believe that the packages not yet reviewed are not free of bugs too.

What should be learned from these citations is that packages may suffer from everything starting with numerical inaccuracy and going

to inflexible and inconsistent management of statistical analysis.

To make explizit that we are looking for more let us coin a new name for it. I think statistical environment would be the right name. By this I mean a computing environment made up through an integrated collection of tools for exploiting as much of the hardware and software at hand (not calling for special devices) in an easy and natural way. This should be embedded in an concise consistent concept of a language well suited to the needs of statisticians. The language should have a syntax strongly influenced by mathematics and logic, with basic statistical terms such as mean, median etc. but also - and this is essential - it must contain tools to enlarge the language for personal or common usage.

At the moment I just see two such systems which are more than experimental prototypes. I speak of the S-system and APL.

Let them introduce themselves.

"S is an interactive environment for data analysis and graphics with two components: a language and a support system. The S language is a very high-level language for specifying computations. S is also a system in that it provides a total environment for the user, including data management, documentation, and graphics.

The primary goal of the S environment is GOOD DATA ANALYSIS. The facilities in S are directed toward this goal. S encourages the iterative, interactive style of data analysis which leads to unterstanding. In this way, S is quite unlike most statistical 'packages'.

S provides the user with interactive computation, both simple and complex, graphical displays on a wide vairety of graphics devices, data management and structuring." (2, p. i)

The S-system was developed at the Bell Telephone Laboratories by Becker and others. It runs on UNIX machines (for instance I made my experiences with S on a HP 9000). The book by Becker and Chambers: S An interactive environment for data analysis and graphics describes the system - it is the reference manual. The quote given was from this book.

Turning to APL let me quote Dr. Iverson (the father of APL) "Nearly all programming languages are rooted in mathematical notation, employing such fundamental notions as functions, variables, and the

decimal (or other radix) representation of numbers ... APL has, in its development, remained much closer to mathematical notation, ..." (4, p. 39)

"The primitive objects of the language are arrays. ... The syntax is simple: there are only three statement types (name assignment, branch, or neither), there is no function precedence hierarchy, functions and defined functions ... are treated alike.

The semantic rules are few: the definition of primitive functions are independent of the representations of data to which they apply, all scalar functions are extended to other arrays in the same way (that is item - by - item) ...

The utility of the primitive functions is vastly enhanced by operators which modify their behavior in a systematic manner. ...

External communication is established by means of variables which are shared between APL and other systems. These shared variables are treated both syntactically and semantically like other variables. A subclass of shared variables, called system variables, provides convenient communication between APL programs and their environment." (5, p. 40f)

APL was first developed by Iverson, Falkoff and others within IBM. But it began to spread out into other companies and universities. As a result we have different implementations of APL grouped around a core of the language which is standardized.

It is impossible to give a complete review of both systems within this short paper. Nor do I intended to rank one against the other.

As I consider them both to be advanced prototypes of a statistical environment we should look for I will present certain of their features in a number of small examples to exemplify some essential points.

Just two further remarks before I start:

i) all "lines of code" presented so far were written down either in S or APL.

ii) Although being unique S and APL have some things in common. Both are interpretative languages - so interactive work is eased. Both have a concept of workspace, allowing the user to tailor his own special environment without loosing contact to the overall system or other users if he wishes to cooperate.

Example 1: Closeness to mathematical notation

APL offers matrices as primitive objects and a set of primitive functions and operators to deal with them. Talking about requirements we presented the formula

$$SS = \hat{B}'C(C'GC)^{-1}\ C'\hat{B}\ .$$

and its transformation into an APL expression

```
SS ← CB +.× (⌹ (⍉C) +.× G +.× C) +.× CB ← (⍉C) +.× B .
```

This is almost a one-to-one mapping of our well acquainted mathematical notation. For instance "⍉" mirrors matrix transpose, "+.×" tells exactly what a scalar product of vectors is. If you learn that "←" stands for assignment and become acquainted with the (at first somewhat peculiar) right to left anti-hierachic style of formula evaluation unique in APL you will read and unterstand such expressions as easy as you do with normal mathematical notation.

Probably someone knows Conway's game of life. Let a rectangular grid divide the plane into cells. A cell can be dead or alive. A living cell will survive, if its neighbourhood is neither overcroweded (i.e. 4 or more living cells) nor it is isolated (i.e. 0 or 1 living cell) otherwise it dies. A cell will be born if exactly 3 cells in its neigbourhood are alive.

A simple APL solution - exploiting direct function definition is shown below. The plane is mapped on a bit-matrix, where 1 stands for a living cell.

```
GENER : HH ∨ H RULE2 HH ← (H ← NB ⍵) RULE1 ⍵
RULE1 : ⍵ ∧ ⍺ ∈ 2 3
RULE2 : (~⍵) ∧ 3 = ⍺
NB    : (HOR H) + (VERT H) + DIAG H ← BORD ⍵
BORD  : 0 ,[1] (0 , ⍵ , 0) ,[1] 0
DIAG  : (¯2 2 ↓ ⍵)+(2 ¯2 ↓ ⍵)+(¯2 ¯2 ↓ ⍵)+2 2 ↓ ⍵
VERT  : (¯2 1 ↓ H) + 2 1 ↓ H ← 0 ¯1 ↓ ⍵
HOR   : (1 ¯2 ↓ H) + 1 2 ↓ H ← ¯1 0 ↓ ⍵
```

Besides easily identified logical notation the main features of APL applied in this solution are the "take" and "drop" function. L↑R ≙ pull L items from one end or corner of R, L↓R ≙ wipe L items from one end or corner of R. Note that you may specify which dimensions of your objects are envolved such as in L,[I]R ≙ join values of L and R together into a larger array by catenating along L's or R's I-th coordinate.

Example 2: Extensibility of the language

Look at those two stem and leaf display done with the S function stem.

```
> stem  income, depth=T
N = 50  Median = 695.665
Quartiles = 287.77, 1813.93
Decimal points is 3 places to the right of the colon
   19   19   0 : 11122222223333334444
        11   0 : 56667778889
   20    3   1 : 134
   17    6   1 : 557789
   11    5   2 : 12234
    6    3   2 : 556
    3    2   3 : 03
    1    0   3 :
    1    1   4 : 0
> stem  income, depth=T, twodig=T
N = 50  Median = 695.665
Quartiles = 287.77, 1813.93
Decimal points is 3 places to the right of the colon
   20   20   0 : 09,12,14,15,19,21,22,23,24,25,25,28,29, ...
        10   0 : 57,58,60,65,66,73,77,77,81,87
   20    4   1 : 14,26,39,49
   16    5   1 : 51,68,74,81,90
   11    6   2 : 11,21,23,33,45,46
    5    3   2 : 50,63,98
    2    1   3 : 30
    1    0   3 :
    1    1   4 : 00
```

The only difference should be the difference in leaf size, i.e. 1 or 2 digits. But more has happened. Some items changed their stem. The reason for this is, that the stem function rounds the data. There was no way to avoid it. So I decided to extend the abilities of the S-function. The next lines show what has to be done. The subroutine stemw as delivered by Bell reads as follows.

```
ROUTINE(stemw,produce stem and leaf portion for stem)
subroutine stemw(a,n,fc,kk1,kku,ksl,nl,wd,twodig,depth)
real a(n)
integer n,fc,kk1,kku,ksl,nl,width,wd
:
:
   if (twodig) {#... ditto above, but for 2 digit leaves.
      t = s*a(k)+0.0499
      if (a(k) < 0.0) t = t-10.1
   }
   else {#...adjusts round-off
      t = s*a(k)+0.499
      if (a(k) < 0.0) t = t-11.0
   }
:
:
```

This has to be changed to

```
ROUTINE(stemw,produce stem and leaf portion for stem)
subroutine stemw(a,n,fc,kkl,kku,ksl,nl,wd,twodig,depth,ritup)
real a(n)
integer n,fc,kkl,kku,ksl,nl,widt,wd,ritup
.
.
.
   if (twodig) {# ...ditto above, but for 2 digit leaves.
      t = s*a(k)+(0.0499*ritup)
      if (a(k)< 0.0) t = t-10.0-(0.1*ritup)
   }
   else {# ...adjusts round-off
      t = s*a(k)+(0.499*ritup)
      if (a(k)< 0.0) t = t-10.0-(1.0*ritup)
  }
.
.
.
```

The essential calculations are done by function stem.

The old version

```
FUNCTION stem(
              x/REAL,NAOK/
             nl/INT,1,0/
          scale/INT,1,1000/
         twodig/LGL,1,FALSE/
          fence/REAL,1,2./
           head/LGL,1,TRUE/
          depth/LGLG,1,FALSE/
)
INCLUDE(option,io)
if(nl!0&nl!=2&nl!=5&nl!=10)
  FATAL(number of leaves per stem is not 0 2 5  or  10)
n=LENGTH(x)
NAOUT(`x') # remove NAs
if(n!=LENGTH(x) && head)
FPRINT(OUTFC,"Contained",I(n-LENGTH(x)),"NAs")
call stems(x,LENGTH(x),OUTFC,0,lwidth,nl,2.,FALSE,
           twodig,!head,scale,fence,FALSE,depth)
END
```

has to be changed to

```
FUNCTION STEM(
              x/REAL,NAOK/
             nl/INT,1,0/
          scale/INT,1,1000/
         twodig/LGL,1,FALSE/
          fence/REAL,1,2./
           head/LGL,1,TRUE/
          depth/LGL,1,FALSE/
          round/LGL,1,FALSE/
)
INCLUDE(option,io)
```

```
if(nl!=0&nl!=2&nl!=5&nl!=10)
  FATAL(number of leaves per stem is not 0 2 5  or  10)
n=LENGTH(x)
NAOUT(`x') # remove NAs
if(n!=LENGTH(x) && head)
FPRINT(OUTFC,"Contained",I(n-LENGTH(x)),"NAs")
if(round)
   iritup=1
else
   iritup=0
call stems(x,LENGTH(x),OUTFC,0,lwidth,nl,2.,FALSE,
          twodig,!head,scale,fence,FALSE,depth,iritup)
END
```

I do not present the compilation - and binding steps necessary to include the new version in S. Your are well supported by the S-system (and the operating system UNIX without which S would not be so powerful) in going through this procedures. ●

Example 3: It is easy to make a plot

Imagine you want to build a plot function for histogramms. Here are some ideas how the result should look like.

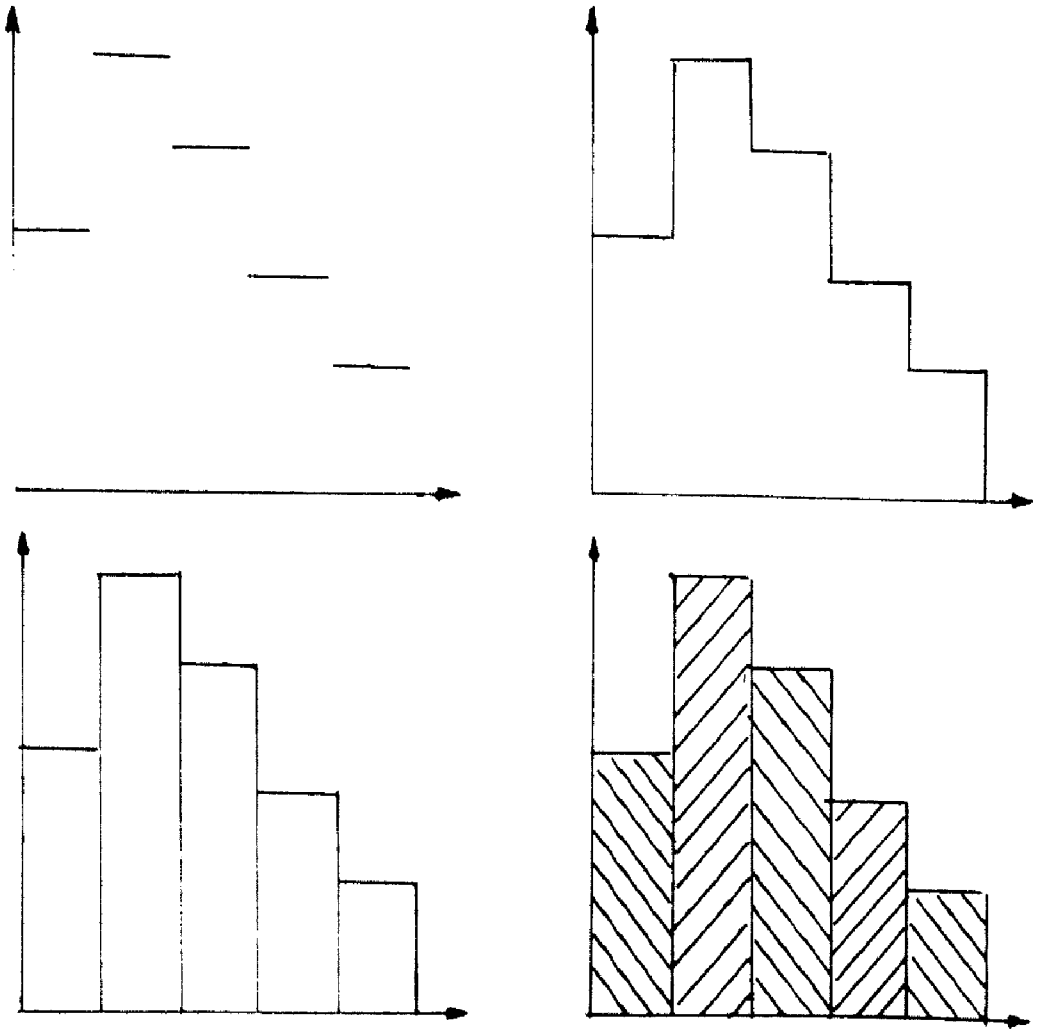

figure 1: Some ideas for histogram plots

In the APL implementation APL*PLUS you are provided with a set of system functions which give you control over your plotting device.

Some system functions for graphics in APL*PLUS:

□*GINIT*	initializes the graphics display for the particular adapter in use
□*GVIEW*	selects the rectangular area on the screen onto which graphics displays are projected
□*GLINE*	plots individual points, draws straight line segments, and fills rectangular bars
□*GCIRCLE*	creates displays of entire circles or ellipses, or arcs from them, or pie-shaped wedges formed by such arcs and two radii
□*WRITE*	writes text on the graphics screen

One thing more you have to learn is that different people (or graphics devices) have different coordinate systems to structure a picture.

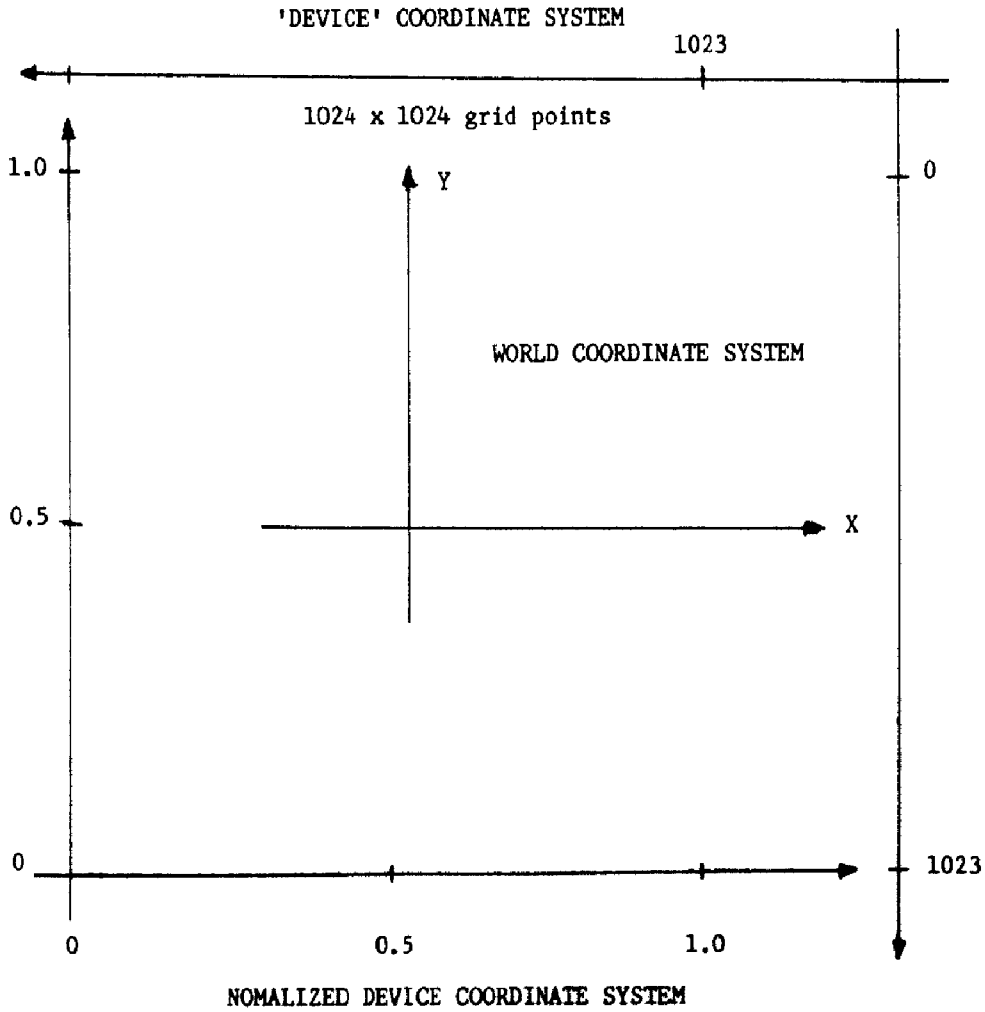

figure 2: Different coordinate systems

Taking this into account the simple plot function is easyly coded.

```
∇ HIST XY
  ⎕ELX←⎕ALX←' ← END ◊ ⎕DM
  0 0 ρ0 ⎕GINIT 'IBMCOLOR
  XY←NORM XY
 ⍝ NEXT LINE FOR SOLID BARS AND BOXES ONLY
 ⍝ XY←(¯1 0 ↓XY),(1↓XY[;1]),[1.5]0ρ¯1↑XY[;2]
 ⍝ SOLID BARS
 ⍝ 1 ⎕GLINE XY
 ⍝ BOXES
 ⍝ 1 ⎕GLINE((1↑ρXY), 5 2)ρXY[; 1 2 1 4 3 4 3 2 1 2]
 ⍝ HORIZONTAL LINES
 ⍝ 1 ⎕GLINE((¯1+1↑ρXY),2 2)ρ(¯1 0↓XY),(1↓XY[;1]),[1.5]¯1↓XY[;2]
 ⍝ SKY LINE
 ⍝ 1 ⎕GLINE((¯1+1↑ρXY),3 2)ρ(¯1 0↓XY),(1↓XY[;1]),(¯1↓XY[;2]),1 0↓XY
 ⍝ ⎕GLINE 1 2 2 ρXY[1;],XY[1;1],¯1↑XY[;2]
  0 0 ρ⎕INKEY
 END: 0 0 ρ3 ⎕INT 16
∇
```

If you are willing to put somewhat more effort into the plotting business you may produce a result like that shown in the following figure 3.

I want to demonstrate by this example that is rather easy to create a plot function for the essential part of a picture. But it is always this essential part that matters if you try to use the graphical mode to express your ideas. So if a graphical output is just for your sake why spent a lot of time in making it look nice or in making the plot function foolproof? ●

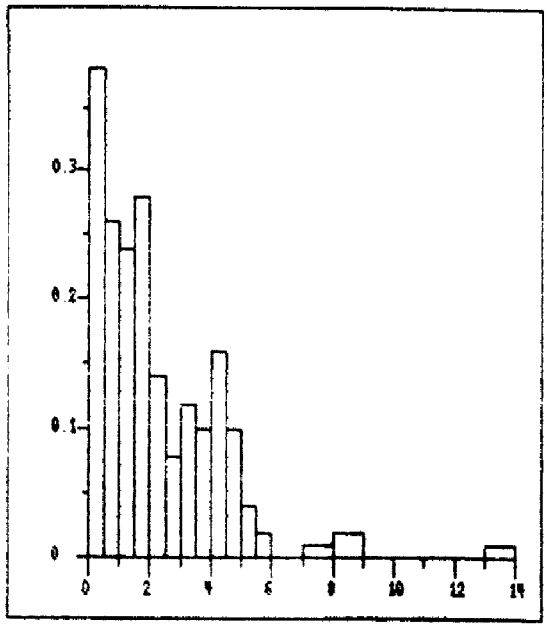

figure 3: One nice histogram plot

I hope this set of examples has made the vision of a computing environment for statistician a little bit more concrete. As I said I do not want to rank those two candidates. At the moment you may praise yourself if you can lay your hand on either of them. But I think it is fair to confess that I myself still are biased towards APL.

And it is also fair to report that both systems are not free of bugs. The three APL systems are all implemented on the same computer (an IBM compatible PC with a 8088 and 8087 processor).

'Inconsistencies' in the implementation of the Gamma function and the Binomial coefficient in APL on an IBM PC

L I M I T S for X

! X	SHARP	: ≤ 56.5452	1 ! X	SHARP	: ≤ 3.602 E 16
	APL*PLUS	: ≤ 170.6243		APL*PLUS	: ≤ 255
	IBM 1.0	: ≤ 170.6243		IBM 1.0	: ≤ 1.796 E 308
0 ! X	SHARP	: ≤ 3.602 E 16	X ! X	SHARP	: ≤ 3.602 E 16
	APL*PLUS	: ≤ 255		APL*PLUS	: ≤ 1.796 E 308
	IBM 1.0	: ≤ 1.796 E 308		IBM 1.0	: ≤ 1.796 E 308

Ironically enough when we had implemented S on our HP 9000 we got a run time error when calling the gamma function with argument 6, i.e. the result should be 120. The reason was an inconsistent handling of the machine dependent number ranges.

To bring this paper to an end let me summarize. Not what a computer language offers but what a computer language should offer should be the point of concern. One might put it in one sentence. The computer language must fit to the statistician's way of thinking and working. In some detail this plea was worked out in requirements 1 - 9 and examples 1 - 3. The history of APL reveals that user's demands can be fruitfully incorporated into the development of a language. So do not be a devote beggar grasping what will be given to you by the computer scientists. It is you who has the real problems to solve not they. And S and APL show that it can be done.

References

1. Becker, R.A. Chambers, J.M. (1984). Design of the S System for data analysis. Communications of the ACM, vol 27.
2. Becker, R.A., Chambers, J.M. (1984). S an interactive environment for data analysis and graphics. Wadsworth Inc., Belmont.
3. Becker, R.A., Chambers, J.M. (1985). Extending the S System. Wadsworth Inc., Belmont.
4. Falkoff, A.D., Iverson, K.E. (1978). The evolution of APL. APL Quote Quad, vol 9, no 1.
5. Falkoff, A.D., Iverson, K.E. (1978). APL language. APL Quote Quad, vol 9, no 1.
6. Grenander, U. (1982). Mathematical experiments on the computer. Academic Press, New York.
7. Ling, R.F. (1974). Comparison of several algorithmus for computing sample means and variances. Journal of the American Statistical Association, vol 69.
8. Longley, J.W. (1967). An appraisal of least squares programs for the electronic computer from the point of view of the user. Journal of the American Statistical Association, vol 62.
9. Mustonen, S.(1981). Statistical computing with text editor. In: Computational Statistics, Naeve, P., Büning, H. (Ed.). de Gruyter, Berlin.
10. Programming languages issuess for the 1980's (1984). SIGPLAN-notices, vol 19, no 8.
11. The American Statistician (1985), vol 39, no 3.
12. The American Statistician (1986), vol 40, no 2.
13. The Software Catalog (1984). Science and Engineering. Elsevier, New York.
14. Wilkinson, L., Dallal, G.E. (1977). Accuracy of sample moment calculations among widely used statistical programs. The American Statistician, vol 31.