TAMING STATISTICS FOR STUDENTS WITH APL

Stephen M. Mansour
University of Scranton, Scranton, PA 18505
stephen.mansour@scranton.edu

Statistics can be overwhelming for students due to the vast number of functions available in many software packages. In the computer language APL, "operators" are higher order functions which modify or combine existing functions to produce new functions. These operators allow us to use a reduced set of functions to produce a wide variety of useful results, using expressions which are surprisingly English-like. This approach simplifies the teaching – and learning – of statistics using software, by dramatically reducing the vocabulary that a student needs to master, and clarifying the fundamental principles of entry-level statistics. A working software program, TamStat, with a user interface and an optional interface to "R" has been implemented in Dyalog APL.

INTRODUCTION

The computer language APL is ideal for expressing statistical concepts. The basic syntax and structure of APL is described in detail in (Iverson, 1962) and its extensions including defined operators are described in (Brown, 1984). Since the language is array-based, sample data can easily be represented as a vector. Many statistical operations can be represented as functions, or as operators--examples of which are described in (Iverson, 1979) and (Smith, 1981). And since APL uses infix notation and right-to-left precedence of operations, we can define a set of functions to make statistical expressions in APL appear English-like (Mason, 1986). Finally, complex statistical output with multiple results can be stored in namespaces (Scholes, 1994).

STATISTICAL FUNCTIONS

There are four basic types of functions used in statistics. These consist of summary or measurement functions used in descriptive statistics, probability distributions, relational functions for probability and hypothesis testing, and logical functions used to define the rules of probability.

Summary functions are of the form $y = f(x_1, x_2, \cdots, x_n)$. A *parameter* is a summary function applied to a population; a *statistic* is a summary function applied to a sample. Probability distributions are dyadic functions whose left argument is a vector of parameters and whose right argument is usually a single numeric value. Discrete distributions are defined by the probability mass function; continuous distributions are defined by the density function. Relational and logical functions described in (Falkoff & Iverson, 1973) produce Boolean results, 1 being true and 0 being false. Examples of these functions are described in detail in (Mansour, 2017).

PROBABILITY

Calculating probabilities requires not only a distribution and its associated parameters, but also a value and a relationship. We need to construct a function which combines the distribution with the relationship. This requires creating a dyadic operator **probability** whose left operand can be any distribution function and whose right operand is a relational function. For example, the following expression calculates the upper tail probability for the standard normal distribution:

```
      normal 1.5              ⍝ Standard normal density at 1.5
0.12952
      normal probability < 1.5  ⍝ Cumulative normal probability
0.93319
```

If certain parameters are needed, we can include them; thus, to find the probability that a randomly selected person is taller than six feet, given the mean and standard deviation of heights are 68 and 3 inches respectively:

```
      68 3 normal probability > 72   ⍝ Upper-tail probability
```

In M. A. Sorto, A. White, & L. Guyot (Eds.), *Looking back, looking forward. Proceedings of the Tenth International Conference on Teaching Statistics (ICOTS10, July, 2018), Kyoto, Japan.* Voorburg, The Netherlands: International Statistical Institute. iase-web.org

```
0.091211
```

This works just as well for discrete distributions. Thus, the probability of tossing 7 fair coins and getting exactly 3 heads is about 27%.

```
      7 0.5 binomial probability = 3
0.27344
```

The general syntax of the *probability* operator is:

Result ← [Parameters] (distribution ***probability*** relation) Value

DISTRIBUTIONS

Distributions can be useful for obtaining probabilities. But we can also find critical values from them, determine their expected values or generate random variables from them. Other software programs such as Excel or R simply create variations of these distributions as separate functions. A better approach is to simply have one function for each distribution and modify it using the appropriate operator.

To find the critical value of a distribution, we start with a probability. This is an inverse process, so we use another operator ***criticalValue*** to do the job. The syntax is similar to that of ***probability***:

Result ← [Parameters] (distribution ***criticalValue*** relation) Proportion

The following expression gives us the critical value of the Student t distribution with 9 degrees of freedom. The syntax suggests that it is the value which is less than 5% of all values.

```
      9 tDist criticalValue < 0.05
1.8331
```

We can calculate the expected value and the variance of a distribution using the ***theoretical*** operator by combining it with the appropriate summary function. We use a dummy argument on the right to allow for the consistent representation of the distribution with parameters on the left.

```
      7 0.5 binomial theoretical mean '' ⍝ Expected Value
3.5
      7 0.5 binomial theoretical var '' ⍝ Variance
1.75
```

We can also use distributions to generate random variables from various distributions. To simulate the random selection of 5 individuals by measuring their heights, we use the ***randomVariable*** function which takes a distribution on its left and a sample size on the right and produces a vector of the appropriate length:

```
      68 3 normal randomVariable 5
66.974 67.566 69.933 68.071 74.094
```

INFERENTIAL STATISTICS

There are two main areas of inferential statistics: confidence intervals and hypothesis tests. We define two operators, ***confInt*** and ***hypothesis***, to handle these cases. The former is a monadic operator, which takes a summary function as its operand, while the latter is a dyadic operator which requires a summary function on its left and a relation on the right.

Confidence Intervals

The *confInt* operator modifies a summary function to produce a confidence interval instead of a point estimate. We represent a confidence interval as a 2-item vector containing the lower and upper bounds respectively. The optional left argument is the confidence level which defaults to 0.95. The syntax for the *confInt* operator is:

(*Lower*, *Upper*) ← [*ConfidenceLevel* | **0.95**] *summaryFunction* **confInt (***Data* | *Stats*)

```
      mean Height                ⍝ Point estimate for mean
68.98
      mean confInt Height        ⍝ 95% confidence interval
67.815 70.144
      mean confInt stats 49 68.98 4.053  ⍝ Using summary statistics
67.816 70.144
      0.99 mean confInt Height   ⍝ 99% confidence interval
67.427 70.533
```

Hypothesis Tests

Hypothesis tests are a more complex area of inferential statistics, and thus are harder to define as functions because there are a lot of moving parts and there is no clear-cut simple result. The p-value makes sense in some respects because the closer it is to 0 the more "false" the null hypothesis is. Unfortunately, the p-value has somewhat fallen out of favor with the statistical community, because it varies widely from sample to sample. (Nuzzo, 2014). Statistics textbooks usually list two approaches to hypothesis tests: the critical-value approach and the p-value approach. To test whether the average height exceeds 68 inches, we use the sample vector **Height** along with the *hypothesis* operator which produces a namespace that includes the test statistic and the p-value.

```
HYP ← Height mean hypothesis ≤ 68  ⍝ create namespace HYP
HYP.TestStatistic          ⍝ t-Statistic          1.691876225
HYP.P                      ⍝ p-Value          0.04857633797
```

To generate presentable output we apply the *report* function to this namespace, using the significance level as optional left argument.

```
0.05 report HYP          ⍝ Generate hypothesis report

_
X = 68.97959
s = 4.05298
n = 49
Standard Error: 0.57900

Hypothesis Test

 Ho: μ≤68        H1: μ>68
┌──────────────┬──────────────────┐
│Test Statistic:│P-Value:         │
│t=1.6919      │p=0.04858         │
├──────────────┼──────────────────┤
│Critical Value:│Significance Level:│
│t(α;df=48)=1.6772│α=0.05          │
└──────────────┴──────────────────┘
```

The hypothesis report above contains a comparison table which combines the critical-value and p-value approaches. The test statistic in the upper-left hand corner is calculated from the

data, and the significance level in the lower right is usually given; however comparing these values is impossible since they are scaled differently. To complete the table, we calculate across—the p-value from the test statistic and the critical value from the significance level; to complete the test, we compare down – the test statistic to the critical value and the p-value to the significance level.

SOFTWARE COMPARISON

(Chen, 2016) states that the suitability of a programming language for a particular task is less about functionality and more about the expressability. The use of operators allows us to express statistical concepts in a consistent way without a proliferation of functions.

Other statistical programs attempt to incorporate some of these concepts. *Excel* uses a dot separator, e.g. *NORM.DIST* and *NORM.INV*, while *R* uses a single-letter prefix followed by a distribution name, e.g. *pnorm*, and *qnorm* , but these are still distinct functions. *Mathematica* uses relations in conjunction with its probability functions to handle upper-tail and interval probabilities. While these developments are encouraging, they don't go far enough.

TamStat expresses ideas in a more readable, English-like form. Suppose the mean IQ is 100 with a standard deviation of 15. What is the probability that a randomly selected individual has an IQ above 90? Table 1 shows how to calculate this probability using Excel, R, Mathematica and TamStat:

Program	Statistical Expression
Excel	=1-NORM.DIST[90,100,15,TRUE]
Mathematica	Probability[x > 90, x ~ NormalDistribution[100,15]]
R	pnorm(90,100,15,lower.tail=FALSE)
TamStat	100 15 normal probability > 90

Table 1 - Probability expressions using various statistical programs

CONCLUSION

TamStat can be used by an instructor as an electronic blackboard to illustrate various statistical concepts. Students can use the program for homework assignments and tests. The elimination of tedious calculations and obscure syntax allows more material to be covered. Student reaction has been favorable over the past two years, especially to a graphical user interface which was added to provide expression builders for various statistical operations; one example is the probability wizard which can be seen at www.tamstat.com.

The unique structure of APL not only allows a domain expert to produce a system quickly without involving teams of programmers, it also provides a superior platform which to design a statistical program unlike any other. Arrays as first-class objects are a natural way to represent sample data. Functions can be used to define basic concepts such as a sample statistic or a probability distribution. And finally, the operator paradigm allows students to use previously-learned concepts to expand their knowledge and experiment with new ideas.

REFERENCES

Brown, J. (1984, June). Function Assignment and Arrays of Functions. *SIGAPL Quote Quad, 14*(4), 81-84.

Chen, J. (2016). Linguistic Relativity and Programming Languages. *JSM-2016.* ASA.

Falkoff, A., & Iverson, K. (1973, July). The Design of APL. *IBM Journal of Resarch and Development, 17*.

Iverson, K. (1962). A Programming Language. *May 1-3, 1962 Spring Joint Computer Conference* (pp. 345-351). New York: AIEE-IRE.

Iverson, K. (1979, June). The Role of Operators in APL. *APL Quote Quad, 9*(4), 128-133.

Mansour, S. (2017). Taming Statistics with Limited Domain Operators. *Vector, 27*(1).

Mason, J. (1986). Making APL Expressions Look Like English. In J. Mason, *Learning APL: An Array Processing Language* (pp. 190-228). New York: Harper & Row.

Nuzzo, R. (2014). Statistical Errors. *Nature, 506*(7487).

Scholes, J. (1994, July). Meeting: Dyalog APL Namespaces. *Vector, 11*(1), 101-107.

Smith, B. (1981). Nested Arrays, Operators and Functions. *SIGAPL Quote Quad, 12*(1), 286-290.