The functions form a pedagogic sequence in the sense that to understand any one of them you must first understand those that precede it. Each function can be directly defined in a single line, and each takes the original data as its argument.

Next, in direct definition:

```
MEAN:(+/ω)÷0⊥ρω
DEV:ω-(MEANω)∘.+(0⊥ρω)ρ0
SS:(DEVω)+.*2
VAR:(SSω)÷¯1+0⊥ρω
SD:(VARω)*0.5
SP:M+.×⍉M←DEVω
COV:(SPω)÷¯1+0⊥ρω
COR:(COVω)÷S∘.×S←SDω
```

Using Iverson's new dialect J,[29-31] the same functions can be defined even more succinctly, and without parentheses. Not only are no variables assigned, no explicit reference is made to the arguments. This is *tacit definition*, or pure functional programming (Backus, 1978), which leads to efficient execution and invites parallel processing. (Version 3.3 of Iverson's J is used for the examples that follow.)[32]

```
mean=.+/%#

dev=.-mean

ss=.+/@*:@dev

var=.ss%<:@#

sd=.%:@var

sp=.+/ .*~|:@dev

cov=.sp%<:@#

cor=.cov%*/~@sd
```

The sequence of functions starts with the mean and ends with the correlation coefficient. Is this structured programming? Is it top down or bottom up? Such questions seem to vanish in a sequence that is almost self-documenting.

The style of programming brings to mind the words of Babbage: "The almost mechanical nature of many of the operations of Algebra, which certainly contributes greatly to its power, has been strangely misunderstood by some who have even regarded it as a defect. When a difficulty is divided into a number of separate ones, each individual will in all probability be more easily solved than that from which they spring. In many cases several of these secondary ones are well known, and methods of overcoming them have already been contrived: it is not merely useless to re-consider each of these, but it would obviously distract the attention from those which are new: something very similar to this occurs in Geometry; every proposition that has been previously taught is considered as a known truth, and whenever it occurs in the course of an investigation, instead of repeating it, or even for a moment thinking on its demonstration, it is referred to as a known datum. It is this power of separating the difficulties of a question which gives peculiar force to analytical investigations, and by which the most complicated expressions are reduced to laws and comparative simplicity."[33]

Revisiting our roots

Being aware of the long history of functions in mathematics, and having seen examples written in current APL, we can now use APL to illuminate our roots, which reach back to Egyptian hieroglyphics. The word *algorithm*, according to the Oxford English Dictionary, is an erroneous refashioning of *algorism*, a word derived from "al-Khowarazmi, the native of Khowarazm, surname of the Arab mathematician who flourished early in the 9th Century, and through the translation of whose work on Algebra, the Arabic numbers became generally known in Europe." In its original form it was used by Chaucer, and the Oxford dictionary cites the use of *algorithm* in 1774.[34,35] I found it first used by Sylvester[36] in one of the earliest papers to speak of *matrices* (compare References 27 and 37 for APL treatment of polygons and polyhedra).

The earliest known book of algorithms is the Rhind Papyrus, based on work written 2000–1800 BC and copied by Ahmes the scribe in 1650 BC.[38-40] It is a textbook on solving practical problems. Consider a simple example, shown in Figure 6 and using Figure 2, again, as the key; to multiply 12 by 12 begin by writing down 12, and by successive doublings obtain 1, 2, 4, and 8 times 12. Check the rows 4× and 8× (on the papyrus the check marks are red) and add them to get the required result. The symbol preceding the answer is a rolled-up scroll (*quod erat demonstrandum*), which in fancy we may take as the ancestor of our *equals* and APL's *assignment* symbols.[41]