June 1995

Some Notes on Introducing J with Statistical Examples

Revised Edition

Keith Smillie
Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2H1
smillie@cs.ualberta.ca

J, a modern dialect of APL, is introduced with examples chosen from probability and statistics. The examples include means, variances, medians and quartiles, one- and two-dimensional frequency distributions, regression and analysis of variance, ranks, simulation and random number generation, probability distributions, and the use of **J** with other software packages.

Table of contents

Introduction
Arithmetic mean
Precedence
Geometric and harmonic means
Forks and hooks
Variance and standard deviation
Valence
Covariance and correlation I
Median and quartiles
Noun rank
Discrete frequency distributions
Mode
Continuous frequency distributions
Box and open
Barcharts
Character frequency distributions
Discrete frequency tables
Stem-and-leaf diagrams
Explicit definition I
Summary statistics
Verb rank
Linear regression
Base and antibase
Covariance and correlation II
Permutations and combinations
Ranks
Rank tests
Runs
Explicit definition II
Analysis of variance
Simulation I
Central limit theorem
Simulation II
De Méré's problems
Derivatives and integrals
Probability distributions
Binomial distribution
Poisson distribution
Locales
Utilities
Other software - Graphics
Other software - Spreadsheets
Other software - Statistics
Gerunds
Windows programming
Coupon collector's problem
Cut and word formation
J and statistics
Acknowledgements
References

Appendices
1. Language summary
2. Multiple regression
3. Analysis of variance
4. Statistics and literary style
5. Script file

Introduction

J is a modern dialect of APL using the standard ASCII character set which has been developed by Kenneth Iverson and his associates at Iverson Software Inc. The language, **J** Release 2.05 for Windows, is fully described in ***J*** *Introduction and Dictionary*, Iverson (1994), which is an indispensible reference in learning and using the language. The language summary of Appendix 1 is from this reference. The ***J*** *Introduction and Dictionary* and the ***J*** *User Manual*, Burke (1994), are available online in the present version of J.

Primitive functions are represented by single characters which may be suffixed by a period or a colon, and which, as in APL, have no hierarchy for order of execution. The valency of functions is determined by context as with the minus sign in conventional arithmetic which may represent either the difference of the arguments if there are two or the negation of a single argument.

Some of the terms used in **J** have been taken from English grammar rather than from mathematics or computing. Thus functions, either primitive or defined, are called *verbs* since they act upon their argument or arguments which may be called *nouns* although this last term is seldom used in this report. Furthermore verbs may be modified by *adverbs* and *conjunctions* to extend their meaning and utility. The terms *atoms*, *lists* and *tables* are used in place of the usual mathematical terms scalars, vectors and matrices. Arrays of more than two dimensions are called *reports*.

Verbs may be defined in **J** in a *functional* or *tacit* style without the use of named arguments. However, arguments may be specified in *explicit* verbs where the definition may extend over several lines and involve control structures similar to those in conventional programming languages.

For example, the expression **2+3** has the value **5** and the expression **+3** is **3**, illustrating the dyadic verb *plus* and the monadic verb *conjugate* both represented by **+**. The expression **+/3 2 5**, where **+/** is a verb derived from **+** by the adverb **/**, gives the sum **10** over the items in the list. Both the list and the derived verb may be named and subsequently used so that, if **sum=. +/** and **a=. 3 2 5**, then **sum a** is **10**. Finally,

```
add2=. 2&+
```

where the conjunction **&** binds the left argument **2** to the verb, is a monadic verb which adds **2** to its argument, and **add2 a** is **5 4 7**.

The present report introduces **J** by means of a number of examples chosen somewhat arbitrarily from elementary probability and statistics. Syntactic and semantic details of the features of **J** which are introduced are usually limited to those required for an understanding of the examples. Again the reader is encouraged, even implored, to consult the ***J*** *Introduction and Dictionary* while reading this report.

Arithmetic mean

The arithmetic mean of a list of observations is the sum of the observations divided by the number of observations. If the observations are

```
w=. 2.3 5 3.5 6
```

then the arithmetic mean is **+/w % #w** which has the value **4.2**. In this expression the verb **+/**, derived from the verb *plus* **+** by the adverb *insert* **/**, gives the sum of the items of **w**, **%** is the verb *divided by*, and **#** is the verb *tally* which gives the number of items in its argument. Since this expression may be written more simply as **(+/ % #)w**, we may define the verb

```
am=. +/ % #
```

so that **am w** is **4.2**.

If we define the synonyms

```
sum=. +/
```

and

```
tally=. #
```

we may write the definition of the arithmetic mean as

```
am=. sum % tally
```

Precedence

Precedence amongst verbs is determined by parentheses, and in their absence the right argument is the entire expression on the right and the left argument is the noun immediately on the left. For example, each of the expressions **2*3+4** and **2*(3+4)** is equal to **14**, while **(2*3)+4** is equal to **10**. As another example, the expression **(+/w) % (#w)** for the arithmetic mean of the list **w** may be written in the form **(+/w) % #w** used in the last section. Adverbs and conjunctions take precedence over verbs with the left argument being the entire verb phrase on the left as shown by the expression **(+/ % #)w** given above for the arithmetic mean.

Geometric and harmonic means

The geometric mean of a list of *n* observations is the *n*th root of the product of the observations. We may define the verb

```
gm=. # %: */
```

where `%:` is the dyadic verb *root*, for the geometric mean, and the expression **gm w** is equal to **3.94211**.

The harmonic mean is the reciprocal of the arithmetic mean of the reciprocals of the observations, and is given by the verb

```
hm=. % @ am @ %
```

and **hm w** is equal to **3.6793**. We note the use of the conjunction *atop* (or *after*) **@** so that the definition of **hm** may be read as "apply **%** (*reciprocal*) to the result of applying **am** to the result of applying **%** to each of the observations".

A three-item list of the arithmetic, geometric and harmonic means is given by the verb

```
means=. am,gm,hm
```

where **,** is the verb *append*, and **means w** is equal to **4.2 3.94211 3.6793**.

If we introduce the dyadic adverb *pass* **~** which interchanges the arguments of the verb, the geometric mean may be expressed in a similar manner to the arithmetic mean as

```
gm=. */ %:~ #
```

Finally, the geometric mean, which may be defined also as the antilogarithm of the arthmetic mean of the logarithms of the observations, may be expressed as

```
gm=. ^ @ am @ ^.
```

where **^** and **^.** are the monadic verbs *exponential* and *natural log*, respectively.

Forks and hooks

A sequence of three verbs such as occurs in the definition of the arithmetic mean as **+/ % #** is known as a *fork*. It is used in conventional mathematical notation, where, for example, $(f_1+f_2)x$ represents the sum $f_1(x)+f_2(x)$. In general, if **f**, **g** and **h** are verbs, then the fork **(f g h)y** is equal to **(f y) g (h y)**.

A sequence of two verbs is called a *hook*. The hook **(g h)y** is equal to **y g (h y)**. An example is the expression for deviations from the mean given in the next section since **(- am)w** is equal to **w - am w**.

Forks and hooks may also have two arguments. The fork **x(f g h)y** is equal to

```
(x f y) g (x h y)
```

and the hook **x(g h)y** is **x g (h y)**.

Forks and hooks are special cases of *isolated*, i.e., uninterrupted, sequences of an arbitrary number of verbs which may be evaluated by repeated resolution into zero or more forks followed by a final fork or hook. For example, the sequence of verbs **d e f g h** represents the fork **d e (f g h)** where the verb on the right **f g h** is itself a fork. Two other examples are deviations from the mean which may be expressed as **dev=. - +/ % #**, a fork followed by a hook, and the verb **means** defined in the previous section which is a fork followed by another fork.

Variance and standard deviation

The variance of a list of observations is defined as the sum of squares of the deviations of the observations from the arithmetic mean divided by one less than the number of observations. The deviations of the items **w** from their mean **am w** are given by

```
(- am)w
```

which is equal to

```
_1.9 0.8 _0.7 1.8
```

(We may note that the underbar **_** represents the *negative sign*, and that the decimal point is necessarily preceded by at least one digit so that the second deviation, for example, must be expressed as **0.8**.) Therefore, if we define

```
dev=. - am
```

then verbs for the sum of squares of deviations from the mean may be defined as

```
ss=. +/ @ *: @ dev
```

and the variance as

```
var=. ss % <: @ #
```

where ***:** is the verb *square* and **<:** is *decrement* which subtracts **1** from its argument. (The monadic verb *increment* **>:** adds **1** to its argument, and, for example, **<:5** is **4** and **>:5** is **6**.) The variance of **w** is then **var w** which is equal to **2.66**.

The standard deviation, defined as the positive square root of the variance, is given by

```
sd=. %: @ var
```

where **%:** is the monadic verb *square root*, and **sd w** is equal to **1.63095**.

Valence

Verbs are either *monadic* with a single right argument or *dyadic* with left and right arguments. For example, **%** represents the monadic verb *reciprocal* and the dyadic verb *divided by*, and **%4** is **0.25** and **3%4** is **0.75**. The conjunction *bond* **&** may be used to bind an argument to a dyadic verb. For example, **mod7=. 7&|**, where **|** is the dyadic verb *residue*, gives 7-residues, and

mod7 10 is **3**, and **cube=. ^&3**, where **^** is the verb *power*, gives cubes, and **cube 5** is **125**.

Covariance and correlation I

The covariance of two sets of observations is defined as the sum of the products of the deviations of the observations from their respective means divided by one less than the number of pairs of observations, and the correlation coefficient is the covariance divided by the the product of the standard deviations. To illustrate the calculations in this section we shall use the observations

```
y1=. 5.27 5.68 6.25 7.21 8.02
                    8.71 8.42
```

and

```
x1=. 12 18 24 30 36 42 48
```

from Hoel (1966) representing the yield of a crop and the corresponding amount of water applied, respectively.

Since the sum of products is given by

```
sp=. +/ @ (*&dev)
```

,

the covariance may be defined as

```
cov=. sp % <:@#@]
```

where the verb *right* **]** gives its right argument. (The verb *left* **[** gives its left argument so that, for example, **2]3** is **3** and **2[3** is **2**.) With the sample data, **x1 sp y1** is **103.68** and **x1 cov y1** is **17.28**. The correlation coefficient is given by

```
cor=. cov % *&sd
```

and **x1 cor y1** is **0.972408**.

Median and quartiles

The median of a list of observations is defined as the middle observation when they are arranged in sorted order if the number of observations is odd, and the average of the two middle observations if the number is even. For example, if **u** is the ten-item list

```
22 14 32 30 19 16 28 21 25 31
```

,

then the median is **23.5** since the items in sorted order are

```
14 16 19 21 22 25 28 30 31 32
```

and the middle items are **22** and **25**.

Sorting may be accomplished by the monadic verb *grade up* and the dyadic verb *sort (up)* each represented by **/:**. The monadic verb grades its argument giving the permutation which would sort the items of the argument in ascending order, and, for example, **/:w**, where **w** is the list **2.3 5 3.5 6** introduced previously, is **0 2 1 3**. (We note the use of zero-origin indexing so that the smallest item has the index **0**, the next item the index **1**, etc.) The dyadic verb sorts the left argument to an order specified by the grade up of the right argument, so that **w/:w** gives the list **2.3 3.5 5 6** of the items of **w** sorted in ascending order. This last expression may also be written as **/:~ w**, where **~** is the monadic adverb *reflex* which provides the preceding verb with equal left and right arguments. We shall define the verb

```
sort=. /:~
```

which will sort the items of its argument in non-decreasing order, so that, for example, **sort w** is **2.3 3.5 5 6**.

The verbs *grade down* and *sort (down)* represented by **\:** provide sorting in descending order, and, for example, **\:w** is **3 1 2 0** and **\:~w** is **6 5 3.5 2.3**.

The indices of the middle item or items of a list are given by

```
midindices=. (<.,>.)@-:@<:@#
```

where **<.**, **>.** and **-:** are the monadic verbs *floor*, *ceiling* and *halve*, respectively, and the corresponding middle values are given by

```
midvalues=. midindices { ]
```

where the verb *from* **{** selects from its right argument those items whose indices are given by the left argument. For example, **midindices u** is **4 5** and **midvalues sort u** is the two-item list **22 25**. Therefore, if we define

```
q2=. am @ midvalues
```

to give the median of a sorted list, then the median of an arbitrary list is given by

```
median=. q2 @ sort
```

and **median u** is **23.5**.

The three *quartiles* of a sorted list are defined so that one quarter of the items lie between consecutive quartiles or between an end of the list and the adjacent quartile. The median is thus the second quartile, and we may define the synonym

```
Q2=. median
```

.

The first quartile is calculated simply as the median of those items less than the median. Therefore, the first quartile of a sorted list is given by

```
q1=. q2 @ ((q2 > ]) # ])
```

,

where the dyadic verb *copy* **#** selects the items less than the median. (The verb **#** copies items from its right argument according to the items in its left argument, and, for example, **#~ i. 4** is **1 2 2 3 3 3**.) The first quartile of an arbitrary list is

```
Q1=. q1 @ sort
```

and **Q1 u** is **19**. Similarly, the third quartile may be

defined as

```
Q3=. q3 @ sort
```

where

```
q3=. q2 @ ((q2 < ]) # ])
```

gives the third quartile for a sorted list, and **Q3 u** is **30**.

Finally, a five-statistic summary giving the smallest item, the three quartiles and the largest item may be defined as

```
five=. ({.,q1,q2,q3,{:) @ sort
```

where the verbs *head* **{.** and *tail* **{:** give the first and last items, respectively, of a list, and **five u** is the five-item list **14 19 23.5 30 32**.

Noun rank

Nouns may be single items or *atoms*, one-dimensional arrays or *lists*, two-dimensional arrays or *tables*, or arrays of higher dimension or *reports*. The dimension of a noun is its *rank*, and atoms, lists and tables are of rank 0, 1 and 2, respectively. For example, the table **t** whose value is

```
0  1  2  3
4  5  6  7
8  9 10 11
```

given by the expression **t=. i. 3 4** where **i.** is the monadic verb *integers*, has three rows of four columns each and has shape **3 4** and rank **2**. The last *k* axes of a noun determine a *rank k-cell* or more simply a *k-cell* of the noun. For example, the 0-cells of **t** are the atoms **0**, **1**, **2**, ... , the 1-cells are the rows, and the 2-cell is **t** itself. For an array of rank r the (r-1)-cells are known as the *(major) items*.

The *rank* conjunction **"** is used to apply a verb, primitive or defined, to each of the k-cells of its argument so that in the expression **u"k y** the verb **u** is applied to each **k**-cell of the argument **y**. For example, **am"1 t** is the list **1.5 5.5 9.5** of row means of **t**, and **am"2 t** is the list **4 5 6 7** of column means. These results may be summarized by the expression

```
(] ; am"1 ; am"2) t
```

which has the value

```
+---------+-----------+-------+
|0 1  2  3|1.5 5.5 9.5|4 5 6 7|
|4 5  6  7|           |       |
|8 9 10 11|           |       |
+---------+-----------+-------+
```

where the dyadic verb *link* **;** boxes its arguments. The expression **am"0 t** gives the means of the atoms and is equal to **t**.

We might emphasize that the adverb **/** produces a verb which may be further modified by the rank conjunction which is applied along the first axis of its argument. Thus for the table **a** above **am"2 a** or more simply **am a** gives the column means. The following example using the report **r=. i. 2 3 2** provides further illustrations of insertion:

```
(];+/"1;+/"2;+/"3) r
+-----+--------+-----+-----+
| 0  1| 1  5  9| 6  9| 6  8|
| 2  3|13 17 21|24 27|10 12|
| 4  5|        |     |14 16|
|     |        |     |     |
| 6  7|        |     |     |
| 8  9|        |     |     |
|10 11|        |     |     |
+-----+--------+-----+-----+
```

Discrete frequency distributions

We shall define verbs which give the frequency distribution for discrete data, i.e., data whose range is restricted to the non-negative integers **0**, **1**, **2**, ... We shall need the adverb *table* **/** which gives an array formed by inserting the verb it modifies between all possible pairs of items chosen from the two arguments. For example, if **p=. i. 5** and **q=. i. 6,** then the expression **p+/q** gives the first five rows and six columns of an addition table. This table may be given in the convenient form

```
+-+----------+
| |0 1 2 3 4 5|
+-+----------+
|0|0 1 2 3 4 5|
|1|1 2 3 4 5 6|
|2|2 3 4 5 6 7|
|3|3 4 5 6 7 8|
|4|4 5 6 7 8 9|
+-+----------+
```

by the expression

```
p by q over p+/q
```

where

```
over=. ({.,.@;}.)@":@,
```

and

```
by=. ' '&;@,.@[,.]
```

are utility verbs given by Kenneth Iverson and are intended for use rather than immediate study,

The monadic verb *nub* **~.** selects the distinct items from its list argument. For example, if **a** is the list **1 1 0 3 1 3 1**, then **~. a** is **1 0 3**. The monadic verb *self-classify* **=** gives the distribution table which relates the items of its argument to the nub of the argument, and, for example, **= a** is

```
1 1 0 0 1 0 1
0 0 1 0 0 0 0
0 0 0 1 0 1 0 .
```

Since the row sums of the distribution table give the frequency of occurrence of the items of the nub, we define

```
fr=. +/"1 @ =
```

to give the list of frequencies so that **fr a** is **4 1 2** which means that **a** has four **1**s, one **0** and two **3**s. A simpler definition of **fr** is

```
fr=. #/.~
```

where **/.** is the adverb *key* which groups the items of the right noun argument according to the key given by the left noun argument and then applies its verb argument to each group.

The verb

```
sfr=. fr @ sort
```

will give the frequencies of the items in the nub in their sorted order, and **sfr a** is **1 4 2**.

Now consider the frequencies over a specified range of values. For example, for the list **a** the frequencies over the range **0 1 2 3** are **1 4 0 2** indicating that **a** has one **0**, four **1**s, zero **2**s and two **3**s. These range frequencies may be found very simply if we note that the expression **0 1 2 3 =/ a** has the value

```
0 0 1 0 0 0 0
1 1 0 0 1 0 1
0 0 0 0 0 0 0
0 0 0 1 0 1 0 .
```

Therefore, the range frequencies are given by the verb

```
rfr=. +/"1 @ (=/)
```

and **0 1 2 3 rfr a** is **1 4 0 2**.

Finally, the frequencies over the range from **0** to the maximum value are given by the verb

```
mfr=. (i.@>:@(>./)) rfr ]
```

where **>.** is the dyadic verb *larger of*, and, for example, **mfr a** is **1 4 0 2**.

Any of these lists of frequencies may be conveniently displayed in a two-column table with the items of the nub or the range in the first column and the corresponding frequencies in the second. The verbs

```
frtab=. ~. ,. fr
```

where **,.** is the dyadic verb *stitch*, and

```
sfrtab=. frtab @ sort
```

give frequency tables for the nub and sorted nub, respectively. A range frequency table is given by the verb

```
rfrtab=. [ ,. rfr .
```

A maximum frequency table is given by

```
mfrtab=. (i.@>:@(>./)) ,. mfr .
```

As an example, the expression

```
(frtab;sfrtab;0 1 2 3 4&rfrtab;
                mfrtab) a
```

has the value

```
+---+---+---+---+
|1 4|0 1|0 1|0 1|
|0 1|1 4|1 4|1 4|
|3 2|3 2|2 0|2 0|
|   |   |3 2|3 2|
|   |   |4 0|   |
+---+---+---+---+
```

As another example, the list **c** given by

```
1 5 2 4 6 5 1 6 2 5
```

might represent the results of rolling a die ten times. Then **fr c** is the list **2 3 2 1 2** which shows that their are two **1**s, three **5**s, two **2**s, one **4** and two **6**s, and

```
1 2 3 4 5 6 rfr c
```

is the list **2 2 0 1 3 2** which gives the frequencies of the faces from **1** to **6** in natural order. If we define the monadic verb

```
DieFr=. 1 2 3 4 5 6 & rfr
```

so that the left argument of **rfr** is always the list **1 2 3 4 5 6**, the frequencies for rolling a die are given simply by **DieFr c**.

Mode

The mode is defined as that item which occurs most frequently, and, for example, for the list **a** of the last section, which has the value **1 1 0 3 1 3 1**, the mode is **1**. The verb **frtab** given in the last section may be used to find the mode which is simply the item (or items) in the first column in the rows corresponding to the maximum item in the second.

First we shall define the verb

```
imax=. [] e. >./) # i.@#@]
```

to give the index of the maximum item in a list, and, for example,

```
imax z=. 4 6 2 1 3 6 0
```

is equal to **1 5** which gives the indices of the maximum item **6**. The dyadic verb *member* or *in* **e.** gives a list of **0**s and **1**s with the **1**s indicating the matches of the right argument in the left, and, for example, **z e. 6** is the list **0 1 0 0 0 1 0**.

The verb **mode** may now be defined as

```
mode=. (([:imax{:"1){{."1)@frtab
```

and **mode a** is **1**. Two more examples are

```
mode 1 2 3 2 3 2 3 4
```

which is **2 3**, and

```
mode 1 2 3 4
```

which is **1 2 3 4**.

In the definition of **mode** we note the use of the verb *cap* **[:** which caps the left branch of a fork often allowing a verb to be defined with fewer parentheses as a train. Without the cap **mode** would have had to be defined as

```
mode=. ((imax@({:"1@])){{."1)
       @frtab
```

Continuous frequency distributions

Consider now the problem of finding a frequency distribution table for continuous data, i.e., data whose values are not restricted to integers. For example, suppose we have the list **b** given by

```
5.2 8.6 3.4 8.1 9.2 5.3 4.1 8.2
                    9.9 4.8
```

and we wish to find the number of items in each of the intervals defined by the two-item lists **1 3**, **3 5**, **5 7**, **7 9** and **9 11**. The resulting frequency classification is the list **0 3 2 3 2** indicating that there are no items of **b** in the first interval, three in the second interval, etc. In this section we shall use the verb **rfr** in the definition of the verb **cfrtab** for continuous frequency distributions.

The intervals of classification may be specified by three parameters - the left-hand side of the first interval, interval width, and number of intervals - which in this example have the values **1**, **2** and **5**, respectively. The classification may be done by applying the verb **rfr** to the result of taking the floor of the quotient of the interval width divided into the items less the left-hand side of the first interval. For the sample data these calculations are as follows:

```
   b
5.2 8.6 3.4 8.1 9.2 5.3 4.1 8.2 9.9 4.8
   b - 1
4.2 7.6 2.4 7.1 8.2 4.3 3.1 7.2 8.9 3.8
 (b - 1) % 2
2.1 3.8 1.2 3.55 4.1 2.15 1.55 3.6 4.45
                              1.9
 <.(b-1)%2
2 3 1 3 4 2 1 3 4 1
 (i. 5) rfr <.(b-1)%2
0 3 2 3 2
```

The frequency classification may be accomplished by the verb **cfr** defined as

```
cfr=.i.@{:@[rfr[:<.()-{.@[)%1&{@[
```

The midpoints of the intervals, **2 4 6 8 10** in the example, are given by the verb

```
cfrpts=. -:@(1&{)+{.+1&{ * i.@{:
```

The required continuous frequency table is given by

```
cfrtab=. cfrpts@[ ,. cfr
```

and

```
1 2 5 cfrtab b
```

is the table

```
 2 0
 4 3
 6 2
 8 3
10 2
```

Box and open

The expression **i. 5** gives the five-item list **0 1 2 3 4**, whereas the expression **<i. 5**, where **<** is the verb *box*, gives the atom

```
+---------+
|0 1 2 3 4|
+---------+
```

Boxing is useful in displaying the action of verbs. For example, **+/\i. 5**, where **** is the adverb *prefix*, gives the list **0 1 3 6 10** of partial sums, and the expression **<\i. 5**, which has the value

```
+-+---+-----+-------+---------+
|0|0 1|0 1 2|0 1 2 3|0 1 2 3 4|
+-+---+-----+-------+---------+
```

shows the partitions from which these sums are obtained. The verb inverse to box is *open* **>**, and, for example, **><i. 5** is equal to **i. 5**.

The phrase **f &. g**, where **&.** is the conjunction *under*, is equivalent to **f & g** followed by the application of the verb obverse to **g**. For example,

```
x=. 1 2 3; 4 5; 6 7 8 9
```

has the value

```
+-----+---+-------+
|1 2 3|4 5|6 7 8 9|
+-----+---+-------+
```

and **(+/ &. >) x** is the boxed list

```
+-+-+--+
|6|9|30|
+-+-+--+
```

giving the sums of the items of **x**. (The expression **(+/ & >) x** is the three-item list **6 9 30**.) The harmonic and geometric means may be defined in terms of the arithmetic mean as

```
hm=. am &. %
```

and

```
gm=. am &. ^.
```

respectively.

The verb

```
BoxSort=. /:>
```
may be used to sort the items in a boxed list. For example, `<"0 (6 9 5 7)` is the list

```
+-+-+-+-+
|6|9|5|7|
+-+-+-+-+
```
and
```
BoxSort <"0 (6 9 5 7)
```
is the sorted list
```
+-+-+-+-+
|5|6|7|9|
+-+-+-+-+ .
```

Barcharts

We may wish to append a barchart to one of the frequency tables discussed in previous sections. This may be accomplished by the verb
```
barchart=. ] ;"1 0 bars @
                    ({:"1) @ ]  ,
```
where
```
bars=. #&'*' &. >  ,
```
whose argument is one of the frequency tables as is illustrated by the expression
```
barchart 0 1 2 3 rfrtab a
```
which gives the following result:
```
+---+----+
|0 1|*   |
+---+----+
|1 4|****|
+---+----+
|2 0|    |
+---+----+
|3 2|**  |
+---+----+
```

Character frequency distributions

In this section we shall develop verbs for finding the frequency of occurrence of characters in a list of characters. First we shall give a verb for finding the frequencies of arbitrary characters in a text, and then several verbs derived from this verb where the characters are restricted to the twenty-six lower-case letters.

The verb
```
charfr=. [: +/"1 (=/)
```
will give the frequency of occurrence in the text given as the right argument of the characters specified by the left argument. For example, for a list `t` of text the expression `'.?!' charfr t` gives the frequencies in `t` of the stop characters `.`, `?` and `!`. The verb
```
charfrtab=. [ ;"0 charfr
```
gives a two-column frequency table with the characters in the first column and the frequencies in the second.

To conveniently define a verb to give the frequencies of lower-case letters in a text we shall use the noun
```
alpha=. 'abcdefghijklmnopqrst
                        uvwxyz'
```
(This character list is also given by the expression `(97+i. 26) { a.`, where the *alphabet* `a.` gives the list of the 256 elements in the ASCII character set.) The frequency of occurrence of letters in a character list is given by
```
afr=. alpha & charfr
```
and, for example,
```
afr t=. 'toronto'
```
is the 26-item list
```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 3
      0 0 1 0 2 0 0 0 0 0 0 0
```
giving the frequency of the letters. A two-column frequency table with the letters in the first column and the corresponding frequencies in the second is given by
```
afrtab=. alpha & charfrtab  .
```
A condensed table in which rows corresponding to letters which do not appear in the text are removed is given by
```
afrnubtab=. (([: 0&~: >@{:"1)
             # ]) @ afrtab  .
```
Finally, the verb
```
safrnubtab=.(([:\:[:>({:"1)@]) {
            ])@afrnubtab
```
gives a condensed table with the rows in descending order of frequency.

As an example the expression
```
(<afrnubtab t),. <safrnubtab t
```
gives the following condensed tables in alphabetical order and frequency order for the list `'toronto'`:
```
+-----+-----+
|+-+-+|+-+-+|
||n|1|||o|3||
|+-+-+|+-+-+|
||o|3|||t|2||
|+-+-+|+-+-+|
||r|1|||n|1||
|+-+-+|+-+-+|
||t|2|||r|1||
|+-+-+|+-+-+|
+-----+-----+
```
.

Discrete frequency tables

Two-dimensional discrete frequency tables are given by the verb
```
RFR=. [: +/"1 ({@[) =/ ]
```
where the left argument is a two-item list giving the

ranges for each of the axes and the right argument the list of pairs of values to be classified. For example, if

```
y=. 0 1;1 1;1 2;0 1;0 0;1 2;
      0 0;1 0;0 1  ,
```

then

```
(0 1;0 1 2) RFR y
```

is the table

```
2 3 0
1 1 2 .
```

The verb **RFR** gives frequency classifications for an arbitrary number of dimensions. For example, if **y** is a list of triples whose items are either **0** or **1**, then the expression

```
(0 1;0 1;0 1) RFR y
```

will give an array of rank **2 2 2** whose items give the frequency of occurrence of the triples in **y**. As another example, the range frequency **rfr** of a previous section may be defined as

```
rfr=. <"1 @ [ RFR {@ (<"1 @ ]) .
```

A test of independence in a two-way contingency table is provided by the chi-square statistic which gives a measure of the departure of the observed frequencies from those expected if the two variables of classification are unrelated. The expected frequencies are given by $exp_{ij} = r_i c_j / N$, where r_i and c_j are the sums of the ith row and jth column, respectively, and N is the total frequency. If the observed frequencies are obs_{ij}, then the test statistic is $\Sigma(obs_{ij} - exp_{ij})^2/exp_{ij}$ which has an approximate chi-square distribution with $(r-1)(c-1)$ degrees of freedom, where r and c are the number of rows and columns in the table. The corresponding verbs are

```
ExpFrTab=. (+/"1 */ +/)%tsum  ,
```

where

```
tsum=. +/@,
```

and **,** is the monadic verb *ravel*, gives the total for all items in the table, for the expected frequencies, and

```
ChiSqTab=. [: tsum ([: *: ] -
      ExpFrTab) % ExpFrTab
```

for the chi-square statistic. If we use the data **TableData** with the value

```
18 29 70 115
17 28 30  41
11 10 11  20
```

from Hoel (1966), then the expected frequencies are given by **ExpFrTab TableData** which has the value

```
26.68 38.86 64.38 102.08
13.34 19.43 32.19  51.04
 5.98  8.71 14.43  22.88  ,
```

and the value of chi-square is **ChiSqTab TableData** which is **19.9426**. We may conclude that the two variables of classification, level of education and marriage adjustment, are related.

If the frequencies are small in a two-by-two table, then it is suggested that the exact probabilities be calculated for the given table and all other tables less likely to occur if the two variables of classification are independent. For a given table with frequencies a, b, c and d the associated probability is

$$r_1!r_2!c_1!c_2!/a!b!c!d!N!\ .$$

Therefore, the corresponding verb is

```
pr22=. (*/ @ !@(+/"1 , +/)) %
      */ @ (! @ (, , tsum))  ,
```

where **!** is the monadic verb *factorial*. As an example consider the table **Table22Data**

```
+---+---+---+
|5 2|6 1|7 0|
|3 3|2 4|1 5|
+---+---+---+
```

from Steel and Torrie (1960) where the first item is the given table and the second and third represent the less likely tables. The associated probabilitities are

```
(pr22 & >)Table22Data
```

which has the value

```
0.32634 0.08159 0.00466
```

and sum **0.41259** so thay we may conclude that the two criteria of classification are independent.

Stem-and-leaf diagrams

Discrete data may be summarized by stem-and-leaf diagrams which group the sorted data by their ten-residues. For illustrative purposes we shall use the list **u** introduced earlier which has the value

```
22 14 32 30 19 16 28 21 25 31
```

and the list

```
U=. sort u
```

which is

```
14 16 19 21 22 25 28 30 31 32 .
```

Classification by ten-residues may be done by the verb

```
classify=. (div&10 </. ]) ,
```

where

```
div=. <. @ %
```

gives the integer quotient, and **classify U** is equal to

```
+--------+----------+--------+
|14 16 19|21 22 25 28|30 31 32|
+--------+----------+--------+ .
```

The data in each decade may be represented more concisely by giving only the unit's digits of the observations, the leaves, preceded by a single indication of the associated decade, the stem. This stem-and-leaf representation is given by

```
sl=. stem ; leaves
```
where
```
stem=. {. @ (10&* @ div&10)
```
and
```
leaves=. 10&|
```
and **sl 21 22 25 28** is
```
+--+-------+
|20|1 2 5 8|
+--+-------+
```
.

A stem and leaf list representation is given by the verb
```
SLlist=. (sl &. >) @ classify
```
and **SLlist U** is
```
+----------+-----------+----------+
|+--+-----+|+--+-------+|+--+-----+|
||10|4 6 9|||20|1 2 5 8|||30|0 1 2||
|+--+-----+|+--+-------+|+--+-----+|
+----------+-----------+----------+
```
.
The usual tabular representation is given by
```
SLtable=. ;"0 @ SLlist @ sort
```
where **;** is the monadic verb *raze* which ravels the opened items of its argument and the verb **sort** allows an unsorted argument, and **SLtable u** is
```
+--+-------+
|10|4 6 9  |
+--+-------+
|20|1 2 5 8|
+--+-------+
|30|0 1 2  |
+--+-------+
```
.

Explicit definition I

The *explicit definition* conjunction **:** may be used to define verbs, conjunctions, adverbs and nouns with explicit arguments. As a simple example,
```
AM=. 3 : 0
(+/ % #) y.
:
(+/x.*y.) % +/x.
)
```
gives an explicit definition of **AM** whose monadic use is equivalent to the verb **am** for the arithmetic mean and whose dyadic use gives a weighted mean. For example, if **w=. 2.3 5 3.5 6** as defined previuosly, then
```
AM w
```
is **4.2** and
```
1 3 3 1 AM w
```
is **4.225**.

Left and right arguments are represented by **x.** and **y.**, respectively, monadic and dyadic definitions are separated by a colon on a line by itself, and the definition is terminated by a right parenthesis. The explicit definition of **AM** may be displayed as
```
+-+-+----------------+
|3|:|(+/ % #) y.     |
| | |:               |
| | |(+/x.*y.) % +/x.|
+-+-+----------------+
```
An equivalent definition of **AM** is
```
+-+-+--------------------+
|3|:|((# $ 1:) y.)  AM y.|
| | |:                   |
| | |(+/x.*y.) % +/x.    |
+-+-+--------------------+
```
where **$** is the dyadic verb *shape* which gives an array of items of its right argument shaped according to its left argument.

Conjunctions, adverbs and nouns may be defined similarly by using **2 : 0**, **1 : 0** and **0 : 0**, respectively. Iverson (1994) defines the adverb **table** defined as
```
+-+-+---------------+
|1|:|[ by ] over x./|
+-+-+---------------+
```
which gives a bordered table for an arbitrary dyadic verb, and
```
(i. 5) + table i. 6
```
is equivalent to the expression
```
p by q over p+/q
```
given earlier for the first **5** rows and **6** columns of an addition table.

Summary statistics

Some of the statistics discussed in the previous sections may be combined in a monadic verb **summary**, given below with an example of its use, which will calculate and display the statistics with suitable labels for a given argument. We note the use of the dyadic verb *laminate* **,:** for joining arrays of different shapes, and the verb *format* **":** whose left argument specifies the width and number of decimal places displayed in the right argument and which gives a literal result.
```
+-+-+----------------------------------------+
|3|:|r=.  'Sample size      ',5.0 ": #y.      |
| | |z=. five y.                             |
| | |r=. r,: 'Minimum           ', 8.3 ": {. z|
| | |r=. r, 'Maximum           ',8.3": {: z  |
| | |r=. r, 'Arithmetic mean   ',8.3": am y. |
| | |r=. r, 'Variance          ',8.3": var y.|
| | |r=. r, 'First quartile    ',8.3": 1{ z  |
| | |r=. r, 'Median            ',8.3":2{ z   |
| | |r=. r, 'Third quartile    ',8.3":3{ z   |
| | |r=. r, 'Standard deviation', 8.3": sd y.|
| | |r=. r, 'Geometric mean    ',8.3": gm y. |
| | |r=. r, 'Harmonic mean     ',8.3": hm y. |
+-+-+----------------------------------------+
```

```
   b
5.2 8.6 3.4 8.1 9.2 5.3 4.1 8.2 9.9 4.8

   summary b
Sample size          10
Minimum               3.400
Maximum               9.900
Arithmetic mean       6.680
Variance              5.531
First quartile        4.800
Median                6.700
Third quartile        8.600
Standard deviation    2.352
Geometric mean        6.283
Harmonic mean         5.887
```

Verb rank

The definition of verbs in the *Dictionary* gives the ranks of their monadic and dyadic forms. For example, the ranks associated with **-**, whose monadic form is negate and dyadic form is minus, are given as **_ 0 0** where the first item in the triple is the monadic rank and the remaining two are the left and right dyadic ranks. The monadic rank, represented by **_**, is said to be *unbounded* since negate applies to the entire argument; the dyadic ranks of **0** indicate that the dyadic verb is applied to the 0-cells of its arguments. The rank of any verb, primitive or defined, may be determined by the adverb **jrank** defined as

```
+-+-+-------+
|1|:|x. b. 0|
+-+-+-------+ ,
```

where **b.** is the adverb *basic characteristics*, and, for example, **- jrank** is **_ 0 0**.

As another example, the verb **,** with a monadic form ravel and a dyadic form append has ranks **_ _ _**. If **r=. i. 2 3 2** which was introduced in the discussion of noun ranks, then **,r** is the list

```
0 1 2 3 4 5 6 7 8 9 10 11
```

since ravel applies to the entire array. However, **,"2** has rank **2 2 2** so that **,"2 r** is

```
0 1 2 3  4  5
6 7 8 9 10 11
```

since the verb now applies to the 2-cells of **r**. For the dyadic form consider the verbs **,"1 0** which allows appending a list to the right of the 1-cells of a table and **,"0 1** which allows appending the list to the left of a table. Thus if **t=. i. 2 3** and **x=. 6 7 8** and **y=. 6 7**, then the expression

```
(x,t);(t,"1 0 y);(t,x);y,"0 1 t
```

has the value

```
+-----+-------+-----+-------+
|6 7 8|0 1 2 6|0 1 2|6 0 1 2|
|0 1 2|3 4 5 7|3 4 5|7 3 4 5|
|3 4 5|       |6 7 8|       |
+-----+-------+-----+-------+  .
```

As a final example the monadic verb *matrix divide* **%.** gives the inverse of its non-singular square arument. However, since it has a rank of 2, it is applied to all 2-cells of the argument, so that, for example, **%. t**, where **t** is of rank **3 4 5 5**, say, gives an array of the same rank as **t** whose 2-cells are the inverses of the corresponding 2-cells of **t**.

Linear regression

To introduce a simple regression example we shall use the data

```
y1=. 5.27 5.68 6.25 7.21 8.02
                    8.71 8.42
```

and

```
x1=. 12 18 24 30 36 42 48
```

on crop yield and the amount of water applied which were introduced in a previous section. Then

```
X=. 1,"0 x1
```

is a two-column table with **1**s in the first column and the values of **x1** in the second column. The regression coefficients are given by

```
b=. y1 %. X
```

which has the value **3.99 0.103** rounded to three figures. Thus the least-squares estimate of yield $\hat{y}$ for amount of water x is $\hat{y} = 3.99 + 0.103x$. The estimated values of the yield are given by **X +/ . * b**, where **.** is the conjunction *dot product* which gives the matrix or inner product, and are, rounded to two decimal places, equal to

```
5.23 5.85 6.46 7.08 7.70 8.31
                          8.93 .
```

(Since the matrix product often occurs in calculations, it is convenient to represent it by a verb such as

```
mp=. +/ . * .
```

Then the estimated values are given by the expression **X mp b**.)

The verb **sr** given at the end of the section finds the regression coefficients and some of the statistics required to test for the significance of the linear regression. We note the use of the monadic verb *shape of* **$** which gives the shape of its argument, the indirect use of assignment to define both the intercept **b0** and the slope **b1** in the same statement, and the verb *is (global)* **=:** which allows the estimated values of the dependent variable to be calculated as the global variable **yest**.

```
+-+-+------------------------------+
|3|:|:                             |
| | |('b0';'b1')=. b=. y.%.X=.1,"0 x.|
| | |yest=: b0+b1*x.               |
| | |sst=. +/*:y.-am y.            |
| | |sse=. +/*:y.- X +/ . * b      |
| | |mse=. sse%<:<:$y.             |
| | |seb=. %:mse%+/*:x.-am x.      |
| | |rsq=. 1-sse%sst               |
| | |fmt=. 10.5&":                 |
| | |r=. 'Slope       ',fmt b1     |
| | |r=. r,: ' S.E.       ',fmt seb|
| | |r=. r,'Intercept   ',fmt b0   |
| | |r=. r,'S.E. of est.',fmt %:mse|
| | |r=. r,'Corr. sq.   ',fmt rsq  |
+-+-+------------------------------+
```

The expression **x1 sr y1** gives the following results:

```
Slope          0.10286
 S.E.          0.01104
Intercept      3.99429
S.E. of est.   0.35036
Corr. sq.      0.94558
```

Since the estimated values of the dependent variable are given by the global variable **yest**, we may now define the three-column array

```
RegEx=. x1,.y1,.yest
```

which has the value

```
12 5.27 5.22857
18 5.68 5.84571
24 6.25 6.46286
30 7.21    7.08
36 8.02 7.69714
42 8.71 8.31429
48 8.42 8.93143
```

and which gives the observed values of the independent and dependent variables in the first two columns and the estimated values of the dependent variable in the third column.

Appendix 2 gives the explicit verb **reg** for multiple regression calculations and some examples of its use.

Base and antibase

The two verbs *base* **#.** and *antibase* **#:** are useful for converting from one number representation to another. For example, **#: 13** is **1 1 0 1**, the binary representation of the decimal number **13**, and, conversely, **#. 1 1 0 1** is **13**. The dyadic forms allow conversion between arbitrary bases, and, for example, **8 8 8 #: 123** is **1 7 3** which is the octal representation of the decimal number **123**, and **8 #. 1 7 3** is **123**. The number base may be mixed as in **24 60 60 #: 19510** which is **5 25 10**, the number of hours, minutes and seconds in **19510** seconds.

A verb of considerable usefulness is

```
tt=. #: @ i. @ (2&^)
```

which gives a truth table, i.e., all possible arguments for a logical function of an arbitrary number of variables. An ordered truth table with the rows sorted according to the number of **1**s is given by

```
ott=. (/: +/"1) @ tt .
```

For example, **(tt ; ott) 3** is equal to

```
+-----+-----+
|0 0 0|0 0 0|
|0 0 1|0 0 1|
|0 1 0|0 1 0|
|0 1 1|1 0 0|
|1 0 0|0 1 1|
|1 0 1|1 0 1|
|1 1 0|1 1 0|
|1 1 1|1 1 1|
+-----+-----+ .
```

A partial truth table, given by

```
ptt=. (+/"1 @ tt@] e. [) # tt@]
```

gives only those rows of the truth table whose number of **1**s is given by the left argument, and **2 ptt 3** is

```
0 1 1
1 0 1
1 1 0 .
```

All indices of an array of specified shape are given by the verb

```
indices=. <"1 @ (, #: i.)
```

and, for example, **indices 2 4** is

```
+---+---+---+---+
|0 0|0 1|0 2|0 3|
+---+---+---+---+
|1 0|1 1|1 2|1 3|
+---+---+---+---+  .
```

Covariance and correlation II

In previous sections we developed verbs for calculating the variance of a list of observations and the covariance and correlation coefficient of two lists of paired observations. In this section we shall consider the calculation of these statistics for an arbitrary number of sets of observations. The covariances will be given in a square variance-covariance table where the items on the main diagonal are variances and the other items are covariances. The correlations are given similarly in a correlation table. The illustrative data we shall use have been taken from Searle (1966) and represent six observations on each of three variables and will be given

as the three-item list **d**:

```
+----------+----------+----------------+
|1 4 2 2 1 3|0 6 4 3 1 5|10 17 13 14 12 15|
+----------+----------+----------------+
```

The variance-covariance table may be calculated by first finding all pairs of observations corresponding to any given list of observations by using the verb **pairs** defined as

```
pairs=. (>@indices @
              (2: $ $@])) {"1 1 ] ,
```

where, for example, **pairs d** is an array of shape **3 3 2**. Given this array the desired table may be found by the verb

```
covtable=. (({.@]) cov ({:@]))"2
             @ (>@pairs)  .
```

The correlation table is given similarly by the verb

```
cortable=. (({.@]) cor ({:@]))"2
             @ (>@pairs)  .
```

Using the sample data we have **10.5": covtable d** equal to

```
1.36667   2.56667   2.70000
2.56667   5.36667   5.30000
2.70000   5.30000   5.90000
```

and **10.5": cortable d** equal to

```
1.00000   0.94773   0.95084
0.94773   1.00000   0.94188
0.95084   0.94188   1.00000  .
```

The last two verbs may be defined more simply as

```
covtable=. cov ctable
```

and

```
cortable=. cor ctable
```

where **ctable** is the adverb

```
+-+-+--------------------------------+
|1|:|(({.@]) x. ({:@]))"2 @ (>@pairs)|
+-+-+--------------------------------+
```

Permutations and combinations

The six permutations of the integers **0**, **1** and **2**,

```
+-----+-----+-----+-----+-----+-----+
|0 1 2|0 2 1|1 0 2|1 2 0|2 0 1|2 1 0|
+-----+-----+-----+-----+-----+-----+ ,
```

may be represented by the indices, or atomic representations, **0**, **1**, **2**, ..., **7**. The monadic verb *atomic index* **A.** gives the atomic representations corresponding to the permutations, and, for example, **A. 0 2 1** is **1** and **A. 2 0 1** is **4**. The dyadic form *atomic permute* gives the permutation of the right argument whose atomic representation is the left argument, and **1 A. 0 1 2** is **0 2 1** and **4 A. 0 1 2** is **2 0 1**. The expression **perm n**, where

```
perm=. i.@! A. i. ,
```

gives a table of all **!n** permutations of the non-negative integers **i. n**, while **PERM x**, where

```
PERM=. perm @ # { ] ,
```

gives all permutations of the list **x**. For example,

```
(perm 3);PERM 'cat'
```

gives the table

```
+-----+---+
|0 1 2|cat|
|0 2 1|cta|
|1 0 2|act|
|1 2 0|atc|
|2 0 1|tca|
|2 1 0|tac|
+-----+---+ .
```

The expression **<"1 perm 3** gives the six-item list of permutations at the beginning of this section.

Although combinations may be generated from tables of permutations, we choose to obtain them by means of the partial truth table verb **ptt** introduced in a previous section. Thus the expression **m comb n**, where

```
comb=. |.@ptt # i.@]
```

and **|.** is the monadic verb *reverse*, gives a table of all combinations of the integers **i. n** taken **m** at a time, and

```
(0&comb;1&comb;2&comb;3&comb) 3
```

is equal to

```
++-+---+-----+
||0|0 1|0 1 2|
||1|0 2|     |
||2|1 2|     |
++-+---+-----+ .
```

Combinations from an arbitrary list are given by

```
COMB=. ([ comb #@]) { ] ,
```

and, for example, **2 COMB 'cat'** is

```
ca
ct
at .
```

Ranks

In some non-parametric tests the original observations are replaced by their ranks. For example, for the four-item list **w** equal to **2.3 5 3.5 6** the ranks are **1**, **3**, **2** and **4** since the first observation is the smallest, the second is the third smallest, the third is the second smallest, and the last is the largest. These ranks are given by the verb

```
UNADJranks=. >: @ /:^:2
```

and **UNADJranks w** is **1 3 2 4**. The conjunction *power* **^:** is used to repeat the verb left argument a number of times specified by the right argument, and, for

example, **(*:^:2) 5** is **625**.

If there are tied observations, the ranks of equal observations are replaced by the average of their ranks neglecting ties. For example, if

w1=. 2.3 5 2.3 3.5 6 2.3 3.5 ,

then **UNADJranks w1** is equal to **1 6 2 4 7 3 5**, and the ranks of the first, third and sixth observations are replaced by the average of the ranks **1**, **2** and **3** or **2**, and similarly the ranks of the fourth and seventh observations are replaced by **4.5**. Thus the ranks of **w1** adjusted for ties are given by the list

2 6 2 4.5 7 2 4.5 .

The ranks of any list of observations, adjusted for ties, are found by first finding the sum of the ranks neglecting ties of each item in the nub, then dividing each sum by the number of times each item in the nub occurs in the list, and finally distributing these adjusted ranks over the list of ranks. The ranks of the items in the nub are given by

NUBranks=. (= mp UNADJranks) %
[: +/"1 = ,

where **mp** is the matrix product **+/ . *** introduced earlier, and **NUBranks w1** is **2 6 4.5 7**. The ranks adjusted for ties are given by

ranks=. [: , |:@= # NUBranks ,

where **|:** is the monadic verb *transpose*, and **ranks w1** is **2 6 2 4.5 7 2 4.5**.

Often, for example with class marks, ranks are given in the inverse order to that given above so that the largest item in a list has the smallest rank of **1**. These inverse ranks are given by the verb

invranks=. [: ranks - ,

and, for example, **invranks w** is **4 2 3 1**.

Rank tests

In this section we shall discuss very briefly a few statistical tests based upon the ranks of observations rather than on the observations themselves. The first is Spearman's rank correlation coefficient which is the usual correlation coefficient computed from the ranks. Rather than give the customary expression involving the ranks, we shall give the verb

rcor=. ranks@[cor ranks@]

which calculates the rank correlation coefficient from the original observations. If we use the two lists **French** and **German** with the values

83 27 42 51 53 44 47 55 61 33

and

74 22 49 54 48 47 55 61 59 29

which give the marks of ten students in these two subjects and which have been taken from Sprent (1981), we find that

French rcor German

has the value **0.879**, rounded to three decimal places, while the correlation coefficient found from the original marks by

French cor German

is equal to **0.925**.

The second non-parametric test involving ranks we shall consider is the Wilcoxon two-sample test for determining any difference between two sets of observations. For sample data we shall use the data

ht=. 7.9 6.5 17.8 4.2 13.2

and

hc=. 5.6 0.4 6.7 1.2 2.9

taken from Hodges and Lehman (1964) representing the number of years ten patients survived a heart attack, only the first five of whom had received a specific treatment. We wish to determine if the treatment significantly prolonged the lives of the patients who received it.

The ranks of the combined observations are

r=. ranks ht,hc

which is equal to

8 6 10 4 9 5 1 7 2 3

where the first five ranks refer to treated patients and the last five to control patients. The sums of these two groups are **+/5{.r** or **37** and **+/_5{.r** or **18**, respectively. Now we ask if the treatment is of no effect, what is the probability that the sum of the five ranks for the control patients would be less than or equal to the observed value of **18**? There are **5!10** or **252** combinations of the ranks **1**, **2**, ..., **10** taken **5** at a time and they are given by the expression

t=. >: 5 comb 10

which is a table with **252** rows and **5** columns. The row sums are **s=. +/"1 t** and the number of these sums less than or equal to **18** is **+/s <: 18** which is equal to **7**. (This value may be calculated in a single expression as

+/ 18 >: +/"1 >: 5 comb 10 .)

Thus the probability that the sum of the control ranks is less than or equal to **18** is **7%252** or **0.028** and we may conclude that the treatment was effective in prolonging life.

The above calculations could be performed simply by a specially defined verb, but we shall merely give the verb

Wsum=. (#@[<: i.@#@([,])) +//.
ranks @ ([,])

for the sums of the ranks, where

```
ht Wsum hc
```

is the two-item list **37 18**.

The third and last non-parametric test we shall consider is the Wilcoxon test for paired comparisons. Again we shall use sample data from Hodges and Lehman (1964), this time representing the degree of burning when each of seven subjects uses an old lotion and receives burns measured on some scale of

```
s1=. 42 51 31 61 44 55 48
```

and also a new lotion where the burn measurements are

```
s2=. 38 53 36 52 33 49 36 .
```

To estimate the efficacy of the new lotion we shall first find the differences of the pairs of measurements or **s1 - s2** which is

```
4 _2 _5 9 11 6 12 ,
```

and then the ranks of the absolute value of these differences or

```
ranks |s1 - s2  ,
```

where **|** is the monadic verb *magnitude,* which is

```
2 1 3 5 6 4 7 .
```

These last two lists may be compared more conveniently in the table

```
4 _2 _5 9 11 6 12
2  1  3 5  6 4  7
```

given by the expression

```
(s1 - s2) ,: ranks |s1 - s2 .
```

The sum of the ranks corresponding to negative differences, which indicate that there is more burning with the new lotion, is **4** so we should find the probability that of the **2^7** or **128** ways of assigning signs to the ranks the sums associated with negative ranks would be less than or equal to this number. If we use an argument similar to that used for the Wilcoxon two-sample test, we see that the number of ways is

```
+/ 4 >: +/"1 (tt 7) # >: i. 7
```

which is **7**. Thus the probability is **7%128** or **0.0546875** which casts some doubt at the 5% level on the hypothesis that the new lotion is an improvement.

As with the last test we will not define a verb to carry out all of these calculations, but give only the verb

```
Wpsum=. (* +//. ranks@|) @
                        sort @ ([-])
```

for finding the sums associated with the negative and positive ranks, where

```
s1 Wpsum s2
```

is the two-item list **4 24**. The verb **sort** is used so that the sum associated with the negative ranks is given first.

Runs

In this section we shall require the dyadic verb *infix* which we shall introduce together with the monadic verb *prefix*, both of which are represented by ****. The expression **u\y** applies the verb **u** to each of the partial lists derived from the noun **y**. For example, the expression **<\i. 4** shows the partial lists

```
+-+---+-----+-------+
|0|0 1|0 1 2|0 1 2 3|
+-+---+-----+-------+
```

and **><+/\i. 4** is the list **0 1 3 6** of their sums. The expression **a u\y** gives the result of applying **u** to each infix of length **a** of **y**, the infixes being non-overlapping if **a** is negative. Thus **2<\i. 4** is

```
+---+---+---+
|0 1|1 2|2 3|
+---+---+---+ ,
```

and **_2<\ i. 4** is

```
+---+---+
|0 1|2 3|
+---+---+ .
```

In the discussion of the median we introduced the list **u**,

```
22 14 32 30 19 16 28 21 25 31 ,
```

whose median was found to be **23.5**. If we are interested in determining if the sampled population, whatever it may be, changed with time, we could examine how the sequence of observations fluctuated about the median. Thus if we were to calculate

```
s=. u > median u
```

which is equal to

```
0 0 1 1 0 0 1 0 1 1 ,
```

then the **0**s would represent items of **u** less than or equal to the median and the **1**s values greater than the median. If the number of consecutive sequences or runs of **0**s and **1**s lay between reasonable limits, which may be calculated from probabilistic considerations and which have been tabulated, then we might conclude that the observations represented a random sample from the population. In this section we shall be concerned only with the calculation of the number of runs and not with any tests for randomness based on their number.

There are six runs of **0**s and **1**s in the sequence in the last paragraph, viz., **0 0**, **1 1**, **0 0**, **1**, **0** and **1 1**. Now the beginning of a run, other than the first, is indicated by a change from **0** to **1** or from **1** to **0**. Therefore, the number of these is given by the number of unequal consecutive overlapping pairs. The overlapping pairs are given by **2 <\s** which is

```
+---+---+---+---+---+---+---+---+---+---+
|0 0|0 1|1 1|1 0|0 0|0 1|1 0|0 1|1 1|
+---+---+---+---+---+---+---+---+---+---+
```
,

the number of unequal pairs given by the **1**s in the list **2 ~:/\s** which is

```
0 1 0 1 0 1 1 1 0
```
,

and the number of unequal pairs by **+/ 2 ~:/\s** which is **5**. Therefore, a verb for the number of runs is given by

```
runs=. [: >: [: +/  2: (~:/\) ]
```

where the verb **2:** is one of the the nineteen *constant* verbs **_9:**, ..., **0:**, ..., **9:** which always return their indicated value regardless of the argument, and

```
runs u > median u
```

is **6**, a number within limits for the randomness of the list **u** of observations. We might note that the number of runs in a ten-item list of **0**s and **1**s can vary from a minimum of one in a list of all **0**s or all **1**s to a maximum of five in a list where the **0**s and **1**s alternate.

The verb **runs** may be used to find the number of runs in a list of characters. For example,

```
runs 'HHHTHTTHHHHTH'
```

is equal to **7** and could represent the number of runs of heads and tails in a coin-tossing experiment. The number of different characters need not be limited to two, and, for example,

```
runs 'aabbbbacacccbddbbbb'
```

is equal to **9**.

Explicit definition II

The sequential execution of instructions in an explicit definition may be changed by *control words* such as **if.**, **do.**, **else.**, **end.** and **while.** whose function is similar to such words in conventional programming languages. Their use may be illustrated by the verb **PrimeTest** which determines if its integer argument is prime:

```
+-+-+----------------------+
|3|:|d=. 2                 |
| | |while. d < y.         |
| | |   do. if. (0: = d&|) y.|
| | |       do. r=. 0      |
| | |           return.    |
| | |       else. d=. d+1+d>2|
| | |   end.               |
| | |end.                  |
| | |r=. 1                 |
+-+-+----------------------+
```

The integer **n** greater than **1** is divided successively by the integers **2**, **3**, **5**, ... up to the largest odd integer less than **n**. If an even divisor of **n** is found, then **n** is not prime; if no even divisor is found, then **n** is prime. The following expressions give some examples of its use:

```
   PrimeTest"0 (N=. 2+i. 15)
1 1 0 1 0 1 0 0 0 1 0 1 0 0 0
   (PrimeTest"0 # ]) N
2 3 5 7 11 13
```

Analysis of variance

The main computational problem in the analysis of variance is the partition of the total variation of a dependent variable as measured by the sum of squares of deviations from the arithmetic mean into a number of orthogonal components for each of several main effects and their interactions. If the data are considered to be arranged in a rectangular array with an axis for each factor and one for replications, then a major aspect of the partitioning process is the calculation of some, or possibly all, of the marginal totals of various subarrays. For example, for a one-factor design the data may be arranged in a table, and the four marginal totals consist of the row and column totals, the grand total, and the items of the array. In general, for an *n*-dimensional rectangular array there are 2^n marginal totals. For each set of marginal totals a weighted sum of squares is found by squaring each item, summing and dividing by the number of items occurring in each total. From these weighted sums of squares the required analysis-of-variance table may be found relatively simply. In this section we shall develop verbs for finding all weighted sums of squares in a rectangular array of arbitrary rank.

We shall need the dyadic verb *transpose* **|:** which interchanges the axes of the array right argument as specified by the left argument, and the monadic verb *not* **-.** which gives the complement of its boolean argument so that, for example, **-. 0 1** is **1 0**. (The dyadic form of **-.**, which will be used in Appendix 4, is *less* and gives all those items in the left argument which are not in the right argument. For example,

```
(i. 6) -. 4 3
```

is **0 1 2 5**.)

The calculation of any set of marginal totals in an arbitrary array may be accomplished by first transposing the array so that the axes to be summed over occur first and then summing over these axes. We shall specify the axes by a boolean list of length equal to the rank of the array with **0**s indicating the axes being summed over. For example, consider summing over the first, third and fourth axes of an array **a**, where **$a** is **2 3 4 5**, say, so that the boolean list specifying the summation is **0 1 0 0**. Since **/:0 1 0 0** is equal to

0 2 3 1, the transposition may be accomplished by the expression **0 2 3 1|:a** which gives an array of shape **2 4 5 3**. Application of the verb **+/** three times, the number of **0**s in the boolean list specifying the marginal totals, to this array will give the desired marginal totals. These calculations may be accomplished simply by the explicit verb **T** defined as

```
+-+-+--------------------+
|3|:|:                   |
| | |+/^:(+/-.x.)(/:x.)|:y.|
+-+-+--------------------+
```

where the right argument gives the array and the left argument specifies the marginal totals required. Then the above example is given by **0 1 0 0 T a**. The weighted sum of squares is then given by

```
S=. (+/@((*:@,)@T)) %
          (*/@(-.@[ # $@]))
```

where the arguments are the same as for **T**.

As an example of the use of the above verbs consider the three-by-four table **x** given by

```
4 7 5 6
9 4 3 8
2 5 7 3 .
```

Then **0 0 T x** is the sum **63** of the items of **x**, **0 1 T x** is the column sums **15 16 15 17**, **1 0 T x** is the row sums **22 24 17**, and **1 1 T x** is the array **x**. The corresponding values of the weighted sums of squares given by **S** are **330.75**, **331.667**, **337.25** and **383**, respectively.

Appendix 3 gives the explicit verb **AOV** for analysis-of-variance calculations for factorial experiments with an arbitrary number of factors. The defined verbs which are used in **AOV** are given in the script file in Appendix 5. We note that the complete analysis-of-variance table with the degrees of freedom, sum of squares and mean squares is always given as a three-column numeric table in the global variable **AOVtable**.

Simulation I

Simulation may be accomplished by the monadic verb *roll* **?** which gives sampling with replacement, and **? y** gives a uniform random selection from the population **i. y**. As an example, if **t=. ? 6 6 6 6 6** or **t=. ? 5$6**, then **t** could have the value **3 0 5 3 2**, and **>: t**, which would then be equal to **4 1 6 4 3**, could represent the results of rolling one die **5** times.

The dice-rolling simulation could be accomplished simply by the verb

```
die=. >: @ ? @ $&6
```

and, for example, **die 10** could have the value **4 6 3 5 1 3 6 5 4 2**. The verb

```
dice=. >: @ ? @ (6&($~)@,)
```

could be used to simulate the rolling of an arbitrary number of dice an arbitrary number of times, and, for example, **2 dice 10**, which could have the value

```
1 3 3 1 5 5 4 1 6 5
1 4 1 4 5 6 4 2 5 4 ,
```

would represent the rolling of **2** dice **10** times. The verb

```
TwoDice=. 2&dice
```

could be used to simulate the rolling of two dice, and **TwoDice 10** would be equivalent to **2 dice 10**.

If we are interested in rolling two dice and summing the numbers which occur on each roll we could define the verb

```
SumTwoDice=. +/ @ TwoDice ,
```

and **SumTwoDice 10** might have the value

```
5 7 11 7 6 6 11 5 11 9  .
```

A frequency distribution of such sums may be found very simply by using the range frequency verb **rfr** Since the expression **2+i. 11** gives the eleven-item list

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

of all possible sums occurring when two dice are rolled, the expression

```
(2+i. 11) rfr SumTwoDice 100
```

would represent the frequency distribution of sums occurring when two dice are rolled **100** times and could have the value

```
3 4 9 14 15 12 9 18 10 2 4 .
```

The verb

```
SumTwoDiceFr=. (2&+@i.11)&rfr @
                    SumTwoDice
```

could be used to accomplish this simulation more simply.

If we wished to simulate the rolling of two dice and testing to see if the two faces on each roll were equal, we could define the verb

```
EqualTwoDice=. =/ @ TwoDice ,
```

and, for example, **EqualTwoDice 10** could have the value

```
0 0 1 0 0 0 0 0 1 0
```

which would indicate that the faces were equal on the third and ninth rolls. The expression

```
+/EqualTwoDice 500
```

would give the number of times in **500** rolls that the two faces were equal. Five evaluations of this expression gave the values **84**, **72**, **91**, **85** and **80** which may be compared with the expected number **500%6** or approximately **83** of equal faces in **500** rolls.

The use of one of the verbs given in this section for simulating and tabulating an increasingly large number of rolls, say several thousand, will eventually cause a limit error and the termination of the computation. This situation may be avoided simply by using the rank conjunction to perform independent simulations of smaller numbers of rolls and then combining the results. For example, to tabulate the sums occurring when two dice are rolled **2500** times, the expression

```
SumTwoDiceFr"0 (1000 1000 500)
```

would give a three-row table, with the first two rows giving the results of two simulations of **1000** rolls each and the third row giving the result of a simulation of **500** rolls. Therefore, the expression

```
+/SumTwoDiceFr"0 (1000 1000 500)
```

would pool the results of the three separate simulations to give the desired results. One evaluation of this last expression resulted in the frequencies

```
82 160 188 274 349 418 351 253
                    214 156 55 .
```

The result of simulating the rolling of dice may also be represented as a boxed array by the verb

```
NewDice=.(-@[)<\>:@?@(6&($~)@*)
```

, and, for example, the expression **2 NewDice 10** could give the result

```
+---+---+---+---+---+---+---+---+---+---+
|1 5|3 4|2 1|5 5|6 3|4 5|1 1|4 5|1 3|1 3|
+---+---+---+---+---+---+---+---+---+---+
```

The verb

```
SumTwoNewDice=.+/"1 @>(2&NewDice)
```

would give the sums occurring when two dice are rolled an arbitrary number of times, and

```
SumTwoNewDice 10
```

might be equal to

```
4 7 6 9 5 8 4 9 8 2 .
```

The serial test for random digits requires counting the frequency of occurrence of each of the 100 possible pairs of successive digits in a sequence of random digits partitioned into non-overlapping pairs. These counts may be conveniently arranged in a ten-by-ten table where the item in row *i* and column *j* gives the frequency that digit *i* is followed by digit *j*, where *i*,*j* = 0, 1, 2, ..., 9. A sample value of the expression

```
((i.10);i.10) RFR _2<\ ?1000$10
```

which gives such a table for one thousand random digits is

```
 5 6  9 4 2 4 2  5  8 6
 4 4  4 5 7 9 8  5  6 4
 6 2  8 3 4 2 3  8  5 5
 2 7  3 3 6 6 5  2  3 5
 3 4  3 1 9 5 4  7  6 7
 8 2 10 3 3 4 7  9  3 5
 7 2  4 5 4 6 4  7  9 3
 4 4  1 4 1 4 4  2 11 6
 3 4  1 8 0 4 7 10  5 4
10 8 12 9 3 5 2  4  6 5 .
```

Finally let us consider very briefly simulating the tossing of coins. The expression **?2** may be considered to represent the tossing of an unbiased coin with a result of **1** representing a head and **0** a tail. Then the verb

```
Coins=. [: +/ [: ? ] $ 2:
```

may be used to simulate the number of heads obtained when a coin is tossed an arbitrary number of times, and the expression **Coins n**, where **n** is a non-negative integer, may have any value between **0** and **n**, inclusive. The verb

```
CoinTest=. +/\@(Coins"0 @ $) %
      ] * (>: @ i. @ [)
```

may be used to examine the ratio of the cumulative number of heads to the total number of tosses as the number of tosses increases. For example, the expression

```
5 CoinTest 100
```

gives a five-item list whose items give this ratio for **100**, **200**, ..., **500** tosses; one sample value, rounded to three places of decimals, is

```
0.490 0.505 0.530 0.543 0.534 .
```

Central limit theorem

A very important and remarkable theorem in probability theory is the central limit theorem which states that under very general conditions the distribution of the standardized sum of a number of independent random variables approaches asymptotically that of a standard normal distribution with zero mean and unit variance. In this section we shall develop a dice-rolling simulation to illustrate this theorem.

It may be shown that the sum *T* occurring when *n* dice are rolled has mean $\mu = 7n/2$ and standard deviation $\sigma = \sqrt{35n/12}$. Thus according to the central limit theorem the distribution of the standardized sum $(T - \mu)/\sigma$ approaches that of the standard normal distribution as *n* increases. In the following discussion of the verbs to perform the simulation we shall consider throwing three dice five times although, of course, a reasonable simulation will involve a variation in the number of dice which will be rolled a much larger number of times.

The verb **NewDice** given at the end of the

previous section will be used to simulate the rolling of the dice, and, for example, **3 NewDice 5** could have the value

```
+-----+-----+-----+-----+-----+
|1 3 3|6 1 3|5 2 6|6 6 5|3 5 2|
+-----+-----+-----+-----+-----+
```

Then the verb

```
SumDice=. +/"1 @ > @ NewDice
```

will give the sums occurring when the dice are rolled, and **3 SumDice 5** could be equal to

```
12 16 7 8 10 .
```

These sums may be converted to standard units by the verb

```
SU=. (] - ExpSum@[) % SDSum@[ ,
```

where

```
ExpSum=. 3.5&*
```

and

```
SDSum=. %:@(2.91667&*)
```

give the mean and standard deviation, respectively, for the sum of the dice. If we use the sums given above for rolling **3** dice, we have that

```
3 SU 12 16 7 8 10
```

is equal to

```
0.507092 1.85934 _1.18322
                 _0.845154 _0.169031
```

since **ExpSum 3** is equal to **10.5** and **SDSum 3** is equal to **2.95804**. The verb

```
ISU=. -: @ <. @ +: @ (0.25&+@])
```

may be used to classify these continuous values into discrete intervals centred at the origin and with a width equal to one-half a standard deviation. The continuous values given above are converted to **0.5 2 _1 _1 0**, where, for example, the second item **2** indicates that the deviate **1.85934** is in the interval **1.75** to **2.25** whose centre is **2**.

A barchart for an arbitrary list of interval centres is given by

```
CLTchart=. rng&(;"0) @
                bars@(rng&rfr))
```

where

```
rng=. (-@|.,0:,])@-:@>:@i. 6
```

gives the constant range **_3**, **_2.5**, ..., **0**, ...,**3** of interval centres.

All of the above calculations to simulate the central limit theorem are given by the single verb

```
CLTdemo=. CLTchart @ ISU @ ([ SU
                SumDice)
```

and, for example, **3 CLTdemo 100** will display the results corresponding to rolling **3** dice **100** times. The examples on the next page give the results of rolling various numbers of dice **100** times.

Simulation II

Random numbers uniformly distributed between **0** and **1** may be generated by the verb

```
urand=. (? % ]) @ ($&1e9)  ,
```

and, for example, **urand 3** would be a three-item list which might have the value

```
0.1315378 0.7556053 0.4586501
```

and **urand 4 5** would be a four-by-five table of random numbers.

As a simple example of the use of uniformly distributed numbers consider the simulation of the repeated selection of random points within a unit square lying in the first quadrant and with the lower left vertex at the origin. Since the area of the quadrant of a unit circle centred at the origin inscribed in the square is $\pi/4$, the proportion of points lying within the circle may be used to obtain an estimate of the value of π. We shall now give some verbs which will allow us to estimate π by simulating the selection of points in the unit square.

The coordinates of random points within the square are given by

```
coords=. urand@,&2 ,
```

and **coords 3** could have the value

```
0.9683066 0.3283392
0.3966709 0.8477068
0.4082389 0.2704737
```

representing the coordinates of three random points within the unit square. Now the row sums of the squares of the items in such a table would gives the squares of the distances of the points from the origin. With the above table this value would be the list

```
1.04542 0.8759546 0.239815
```

indicating that the first second and third points would lie within or on the circle. Thus the verb

```
incircle=. [: +/ 1: >: +/"1 @ *:
                    @ coords
```

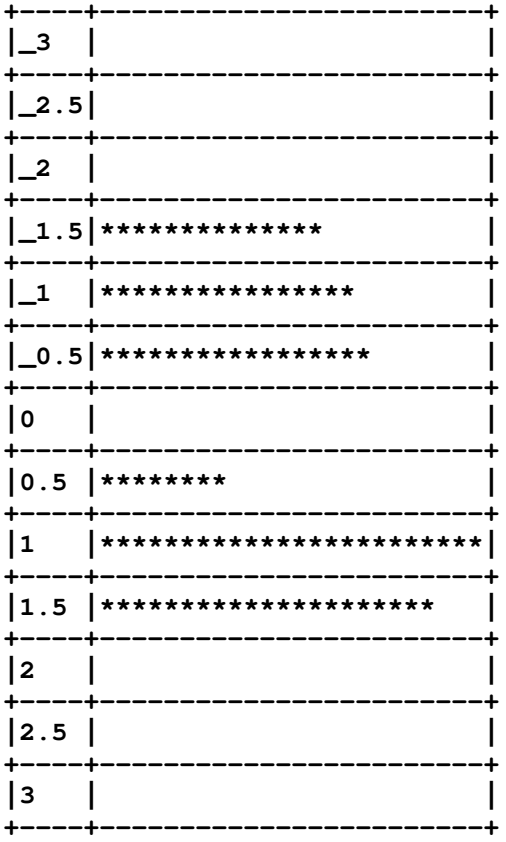
will give the number of points within or on the circle, and, for example, **incircle"0 (5$100)** could be equal to **78 77 74 86 75**. Finally the verb

```
PIest=. 4: * incircle % ]
```

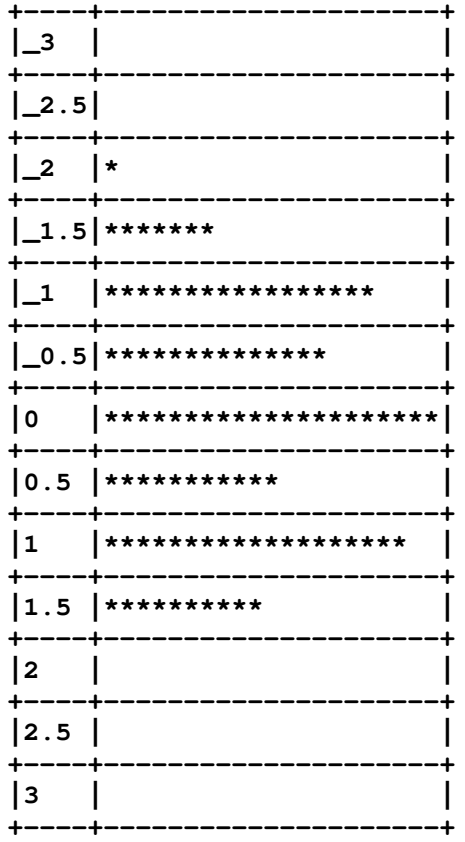
will give an estimate of the value of π for an arbitrary number of points. Five evaluations of **PIest 10000** gave **3.136**, **3.1588**, **3.1208**, **3.156** and **3.1452** for estmates of the value of π.

Pairs of standard normal random variables may be generated by the following algorithm:

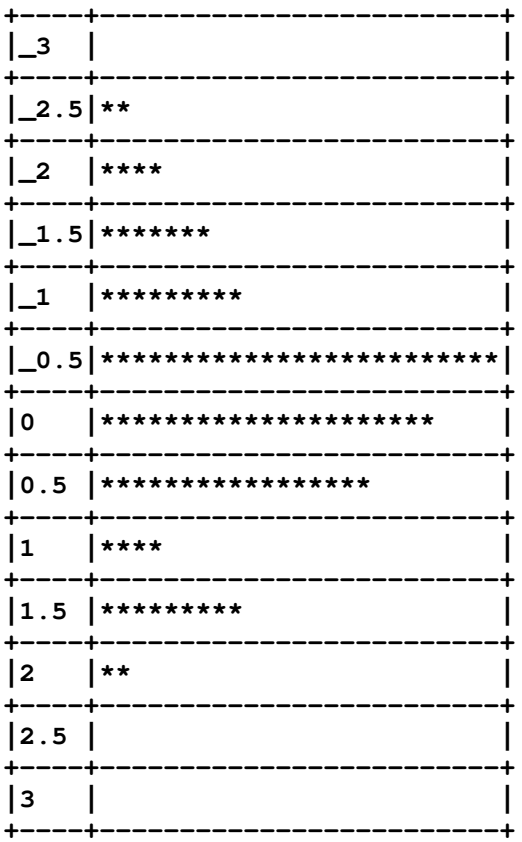
1 CLTdemo 100



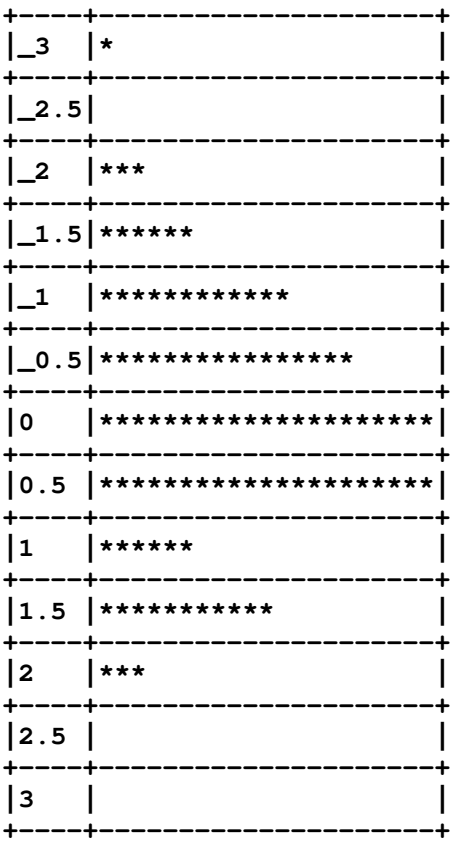
2 CLTdemo 100



5 CLTdemo 100



10 CLTdemo 100



1. Generate two independent uniform random variables U_1 and U_2.

2. Let $V_1 \leftarrow 2U_1 - 1$ and $V_2 \leftarrow 2U_2 - 1$.

3. Compute $S \leftarrow V_1^2 + V_2^2$.

4. If $S \geq 1$, return to Step 1; otherwise, go to Step 5.

5. Compute

$$X_1 \leftarrow V_1\sqrt{(-2\ln S)/S}$$

and

$$X_2 \leftarrow V_2\sqrt{(-2\ln S)/S}$$

which will be independently distributed standard normal variables.

This algorithm may be expressed in **J** by the verb **stnrand** defined as follows:

```
+-+-+--------------------------+
|3|:|r=. i. 0 2                |
| | |while. y. > {.$r          |
| | |   do. whilst. S >: 1     |
| | |      do. V=: <:+:urand 2 |
| | |          S=: +/ *: V     |
| | |   end.                   |
| | |r=. r, V * %: -+:(^.%])S  |
| | |end.                      |
+-+-+--------------------------+
```

We note the use of the control word **whilst.** which is similar to **while.** except the test is skipped the first time. The expression **stnrand 3** could give

```
   1.16495    1.16777
 0.6268391 _0.1679932
0.07508015    1.27979
```

representing three pairs of independently distributed standard normal variables.

The central limit theorem provides a simple means of using uniformly distributed random variables to generate random variables which are approximately normally distributed by noting that the sum of an arbitrary number of uniformly distributed random variables is approximately normally distributed with a given mean and standard deviation. The verb

```
stnrand1=.  4: %~  _6: + [: +/"1
                    urand@,&12
```

gives approximate standard normal random variables from the sum of **12** uniformly distributed random variables whose mean is **6** and standard deviation is **4**. The expression **stnrand1 5** could have the value

```
_0.1958305 0.01734841 0.3329121
_0.005577353 _0.1598029.
```

Finally random variables with the exponential distribution may be found by noting that if U is a uniformally distributed random variable, then $X = -\ln U$ has an exponential distribution with unit mean and μX has an exponential distribution with mean μ. The verb

```
exprand=. [ * -@^.@urand@]
```

gives exponentially distributed random numbers, and, for example, **1.5 exprand 3** could have the value

```
0.1351867 0.9530325 1.15037 .
```

Normally distributed random variables with arbitrary mean and standard deviation may be found very simply from standard normal random variables since if X has a standard normal distribution then $\mu + \sigma X$ has a normal distribution with mean μ and standard deviation σ.

De Méré's problems

In 1654 the French nobleman Chevalier de Méré consulted the philosopher Blaise Pascal about a problem with rolling dice that had been troubling him. He had observed that it was reasonable for a gambling establishment to bet even money that a player will roll at least one 6 in four rolls of a single die. Why, he asked, was it also not reasonable to bet that the player will roll at least one double 6 in twenty-four rolls of a pair of dice? This question troubled him because it contradicted an old gamblers' rule concerning the critical number of rolls, i.e., the number of rolls at which the odds changed from unfavourable to favourable. Applied to this problem the rule said that if four is the critical number for rolling one die, then six times four, or twenty-four, should be the critical number for two dice. The multiple of six arose because there are six times as many outcomes with two dice as there are with one die. In this section we shall have a brief look at this problem, the solution of which by Pascal is said to mark the beginnings of the theory of probability.

We shall need the verb *catalog* **{** which is a generalization of the Cartesian product which gives the array formed by selecting in all possible ways one item from each item of its argument with a shape determined by the shapes of the items in its argument, as illustrated by the following example:

```
   {1 2;3 4 5
+---+---+---+
|1 3|1 4|1 5|
+---+---+---+
|2 3|2 4|2 5|
+---+---+---+
```

If we represent the faces on a die by the list **f=. >: i. 6**, then a representation of the thirty-six

possible outcomes when a die is rolled twice is given by **{f;f**. We may note that at least one **6** occurs eleven times so that the probability of at least one **6** on two rolls is **11%36** or **0.30556**.

```
          {f;f=. >:i.6          +./~g=. 6=f
+---+---+---+---+---+---+     0 0 0 0 0 1
|1 1|1 2|1 3|1 4|1 5|1 6|     0 0 0 0 0 1
+---+---+---+---+---+---+     0 0 0 0 0 1
|2 1|2 2|2 3|2 4|2 5|2 6|     0 0 0 0 0 1
+---+---+---+---+---+---+     0 0 0 0 0 1
|3 1|3 2|3 3|3 4|3 5|3 6|     1 1 1 1 1 1
+---+---+---+---+---+---+
|4 1|4 2|4 3|4 4|4 5|4 6|
+---+---+---+---+---+---+
|5 1|5 2|5 3|5 4|5 5|5 6|
+---+---+---+---+---+---+
|6 1|6 2|6 3|6 4|6 5|6 6|
+---+---+---+---+---+---+
```

Since we are interested only in the occurrence of at least one **6**, a simpler representation of rolling one die twice is given by the table **+./~ g**, where **+.** is the verb *or* and **g=. 6=f**, where the **1**s show the outcomes which include at least one **6**. (This second representation may be obtained directly from the first by the expression **>(+./ @ (6&=) &. >){f;f** .) The number of these occurrences is given by the expression **+/,+./~g** which is equal to **11**. Since the shape of the table is **$+./~g** or **6 6**, the total number of outcomes when a die is rolled twice is ***/$+./~g** or **36**. An extension of this argument shows that a representation for rolling one die four times is given by **g+./g+./+./~g** which is a table with shape **6 6 6 6**. Therefore, the number of favourable outcomes is **+/,g+./g+./+./~g** or **671**, so the probability of at least one **6** on rolling a single die four times is **671%1296** or **0.517747**.

The probability of obtaining at least one success in an arbitrary number *n* of independent trials may be calculated more simply as $1 - (1 - p)^n$, where *p* is the probability of success in a single trial. The probability found in the last paragraph is given by substituting *1/6* for *p* and *4* for *n* in this expression. A verb for this calculation is given by

```
P=. -. @ [] ^~ -.@[] ,
```

where the left argument gives the probability of success in a single of trial and the right argument is the number of trials. A verb for the probability of success corresponding to rolling a single die once is

```
P1=. (%6)&P ,
```

and, for example,

```
P1 1 2 3 4
```

has the value

```
0.166667 0.305556 0.421296
                  0.517747 .
```

The probability of getting at least one double **6** on an arbitrary number of rolls of two dice may be found with the verb

```
P2=. (%36)&P ,
```

since the probability of a double **6** on a single roll is **%36**. Since

```
P2 24 25
```

is equal to

```
0.491404 0.505532 ,
```

we see that the critical number of rolls for obtaining a double **6** is closer to **25** than the value of **24** given by the old gamblers' rule.

Derivatives and integrals

The expression **u D. n**, where **D.** is the derivative adverb, gives the *n*th derivative of **u**. For example, if

```
f=. +:@*:"0 ,
```

then **f i. 4** is **0 2 8 18**, **f D. 1 i. 4** is **0 4 8 12**, and **f D. 2 i. 4** is **4 4 4 4.** Now define the scalar derivative adverb

```
D=. ("0)(D. 1)
```

so that **u D x** gives the first derivative of **u** evaluated at **x**, and, for example, **f D i. 4** is **0 4 8 12**.

Iverson (1993) defines the integral or anti-derivative adverb **I** such that the expression **u I x** gives the area under the graph of **u** from **0** to **x**. The sequence of verbs and one conjunction defining **I** are given in the script file in Appendix 5. For example, for the verb **f** given in the last paragraph **f I i. 4** is **0 0.666667 5.33333 18**.

As another example, if

```
snd=. (%:@o.@2:) (%~) ^@-@-:@*:
```

gives the standard normal probability density function, then the expression

```
(([],.snd),.snd D),. snd I) x
```

gives a table of arguments, ordinates, first derivatives and areas for the standard normal distribution for a suitable list argument **x**.

Probability distributions

We shall give verbs for the probability density and cumulative density functions for several of the more commonly used discrete and continuous probability density functions. With each distribution the parameters will be given as the left argument and the value or values of the independent variable as the right argument. The names of the density function and the cumulative density

function will be given in lower case and upper case characters, respectively, as, for example, **binomial** and **BINOMIAL** for the binomial distribution.

The adverb **cdf** given by

```
+-+-+-------------------------+
|1|:|+/ @ ([ x.f. (i. @ >: @ ]))|
+-+-+-------------------------+
```

will be used in the definition of discrete cumulative distributions and the integral adverb **I** introduced in the last section will be used for continuous cumulative distributions. Also the verb

```
B=. (G@[ * G@]) % G@+
```

for the beta function, where

```
G=. !@<:
```

gives the gamma function, will be used in the definition of the *t*- and *F*- distributions.

For each distribution the parameters are defined, the syntax of the verbs are given as well as some numerical examples of their use. The cumulative probabilities for the standard normal distribution and *t*-distribution are measured from the origin.

Binomial distribution
n: Number of trials
p: Probability of success in a single trial

```
n p binomial x
n p BINOMIAL x
   3 0.25 binomial 2
0.140625
   3 0.25 binomial i. 4
0.421875 0.421875 0.140625
      0.015625
   +/3 0.25 binomial i. 4
1
   3 0.25 BINOMIAL 2
0.984375
   3 0.25&BINOMIAL"0 i. 4
0.421875 0.84375 0.984375 1
```

Poisson distribution
m: Mean

```
m poisson x
m POISSON x
   0.3 poisson i. 5
0.740818 0.222245 0.0333368
      0.00333368 0.000250026
   0.3&POISSON"0 i. 5
0.740818 0.963064 0.996401    0.999734
0.999984
```

Geometric distribution
p: Probability of success in a single trial

```
p geometric x
p GEOMETRIC x
   0.8 geometric i. 6
0.8 0.16 0.032 0.0064 0.00128
      0.000256
   0.8 GEOMETRIC"0 i. 6
0.8 0.96 0.992 0.9984 0.99968
      0.999936
```

Standard normal distribution

```
stnormal x
STNORMAL x
   stnormal i. 4
0.398942 0.241971 0.053991
      0.00443185
   STNORMAL"0 i. 4
0 0.341345 0.47725 0.49865
```

Exponential distribution
r: Reciprocal of mean

```
r exponential x
r EXPONENTIAL x
   0.5 exponential i. 5
0.5 0.3032653 0.1839397 0.1115651
                    0.06766764
   1.5 EXPONENTIAL 0 2 4 6
0 0.9502129 0.9975213 0.9998766
```

t-distribution
n: Degrees of freedom

```
n TDISTN x
   5 TDISTN 2.015 2.571 3.365
0.449997 0.475013 0.490001
   15 TDISTN 2.602 2.947
0.48999 0.495003
```

Chi-square distribution
n: Degrees of freedom

```
n CHISQ x
   10 CHISQ 12.5 16 18.3
0.747015 0.900368 0.949891
```

F-distribution
m: Numerator degrees of freedom
n: Denominator degrees of freedom

```
m n FDISTN x
   5 20 FDISTN 2.16 2.71 3.29 4.1
0.900263 0.950012 0.975138
      0.990169
```

Binomial distribution

The probability of x successes in n independent binomial trials with probabilty p of success in a single

trial is equal to ${}^nC_x p^x(1-p)^{n-x}$ for $x = 0, 1, 2, ..., n$, where ${}^nC_x$ is the number of combinations of n things taken x at a time. The probability density function for the binomial distribution is

```
bd=. (x!n)*(p^x)*(-.@p=. {:@[) ^
                  (n=.{.@[) - x=. ]
```

where we note that the verb not **-.** with an argument which is a probability gives the complementary probability, and, for example, **-. 0.4** is **0.6**. We shall consider briefly a sequence of binomial trials where the probability of success in a single trial remains constant, as, for example, occurs in tossing a coin a number of times and counting the number of heads.

Define the verb

```
cp=. ,@  @ (] # < @ (-. , ])@ []
```

whose result is a Cartesian product of probabilities and complementary probabilities replicated a specified number of times. For example, **0.6 cp 2** is

```
+-------+-------+-------+-------+
|0.4 0.4|0.4 0.6|0.6 0.4|0.6 0.6|
+-------+-------+-------+-------+
```

As another example, **1 cp 2** is

```
+---+---+---+---+
|0 0|0 1|1 0|1 1|
+---+---+---+---+
```

which gives the arguments in a truth table for two Boolean variables.

The verb

```
succ=. +/"1 @ > @ (1&cp)
```

gives the number of successes in a specified number of binomial trials, and, for example, **succ 3** is

```
0 1 1 2 1 2 2 3 .
```

The corresponding probabilities are given by

```
prob=. */"1 @ > @ cp
```

and **0.6 prob 3** is

```
0.064 0.096 0.096 0.144 0.096
      0.144 0.144 0.216  .
```

The binomial coefficients are given by

```
bc=. #/.~ @ succ ,
```

and **bc 3** is **1 3 3 1**. The binomial probabilities are given by

```
bp=. succ@] +//. prob ,
```

and

```
0.6 bp 3,
```

which is equal to

```
0.064 0.288 0.432 0.216,
```

gives the probabilities of **0**, **1**, **2** and **3** successes in **3** binomial trials with probability **0.6** of success in a single trial.

The binomial coeeficients may be calculated more simply as **bc=. i.@>: !]** where **!** is the verb *out of*.

Poisson distribution

The Poisson distribution applies when the probability of success on any one trial is very small and the number of trials is large so that the expected number of successes, the product of these two quantities, is of moderate size. If the mean number of successes is λ, then the probability of x successes, where x is a non-negative integer, is $e^{-\lambda}\lambda^x/x!$. The probability density function is

```
poisson=. ^@-@[ * ^ % !@]
```

and, for example, the first four probabilities for a distribution with mean **1.5** are given by the expression

```
1.5 poisson 0 1 2 3
```

which has the value rounded to four decimal places of

```
0.2231 0.3347 0.2510 0.1255 .
```

A classical example, given in Weaver (1963), is the number of deaths which occurred from 1875 to 1894 in various German army corps due to kicks from horses, and are given here in the table **h** where the **14** rows represent army corps and the **20** columns represent years which may be displayed by the expression **1":h**:

```
02210011030210010101
00020302000111020310
00020200110021100200
00011120200010121000
01011110000100001100
00002100100101111110
00102001201131110300
10100010110020021020
10001001000010001101
00000211102110120100
00110102020000213011
00002401301111213131
11211304010321021100
01000001011000220000
```

The maximum number of deaths is **>./,h** which is **4**, and the frequencies of **0** to **5** deaths, inclusive, are

```
0 1 2 3 4 5 rfr ,h
```

which is **144 91 32 11 2 0**. The average number of deaths is **am ,h** or **0.7**. An estimate of the Poisson probabilities for **0** to **5** deaths, inclusive, is given by **0.7 poisson i. 6** which is

```
0.49659 0.34761 0.12166 0.02839
                0.00497 0.00070 .
```

The expected frequencies are **280** times these values which, rounded to one decimal place, are

```
139.0 97.3 34.1 7.9 1.4 0.2 .
```

If we let the observed and estimated frequencies be represented by the lists **obs** and **exp**, respectively, then these calculations may be conveniently summarized by the table

```
5.0 5.0 8.1": (i. 6),.obs,.exp
```

which has the value

```
0  144   139.0
1   91    97.3
2   32    34.1
3   11     7.9
4    2     1.4
5    0     0.2 .
```

Locales

Objects in **J** - nouns, verbs, etc. - are stored in one or more *locales* which prevent name conflicts between objects in different applications which might be in use at the same time. Locales are designated by suffixes, as for example **mean_temp_** which would refer to the name **mean** in locale **temp**. A locale is created by creating one or more names in the locale, and is removed by deleting all names in it.

The default locale is known as the *base* locale and all names for which a locale is not specified are stored in it. Another locale is the **z** locale which contains several sets of utilities which are loaded by the standard profile. Reference to a name produces a search of the specified locale or the base locale if no locale is given, and then, if the name has not been found, the **z** locale.

As a simple example consider the verb

```
mean=. +/ % #
```

for the arithmetic mean defined in the base locale, and the verb

```
mean_temp_=. # %: */
```

for the geometric mean defined in the locale **temp**. Then we may define the verbs

```
test=. <./,mean,>./
```

and

```
test_temp_=. <./,mean,>./
```

so that if **w** is the list **2.3 5 3.5 6**, **test w** is the list **2.3 4.2 6** of minimum, arithmetic mean and maximum, and **test_temp_ w** is the list **2.3 3.94211 6** of minimum, geometric mean and maximum.

The **z** locale is the parent or root locale and is populated with a number of scripts specified in the standard profile. Two of these are **stdlib.js** which contains a number of nouns and verbs that may be assumed to be available for all applications, and **winlib.js** which contains nouns and verbs assumed to be available for Windows applications. These two scripts should not require modifying. The script **myutil.js** is initially empty and may be modified as required.

Utilities

The computational facilities of **J** may be used in conjunction with those of other software packages. For example, the results of a dice-throwing simulation in **J** could be plotted with some appropriate graphics program and then incorporated into a word-processing document. As another example, data from some experimental design that had been assembled in a spreadsheet could be passed to **J** for processing by an analysis-of-variance program, or data assembled in **J** could be passed to a statistical package for processing. **J** Release 2 for Windows permits the very simple sharing of data, graphics and text with a variety of Windows and even DOS programs. We shall introduce in this section some utilities that simplify the exchange of data between **J** and other Windows and DOS software, and in the following three sections give a few examples of their use with some software packages. Some of these utilities will be used also in Appendix 4.

The following two verbs are available in the script **stdlib.js**:

list List the data given as the right argument in columns to a maximum width given by the left argument.

toLF Convert character strings to be delimited by **LF**.

The following two verbs are available in the script **winlib.js**:

wdclipread Read clipboard. An argument is required but is ignored.

wdclipwrite Write character data to clipboard. The result is the number of characters written.

The following verb is available in the script **convert.js:**

tolower Convert character argument to lower case.

The following two verbs are available in the

script **files.js**:

freads Read characters from a file, converting any **CRLF** pairs or isolated **CR** characters into **LF** characters.

fwrites Write character text to a file. The text is first ravelled into a list with each row teminated by the **CRLF** pair. If the file exists, it is overwritten. The result is the number of characters written.

The following verb is available in the script **format.js**:

clipfmt Format a scalar, list or table to that required by other software reading the clipboard.

The following six verbs are available in the script **misc.js**:

charsub Substitute characters in character data. The left argument is one or more character pairs listed sequentially and the right argument is the character array. For example,

```
'ch' charsub 'cat'
```

is the character list **hat**.

chop Chop a character list into items and a table into columns. The left argument is the delimiter with a default of **LF**; the right argument is the array to be chopped. For example, **chop ": i. 3** is

```
+-+-+-+
|0|1|2|
+-+-+-+
```

.

deb Delete extra blanks, i.e., leading and trailing blanks and any extra blanks, from a character list.

nubcount The nub of the list argument linked with the count of each item as a two-column table.

rplc Replace characters in a character string. The left argument is the character string and the right argument is a two-item boxed list giving the old and new characters being replaced. For example,

```
'The Tail of Peter Rabbit'
   rplc 'Tail';'Tale'
```

is **The Tale of Peter Rabbit**.

The verbs in the last four files may be copied to the script **myutil.js** or the corresponding files may be copied in their entirety by the commands **load 'convert'**, **load 'files'**, etc.

The first problem to be considered is the conversion between the underscore used to represent negative numbers in **J** and the conventional hyphen used in other applications. McDonnell (1992) considers this problem in a short paper on the verb *amend* **}** which may be consulted for an understanding of the process of substitution. However we shall use the verb **charsub** while retaining the designation **hfu** for the verb for hyphen-from-underscore substitution given in the paper and also introducing the verb **ufh** for the converse substitution. These verbs may be defined simply as

```
hfu=. '_-'&charsub
```

and

```
ufh=. '-_'&charsub
```

, and, for example,

```
hfu '123 _45 _6 789'
```

is

```
123 -45 -6 789
```

and

```
ufh hfu '123 _45 _6 789'
```

is

```
123 _45 _6 789
```

.

We may now define the verb

```
CLIPwrite=.
      wdclipwrite@hfu@clipfmt
```

to copy data to the clipboard in a format that may be read by other software. The converse operation of pasting the clipboard contents into **J** may be accomplished by

```
CLIPread=. Jfmt@ufh@wdclipread
```

where

```
Jfmt=. (".&>)@chop@toLF
```

and **".** is the verb *do* which executes its argument. For example, the expression

```
CLIPwrite i. 3 4
```

would copy the designated table to the clipboard and produce a result of **29**, the number of characters copied, and

```
x=. CLIPread ''
```

would assign to **x** the current contents of the clipboard.

Similar operations of writing to, and reading from, DOS files may be carried out by the verbs

```
DOSwrite=. (hfu@":@[) fwrites ]
```

and

```
DOSread=. Jfmt@ufh@freads
```

.

For example, the expression

```
(i. 3 4) DOSwrite'c:\j2\temp\y'
```

would write the table to the specified file, and

```
x=. DOSread 'c:\j2\temp\y'
```

would assign to **x** the contents of the file. Files containing text only and no numeric data should be written and read by the verbs **fwrites** and **freads**, respectively.

Other software - Graphics

The example of a graphics package we shall use is GNUPLOT, a widely used interactive plotting program available by anonymous ftp at a several sites, which allows the plotting of two- and three- dimemsional functions and data. Fraser Jackson has provided in his JSTAT statistical package a number of verbs which permit communication with GNUPLOT from within **J**. In this section, however, we shall give only one simple example of the use of GNUPLOT in which a file containing a **J** array is created in the GNUPLOT directory.

In a previous section we defined the three-column table **RegEx** giving the values of the observed values of an independent variable and one dependent variable and the estimated values of the dependent variable in an example using linear regression to show the relationship between crop yield and the amount of water applied. To display these data graphically using GNUPLOT, the table is first written to a file, `regex`, say, in the GNUPLOT directory by the command

```
RegEx DOSwrite 'c:\gnu\regex' .
```

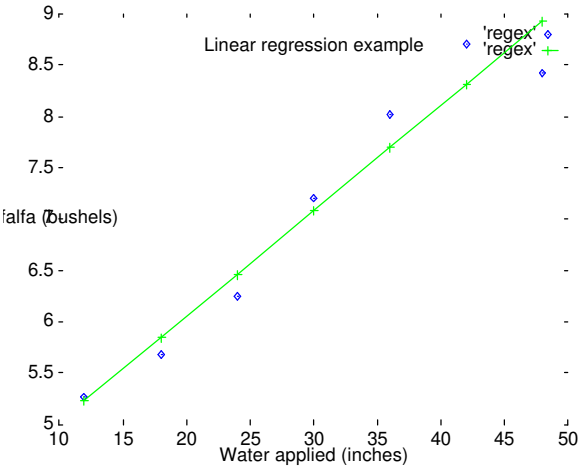
If GNUPLOT is then opened, execution of the four commands

```
set title 'Linear regression example',-3
set xlabel 'Water applied (inches)'
set ylabel 'Yield of alfalfa (bushels)'
plot 'regex' using 1:2, 'regex' using 1:3
                with linespoints
```

will produce Chart (a) showing observed values of yield together with the estimated values and the regression line. If the GNUPLOT commands are given in a file, named, for example, `regex.prg`, which is saved in the GNUPLOT directory, then execution of the GNUPLOT command

```
load 'regex.prg'
```

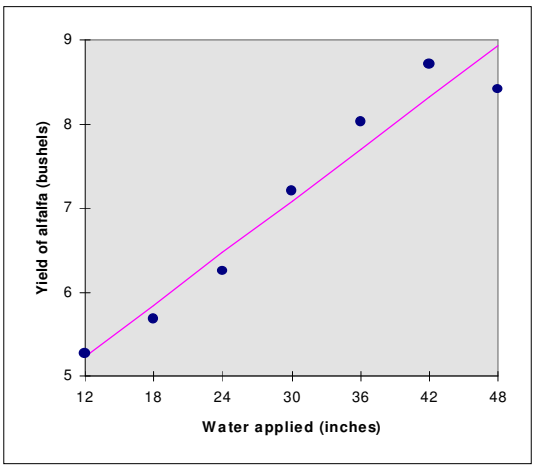
will produce the graph.



(a) Linear regression (GNUPLOT)

Other software - Spreadsheets

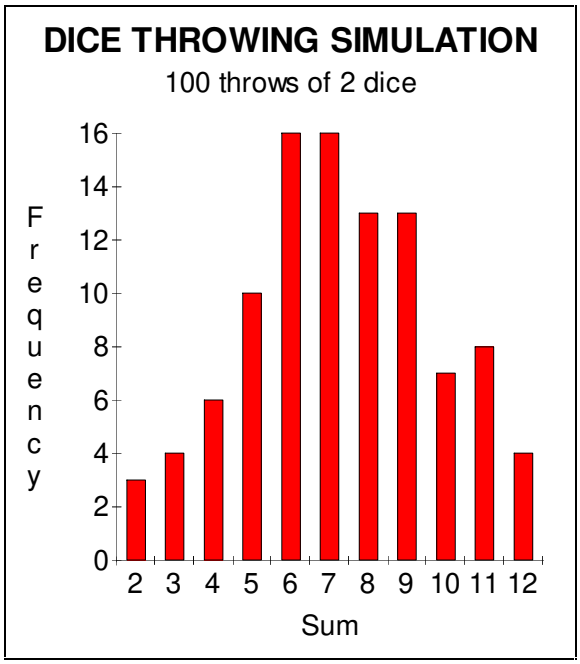
J may be used effectively with spreadsheets such as Microsoft Excel or those included with integrated packages such as Microsoft Works with the Windows Clipboard providing a simple means of communication



(b) Linear regression (Excel)

between **J** and whatever other software is being used. In Chart (b) the regression example referred to above is plotted using Excel, and in Chart (c) data corresponding to rolling two dice generated by the verb **SumTwoDiceFr** are plotted using Microsoft Works.

As an example of transferring data from a spreadsheet to **J** consider the analysis-of-variance data in the three-dimensional array **TestData** in Appendix 3. Suppose these data had been entered originally in a spreadsheet as an eight-by-six array with



(c) Dice throwing (MS Works)

```
25 7 21 4 10 16
```

as the first row,

```
5 21 4 25 7 6
```

as the second row, etc., and then copied to the Clipboard. The array could then be retrieved in **J** and restructured to the desired three- dimensional array by the expression

```
TestData=. 2 4 6 $, CLIPread '' .
```

If the data were saved as the textfile

```
c:\msworks\mydata\testdata.txt ,
```

the file could be read by the expression

```
DOSread 'c:\msworks\mydata\testdata.txt' .
```

Other software - Statistics

Let us consider an example of the use of **J** with the widely used statistical package MINITAB. Suppose that for some reason we wish to use the Balanced ANOVA design in MINITAB to analyze the data given in **TestData**. MINITAB requires that the data be arranged in columns with the response in one column and corresponding levels of the responses in other columns. For example, the row corresponding to the first item would be `25 1 1 1`, the second `7 1 1 2`, ... and the last `12 2 4 6`. These levels are given by the verb

```
AOVlevels=. >:@(] #: i.)@$ ,
```

and **AOVlevels TestData** would give an array of shape **2 4 6 3**. This array may be converted to the required table by

```
Flatten=. (*/@}:,{:)@$  $ ,@]  ,
```

where the verb *curtail* **}:** gives all items except the last, so that

```
Flatten AOVlevels TestData
```

gives the table of levels with **48** rows and **3** columns. The data table and levels table may be combined by the verb

```
AOVdata=. , ,. Flatten@AOVlevels
```

so that **AOVdata TestData** would give the data in the four-column format required by MINITAB, and the expression

```
CLIPwrite AOVdata TestData
```

would copy the table to the Clipboard from where it may be pasted to a MINITAB worksheet.

Analysis-of-variance data of the form given by the verb **AOVdata** may also be analyzed by a DOS package such as MYSTAT which is an instructional version of the statistical package SYSTAT. The data are simply written to the file `mytest.dat`, say, in the MYSTAT directory by the expression

```
(AOVdata TestData) DOSwrite
        'c:\mystat\mytest.dat'
```

from where it may be accessed by the appropriate MYSTAT commands.

Gerunds

In English grammar a gerund is a verb form, ending in *-ing*, which may be used as a noun, e.g., *The Art of Teaching*. In **J** the conjunction *tie* **`** may be used with two verb arguments to form a gerund whose elements may be selected and executed. Gerunds may be used with the adverb insert with the elements being selected cyclically if required, and, for example,

```
+`*/1 2 3 4 5
```

is equivalent to **1+2*3+4*5** and is equal to **47**.

A gerund used with the *agenda* conjunction **@.** gives a case statement since in the expression **m@.v** the result of the verb **v** selects the appropriate item of the agenda **m**. For example,

```
3  +`-`*`% @. 1: 4
```

is **3-4** or **_1**, and

```
3  +`-`*`%  @. 3: 4
```

is **3%4** or **0.75**. Also

```
+`-`*`% @. (4&|)"0 i. 5
```

is **0 _1 1 0.3333333 4**. As another example, if

```
SD=. (_:-_:)`sd @. (#>1:) ,
```

then **SD w**, where **w** is the list **2.3 5 3.5 6**, is **1.63095** and **SD 3.5** is **_.** representing *indeterminate* since the standard deviation is undefined for a single observation.

The gerund may be used in recursive definitions.

For example, the verb

```
rsum=. 0:`({:+rsum@}:) @. (#>0:)
```

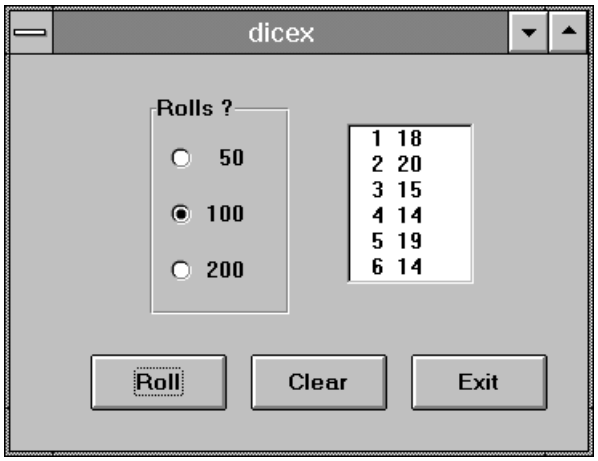
represents the recursive definition of a sum of a list as the last observation plus the sum of all observations except the last, and **rsum w** is **16.8**. The recursion terminates with an empty list whose sum is **0**. The factorial function, a classic example of recursion, may be defined as

```
fact=. 1:`()*fact@<:) @. * ,
```

and **fact"0 (i. 7)** is **1 1 2 6 24 120 720**.

Windows programming

In this section we shall introduce some the facilities available in **J** for Windows programming. The remarks will be limited mostly to a brief discussion of the construction of a **J** program to produce the following dialogue box for a demonstration of rolling a die and tabulating the frequencies of the six faces that may occur:



The number of rolls, 50, 100 or 200, is first selected, the "roll" button is clicked and the frequency table appears in the box. This may be repeated as often as is desired. The dialogue box may be closed by clicking the "Exit" button. The "Clear" button, which need not be used, simply clears the results of the last simulation from the frequency table box.

The verbs for this example may be found in the script file in Appendix 5. The frequency table is generated by the verbs

```
diefr=. [: +/"1 (i.6)&
                (=/)@(?@$&6)
```

which gives the frequencies for an arbitrary number of rolls of a single die,

```
dietab=. (>:@i.@6:),. diefr
```

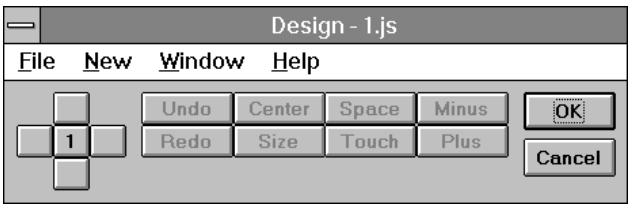
which gives the frequenciy table, and

```
dielist=._8: <\ ((4.0 &":)
                @,@dietab)
```

which gives the tables as a six-item character list in a format for convenient displaying in the frequency box.

Details on the Windows interface and Windows programming may be found in the *User Manual*, Burke (1994) and in the fourteen-page supplementary documentation *The Form Editor* provided with **J** Release 2.05 for Windows.

To create a script file for a form, create a new script file (**New JS**), then select the menu command **Edit/Form Edit**, and give the form a name, **dice** in this example. The new form is then given together with the following Design dialog box:



If the form is run, by selecting **File/Run** from the Dialog, the following form is given:

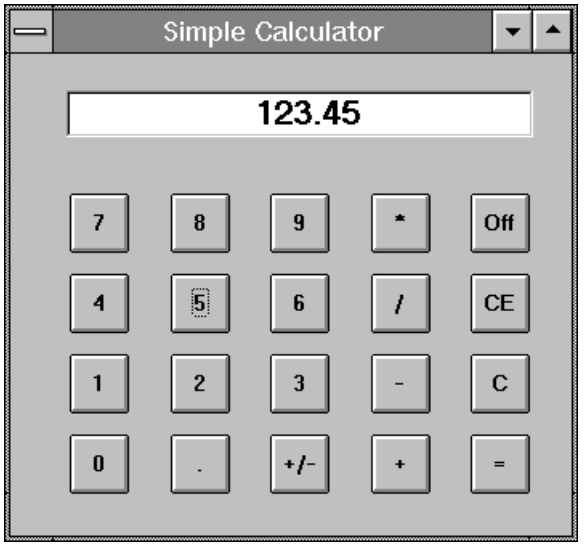
Selecting the button **Nil** has no effect, while selecting **Exit** closes the form. In addition to the Dialog box and the form, the following script is created in the new script file which should be saved with an appropriate name:

```
DICEX=: 0 : 0
pc dicex;
xywh 136 8 34 14;cc nil button;cn "Nil";
xywh 136 30 34 14;cc exit button;cn "Exit";
pas 10 10;pcenter;pcloseok;pescclose;pscale;
rem form end;
)

dicex_init=: 3 : 0
wd DICEX
wd 'pshow;'
)

dicex_button_nil=: 3 : 0
wd 'psel dice;'
)

dicex_button_exit=: 3 : 0
```

```
wd 'psel dicex;pclose;'
)

dicex_run=: 3 : 0
dicex_init''
wdml 1
)
```

The above script has been generated from the standard script in the file **c:\j2\main\forminit.js** which represents a slightly simplified form of the script supplied with the current version of **J**. The noun **DICE** defines the structure of the form, the verb **dice_init** allows for any initialization, the next two verbs define the action of the two buttons, while the last verb allows the form to be run apart from the Dialog box by the statement **dice_run ''**. The completed script file for this example is given in Appendix 5.

The following figure gives a simple four-function calculator similar to one shown in a brochure advertising APL*PLUS III for Windows:



Its construction is a relatively simple process not much more complicated than the previous example, and details as well as the corresponding script file are not given.

DOS and Windows programs may be called from within **J** by the verb **winexec**, and, for example, the expression

```
wd 'winexec
      c:\msworks\msworks.exe;'
```

will call MS Works. Furthermore, the verb

```
works=. wd bind 'winexec
      c:\msworks\msworks.exe;' ,
```

where **bind**, given in **stdlib.js**, binds the argument to a monadic verb, allows MS Works to be called by the expression

```
works ''.
```

Finally, we may note that **J** also supports controls for Dynamic Data Exchage (DDE), Object Linking and Embedding (OLE) and Visual BASIC Extended (VBX). Details and several examples may be found in the *User Manual*.

Coupon collector's problem

As a final example of the use of **J** we shall consider the coupon collector's problem of sampling with replacement from a finite population until all of the items are represented in the sample. Such a sampling procedure may serve as a model for collecting a complete set of prizes included, one prize in a package, in products such as breakfast cereal. If there are *n* different prizes, the problem is equivalent to sampling with replacement from the first *n* positive integers until all *n* integers are represented in the sample. The expected sample size may be shown to be "*n* times the sum of the reciprocals of the first *n* positive integers". The table on the next page gives the expectations rounded to the nearest integer for values of *n* from 1 to 100, inclusive.

An expression for the expected sample size for a population of **5** prizes, say, is

```
5 * +/ % >: i. 5
```

which has the value **11.4167**. An expression for **n** prizes is given by

```
n * +/ % >: i. n
```

which, if **n** is **5**, has the same value as the previous expression. If we define the verb

```
pos=. >: @ i.
```

so that, for example, **pos 5** is **1 2 3 4 5**, then a verb to give the expectation for an arbitrary number of prizes is given by

```
cc=. * +/ @ % @ pos
```

and, for example, **cc 5** is **11.4167** and **cc 10** is **29.2897**. We note that **cc 0** is **0** which is reasonable since if there are no prizes to collect there is nothing to purchase. The table of expectations on the next page is given by **20 cctable 5**, where

```
cctable=. [] * i.@[) (int@cc@+)
                table pos@ ]
```

and

```
int=. ". @ (6.0&":)
```

rounds the items of its argument to the nearest integer.

In order to simulate the coupon collector's problem let us first define the verb

```
Prize=. >: @ ?
```

```
+--+------------------+
|  |  1   2   3   4   5|
+--+------------------+
| 0|  1   3   6   8  11|
| 5| 15  18  22  25  29|
|10| 33  37  41  46  50|
|15| 54  58  63  67  72|
|20| 77  81  86  91  95|
|25|100 105 110 115 120|
|30|125 130 135 140 145|
|35|150 155 161 166 171|
|40|176 182 187 192 198|
|45|203 209 214 219 225|
|50|230 236 242 247 253|
|55|258 264 269 275 281|
|60|286 292 298 304 309|
|65|315 321 327 332 338|
|70|344 350 356 362 368|
|75|374 379 385 391 397|
|80|403 409 415 421 427|
|85|433 439 445 451 457|
|90|464 470 476 482 488|
|95|494 500 506 513 519|
+--+------------------+
```

which will be used to select one random positive integer, and, for example, **`Prize 4`** would simulate the selection of one prize out of a total of four different prizes, and the simulation of **`10`** random selections of **`4`** prizes is given by **`Prize 10$4`** could have the value

```
1 4 2 3 1 1 3 3 4 2 .
```

Repeated simulation of selecting prizes until all prizes are represented in the sample may be accomplished by the verb **`ccsim`** defined as

```
+-+-+------------------+
|3|:|r=. 0$0           |
| | |while. y. > #~.r do.|
| | |   r=. r, Prize y.|
| | |end.              |
+-+-+------------------+
```

The use of this verb to simulate the coupon collector's problem with four prizes is given by the expression **`ccsim 4`** whose values on two occasions were **`4 1 1 2 4 3`** and **`1 3 2 2 2 1 3 4`**.

The verb

```
ccsample=. (#@ccsim)"0@#
```

gives a list of the sample sizes for an arbitrary number of simulations of the coupon collector's problem, and, for example, **`10 ccsample 4`** could have the value

```
4 6 8 5 5 5 7 8 13 6 .
```

The expression

```
s=. 100 ccsample 4
```

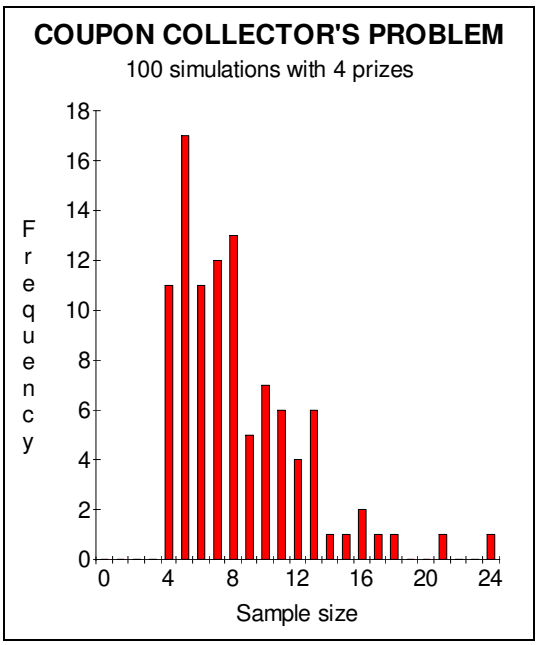
gives a list **`s`** of the sample sizes resulting from **`100`** simulations for **`4`** prizes, where the expected sample size is **`cc 4`** or **`8.3`**. One execution of this expression gave a value of **`s`** of

```
 8  7 13 11  6  7 10  6 16  5
 8  5 13  6  5 10 12  4  8 24
18  5  4  8 10  5  5 17  5  4
 8 11 10  7  4  6  8  7  7 12
 7  7 10 15  6  9  9 11 10 12
 5  8  6 13  5  5  6  5  8  7
 8  7  4  8  9  4  8 11  5  5
 4 13  9  7  4 21  4  4 12  6
 7 11  6 13 10  5 13 11  8  7
 9  5  6  5  8 14  4  6  5 16
```

when displayed as a table with **`10`** rows and **`10`** columns. Minimum, mean and maximum values of **`s`** are given by **`(<./,am,>./) s`** which is equal to **`4 8.31 24`**. Frequency classification with the verb **`mfr`** gave the list

```
0 0 0 0 11 17 11 12 13 5 7 6 4
           6 1 1 2 1 1 0 0 1 0 0 1
```

indicating that there were **`11`** samples of size four, **`17`** of size five, ..., and **`1`** of size twenty-four. The following is the barchart of the frequencies:



Cut and word formation

In this secion we shall introduce a conjunction and a verb which will be used in the extended example presented in Appendix 4. The first is the conjunction *cut* **`,.`** and the second is the monadic verb *word formation* **`;:`**.

In general, the expression **u,. 2 y** applies **u** to each set of intervals marked by the last item, called the *fret*, of **y**. For example, if

```
y=. 1 2 3 0 4 5 0 6 7 8 9 0
```

, then the expression **<,. 2 y** is

```
+-------+-----+---------+
|1 2 3 0|4 5 0|6 7 8 9 0|
+-------+-----+---------+
```

. If the digit is negative, then the frets are excluded, and **<;._2 y** is

```
+-----+---+-------+
|1 2 3|4 5|6 7 8 9|
+-----+---+-------+
```

, and **+/;._2 y** is the list **6 9 30** of the sums of items in the intervals. The right argument may be a character string, and, for example, the expression

```
<;. _2 t=. 'The Tale of Peter
                          Rabbit '
```

is

```
+---+----+--+-----+------+
|The|Tale|of|Peter|Rabbit|
+---+----+--+-----+------+
```

and **#,. _2** is the list **3 4 2 5 6** of word lengths.The expressions **u;. 1 y** and **u;. _1 y** are similar to **u;. 2 y** and **u;. _2 y** except that the fret is the first item and marks the beginnings of the intervals.

The expression **;: y** gives the list of boxed words in **y**, and, for example,

```
;: '  The  Tale of Peter Rabbit '
```

is

```
+---+----+--+-----+------+
|The|Tale|of|Peter|Rabbit|
+---+----+--+-----+------+
```

.

The verb **BoxSort** may be used to sort a boxed list of words into alphabetical order, and, for example, **BoxSort ;: t** is

```
+-----+------+----+---+--+
|Peter|Rabbit|Tale|The|of|
+-----+------+----+---+--+
```

and

```
BoxSort ;: tolower t
```

is

```
+--+-----+------+----+---+
|of|peter|rabbit|tale|the|
+--+-----+------+----+---+
```

.

The arguments of the verb less **-.** may be boxed lists so that, for example,

```
BoxSort (;: t) -. 'The';'of'
```

is

```
+-----+------+----+
|Peter|Rabbit|Tale|
+-----+------+----+
```

.

.

J and statistics

When APL first became available almost thirty years ago, there were relatively few statistical programs available and almost all of these were written in either machine-language for specific machines or in FORTRAN. For a person familiar with APL who wished to do statistical computing it appeared obvious that APL with its array capabilities and interactive implementation offered a much superior means of performing statistical calculations and developing libraries of statistical programs. Now there is not only a large number of statistical packages, some of them very comprehensive, but there are languages designed specifically for statistical computation. Moreover, many spreadsheets are sufficient for many simple calculations, and some are supplemented by statistical libraries of considerable versatility. Indeed, an integrated package offering wordprocessor, spreadsheet, graphical, database and communication facilities may accommodate most of the computing needs of many people.

What, then, is the place of **J** amidst such a variety of computing software? It was a desire to answer this question, as well as an interest in gaining a better understanding of the language, that prompted the preparation of this report. In retrospect, the answer seems very simple: **J** is just one of the many tools available to the statistician and should be used in conjunction with them for the efficient and convenient performance of statistical calculations. The use of the ASCII character set in **J** together with the Windows implementation and the large library of utilities allow **J** to be used with ease with any Windows text-processing, number-handling or graphics package and even with the older DOS software. Indeed, looked at this way, **J** is most attractive, powerful and intellectually satisfying part of the statistician's computing library.

Acknowledgements

I would like to thank Chris Burke and Brian Schott for their very helpful comments on earlier versions of this paper.

References

Abramowitz, M. and I. A. Stegun, 1964. *Handbook of Mathematical Functions*. Applied Mathematics

Series 55, National Bureau of Standards, Washington, D.C.

Burke, Chris, 1994. ***J*** *User Manual*. Iverson Software, Inc., Toronto.

Hoel, P. G., 1966. *Elementary Statistics. Second edition*. John Wiley & Sons, Inc., New York.

Hodges, J. L. and E. L. Lehman, 1964. *Basic Concepts of Probability and Statistics*. Holden-Day, Inc., San Francisco.

Iverson, K. E., 1991. *Arithmetic*. Iverson Software, Inc., Toronto.

Iverson, K. E., 1993. *Calculus*. Iverson Software Inc., Toronto.

Iverson, K. E., 1994. ***J*** *Introduction and Dictionary*. Iverson Software Inc., Toronto.

Jackson, L. F., 1994. Personal correspondence.

Knuth, D. E., 1969. *The Art of Computer Programming, Volume 2. Seminumerical Algorithms*. Addison-Wesley Publishing Company, Reading, Mass.

McDonnell, E. E., 1992. *Amend.* Iverson Software, Inc., Toronto.

Searle, S. R., 1966. *Matrix Algebra for the Biological Sciences*. John Wiley and Sons, Inc., New York.

Smillie, K. W., 1993. "Some J verbs for orthogonal factorial experiments." *Vector*, vol. 9, no. 3, pp. 117 - 124.

Sprent, P., 1981. *Quick Statistics*. Penguin Books, Ltd., Harmondsworth, Middlesex.

Steel, R. G. D. and J. H. Torrie, 1960. *Principles and Procedures of Statistics*. McGraw-Hill Book Company, Inc., New York.

Weaver, W., 1963. *Lady Luck*. Doubleday & Company, Inc., Garden City, N. Y.

Appendix 1. Language summary

		.	:
`=`	Self-Classify · Equal	Is (Local)	Is (Global)
`<`	Box · Less Than	Floor · Lesser of	Decrement · Less Or Eq
`>`	Open · Larger Than	Ceiling · Larger of	Increment · Larger Or Eq
`_`	Negative Sign / *Infinity*	*Indeterminate*	Infinity
`+`	Conjugate · Plus	Real/Imag · GCD (Or)	Double · Not-Or
`*`	Signum · Times	Len/Angle · LCM (And)	Square· Not-And
`-`	Negate · Minus	Not (1-) · Less	Halve · Match
`%`	Reciprocal · Divided By	Matrix Inv · Mat Divide	Square Root · Root
`^`	Exponential · Power	Natural Log · Logarithm	**Power**
`$`	Shape Of · Shape		Self-Reference
`~`	***Reflex*** · ***Pass*** · EVOKE	Nub ·	Nub Sieve · Not-Eq
`|`	Magnitude · Residue	Reverse · Rotate (Shift)	Transpose
`.`	**Det** · **Dot Product**	**Even**	**Odd**
`:`	**Explicit** · **Monad/Dyad**	**Obverse**	**Adverse**
`,`	Ravel · Append	Ravel Items · Stitch	Itemize · Laminate
`;`	Raze · Link	**Cut**	Word Formation ·
`#`	Tally · Copy	Base 2 · Base	Antibase 2 · Antib
`!`	Factorial · Out Of	**Fit** (**Customize**)	**Foreign**
`/`	***Insert*** · ***Table*** · INSERT	***Oblique*** · ***Key*** · APPEND	Grade Up · Sort
`\`	***Prefix*** · ***Infix*** · TRAIN	***Suffix*** · ***Outfix***	Grade Down · Sort
`[`	Same · Left	**Lev**	Cap
`]`	Same · Right	**Dex**	
`{`	Catalog · From	Head · Take	Tail ·
`}`	***Item Amend*** · ***Amend***	Behead · Drop	Curtail ·
`"`	**Rank** · CONSTANT	Do ·	Format
`` ` ``	**Tie** (**Gerund**)		**Evoke Gerund**
`@`	**Atop**	**Agenda**	**At**
`&`	**Bond / Compose**	**Under** (**Dual**)	**Appose**
`?`	Roll · Deal		
	`a.` *Alphabet*	`a:` Ace (boxed empty)	`A.` Atomic Permute
	`b.` ***Boolean/Basic***	`c:` Characteristic values	`C.` Cycle-Direct · Perm
	`D.` **Derivative**	`D:` **Secant slope**	`e.` Raze In · Member (In)
	`E.` · Member of Interval	`f.` ***Fix***	`i.` Integers · Index Of
	`j.` Imaginary · Complex	`NB.` Comment	`o.` Pi Times · Circle
	`p.` Polynomial	`r.` Angle · Polar	`t.` ***Taylor coefficient***
	`T.` **Taylor Function**	`x.` Left Argument	`y.` Right Argument
	`_9:` to `9:` Constant fns		

Appendix 2. Multiple regression

```
      reg
+-+-+-------------------------------------------------------+
|3|:|b=. (y=. ;@{: y.)%.X=.(1&,"1)@|:@(>@}:) y.              |
| | |sst=. +/*:(y-am y)                                     |
| | |ssr=. sst-sse=. +/*:(y- X +/ . * b)                    |
| | |F=. (msr=. ssr%k)%mse=. sse%_1+(n=. $y)-k=. <:#y.      |
| | |rsq=. ssr%sst                                          |
| | |seb=. %:(0{mse)*(<1 0)|:%.(|:X)+/ . * X                |
| | |r=. 49{.'             Var.    Coeff.      S.E.        t'|
| | |r=. r, 15.0 12.5 12.5 10.2 ": (i. >:k),. b,. seb,. b%seb |
| | |r=. r, ''                                              |
| | |r=. r, '  Source     D.F.    S.S.        M.S.         F' |
| | |r=. r, 'Regression', 5.0 12.5 12.5 10.2 ": k, ssr,msr,F  |
| | |r=. r, 'Error     ', 5.0 12.5 12.5": (n-k+1), sse, mse   |
| | |r=. r, 'Total     ', 5.0 12.5 ": (n-1), sst              |
| | |r=. r, ''                                              |
| | |r=. r, 'S.E. of estimate    ', 10.5":%:mse             |
| | |r=. r, 'Corr. coeff. squared', 10.5": rsq              |
+-+-+-------------------------------------------------------+

   reg x1;y1
            Var.    Coeff.      S.E.         t
             0     3.99429     0.35656     11.20
             1     0.10286     0.01104      9.32

  Source     D.F.    S.S.        M.S.         F
Regression    1    10.66423    10.66423     86.87
Error         5     0.61377     0.12275
Total         6    11.27800

S.E. of estimate       0.35036
Corr. coeff. squared   0.94558

   x21=. 1 4 2 2 1 3
   x22=. 0 6 4 3 1 5
   y2=. 10 17 13 14 12 15
   reg x21;x22;y2
            Var.    Coeff.      S.E.         t
             0     9.59821     0.94950     10.11
             1     1.18750     1.06075      1.12
             2     0.41964     0.53529      0.78

  Source     D.F.    S.S.        M.S.         F
Regression    2    27.15179    13.57589     17.34
Error         3     2.34821     0.78274
Total         5    29.50000

S.E. of estimate       0.88472
Corr. coeff. squared   0.92040
```

Appendix 3. Analysis of variance

```
      AOV
+-+-+-----------------------------------------------+
|3|:|(alltags #$y.) AOV y.                          |
| | |:                                              |
| | |AOVtable=: x. allterms y.                      |
| | |Labels=. 8{."1 >x.                             |
| | |dfTotal=. <:*/$y.                              |
| | |ssTotal=. (+/*:,y.) - '' ss0 y.                |
| | |dfError=. dfTotal - +/0{"1 AOVtable            |
| | |if. dfError>0 do.                              |
| | |   ssError=. ssTotal - +/1{"1 AOVtable         |
| | |   AOVtable=. AOVtable,ssError(],[,%)dfError   |
| | |   Labels=. Labels,'Error'                     |
| | |end.                                           |
| | |r=. Labels,"1 (5 12.5 12.5)":AOVtable          |
| | |AOVtable=: AOVtable,TotalRow=. dfTotal,ssTotal|
| | |r=. r, 'Total   ',5 12.5":TotalRow             |
+-+-+-----------------------------------------------+
```

```
   TestData
25  7 21  4 10 16
 5 21  4 25  7  6
 3  6 16 18 20 18
19 17 16  2 15  6

13 23  8 25 19 19
20 14 23  9 10 18
27 26 19 24 13  9
19 13 20 18 16 12
```

```
   AOV TestData
A           1   252.08333   252.08333
B           3    69.16667    23.05556
C           5    74.66667    14.93333
AB          3    10.41667     3.47222
AC          5    51.91667    10.38333
BC         15   309.83333    20.65556
ABC        15  1495.58333    99.70556
Total      47  2263.66667
```

```
   ('A';'B';'AB';'C') AOV TestData
A           1   252.08333   252.08333
B           3    69.16667    23.05556
AB          3    10.41667     3.47222
C           5    74.66667    14.93333
Error      35  1857.33333    53.06667
Total      47  2263.66667
```

Appendix 4. Statistics and literary style

In this paper we have introduced the **J** language by means of a number of statistical examples. Now we wish to give an extended example of the application of **J** to a specific problem. Many examples given in textbooks are too complicated or make use of artificial data to which names, sometimes rather cute, have been attached to give them an appearance of verisimilitude. To avoid both of these problems we shall consider a few examples of the use of elementary statistics in the numerical study of literary style with short passages of English text providing the data.

Of the very many references to the quantitative study of literary style we shall mention only three here. The first is C. B. Williams's delightful little book *Style and Vocabulary: Numerical Studies* (Griffin, 1970). For the author, a retired entomologist who was eighty when the book was published, the book summarized the results of an avocation of many years which allowed him to combine his interest in quantitative methods with his love of English literature. The book is of particular interest for its summary of early work in the subject with a dependence on manual methods of tabulation and analysis. *The Computation of Style* by Anthony Kenny (Pergamon, 1982) gives a brief introduction to some of the statistical methodology used together with a large number of examples drawn from a variety of literary sources. Again the emphasis is on manual methods with a few remarks in the last two pages on the use of computers. The third book, *Computers in Linguistics* by Christopher Butler (Blackwell, 1985), gives an introduction to the use of computers and the SNOBOL4 language in linguistics research and has many interesting examples and a ten-page bibliography.

In this Appendix, then, we shall give some **J** verbs which together with the verbs developed earlier allow us to find distributions of word and sentence length, vowel frequencies, and the occurrence of specific words and classes of words. Other software will be used as required with **J** for the convenient presentation of results.

There are only two remarks to be made about the definition of a word. The first is that in the sample texts we shall be using hyphens will be ignored so that hyphenated words will be considered as single words; the second is that an abbreviation must not end with a period since the period is used to indicate the end of a sentence. The definition of a sentence, however, presents more of a problem. The Second Edition of Fowler's *Modern English Usage* says that a sentence "is defined in every grammar book and every dictionary, but it would not be easy to find two that gave the same definition, and some make heavy weather of it", and then gives ten different definitions none of which was considered suitable for the present purposes. The definition we shall use, from the *Concise Oxford Dictionary of Current English (Eighth Edition*), is that a sentence is "a piece of writing or speech between two full stops or equivalent pauses, often including several grammatical sentences (e. g. *I went; he came*)". We shall define a full stop as a period, question mark or exclamation mark. Even then there may be problems with dialogue such as

"Meow! Meow!" said the little cat.
"What do you want?" she asked.

which will be considered as two sentences, each of which has more than one full stop so that the two exclamation marks in the first sentence and the question mark in the second should be elided. Such editing of the text can probably be best done manually before, or even during, its entry into the appropriate computer file. In addition, manual editing should be used to remove any periods from the end of abbreviations.

When the text is entered into a computer file, it is essential that the last character in the file be a period. Otherwise, the verb **`Sentences`** given below which makes use of the verb cut **`;.`** for dividing the text into sentences will give incorrect results. If there is any doubt regarding the terminal character, use should be made of the verb **`hexdump`** given in the script **`format.js`**.

After any manual editing is done to text, it must be then edited by a sequence of verbs which accomplish the following transformations: line-feed characters **`LF`** which mark the end of lines in **J** are replaced by spaces; the punctuation characters comma **`,`**, semicolon **`;`**, colon **`:`**, left and right parentheses **`( )`** and double quote **`"`** are replaced by spaces and the question mark **`?`** and exclamation mark **`!`** by a period **`.`**; single quotes **`'`** and hyphens **`-`** are deleted; leading, trailing and duplicate spaces are removed; and any upper case characters are converted to lower case. These transformations may be accomplished by the verb

```
EditText=. tolower @ deb @
      delPunct @ repPunct @ SPfLF
```

where

```
SPfLF=. rplc&(LF;' ')   ,
```

```
repPunct=. ', ; : ( ) " ?.!.'
           &charsub  ,
```

and

```
delPunct=. rplc&('''';'') @
           (rplc&('-   ';''))  .
```

For a sample text we shall use the following rather plaintive quatrain of unknown origin:

Do you love me,
or do you not?
You told me once
but I forgot.

It appears in an early edition of Kenneth Iverson's *Programming in J*, but regrettably has been omitted from later works. It is defined in the script file as the noun **TextOne**. The expression

```
EditText TextOne
```

gives

```
do you love me or do you not. you
      told me once but i forgot.
```

which is a sixty-character list.

Once a text has been edited in this manner, it may be cut into sentences very simply if we note that the period marking the end of each sentence can serve as the terminal fret. Thus we define the verb

```
Sentences=. (;:&.>) @ (<;._2)
                    @ EditText
```

and

```
Sentences TextOne
```

is

```
+-----------------------------
|+--+---+----+--+--+--+---+---+
||do|you|love|me|or|do|you|not|
|+--+---+----+--+--+--+---+---+
+-----------------------------
       +-------------------------------+
       |+---+----+--+----+---+-+------+|
       ||you|told|me|once|but|i|forgot||
       |+---+----+--+----+---+-+------+|
       +-------------------------------+
```

The text may also be cut into words by replacing the periods by spaces which now serve as frets. Thus

```
Words=. ;: @ SPfSTOP @
           EditText  ,
```

where

```
SPfSTOP=. rplc&('.';' ')
```

replaces periods with spaces, and

```
Words TextOne
```

is

```
+--+---+----+--+--+--+---+---+
|do|you|love|me|or|do|you|not|
+--+---+----+--+--+--+---+---+
       ---+----+--+----+---+-+------+
       you|told|me|once|but|i|forgot|
       ---+----+--+----+---+-+------+
```

If we define the verb

```
ItemLengths=. # & >  ,
```

then the verb

```
SL=. ItemLengths @ Sentences
```

will give the list of sentence lengths, and **SL TextOne** is the two-item list **8 7**. Similarly the verb

```
WL=. ItemLengths @ Words
```

gives a list of word lengths, and **WL TextOne** is

```
2 3 4 2 2 2 3 3 3 4 2 4 3 1 6  .
```

The verb

```
WC=. WordCount @ Words  ,
```

where

```
WordCount=. (>@{."1 ,"0 {:"1) @
                    nubcount
```

and **nubcount**, which was introduced earlier, from **convert.js** gives the nub of its list argument and the count of each item in the nub, gives a frequency count of the words in a text arranged in descending order of frequency of occurrence. Similarly,

```
LC=. LetterCount @ Words  ,
```

where

```
LetterCount=. [: alpha&(;"0) [:
                    +/ (afr & >)
```

and the verb **afr** and the noun **alpha** were defined earlier for a frequency distribution of letters in a text, gives a frequency count of the letters in a text arranged in alphabetical order. A table of the letters which actually occur in a text is given by

```
LCnub=. (([: 0&~: >@{:"1)#])@LC
```

The frequency tables for **WC TextOne** and **LCnub TextOne** are given on the next page.

Suppose now that we wish to find those words or letters which occur in a text a given number of times, e.g., those words which occur only once or those letters which do not occur at all. First let us define the verb

```
SelectRows=. (>@({:"1@])e.[) #
                    {."1@]
```

which will select those items occurring in the first column of a two-column frequency table corresponding to specified frequencies occurring in the second column. For example,

```
2 3 SelectRows WC TextOne
```

is the list

```
+---+--+--+
|you|do|me|
+---+--+--+
```

of words which occur two or three times in **TextOne**. Now the verb

```
OnceOnly=. BoxSort @
                (1&SelectRows) ,
```

where **BoxSort** which sorts the items of a boxed list has been defined earlier, will select the items from a

```
+----------+------+
|+------+-+|+-+--+|
||you   |3|||b|1 ||
|+------+-+|+-+--+|
||do    |2|||c|1 ||
|+------+-+|+-+--+|
||me    |2|||d|3 ||
|+------+-+|+-+--+|
||love  |1|||e|4 ||
|+------+-+|+-+--+|
||or    |1|||f|1 ||
|+------+-+|+-+--+|
||not   |1|||g|1 ||
|+------+-+|+-+--+|
||told  |1|||i|1 ||
|+------+-+|+-+--+|
||once  |1|||l|2 ||
|+------+-+|+-+--+|
||but   |1|||m|2 ||
|+------+-+|+-+--+|
||i     |1|||n|2 ||
|+------+-+|+-+--+|
||forgot|1|||o|12||
|+------+-+|+-+--+|
|          |||r|2 ||
|          |+-+--+|
|          |||t|4 ||
|          |+-+--+|
|          |||u|4 ||
|          |+-+--+|
|          |||v|1 ||
|          |+-+--+|
|          |||y|3 ||
|          |+-+--+|
+----------+------+
```

frequency table which occur only once, and

```
OnceOnly WC TextOne
```

is the list

```
+---+------+-+----+---+----+--+----+
|but|forgot|i|love|not|once|or|told|
+---+------+-+----+---+----+--+----+ .
```

Such a list may be displayed more conveniently by the verb **list** given in **stdlib.js**, and, for example,

```
40 list OnceOnly WC TextOne
```

is

```
but    forgot i      love   not
once   or     told
```

The verb

```
NoneOnly=. BoxSort @
              (0&SelectRows)
```

will give the items appearing in a frequency table with zero frequencies, and

```
NoneOnly LC TextOne
```

is the list

```
+-+-+-+-+-+-+-+-+-+-+-+-+
|a|h|j|k|p|q|s|w|x|z|
+-+-+-+-+-+-+-+-+-+-+-+-+
```

of letters which do not occur in **TextOne**.

A list of all of the words occurring in a text is given by

```
WordList=. BoxSort @ ~. @ Words
```

and

```
40 list WordList TextOne
```

is

```
but    do     forgot i      love
me     not    once   or     told
you
```

We shall now discuss briefly six of the many text processing problems given in the books mentioned at the beginning of this Appendix, giving **J** expressions rather than verbs for each. The noun **T** represents unedited text such as **TextOne**.

i) Frequency of punctuation

A frequency table for the six punctuation characters **!**, **,**, **?**, **:**, **;** and **.** is given by

```
'!,?:;.' charfrtab T .
```

ii) Words beginning with a vowel

A boxed list of all words beginning with a vowel is

```
(({."1 >w) e. 'aeiou') # "1 w
```

where

```
w=. WordList T .
```

iii) Sentences beginning with a specific word

The number of sentences beginning with **word** is

```
+/(<,'word') ="0 ({."1 & >)
                    Sentences T
```

iv) Frequency of occurrence of words

The number of times each of **word1**, **word2** occurs is

```
+/(Words T) ="0/'word1';'word2'
```

We note that list of words to be searched for must be a boxed list not containing atoms so that any one-letter words in a list are given as in the example

```
'abc';(,'d');'ef'
```

and a list containing a single word as in the example **<,'abc'**.

v) Short and long words

The number of short (e.g., <5 letters) words followed by long (e.g., >=5 letters) words in a sentence is

```
+/2</\,(5&<:@$ & >)>3{Sentences T
```

vi) Grammatical and lexical words

The majority of words in a text are either grammatical words (pronouns, conjunctions, prepositions and auxiliary verbs) or lexical words (nouns, adjectives, adverbs and main verbs). Some words, however, may be ambiguous in that they can be grammatical or lexical according to context. If we let **A** and **G** be the lists of ambiguous and grammatical words, respectively, then

```
(w e. A) # w=. Words T
```

is the list of ambiguous words,

```
(w e. G) # w=. Words T
```

the list of grammatical words, and

```
((Words T) -. A) -. G
```

the list of lexical words in the text **T**.

For a longer sample of text to be used in the next example we have selected the opening paragraph of *A Scandal in Bohemia* by Sir Arthur Conan Doyle which will be stored as a DOS file which may be read with the verb **freads** to define the noun **TextTwo**:

To Sherlock Holmes she is always *the* women. I have seldom heard him mention her under any other name. In his eyes she eclipses and predominates the whole of her sex. It was not that he felt any emotion akin to love for Irene Adler. All emotions, and that one particularly, were abhorrent to his cold, precise but admirably balanced mind. He was, I take it, the most perfect reasoning and observing machine that the world has seen, but as a lover he would have placed himself in a false position. He never spoke of the softer passions, save with a gibe and a sneer. They were admirable things for the observer -- excellent for drawing the veil from men's motives and actions. But for the trained reasoner to admit such intrusions into his own delicate and finely adjusted temperament was to introduce a distracting factor which might throw a doubt upon all his mental results. Grit in a sensitive instrument, or a crack in one of his high-power lenses, would not be more disturbing than a strong emotion in a nature such as his. And yet there was but one woman to him, and that woman was the late Irene Adler, of dubious and questionable memory.

A list of word lengths for the Doyle text is given by the expression

```
w=. WL TextTwo
```

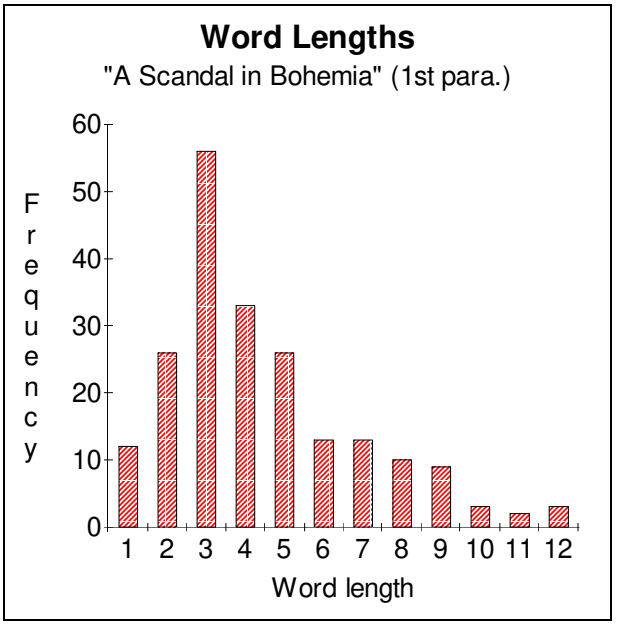
and we have that **>./w** is **12** and **am w** is **4.4**. A frequency table is given by

```
(>:i.12) rfrtab w
```

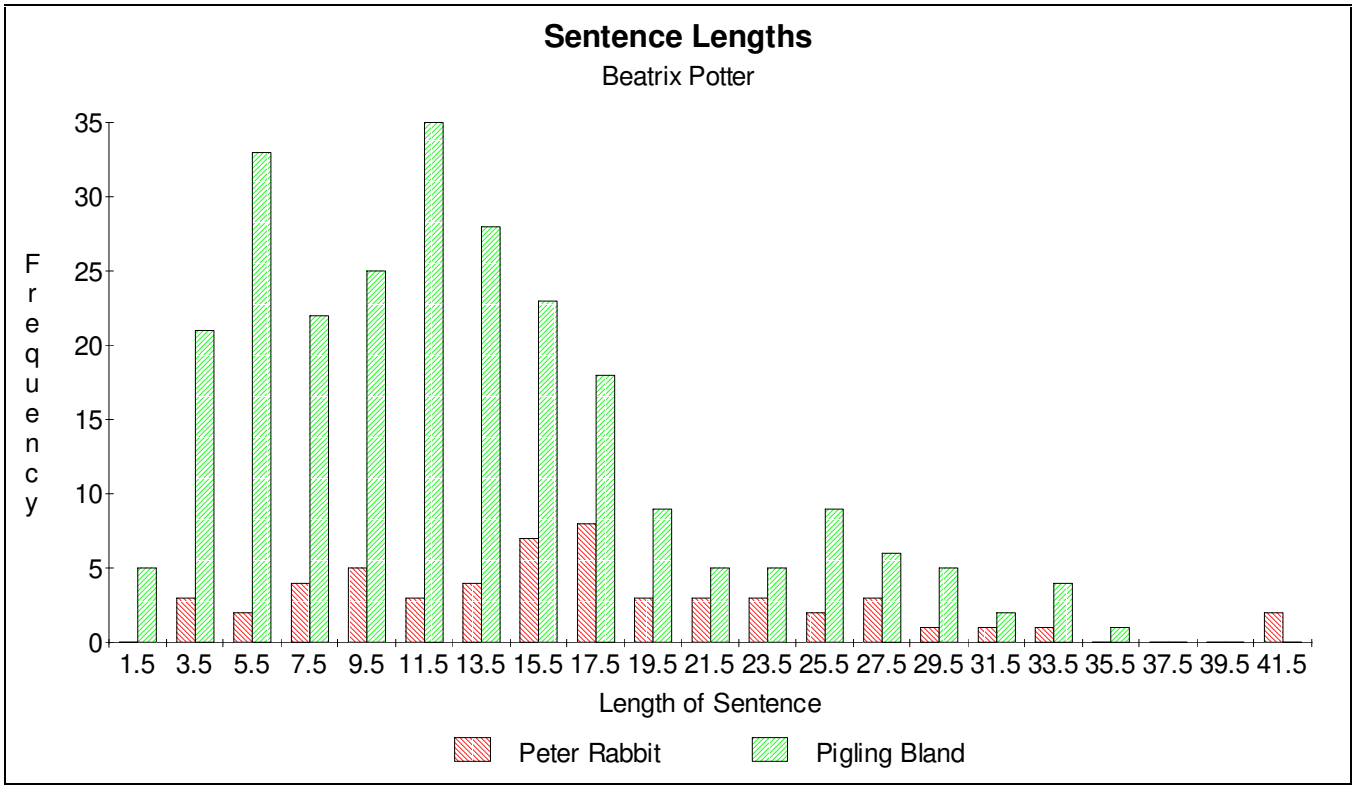
which, transposed for convenience of presentation, is

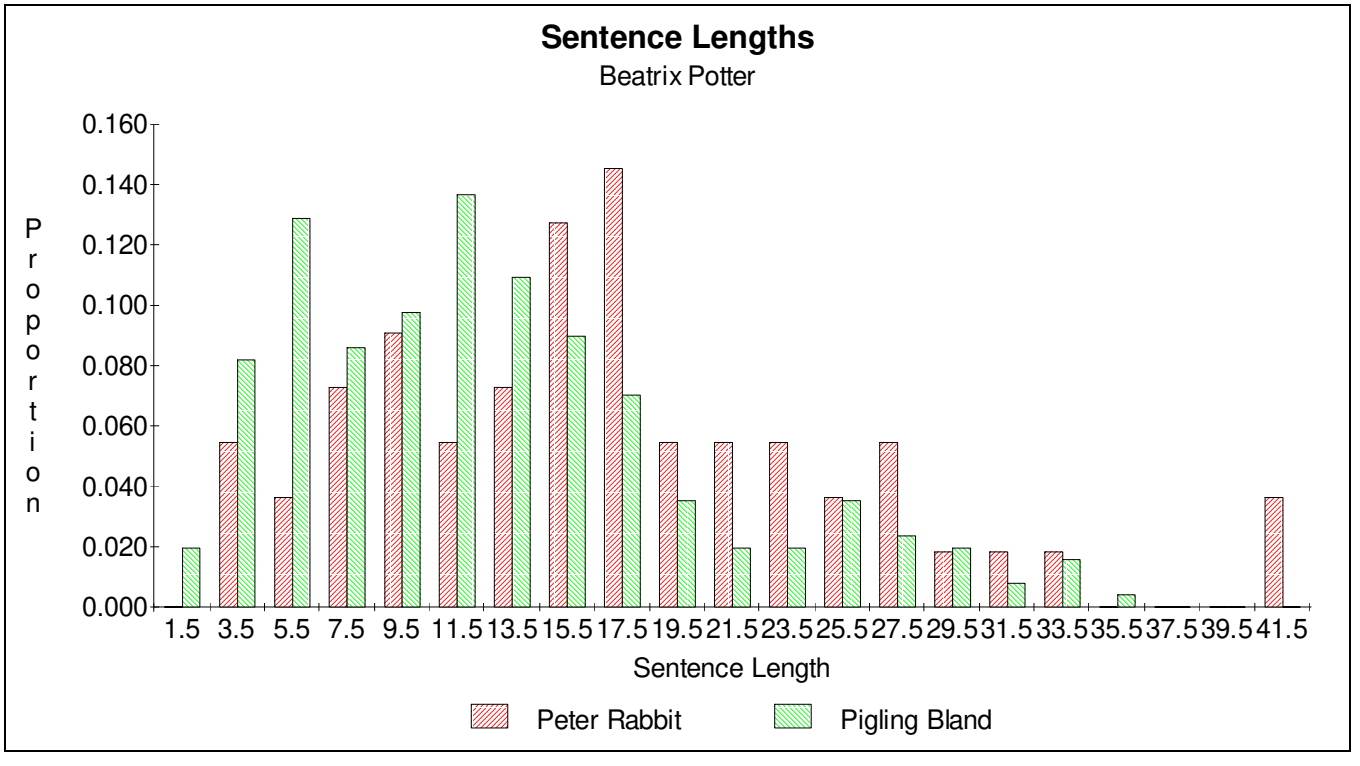
```
 1  2  3  4  5  6  7  8 9 10 11 12
12 26 56 33 26 13 13 10 9  3  2  3
```

The corresponding barchart is given in the next column. Distributions of word lengths often have a lognormal distribution. However, a plot of these frequencies against the logarithm of the word length showed no such property, possibly because of the small sample size of 206 words, and is not given here.



In the final example for an examination of the distributions of sentence lengths we shall use the complete texts of the two books *The Tale of Peter Rabbit* and *The Tale of Pigling Bland* by Beatrix Potter. An analysis of the sentence lengths for *Peter Rabbit*, shows that there are **55** sentences with minimum, mean and maximum lengths of **4**, **17.3** and **41** words, respectively. For *Pigling Bland* there are **256** sentences and the minimum, mean and maximum sentence lengths are **1**, **12.8** and **35** words, respectively. On the next page are given two charts for the distribution of sentence lengths, the first giving the frequency of sentence lengths and the second the proportions of sentence lengths for each text. Although sentence lengths often have a lognormal distribution, there was no evidence of this with these data.


Sentence Lengths
Beatrix Potter
Frequency
0
5
10
15
20
25
30
35
1.5 3.5 5.5 7.5 9.5 11.5 13.5 15.5 17.5 19.5 21.5 23.5 25.5 27.5 29.5 31.5 33.5 35.5 37.5 39.5 41.5
Length of Sentence
Peter Rabbit
Pigling Bland


Sentence Lengths
Beatrix Potter
Proportion
0.000
0.020
0.040
0.060
0.080
0.100
0.120
0.140
0.160
1.5 3.5 5.5 7.5 9.5 11.5 13.5 15.5 17.5 19.5 21.5 23.5 25.5 27.5 29.5 31.5 33.5 35.5 37.5 39.5 41.5
Sentence Length
Peter Rabbit
Pigling Bland

Appendix 5. Script file

```
NB. Arithmetic mean
w=. 2.3 5 3.5 6
am=. +/ % #
sum=. +/
tally=. #
am1=. sum % tally

NB. Geometric and harmonic means
gm=. # %: */
hm=. % @ am @ %
means=. am,gm,hm
gm1=. */ %:~ #
gm2=. ^ @ am @ ^.

NB. Variance and standard deviation
dev=. - am
ss=. +/ @ *: @ dev
var=. ss % <:@#
sd=. %: @ var

NB. Valence
mod7=. 7&|
cube=. ^&3

NB. Covariance and correlation I
x1=. 12 18 24 30 36 42 48
y1=. 5.27 5.68 6.25 7.21 8.02 8.71 8.42
sp=. +/ @ (*&dev)
cov=. sp % <:@#@]
cor=. cov % (*&sd)

NB. Median and quartiles
u=. 22 14 32 30 19 16 28 21 25 31
sort=. /:~
midindices=. (<. , >.) @ -: @ <: @ #
midvalues=. midindices { ]
q2=. am @ midvalues
q1=. q2 @ ((q2 > ]) # ])
q3=. q2 @ ((q2 < ]) # ])
Q1=. q1 @ sort
median=. Q2=. q2 @ sort
Q3=. q3 @ sort
five=. ({.,q1,q2,q3,{:) @ sort

NB. Noun rank
t=. i. 3 4
r=. i. 2 3 2

NB. Discrete frequency distributions
over=. ({.,.@;}.)@":@,
by=. ' '&;@,.@[,.]
```

```
a=. 1 1 0 3 1 3 1
fr=. +/"1 @ =   NB. fr=. #/.~
sfr=. fr @ sort
rfr=. +/"1 @ (=/)
mfr=. (i.@>:@(>./)) rfr ]
frtab=. ~. ,. fr
sfrtab=. frtab @ sort
rfrtab=. [ ,. rfr
mfrtab=. (i.@>:@(>./)) ,. mfr
c=. 1 5 2 4 6 5 1 6 2 5
DieFr=. 1 2 3 4 5 6&rfr

NB. Mode
z=. 4 6 2 1 3 6 0
mode=. (([:imax{:"1){{."1) @ frtab
imax=. (]e.>./)#i.@#@]

NB. Continuous frequency distributions
b=. 5.2 8.6 3.4 8.1 9.2 5.3 4.1 8.2 9.9 4.8
cfrpts=. -:@(1&{) + {. + 1&{ * i.@{:
cfr=. i.@{:@[ rfr [: <. (] - {.@[) % 1&{@[
cfrtab=. cfrpts@[ ,. cfr

NB. Box and open
hm1=. am &. %
gm3=. am &. ^.
BoxSort=. /:>

NB. Barcharts
bars=. #&'*' &. >
barchart=. ] ;"1 0 bars @ ({:"1) @ ]

NB. Character frequency distibutions
charfr=. [: +/"1 (=/)
charfrtab=. [ ;"0 charfr
alpha=. 'abcdefghijklmnopqrstuvwxyz'
afr=. alpha&charfr
afrtab=. alpha&charfrtab
afrnubtab=. (([: 0&~: >@{:"1) # ]) @ afrtab
safrnubtab=.(([:\:[:>({:"1)@]) { ])@afrnubtab

NB. Discrete frequency tables
y=. 0 1; 1 1; 1 2; 0 1; 0 0; 1 2; 0 0; 1 0; 0 1
RFR=. [: +/"1 ({@[) =/ ]
rfr1=. <"1 @ [ RFR {@ (<"1 @ ])
tsum=. +/ @ ,
TableData=. 3 4 $ 18 29 70 115 17 28 30 41 11 10 11 20
ExpFrTab=. (+/"1 */ +/)  % tsum
ChiSqTab=. [: tsum ([: *: ] - ExpFrTab) % ExpFrTab
Table22Data=. (2 2&$ &. >) (5 2 3 3); ( 6 1 2 4); 7 0 1 5
pr22=. (*/ @ !@(+/"1 , +/)) % */ @ (! @ (, , tsum))

NB. Stem and leaf diagrams
U=. 14 16 19 21 22 25 28 30 31 32
div=. <. @ %
```

```
classify=. (div&10 </. ])
stem=. {. @ (10&* @ div&10)
leaves=. 10&|
sl=. stem ; leaves
SLlist=. (sl &. >) @ classify
SLtable=. ;"0 @ SLlist @ sort

NB. Explicit definition I
AM=. 3 : 0
(+/ % #) y.
:
(+/x.*y.) % +/x.
)

AM1=. 3 : 0
((# $ 1:) y.)  AM y.
:
(+/x.*y.) % +/x.
)

table=. 1 : 0
[ by ] over x./
)

NB. Summary statistics
x=. 5.2 3.1 2.9 4.5
summary=. 3 : 0
r=.  'Sample size      ',5.0 ": #y.
z=. five y.
r=. r,: 'Minimum           ', 8.3 ": {. z
r=. r, 'Maximum           ',8.3": {: z
r=. r, 'Arithmetic mean   ',8.3": am y.
r=. r, 'Variance          ',8.3": var y.
r=. r, 'First quartile    ',8.3": 1{ z
r=. r, 'Median            ',8.3":2{ z
r=. r, 'Third quartile    ',8.3":3{ z
r=. r, 'Standard deviation',8.3": sd y.
r=. r, 'Geometric mean    ',8.3": gm y.
r=. r, 'Harmonic mean     ',8.3": hm y.
)

NB. Verb rank
jrank=. 1 : 0
x. b. 0
)

NB. Linear regression
mp=. +/ . *
sr=. 3 : 0
:
('b0';'b1')=. b=. y.%.X=.1,"0 x.
yest=: b0+b1*x.
sst=. +/*:y.-am y.
sse=. +/*:y.- X +/ . * b
mse=. sse%<:<:$y.
```

```
seb=. %:mse%+/*:x.-am x.
rsq=. 1-sse%sst
fmt=. 10.5&":
r=. 'Slope       ',fmt b1
r=. r,: ' S.E.       ',fmt seb
r=. r,'Intercept   ',fmt b0
r=. r,'S.E. of est.',fmt %:mse
r=. r,'Corr. sq.   ',fmt rsq
)

NB. Base and antibase
tt=. #: @ i. @ (2&^)
ott=. (/: +/"1) @ tt
ptt=. (+/"1 @ tt@] e. []) # tt@]
indices=. <"1 @ (, #: i.)

NB. Covariance and correlation II
d=. 1 4 2 2 1 3;0 6 4 3 1 5;10 17 13 14 12 15
pairs=. (>@indices @ (2: $ $@])) {"1 1 ]
covtable=. (({.@]) cov ({:@]))"2 @ (>@pairs)
cortable=. (({.@]) cor ({:@]))"2 @ (>@pairs)
ctable=. 1 : 0
(({.@]) x. ({:@]))"2 @ (>@pairs)
)
covtable1=. cov ctable
cortable1=. cor ctable

NB. Permutations and combinations
perm=. i.@! A. i.
PERM=. perm @ # { ]
comb=. |.@ptt # i.@]
COMB=. ([ comb #@]) { ]

NB. Ranks
w1=. 2.3 5 2.3 3.5 6 2.3 3.5
UNADJranks=. >: @ /:^:2
NUBranks=. (= mp UNADJranks) % [: +/"1 =
ranks=.  [: , |:@= # NUBranks
invranks=. [: ranks -

NB. Rank tests
rcor=. ranks@[ cor ranks@]
French=. 83 27 42 51 53 44 47 55 61 33
German=. 74 22 49 54 48 47 55 61 59 29
ht=. 7.9 6.5 17.8 4.2 13.2
hc=. 5.6 0.4 6.7 1.2 2.9
Wsum=. (#@[ <: i.@#@([,])) +//. ranks @ ([,])
s1=. 42 51 31 61 44 55 48
s2=. 38 53 36 52 33 49 36
Wpsum=. (* +//. ranks@|) @ sort @ ([-])

NB. Runs
runs=. [: >: [: +/  2: (~:/\) ]

NB. Explicit definition II
```

```
PrimeTest=. 3 : 0
d=. 2
while. d < y.
   do. if. (0: = d&|) y.
       do. r=. 0
           return.
       else. d=. d+1+d>2
   end.
end.
r=. 1
)

NB. Analysis of variance
x=. 3 4 $ 4 7 5 6 9 4 3 8 2 5 7 3
T=. 3 : 0
:
+/^:(+/-.x.)(/:x.)|:y.
)
S=. (+/@((*:@,)@T)) % (*/@(-.@[ # $@]))

NB. Simulation I
die=. >: @ ? @ $&6
dice=. >: @ ? @ (6&($~)@,)
TwoDice=. 2&dice
SumTwoDice=. +/ @ TwoDice
SumTwoDiceFr=. (2&+@i. 11)&rfr@SumTwoDice
EqualTwoDice=. =/ @ TwoDice
NewDice=. (-@[) <\ >: @ ? @ (6&($~)@*)
SumTwoNewDice=. +/"1 @ > @ (2&NewDice)
Coins=. [: +/ [: ? ] $ 2:
CoinTest=. +/\@(Coins"0 @ $) % ] * (>: @ i. @ [)

NB. Central limit theorem
SumDice=. +/"1 @ > @ NewDice
ExpSum=. 3.5&*
SDSum=. %:@(2.91667&*)
SU=. (] - ExpSum@[) % SDSum@[
ISU=. -: @ <. @ +: @ (0.25&+ @ ])
rng=. (-@|., 0: , ])@-:@>:@i. 6
CLTchart=. rng&(;"0) @ (bars@(rng&rfr))
CLTdemo=. CLTchart @ ISU @ ([ SU SumDice)

NB. Simulation II
urand=. (? % ]) @ ($&1e9)
coords=. urand@,&2
incircle=. [: +/ 1: >: +/"1 @ *: @ coords
PIest=. 4: * incircle % ]
stnrand=. 3 : 0
r=. i. 0 2
while. y. > {.$r
   do. whilst. S >: 1
      do. V=: <:+:urand 2
          S=: +/ *: V
   end.
r=. r, V * %: -+:(^.%])S
```

```
end.
)
stnrand1=.  4: %~  _6: + [: +/"1 urand@,&12
exprand=. [ * -@^.@urand@]

NB. De Mere's problems
P=. -. @ (] ^~ -.@[)
P1=. (%6)&P
P2=. (%36)&P

NB. Derivatives and integrals
D=. ("0)(D. 1)
W=. +:(#.|.@])"0 1|:@(0:,%.@(^/~@i)%>:@i=. i.@>:@+:)
ew=. +/@(-@(<:@(#@])*i.@[)|."0 1]{.~>:@([*<:@(#@])))
EW=. ([ ew W@]) f.
ai=. 2 : 0
ai=. +/@(x.&space * x.&[ * y.@(x.&grid))"0
)
grid=. space *i.@#@[
space=. ] % <:@#@[
I=. (4 EW 5) ai
snd=. (%:@o.@2:) (%~) ^@-@-:@*:

NB. Probability distributions
cdf=. 1 : 0
+/ @ ([ x.f. (i. @ >: @ ]))
)
G=. !@<:
B=. *&G % G@+
binomial=. (x!n) * (p^x) * (-.@p=. {:@[) ^ (n=.{.@[) - x=. ]
BINOMIAL=. binomial cdf
poisson=. ^@-@[ * ^ % !@]
POISSON=. poisson cdf
geometric=. [ * -.@[ ^ ]
GEOMETRIC=. geometric cdf
stnormal=. (%:@o.@2:) (%~) ^@-@-:@*:
STNORMAL=. 3 : 0
stnormal I y.
)
exponential=. [ * ^@(-@[*])
EXPONENTIAL=. 3 : 0
:
x.&exponential I y.
)
tn=. (G@-:@>:@[) % (G@-:@[) * (%:@o.@[)
tnx=. (>: @ (*:@] % [)) ^ (- @ -: @ >: @ [)
tdistn=. tn * tnx
TDISTN=. 3 : 0
:
x.&tdistn I y.
)
chisq=. ((]^(-:@<:@<:@[))*(^@(-@-:@]))) % ((2: ^ -:@[) * (G@-:@[))
CHISQ=. 3 : 0
:
x.&chisq I y.
```

```
)
fmn=. (([^-:@[) * (]^-:@])) % (-:@[) B (-:@])
fmnx=. (] ^ <:@-:@{.@[) * ({:@[ +{.@[ *]) ^ (-@-:@(+/@[))
fdistn=. ({.@[ fmn {:@[) * fmnx
FDISTN=. 3 : 0
:
x.&fdistn I y.
)

NB. Binomial distribution
cp=. , @ { @ (] # < @ (-. , ]) @ [)
succ=. +/"1 @ > @ (1&cp)
prob=. */"1 @ > @ cp
bc=. #/.~ @ succ  NB. bc=. i.@>: ! ]
bp=. succ@] +//. prob

NB. Poisson distribution
h=. 0 : 0
0 2 2 1 0 0 1 1 0 3 0 2 1 0 0 1 0 1 0 1
0 0 0 2 0 3 0 2 0 0 0 1 1 1 0 2 0 3 1 0
0 0 0 2 0 2 0 0 1 1 0 0 2 1 1 0 0 2 0 0
0 0 0 1 1 1 2 0 2 0 0 0 1 0 1 2 1 0 0 0
0 1 0 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1 0 0
0 0 0 0 2 1 0 0 1 0 0 1 0 1 1 1 1 1 1 0
0 0 1 0 2 0 0 1 2 0 1 1 3 1 1 1 0 3 0 0
1 0 1 0 0 0 1 0 1 1 0 0 2 0 0 2 1 0 2 0
1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 1
0 0 0 0 0 2 1 1 1 0 2 1 1 0 1 2 0 1 0 0
0 0 1 1 0 1 0 2 0 2 0 0 0 0 2 1 3 0 1 1
0 0 0 0 2 4 0 1 3 0 1 1 1 1 2 1 3 1 3 1
1 1 2 1 1 3 0 4 0 1 0 3 2 1 0 2 1 1 0 0
0 1 0 0 0 0 0 1 0 1 1 0 0 0 2 2 0 0 0 0
)
h=. ".  ,;. _2 h

NB. Locales
mean   =. +/ % #
mean_temp_=. # %: */
test=. <./,mean,>./
test_temp_=. <./,mean,>./

NB. Other software - Utilities
hfu=. '_-'&charsub
ufh=. '-_'&charsub
Jfmt=. (".&>)@chop@toLF
CLIPwrite=. wdclipwrite@hfu@clipfmt
CLIPread=. Jfmt@ufh@wdclipread
DOSwrite=. (hfu@":@[) fwrites ]
DOSread=. Jfmt@ufh@freads

NB. Other software - Statistical packages
AOVlevels=. >:@(] #: i.)@$
Flatten=. (*/@}: , {:)@$  $ ,@]
AOVdata=. , ,. Flatten @ AOVlevels
```

```
NB. Gerunds
SD=. (_:-_:)`sd @. (#>1:)
rsum=. 0:`({: + rsum@}:) @. (#>0:)
fact=. 1:`()*fact@<:) @. *

NB. Windows programming
diefr=. [: +/"1 (i.6)&(=/)@(?@$&6)
dietab=. (>:@i.@6:),. diefr
dielist=._8: <\  ((4.0 &":)@,@dietab)
DICEX=: 0 : 0
pc dicex;
xywh 20 75 34 14;cc roll button;cn "Roll";
xywh 60 75 34 14;cc clear button;cn "Clear";
xywh 100 75 34 14;cc exit button;cn "Exit";
xywh 35 10 35 55;cc groupbox button bs_groupbox;cn "Rolls ?";
xywh 40 19 34 14;cc rb0 button bs_autoradiobutton;cn "   50";
xywh 40 33 34 14;cc rb1 button bs_autoradiobutton group;cn " 100";
xywh 40 47 34 14;cc rb2 button bs_autoradiobutton group;cn " 200";
xywh 85 18 30 45;cc lb0 listbox;
pas 10 10;pcenter;pcloseok;pescclose;pscale;
rem form end;
)

dicex_init=: 3 : 0
wd DICEX
wd 'csel rb1;ccheck 1;'
wd 'pshow;'
)

dicex_button_roll=: 3 : 0
dat=: wd 'qd;'
ndx=: ({."1 dat)i. 'rb0';'rb1';'rb2'
sel=: ({:"1 ndx{dat)i.<,'1'
list=: dielist (sel{50 100 200)
wde 'csel lb0;'
wde 'ctext ',EAV,(;list,&.>CR),EAV,';csetsel -1;'
wde 'psel dicex;'
)

dicex_button_clear=: 3 : 0
wde 'csel lb0;'
list=: (_8 <\ 48$' ')
wde 'ctext ',EAV,(;list,&.>CR),EAV,';'
wde 'psel dicex;'
)

dicex_button_exit=: 3 : 0
wd 'psel dicex;pclose;'
)

dicex_run=: 3 : 0
dicex_init''
wdml 1
)
```

```
NB. Coupon collector's problem
pos=. >: @ i.
cc=. * +/ @ % @ pos
int=. ". @ (6.0&":)
cctable=. (] * i. @ []) (int @ cc @ +) table pos @ ]
prize=. >:@?
ccsim=. 3 : 0
r=. 0$0
while. y. > #~.r do.
   r=. r, Prize y.
end.
)
ccsample=. (#@ccsim)"0 @ #

NB. Appendix 2. Regression
reg=. 3 : 0
b=. (y=. ;@{: y.)%.X=.(1&,"1)@|:@(>@}:) y.
sst=. +/*:(y-am y)
ssr=. sst-sse=. +/*:(y- X +/ . * b)
F=. (msr=. ssr%k)%mse=. sse%_1+(n=. $y)-k=. <:#y.
rsq=. ssr%sst
seb=. %:(0{mse)*(<1 0)|:%.(|:X)+/ . * X
r=. 49{.'              Var.    Coeff.      S.E.          t'
r=. r, 15.0 12.5 12.5 10.2 ": (i. >:k),. b,. seb,. b%seb
r=. r, ''
r=. r, '  Source     D.F.    S.S.        M.S.         F'
r=. r, 'Regression', 5.0 12.5 12.5 10.2 ": k, ssr,msr,F
r=. r, 'Error     ', 5.0 12.5 12.5": (n-k+1), sse, mse
r=. r, 'Total     ', 5.0 12.5 ": (n-1), sst
r=. r, ''
r=. r, 'S.E. of estimate    ', 10.5":%:mse
r=. r, 'Corr. coeff. squared', 10.5": rsq
)

NB. Appendix 3. Analysis of variance
T=. 3 : 0
:
r=. +/^:(+/-.x.)(/:x.)|:y.
)
S=. (+/@((*:@,)@T)) % (*/@(-.@[ # $@]))
allS=. (>@tt@#@$) S"1 _ ]
stt=. <"1@|."1@(/: +/"1)@tt
alphabet=. 'ABCDEF'
tag=. ]#({.&alphabet)@#@]
numtag=. (({.&alphabet)@(#@$@])) e. [
alltags=. (tag &. >)@(1&{.)@stt
expand=. /:@\:@[{#@[{.]
ps=. <"1@expand"1 >@stt@(+/)
ss0=. | @ (-/) @ (>@ps@([numtag]) (+/"1@[ +//. S"1 _) ])
df=. */@<:@(numtag # $@])
term=. ss0 (] , [ , %) df
allterms=. (>@[) term"1 _ ]
AOV=. 3 : 0
(alltags #$y.) AOV y.
:
```

```
AOVtable=: x. allterms y.
Labels=. 8{."1 >x.
dfTotal=. <:*/$y.
ssTotal=. (+/*:,y.) - '' ss0 y.
dfError=. dfTotal - +/0{"1 AOVtable
if. dfError>0 do.
   ssError=. ssTotal - +/1{"1 AOVtable
   AOVtable=. AOVtable,ssError(],[,%)dfError
   Labels=. Labels,'Error'
end.
r=. Labels,"1 (5 12.5 12.5)":AOVtable
AOVtable=: AOVtable,TotalRow=. dfTotal,ssTotal
r=. r, 'Total   ',5 12.5":TotalRow
)
Model1=. 'A';'B';'AB';'C';'AC';'BC';'ABC';'D'
TestData=. 25 7 21 4 10 16 5 21 4 25 7 6
TestData=. TestData, 3 6 16 18 20 18 19 17 16 2 15 6
TestData=. TestData, 13 23 8 25 19 19 20 14 23 9 10 18
TestData=. TestData, 27 26 19 24 13 9 19 13 20 18 16 12
TestData=. 2 4 6 $ TestData

NB. Appendix 4. Statistics and literary style
TextOne=. 0 : 0
Do you love me,
or do you not?
You told me once
but I forgot.
)
SPfLF=. rplc&(LF;' ')
repPunct=. ', ; : ( ) " ?.!.'&charsub
delPunct=. rplc&('''';'') @ (rplc&('-';''))
EditText=. tolower @ deb @ delPunct @ repPunct @ SPfLF
SPfSTOP=. rplc&('.';' ')
Sentences=. (;: &. >) @ (<;._2) @ EditText
Words=. ;: @ SPfSTOP @ EditText
ItemLengths=. # & >
WordCount=. (>@{."1 ,"0 {:"1) @ nubcount
LetterCount=. [: alpha&(;"0) [: +/ (afr & >)
SL=. ItemLengths @ Sentences
WL=. ItemLengths @ Words
WC=. WordCount @ Words
LC=. LetterCount @ Words
LCnub=. (([: 0&~: >@{:"1) # ]) @ LC
SelectRows=. (>@({:"1@]) e. [) # {."1@]
OnceOnly=. BoxSort @ (1&SelectRows)
NoneOnly=. BoxSort @ (0&SelectRows)
WordList=. BoxSort @ ~. @ Words


NB. ***** END OF SCRIPT FILE *****
```