March 2003

J Tutorial

and

Statistical Package

Keith Smillie

Department of Computing Science

University of Alberta

Edmonton, Alberta T6G 2E8

smillie@cs.ualberta.ca

The array language **J** is introduced with examples chosen mostly from elementary statistics. The topics include frequency tabulations, measures of central tendency and dispersion, correlation and regression, analysis of variance, probability distributions, random variables, chi-square, and nonparametric methods. The principal **J** verbs are summarized in a format which may serve as a self-contained statistical package. Both the **J** plotting facilities and the Form Editor for the construction of Windows forms are used for the presentation of some of the results. The paper and the corresponding script files are available on the Web at

http://www.cs.ualberta.ca/~smillie/Jpage/Jpage.html.

Table of Contents

J Tutorial and Statistical Package

```
                3
              141
            59265
          3589793
        238462643
      38327950288
    4197169399375
  105820974944592
 30781640628620899
8628034825342117067
```

The quantity of meaning compressed into small space by algebraic signs, is another circumstance that facilitates the reasonings we are accustomed to carry on by their aid.

Charles Babbage, 1827.

By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race.

Alfred North Whitehead, 1911.

Progress in mastering mathematics depends on reducing familiar laborious processes to automatic mental routines which no longer require conscious thought, this then creates mental space to allow the learner to concentrate on new, unfamiliar ideas.

London Mathematical Society, Royal Statistical Society, Institute of Mathematics and its Applications, 1995.

Introduction

J is a general-purpose language that may be used both as a programming language and also as a simple, executable notation for teaching a wide range of subjects. It is available for the Windows, WindowsCE, Mac, UNIX and Linux operating systems. The core language is identical in all versions. **J** can be integrated with other systems giving, for example, computational support to most graphics and spreadsheet packages.

The principles underlying the design of **J** have been simplicity, brevity and generality. The data objects in **J** are scalars, one-dimensional lists, two-dimensional tables, and in general rectangular arrays of arbitrary dimension. In addition to the usual elementary arithmetical operations of addition,

subtraction, multiplication, division and exponentiation, there is a large number of additional operations which are defined for arrays as well as for individual numbers.

J was developed by Kenneth Iverson as a modern dialect of APL, a language which he proposed and which was first implemented in the early 1960s. **J** provides the simplicity and generality of APL, may be printed on most printers since it uses the standard ASCII character set, and takes full advantage of recent developments in computer technology.

J is available in several releases, the latest being J5.01a. Most of the documentation is available online or may be downloaded in either PDF or HTML format. Further information including details of the pricing of J5.01a is available at the Iverson Software Inc. website at *www.jsoftware.com* which also contains links to related sites.

All of the **J** development for the present paper has been done using J4.06 which is almost identical in language to J5.01a but which lacks the extensive enhancements to the environment of the latest release.

Appendix 1 gives a short two-page introduction to **J** which might be read now by the person unfamiliar with the language. Persons wishing to continue their study of **J** should consult ***J*** *Introduction and Dictionary* (Iverson Software Inc., 1998) which gives a complete description of the language. It is an indispensible reference in learning and using **J**.

Appendix 2 lists the principal verbs developed in this paper, except those appearing in the next section and in "Further examples", together with a brief statement of their syntax and forms a **J** Statistical Package which may be used independently of the present paper.

Appendix 3 gives all of the figures referred to in the text that have been prepared using the plotting facilities in **J,** and Appendix 4 shows the Windows forms that have been constructed for a few of the examples.

Appendix 5 gives a summary of the **J** vocabulary and has been taken from the **J** online Help with a few modifications.

A simple example

We shall introduce **J** with a simple example in which we are given the following list of groceries purchased and we are required to find the total cost:

1 bag oranges @ $5.99

3 grapefruit @ 0.59

2 packages cream cheese @ 3.49

1 bag grapes @ 4.39

We may represent the prices by the list `Prices` in which each price occurs once for each item purchased:

```
Prices=: 0.59 5.99 0.59 0.59 3.49 3.49 4.39
```

Then the total cost is given by the sum `+/Prices` or `19.13`, where the verb `+/` is derived from the verb *plus* `+` and the adverb *insert* `/` and gives the sum of the items in its list argument. Thus `+/Prices` is equivalent to the sum

```
0.59 + 5.99 + 0.59 + 0.59 + 3.49 + 3.49 + 4.39  .
```

We may define the verb

```
sum=: +/
```

so that `sum Prices` is `19.13`.

We may also represent the prices by one list giving the price of each item

```
PriceList=: 0.59 5.99 3.49 4.39
```

and a second list

```
Qty=: 3 1 2 1
```

giving the quantity of each item purchased. Thus the total amount spent on each different good is

```
Qty * PriceList  ,
```

where `*` is the verb *times*, or

```
1.77 5.99 6.98 4.39 ,
```

and the total amount spent is

```
+/ Qty * PriceList
```

or `19.13`. We note that `PriceList` is obtained from `Prices` by selecting its unique items, and

```
PriceList=: ~Prices
```

where `~.` is the verb *nub*.

The calculations given in the last paragraph may be performed with the defined verb `wsum`, for "weighted sum", defined as

```
wsum=: +/ @: *
```

so that

```
 Qty wsum PriceList
```

is `19.13`. The conjunction *at* `@:`, which may be interpreted as "after", is used so that the sum is applied after the item-by-item products have been calculated. If a left argument of `1` is used with the verb `wsum`, it is applied as a multiplier to each item of the list right argument to give an unweighted sum of these items. Thus the expression `1 wsum Prices` is `19.13`.

The two primitive verbs *plus* `+` and *times* `*` take two arguments, one on the left and one on the right, as, for example, `3 + 5` which is `8` and `3 * 8` which is `24`. Such verbs are said to be *dyadic*. On the other hand, the verb *nub* `~.` with a single argument on the right is *monadic*, and, for example, `~. 3 2 2 7 2` is `3 2 7`. Defined verbs may be either monadic or dyadic, and we have `sum` and `wsum` as examples. These two verbs may be replaced with one *ambivalent* verb which may be used either monadically or dyadically. If `Sum` is such a verb, then `Sum Prices` is `19.13` and so is `Qty Sum PriceList`. The definition of ambivalent verbs will be discussed later in the paper.

The same symbol may represent either a monadic or dyadic function. It is important to distinguish

between the name of a symbol and the name or names of whatever it represents. For example, the symbol *percent* `%` represents the dyadic verb *divide* and the monadic verb *reciprocal*, and `4 % 5` is `0.8`, and `% 8` is `0.125`. Both forms may appear in the same expression, and, for example, `% 4 % 5` is `1.25`, and is read as "the reciprocal of (`4` divided by `5`).

An alternative definition for the verb `wsum` provides an opportunity to introduce the very important concept of a *fork* which is defined as an uninterrupted sequence of three verbs. In general, if `f`, `g` and `h` are verbs, then the monadic fork `(f g h)y` has the value `(f y) g (h y)`. Before we give a simple example of a fork we shall introduce three verbs. The dyadic verbs *lesser of* `<.` and *larger of* `>.` give the lesser and larger values, respectively, of their arguments, and the dyadic verb *append* `,` joins its arguments. For example, `3.5 <. 6` is `3.5`, `3.5 >. 6` is `6` and `1 2 3 , 4 5` is `1 2 3 4 5`. Now we may define the verb

```
minmax=: <./ , >./ ,
```

where the sequence `<./ , >./` is a fork, which gives the minimum and maximum items of its argument, and `minmax Prices` is `0.59 5.99` and so is `minmax PriceList`.

Now an alternative definition of `wsum` is

```
wsum=: [: +/ * .
```

Here the first verb in the fork is the monadic verb *cap* `[:` which caps the left branch of the fork so that the verb `+/` is applied to the result of the dyadic verb `*` which gives the item-by-item products of the lists used as arguments. Which definition of `wsum`, or indeed of two comparable alternative definitions of any verb, is preferred is a matter of taste although the use of `[:`, especially with extended sequences of verbs, often results in fewer pairs of parentheses in the final expression.

Discrete frequencies

In this section we shall consider observations which are limited to non-negative integers and find the frequencies over either an arbitrary specified range or the nub which is the list of unique items. As an example we shall use the list

```
D=: 5 6 4 2 6 5 5 4 1 4 5 2
```

representing the (simulated) results of throwing a die twelve times, and the list

```
r=: 1 2 3 4 5 6
```

giving the range of possible values that can result on each throw. In this example the nub is the list

```
5 6 4 2 1
```

since a `3` did not occur on any of the throws

We shall need the dyadic 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=: 1 2 3
```

and

```
q=: 1 2 3 4 5  ,
```

then

```
(p+/q) ; (p-/q) ; p*/q
```

is the table

```
┌─────────┬─────────────┬─────────────┐
│2 3 4 5 6│0 _1 _2 _3 _4│1 2 3  4  5  │
│3 4 5 6 7│1  0 _1 _2 _3│2 4 6  8 10  │
│4 5 6 7 8│2  1  0 _1 _2│3 6 9 12 15  │
└─────────┴─────────────┴─────────────┘
```

giving very small upper left portions of the integer addition, subtraction and multiplication tables. The dyadic verb *link* `;` appends its two arguments with boxing if necessary.

Then the expression `r=/D`, where `=` is the dyadic verb *equal*, gives the distribution table

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

where the first row shows that a `1` occurred on the ninth throw, the second row that a `2` occurred on the fourth and twelfth throws, etc. The row sums given by

```
+/"1 r=/D
```

are `1 2 0 3 4 3` and give the required frequencies. The row summation `+/"1` shows the use of the conjunction *rank* `"` to give the row sums rather the column sums.

The calculations in the last paragraph may be combined in the dyadic verb

```
fr=: +/"1 @ (=/)
```

whose left argument gives the range of data and right argument the list of data so that

```
r fr D
```

is the required list of frequencies `1 2 0 3 4 3`. The conjunction `@` *atop*, which is similar to *at* introduced earlier and which also may be interpreted as "after", is required so that the row sums are calculated after the distribution table has been found. An alternative definition is

```
fr=: [: +/"1 =/ .
```

A two-column frequency table with the range in the first column and the corresponding frequencies in the second column is given by

```
frtab=: [  ,.  fr  ,
```

so that `r frtab D` is the table

```
1 1
2 2
3 0
4 3
5 4
6 2 .
```

The dyadic verb *left* `[` gives its left argument, and the dyadic verb *stitch* `,.` joins its arguments in a table. We also mention here the dyadic verb *right* `]` which gives its right argument, and we have that

`1 2 3 [ 4 5` is `1 2 3` and `1 2 3 ] 4 5` is `4 5`. We note the use of a dyadic fork in the definition of `frtab`. In general, the dyadic fork x (f g h) y is (x f y) g (x h y).

A frequency table with two rows rather than two columns may be given simply with the monadic verb *transpose* `|:` which interchanges the rows and columns of its argument so that `|: r frtab D` is

```
1 2 3 4 5 6
1 2 0 3 4 2 .
```

Instead of finding the frequencies over an arbitrary range of values, we may wish to limit the range to only those distinct values which occur in the data. We use the monadic verb *nub* **~.** introduced in the last section, and we have that the nub of the list `D` is `~.D` or `5 6 4 2 1`. 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, `= D` is

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

This table shows that a `5` occurs in the first, sixth, seventh and eleventh throws, a `6` on the second and fifth throws, etc. Since the row sums of the distribution table give the frequency of occurrence of the items of the nub, we define

```
nubfr=: +/"1 @ =
```

to give the list of frequencies so that `nubfr D` is `4 2 3 2 1` which means that `D` has four `5`s, two `6`s, etc.

A frequency table for the nub is given by

```
nubfrtab=:  ~. ,. nubfr  .
```

Therefore, for the dice data `D`, we have that

```
(nubfrtab D) ; nubfrtab sort D
```

or

```
(nubfrtab ; nubfrtab @ sort)D
```

is

```
┌───┬───┐
│5 4│1 1│
│6 2│2 2│
│4 3│4 3│
│2 2│5 4│
│1 1│6 2│
└───┴───┘
```

where in the first table the items of the nub occur in the order in which they occur in `D` and in the second table they occur in sorted order. We have used the "utility verb" `sort`, the details of which need not concern us here, for sorting the items of its list argument in nondecreasing order.

For a simpler definition of `nubfr` we introduce the dyadic adverb *key* `/.` which groups 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. For example, for the dice data

```
D=: 5 6 4 2 6 5 5 4 1 4 5 2
```

which has been given previously the expression `2|D` is the list

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

where the `0`s and `1`s correspond to even and odd numbers, respectively, showing on the corresponding throws. Then

```
(2|D) </. D
```

is the two-item list

```
┌─────────┬─────────┐
│5 5 5 1 5│6 4 2 6 4 4 2│
└─────────┴─────────┘
```

where the first item gives the even faces and the second item gives the odd faces. The expression `(2|D) #/. D`, where `#` is the monadic verb *tally*, is the list `5 7` of the number of even and odd faces. Two more examples are `D </.D` or

```
┌─────────┬───┬─────┬─┐
│5 5 5 5  │6 6│4 4 4│2 2│1│
└─────────┴───┴─────┴───┘
```

which groups the items of the nub and `D #/. D` or `4 2 3 2 1` which counts them.

In the examples of the previous paragraph we have introduced three additional primitive verbs. The dyadic verb *residue* `|` gives the remainder when the left argument divides the right, the monadic verb *box* `<` boxes its argument, and the monadic verb *tally* `#` gives the number of items in its argument. Finally, we introduce the monadic adverb *reflex* `~` which gives its verb argument equal left and right noun arguments, and, for example, `*~ 1 2 3` is equivalent to `1 2 3 * 1 2 3` or `1 4 9`. Then the final example in the previous paragraph may be written as `#/.~ D`.

Finally we give the alternative definition of the verb for nub frequencies as

```
nubfr=: #/.~ .
```

Grouped frequencies

The following data from Sprent (1977) give the sentence length for the first page of the 1973 Presidential Address of the Royal Statistical Society:

```
SentenceLength=: 11 31 45 31 12 31 39 16 21 31 36 28 31 39 31 22 33
```

The transposed nub frequency table is given by the expression

```
|: nubfrtab sort SentenceLength
```

and has the value

```
11 12 16 21 22 28 31 33 36 39 45
1  1  1  1  1  1  6  1  1  2  1
```

which may not be too helpful in getting an overall view of the distribution of sentence lengths. Therefore we shall give in this section verbs for finding the frequencies of an arbitrary list of data when grouped in intervals of equal width where the end points of the class intervals are given, and, for example, the list `2 5 8 11` would indicate that the intervals are `2` to `5`, `5` to `8`, and `8` to `11`.

The verbs for classified data are not as simple as those for discrete data which have just been discussed. They require the two utility verbs `io` which may be considered to be a generalization of the dyadic verb *index of* `i.` and `midpts` which gives the two-term moving averages of its list argument. As an example of the use of the first verb we have

```
1 3 5 7 9 11 io 5.2 8.6 3.4
```

which is the list `2 3 1`, showing that with zero-origin indexing `5.2` is in the third interval (`5`, `7`), `8.6` is in the fourth interval (`7`, `9`), and `3.4` is in the second interval (`3`, `5`). An example of the use of the second verb is

```
midpts 1 3 5 7 9 11
```

which is the list `2 4 6 8 10`. We shall also use the verb `ap` for arithmetic progressions, where, for example, the expression `ap 1 2 6` gives the list `1 3 5 7 9 11` used in the last example.

The verbs for a frequency list and frequency table for classified data are given by

```
grfr=: i.@(<:@$@[) fr io
```

and

```
grfrtab=: midpts@[,.grfr ,
```

respectively, the details of which are left to the interested reader. We shall use these verbs with the data on sentence lengths to find the number of sentences in each of the intervals (`10`,`15`), (`15`,`20`), ..., (`40`,`45`). The required list of intervals is given by

```
c=: ap 10 5 8
```

which has the value

```
10 15 20 25 30 35 40 45  .
```

Therefore, the list of grouped frequencies is

```
c grfr SentenceLength
```

or

```
2 1 2 1 7 3 1  ,
```

and the frequency table is given by

```
c grfrtab SentenceLength
```

which is

```
12.5 2
17.5 1
22.5 2
27.5 1
32.5 7
37.5 3
42.5 1  .
```

J programs may be incorporated into Windows forms designed by the user so that the programs may be used without any knowledge of the details of the computations or their implementation. These forms may be used within the **J** programming environment or independently of it. As an example the form given as Figure 4.1 in Appendix 4 computes summary statistics and also the frequency distribution of a set of

data, which may be either discrete or continuous, given the left-hand end of the first class, the class width and the number of classes. It is shown here for an analysis of the data on sentence lengths. With the dice data given in a previous section the parameters would be `0.5`, `1` and `6`.

Barcharts

Instead of displaying the numerical values of the frequencies, we may represent the frequencies graphically with each frequency being represented by a bar of the appropriate length. The construction of the barchart requires several functions which will be introduced in the following paragraph.

The dyadic verb *copy* `#` copies items from its right argument according to the items of its left argument, and, for example, the expression

```
0 1 0 1 0 1 # 1 2 3 4 5 6
```

is `2 4 6`, and

```
(i. 4) # i. 4
```

or

```
0 1 2 3 # 0 1 2 3
```

is equal to

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

The monadic conjunction *bond* `&` may be used to bind an argument to a dyadic verb. For example, the verb

```
Triple=: 3&* ,
```

or alternatively

```
Triple=: *&3  ,
```

triples its argument, and `Triple 12` is `36`. The utility adverb `EACH` performs the operation specified by the left argument on each of the items specified by the right argument without preserving the boxing. For example,

```
1;1 2;1 2 3;1 2 3 4
```

is the list

```
┌─┬───┬─────┬───────┐
│1│1 2│1 2 3│1 2 3 4│
└─┴───┴─────┴───────┘
```

,

and

```
+/ EACH 1;1 2;1 2 3;1 2 3 4
```

is

```
1 3 6 10  .
```

The above three functions are used in the expression `#&'*' EACH` which replicates the symbol `*` a specified number of times, and, for example,

```
(#&'*' EACH) 1 2 3
```

is the array

```
*
**
***  .
```

We introduce for subsequent use the adverb `each` similar to `EACH` but which preserves boxing, and, for example,

```
+/ each 1;1 2;1 2 3;1 2 3 4
```

is

```
┌─┬─┬─┬──┐
│1│3│6│10│
└─┴─┴─┴──┘   .
```

The verb

```
barchart=: ([: ": [: ,. [) ,"1 [: ' '&,. [: bars ]
```

where

```
bars=: #&'*' EACH @ fr
```

follows from the above discussion. We note the use of the monadic verb *default format* `":` which converts its argument to a character array. The barchart for the dice data

```
D=: 5 6 4 2 6 5 5 4 1 4 5 2
```

considered previously is given by the expression

```
(~. sort D) barchart nubfr sort D
```

or more simply by

```
(~. barchart nubfr) sort D ,
```

and is

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

Barcharts may be drawn very simply with the plotting facilities in **J** which we shall introduce here so that we may use them in the remainder of this paper. Graphics may also be prepared by using the graphics facilities of some software package such as MS Works. We shall not however consider this second alternative here.

The **J** plotting package Plot requires utilities made available by the system command

```
load 'plot' ,
```

which provides the verb `plot` which will handle most simple plots, and also the verb `pd` which handles all calls to Plot and may be used to give greater control over the graphics output. The syntax of `plot` is

```
opt plot data
```

where `data` gives the data to be plotted and `opt` gives the plotting options. We shall use some dice data to give two simple examples of the use of Plot.

A bar chart of the frequencies for the dice data `D` is given by the expression

```
'bar' plot r fr D ,
```

where `r=: 1 2 3 4 5 6` is the range as before, and is shown in Figure 3.1 in Appendix 3. Now consider throwing the die another `12` times and obtaining the results

```
DD=: 1 5 3 4 2 1 5 5 6 3 4 5   .
```

A barchart showing the frequencies for both sets of data is given by

```
'bar' plot (r fr D),: r fr DD
```

and is shown in Figure 3.2. We note the use of the dyadic verb *laminate* `,:` and the expression

```
(r fr D),: r fr DD
```

gives the table of frequencies

```
1 2 0 3 4 2
2 1 2 2 4 1
```

from its left and right list arguments.

Stem-and-leaf diagrams

In a stem-and-leaf diagram the data are grouped by the integer quotient when divided by `10`, i.e., all items between `0` and `9` which have an integer quotient of `0` are grouped together, all items between `10` and `19` which have an integer quotient of `1` are grouped together, etc. The integer quotient when multiplied by `10` is termed the "stem" and the remainders are termed "leaves". Furthermore, each stem is displayed once for the corresponding leaves. For example, the three items `15`, `12` and `18` have a stem of `10` and leaves of `5`, `2` and `8`, and would be displayed in a stem-and-leaf diagram as

```
┌──┬─────┐
│10│2 5 8│
└──┴─────┘ .
```

The stem and leaf of a non-negative integer are given by the verbs

```
stem=: 10&* @ <. @ %&10
```

and

```
leaf=: 10&| .
```

The monadic verb *floor* `<.` gives the largest integer less than or equal to its argument. Therefore, the diagram at the end of the last paragraph is given by the expression

```
(~.@stem;leaf) 12 15 18 .
```

The two verbs of the last paragraph may be used to give the very simple verb

```
SLdiag=: ~.@stem ;"0 stem </. leaf .
```

for a stem-and-leaf diagram where `/.` is the adverb *key* introduced in a previous section..

For the data on sentence lengths used previously we have

```
sort SentenceLength
```

to be the list

```
11 12 16 21 22 28 31 31 31 31 31 31 33 36 39 39 45
```

so that the stem-and-leaf diagram is the following:

```
SLdiag sort SentenceLength
```

10	1 2 6
20	1 2 8
30	1 1 1 1 1 1 3 6 9 9
40	5

A slight modification to the verb for the stem-and-leaf diagram will give the verb

```
SLfrtab=: ~.@stem ,. stem #/. leaf
```

for a "stem-and-leaf frequency table", and we have that

```
SLfrtab sort SentenceLength
```

is the table

```
10  3
20  3
30 10
40  1  .
```

The **J** plot package may be used to display this frequency table as follows:

```
pd 'new'
pd 'type bar'
pd 'ytic 1 0'
pd 'title Stem & Leaf Table'
pd 'xlabel "10" "20" "30" "40"'
pd 3 3 10 1
pd 'show'
```

The resulting graph is shown in Figure 3.3.

Means

The arithmetic mean is defined as the sum of the observations divided by the number of observations. The geometric mean of *n* observations is defined as the *n*th root of the product of the observations. The harmonic mean is the reciprocal of the arithmetic mean of the reciprocal of the observations.

If we have a list of observations

```
w=: 62.3 58.5 44.6 50.3 ,
```

then the arithmetic mean is

```
+/w % #w
```

which has the value `53.925`. The expression for the arithmetic mean may be written as

```
(+/ % #) w ,
```

and we may define the verb

```
am=: +/ % #
```

and `am w` is `53.925`.

The geometric mean may be defined as

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

and `gm w` is equal to `53.4733`. This is another example of a fork and `gm w` is equivalent to

```
(#w) %: */w  .
```

In this definition we have the dyadic verb *root* `%:` which gives an arbitrary root, and, for example, `3 %: 64 is 4` and `2 %: 64` is `8`. Since the monadic form of `%:` is *square root*, this last expression may also be written as `%: 64`.

Similarities in the definitions of the arithmetic and geometric means may be seen from the two definitions

```
am=: +/ % #
```

and

```
am=: # %~ +/
```

for the arithmetic mean, and

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

and

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

for the geometric mean. We note the dyadic adverb *passive* `~` which interchanges the left and right arguments of its associated verb. For example, `5 %~ 12` is `2.4`, and we may define the verb

```
into=: %~
```

and `5 into 12` is `2.4`.

The harmonic mean may be defined as either

```
hm=: % @ am @: %
```

or

```
hm=: [: % [: am %  ,
```

and we have that `hm w` is `53.0169`. In the first definition we note the use of the conjunctions `@` and `@:` which have been introduced previously.

The arithmetic, geometric and harmonic means may be calculated by the single verb

```
allmeans=: am,gm,hm
```

so that `allmeans w` has the value `53.925 53.4733 53.0169`.

The observations in the four-item list `w` represent the yield in bushels per acre resulting from four replications with one variety of oats when the seeds receive one particular treatment before planting and have been taken from Steel and Torrie (1960). In all there were three different seed treatments, and a second variety of oats which we shall ignore in this example but will include in a later one. The data for the three treatments may be represented by the three-by-four array `W` which has the value

```
62.3 58.5 44.6 50.3
63.4 50.4   45 46.7
64.5 46.1 62.6 50.3
```

which may be formed by the statement

```
W=: 3 4 $ 62.3 58.5 44.6 50.3 63.4 50.4 45 46.7 64.5 46.1 62.6 50.3
```

where the dyadic verb *shape* `$` shapes the right argument according to the left argument. The arithmetic means of the replications (columns) are given simply by the expression `am W` which has the value

```
63.4 51.6667 50.7333 49.1 .
```

The treatment (row) means are given by `am"1 W` which is equal to

```
53.925 51.375 55.875
```

where the conjunction rank `"` applies the verb `am` to the rows of `W`. Verbs for row and columns may be defined for convenience if desired, and, for example, if

```
amrows=: am"1
```

and

```
amcols=: am ,
```

then `amrows W` is

```
53.925 51.375 55.875
```

and `amcols W` is

```
63.4 51.6667 50.7333 49.1 .
```

For a further example of the calculation of the means defined in this section we shall use the following data giving the 1988 per capita income for the fifty American states taken from Sternstein (1994) and given by the 50-item list `Income` displayed here as a table with `5` rows and `10` columns:

```
   5 10$Income
126 195 149 122 189 164 228 177 165 150
169 127 176 147 148 159 128 122 150 193
207 164 168 110 155 127 152 174 190 219
125 193 141 127 155 133 150 162 168 128
125 137 146 120 154 176 166 117 154 137
```

The three means are simply calculated as follows:

```
   am Income
155.28
   gm Income
153.009
   hm Income
150.83
   allmeans Income
```

```
155.28 153.009 150.83
```

We have already introduced the rank conjunction " to give the row sums of a two-dimensional array. Two examples are given in the verb

```
fr=: [: +/"1 =/
```

for discrete frequencies and the verb

```
amrows=: am"1
```

of this section for the row means. Before proceeding further with statistical concerns we should have a closer look at the very important concept of the rank of an array.

A variable, or noun, may be a single atom or a list or a table or an array of higher dimension. The number of axes in an array is its *rank*, and atoms, lists and tables are of rank 0, 1 and 2, respectively, while higher-dimensional arrays have ranks greater than 2. The shape of an array is given by the monadic verb *shape of* `$` which gives the length of each axis of its argument, so that the number of items in the shape is the rank. Therefore, for any array `a` the shape is `$a` and its rank is `#$a`. The list

```
w=: 62.3 58.5 44.6 50.3
```

introduced earlier in the section has shape `$w` or `4` and rank `#$w` or `1`. Also the table `W` whose value is

```
62.3  58.5  44.6  50.3
63.4  50.4  45.0  46.7
64.5  46.1  62.6  50.3
```

has `3` rows and `4` columns and its shape is `$W` or `3 4` and its rank is `#$W` or `2`. As a third example the three-dimensional array

```
T=: i. 3 4 ,
```

where `i.` is the monadic verb *integers* which gives non-negative integers, has the value

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

12 13 14 15
16 17 18 19
20 21 22 23
```

and `$T` is `2 3 4` and `#$T` is `3`.

The last *k* axes of a variable determine a *rank k-cell* or more simply a *k-cell*. For example, for the table `T` the 0-cells are the atoms `0, 1, 2 ...`, the 1-cells are the rows `0 1 2 3`, `4 5 6 7`, ..., the 2-cells are the two 3-by-4 tables, and the 3-cell is the array `T`. For an array of rank r the (r-1)-cells are known as the *(major) items*. The rank conjunction " is used to apply a primitive or defined verb 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`.

One example of the use of the rank conjunction was for row and column means. The row means were given by

```
amrows=: am"1
```

and the column means by

```
amcols=: am
```

which could also be defined as

```
amcols=: am"2  .
```

Finally we could define the verb

```
amitems=: am"0
```

to give the means of the items of it argument which are the items themselves although there would be no purpose in doing so. As another example for the array `T` we have `+/"1 T` equal to

```
 6 22 38
54 70 86 ,
```

`+/"2 T` equal to

```
12 15 18 21
48 51 54 57
```

and `+/"3 T` equal to

```
12 14 16 18
20 22 24 26
28 30 32 34 .
```

Other averages

In this section we shall give **J** verbs for two other commonly used measures of central tendency, the median and the mode. The median is the middle of the sorted observations and the mode is the most frequently occurring observation or observations.

The median of a list of observations is defined as the middle observation when the observations 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, for the list

```
u=: 22 14 32 30 19 16 28 21 25 31
```

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`.

For a list with an odd number of items, `7`, say, the index of the middle item is simply the number of items decremented by `1` and then divided by `2`, or `-:<:7` which is equal to `3`, where `<:` is the monadic verb *decrement* which subtracts `1` from its argument and `-:` is the monadic verb *halve* which halves its argument. (Note that indexing starts with `0` so that the indices for a seven-item list are `0`, `1`, `2`, `3`, `4`, `5` and `6`.) However, for a list with an even number of items, `8`, say, a similar calculation would give the value `3.5` which is midway between the required indices `3` and `4`. These two calculations may be combined in the expression

```
(<.,>.) -: <: ,
```

where the monadic verb *ceiling* `>.` gives the smallest integer greater than or equal to its argument. For an argument of `7` this expression is `3 3` and for an argument of `8` is `3 4`. Thus, in either case the

corresponding items need only be selected and averaged to give the median. Therefore, we may define the verb

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

and, for example, `midindices x` is equal to `3 3` if `x` is a seven-item list and is equal to `3 4` if `x` is an eight-item list. A verb for the median is

```
median=: [: am midindices { sort
```

where `{` is the dyadic verb *from* which selects from its right argument those items whose indices are given by the left argument, and `median u` is `23.5`.

The mode is defined as that item which occurs most frequently, and, for example, for the list

```
D=: 5 6 4 2 6 5 5 4 1 4 5 2
```

of dice data the mode is `5` since this value occurs four times. The mode may be found very simply using one of the frequency verbs defined previously.

First we shall define the utility verb

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

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

```
imax 7 10 4 3 10 0
```

is the list `1 4` of indices of the maximum item `10`. The dyadic verb *member* `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,

```
7 10 4 3 10 0 e. 10
```

is the list `0 1 0 0 1 0`.

The verb `mode` may now be defined as

```
mode=: imax@nubfr { ~.
```

and `mode D` is `5`. 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`.

The median and the mode for the income data introduced in the last section are `median Income` which is `153` and `mode Income` which is `150 127`, where the sorted income data are given by

```
5 10 $ sort Income
```

which is the table

```
110 117 120 122 122 125 125 126 127 127
127 128 128 133 137 137 141 146 147 148
149 150 150 150 152 154 154 155 155 159
162 164 164 165 166 168 168 169 174 176
176 177 189 190 193 193 195 207 219 228  .
```

Variability

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 standard deviation is the square root of the variance. 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 second quartile is thus the median.

If we have the list

```
w=: 62.3 58.5 44.6 50.3
```

which was introduced in a previous section, then deviations from the mean which is equal to `53.925` are given by

```
w - am w
```

which is equal to

```
8.375 4.575 _9.325 _3.625 .
```

The expression for deviations from the mean may be represented more simply as `(- am) w`, where the two-verb sequence in parentheses is known as a *hook*. Thus we may define the verb

```
dev=: - am
```

so that `dev w` is the above list of deviations. The verb

```
ss=: [: +/ [: *: dev ,
```

where `*:` is the monadic verb *square*, gives the sum of squares of deviations from mean and `ss w` is `191.167`. The variance is

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

and the standard deviation is

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

Thus `var w` is `63.7225` and `sd w` is `7.98264`.

We noted above the use of a monadic hook in the definition of the verb `dev`. In general, for verbs `g` and `h` the monadic hook `(g h) y` is equal to `y g (h y)`. Dyadic hooks may also be defined. We may alternatively define the verb for deviations from the mean as

```
dev=: - +/ % #
```

which is a fork followed by a hook.

The verbs for the arithmetic mean, variance and standard deviation may be defined using primitive verbs only and are given here for the interested reader:

```
am=: +/ % #
dev=: - +/ % #
ss=: [: +/ [: *: [ - +/ % #
var=: (# - 1:) %~ [: +/ [: *: ] - +/ % #
sd=: [: %: ([: <: #) %~ [: +/ [: *: ] - +/ % #
```

Since the second quartile is simply the median, we may define it by the synonym

```
Q2=: median .
```

The first quartile is then the median of all those items in the original list which are less than the median and thus may be defined as

```
Q1=: [: Q2 ] #~ Q2 > ] .
```

Similarly the third quartile is the median of all items greater than the median and may be defined as

```
Q3=: [: Q2 ] #~ Q2 < ] .
```

If we recall the list

```
u=: 22 14 32 30 19 16 28 21 25 31
```

of the previous section, we have that `sort u` is the list

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

and

```
(Q1,Q2,Q3) u
```

is the three-item list `19 23.5 30` giving the three quartiles.

Summaries

One brief summary of a list of observations is the five-statistic summary consisting of the minimum observation, first, second and third quartiles, and maximum observation of its list argument. This may be defined by

```
five=: <./,Q1,Q2,Q3,>./
```

and, for example, `five Income` is

```
110 128 153 169 228 .
```

A more complete summary is provided by the monadic verb `summary` which provides a number of statistics with suitable labels, and `summary Income` gives the following table:

```
   summary Income
Sample size         50
Minimum            110.000
Maximum            228.000
Arithmetic mean    155.280
Variance           748.369
Standard deviation  27.356
First quartile     128.000
Median             153.000
Third quartile     169.000
Geometric mean     153.009
Harmonic mean      150.830
```

The verb `summary` has been defined explicitly with a given argument and with a definition which extends over several lines. The first three lines and the last line of the verb are as follows with the omitted

lines being indicated by an ellipsis given as a comment:

```
summary=: 3 : 0
r=.  'Sample size      ',5.0 ": #y.
r=. r,: 'Minimum           ', 8.3 ": <./y.
r=. r, 'Maximum           ',8.3": >./y.
NB. ...
r=. r, 'Harmonic mean    ',8.3": hm y.
)
```

We note the use of the dyadic verb *laminate* `,:` for joining arrays of different shapes, and the dyadic verb *format* `":` whose left argument specifies the width and number of decimal places displayed in the right argument and which gives a literal result. Note that the right argument of an explicit verb is represented by `y.`.

To illustrate some of the main features of explicit definition we shall define the following four very simple verbs `f1, f2, f3a` and `f3b`:

```
f1=: 3 : 0      f2=: 3 : 0      f3a=: 3 : 0     f3b=: 3 : 0
% y.           :               % y.           1 f3b y.
)              x. % y.         :              :
               )               x. % y.        x. % y.
                               )              )
```

The monadic verb `f1` gives the reciprocal so that, for example, `f1 2.5` is `0.4`, and the dyadic verb `f2` gives the quotient of its two arguments so that `15 f2 6` is `2.5`. The verbs `f3a` and `f3b` are ambivalent and each gives the reciprocal when used monadically and the quotient when used dyadically, i.e., `f3a 2.5` is `0.4` and `15 f3a 6` is `2.5`, with the same results for `f3b`.

The first line in each definition gives the name and specifies that a verb is being defined. The last line of each definition is a right parenthesis. A colon `:` separates the monadic and dyadic definitions and is omitted for a monadic verb. Left and right arguments are represented by `x.` and `y.`, respectively.

In an introductory section we gave the monadic verb `sum` and and the dyadic verb `wsum` for unweighted and weighted sums, respectively. It was stated that these verbs could be replaced by one ambivalent verb `Sum` that could be used either monadically or dyadically. Such a verb may be defined as

```
Sum=: 3 : 0
1 Sum y.
:
+/x. *  y.
)
```

and we have `Sum Prices` is `19.13` and `Qty Sum PriceList` is `19.13` as stated previously.

Correlation and regression

The correlation coefficient between two sets of observations is defined as the covariance of the two sets of observations divided by the product of the standard deviations, where the covariance is the product of the deviations of the observations from their respective means divided by one less than the number of pairs of observations. The regression coefficients define the least squares best-fitting straight line to the pairs of observations. To illustrate the calculations in this section we shall use the data

```
Yield=: 5.27 5.68 6.25 7.21 8.02 8.71 8.42
```

and

```
Water=: 12 18 24 30 36 42 48
```

from Hoel (1966) on crop yield (bushels of alfalfa per acre) and amount of water applied (inches).

The sum of products is given by

```
sp=: [: +/ *&dev  ,
```

where the `&` is the dyadic conjunction *compose*, and `x *&dev y` is equivalent to `(dev x) * (dev y)`, so that

```
Water sp Yield
```

is `103.68`. The covariance is given by

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

and

```
Water cov Yield
```

is `17.28`. The correlation coefficient is

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

and

```
Water cor Yield
```

is `0.972408`.

In order to find the regression line expressing the dependence of yield on water we first define the two-column table

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

with `1`s in the first column and the values of `Water` in the second column. The regression coefficients are given by

```
b=. Yield %. X
```

where `%.` is the dyadic verb *matrix divide* and gives in this instance the least-squares solution for a coefficient matrix `X` and right-hand side `Yield`. The value of `b` rounded to three figures is `3.99 0.103` so that 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 mp b`, where `mp` is the utility verb for the matrix product, and are, rounded to two decimal places, equal to

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

The following is the verb `SR` which gives the regression coefficients and some of the statistics required to test for the significance of the linear regression:

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

We note the use of indirect assignment to define both the intercept `b0` and the slope `b1` in the same statement. The variable `SRtable` is a three-column table with 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. Its value is given by global assignment *is (global)* `=:` making it available outside the definition of `SR`. All other variables are given by local assignment *is (local)* `=.` and are not available outside the definition.

The expression `Water SR Yield` gives the results

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

and 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
```

to the variable `SRtable`. If we let

```
Yest=:  5.23 5.85 6.46 7.08 7.7 8.31 8.93
```

be the estimated values of the yield, then the **J** plot package may be used as follows to give the observed values and linear regression line as shown in Figure 3.4.

```
pd 'new'
```

```
pd 'type point'
pd Water; Yield
pd 'type line'
pd Water;Yest
pd 'show'
```

The calculations given above may be generalized to accommodate an arbitrary number of independent variables, and verbs for variance-covariance and correlation tables and for multiple linear regression are described in Appendix 2.

Analysis of variance

The main computational problem in the analysis of variance is the partition of the total variation of a number of observations 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 interactions. If the data are arranged in a rectangular array with an axis for each factor and one for replications, then a major task of the partitioning process is the calculation of some, or possibly all, of the marginal totals of various subarrays. For each set of marginal totals a weighted sum of squares is calculated from which the required analysis-of-variance table may be found relatively simply.

In this section we shall discuss briefy the calculation of marginal totals and then give an example of the use of a **J** program that can accommodate data for designs of arbitrary structure and size. For sample data we shall use those of a previous section giving the yield for various treatments on one variety of oats and then include data for a second variety.

For the observations for the first variety-treatment combination given by the list `w` with the value

```
62.3 58.5 44.6 50.3
```

there are the observations themselves and their sum `215.7` given by `+/w`. For a one-dimensional array there are thus `2` different marginal totals.

The three treatments for the first variety are given by the two-dimensional array `W` which is

```
62.3  58.5  44.6  50.3
63.4  50.4  45.0  46.7
64.5  46.1  62.6  50.3  .
```

There are now four different quantities to calculate: the observations `W`, the row sums `+/"1 W` with the value

```
215.7 205.5 223.5  ,
```

the column sums `+/W` with the value

```
190.2 155 152.2 147.3 ,
```

and the overall sum `+/+/W` which is `644.7`. For a two-dimensional array there are thus `2 * 2` or `4` different marginal totals.

To simplify the calculation of the marginal sums for an arbitrary array we introduce the utility verb `msum` where the right argument gives the array and the left argument indicates the marginal sum to be calculated. For the list `w` we have that `0 msum w` is the sum `215.7` of the items of the list, and `1 msum w` is the given list `62.3 58.5 44.6 50.3`. For the two-dimensional array `W` we have that `0 1 msum W` is equal to the column sums, `1 0 msum W` the row sums, and `0 0 msum W` the total sum. The array itself is given by `1 1 msum W`.

If the data for the second variety of oats are given in the array `X`, say, with the value

```
75.4  65.6  54.0  52.7
70.3  67.3  57.6  58.5
68.8  65.3  45.6  51.0  ,
```

then the statement

```
Vboth=: W,: X
```

gives the three-dimensional array

```
62.3  58.5  44.6  50.3
63.4  50.4  45.0  46.7
64.5  46.1  62.6  50.3

75.4  65.6  54.0  52.7
70.3  67.3  57.6  58.5
68.8  65.3  45.6  51.0
```

representing all of the observations for both varieties. This array has the shape `2 3 4` with `2` levels each with `3` rows and `4` columns with the levels representing the varieties and the rows and columns representing treatments and replications, respectively. If we count the array itself and the total over all of the data, there will be `2 * 2 * 2` or `8` different marginal sums to compute. For example,

```
0 1 1 msum Vboth
```

is the sum over the varieties

```
137.7 124.1  98.6 103.0
133.7 117.7 102.6 105.2
133.3 111.4 108.2 101.3  ,
```

and

```
0 1 0 msum Vboth
```

is the sum over varieties and replications

```
463.4 459.2 454.2  .
```

Having introduced and illustrated the calculation of marginal sums, we omit any further details of the use of **J** in the definition of verbs for analysis-of-variance calculations. These calculations may be done by the ambivalent verb `ANOVA` whose right argument is a rectangular array of arbitrary rank giving the observations. If used monadically, the various components are represented by the letters `A`, `B`, `AB`, `C`, ..., `Total` and all `2^n` components are given for an array of rank `n`. If used dyadically, the left argument specifies the components with `Total` being assumed. However specified, the analysis-of-variance table specifies the components in the first column and the corresponding degrees of freedom, sums of squares and mean squares in the second, third and fourth columns. The following shows the use of `ANOVA` with

the above oats data:

```
          ANOVA Vboth                        'A B AB C' ANOVA Vboth
┌─────────────────────────────────────┬──────────────────────────────────────┐
│A        1   318.28167   318.28167  │A          1   318.28167   318.28167 │
│B        2     5.30333     2.65167  │B          2     5.30333     2.65167 │
│C        3  1026.06333   342.02111  │AB         2   106.60333    53.30167 │
│AB       2   106.60333    53.30167  │C          3  1026.06333   342.02111 │
│AC       3   132.34167    44.11389  │Error     15   455.12167    30.34144 │
│BC       6    68.01667    11.33611  │Total     23  1911.37333             │
│ABC      6   254.76333    42.46056  │                                      │
│Total   23  1911.37333              │                                      │
└─────────────────────────────────────┴──────────────────────────────────────┘
```

Finally, we shall give a verb for a one-factor design with an unequal number of replications for each treatment. For this we require the monadic verb

```
wsqs=: ([: *:+/) % #
```

to give the square of the sum of the items of a list divided by the number of items, and, for example,

```
wsqs 1 2 3 4 5
```

is `45`. The analysis of variance is given by the monadic verb `aov1` defined as follows:

```
aov1=: 3 : 0
D=. y.
DFerr=. (DFtot=. <:#;D) - DFtr=. <:#D
SStr=. (+/ wsqs EACH D) - CT=. wsqs ;D
SStot=. (+/ *: ;D) - CT
SSerr=. SStot - SStr
'MStr MSerr'=. (SStr,SSerr) % DFtr,DFerr
F=. MStr%MSerr
r1=. 'Treatments', 5.0 12.5 12.5 8.1":DFtr,SStr,MStr,F
r2=. 'Error     ', (5.0 12.5 12.5":DFerr,SSerr,MSerr),8$' '
r3=. 'Total     ', (5.0 12.5":DFtot,SStot),20$' '
r1,r2,:r3
)
```

The argument is a list, each item of which is a list giving the replications for each treatment. The result is a table with rows giving treatment, error and total terms and columns giving sums of squares, degrees of freedom, mean squares, and F-value.

As an example we shall use the following data given by the variable `D1` from Bennett and Franklin (1954) giving observations from a test of corrosion in a manufacturing process in three different founderies:

```
┌──────────────────┬──────────────────────────────────┬──────────┐
│84 60 40 47 34    │67 92 95 40 98 60 59 108 86       │46 93 100 │
└──────────────────┴──────────────────────────────────┴──────────┘
```

The analysis-of-variance table for these data is given by `aov1 D1` and is

```
Treatments    2  2329.09804  1164.54902     2.2
Error        14  7398.66667   528.47619
```

```
Total        16  9727.76471
```

Probability distributions

We shall give **J** verbs for some of the commonly occurring discrete and continuous probability distributions and also examples of their use. The discrete distributions are the binomial, Poisson, hypergeometric and geometric distributions, and the continuous distributions are the normal, t-, chi-square and F-distributions. First of all we shall introduce some **J** primitive verbs used in their definition.

We have already seen the monadic verb *integer* `i.` which gives the non-negative integers up to but not including the argument, and, for example, `i. 5` is `0 1 2 3 4` and `i. 3 4` is

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

Two related defined verbs which may be considered utilities are

```
pos=: [: >: i.
```

and

```
ei=: [: i. >:
```

for postive integers and extended integers, respectively, and `pos 5` is `1 2 3 4 5` and `ei 5` is `0 1 2 3 4 5`.

The monadic verb *factorial* `!` gives the familiar factorial function for a non-negative integer argument and the gamma function for other arguments. Thus `!5` is `120`, `!0` is `1` and `!2.5` is `3.32335`. The gamma distribution may be defined by the verb

```
gamma=: !@<:  ,
```

and, for example, `gamma x` is equivalent to `!x-1`. The dyadic form of `!` is the verb *out of* which gives combinations, and `3!5` is `10`, the number of combinations of `5` things taken `3` at a time. The verb

```
bc=: ei ! ]
```

gives binomial coefficients, and, for example, `bc 5` is `1 5 10 10 5 1`, and

```
bc EACH ei 5
```

is

```
1 0  0  0 0 0
1 1  0  0 0 0
1 2  1  0 0 0
1 3  3  1 0 0
1 4  6  4 1 0
1 5 10 10 5 1  ,
```

a segment of Pascal's triangle.

The exponential function is given by the monadic verb *exponential* `^` and `^x` is equivalent to e^x so, for example, `^1` is `2.71828`. The dyadic form of `^` is *power*, and, for example, `2 ^ 6` is `64`. The monadic verb *natural log* `^.`, which will be used in the next section, gives the natural logarithm of its argument

and `^. 10` is `2.30259`.

The verb `p` gives various constants based on π, and, for example, `1p1` is `3.14159`, `2p1` is `6.28319`, `1p2` is `9.8696`, and `1p_0.5` is `0.56419` which may also be expressed as `%%:1p1`.

The monadic verb *not* `-.` gives the complement of a boolean argument and the complementary probability if the argument is a probability, and, for example, `-. 0 1` is `1 0` and `-. 0.75` is `0.25`.

The verbs for continuous distributions require the integral or anti-derivative adverb `I` such that the expression `f I x` for some verb `f` gives the area under the graph of `f` from `0` to `x`.

For the discrete distributions we shall give definitions, both in conventional notation and in **J**, and some examples of their use, For the continuous distributions, we give the verbs for the cumulative and probability density functions and a few examples of the use of the latter verbs.

The probability of x successes in n independent binomial trials with probability 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.

```
   binomial=: 3 : 0
   :
   'n p'=. x.
   x=. y.
   (x!n) * (p^x) * (-.p)^n-x
   )
   3 0.6 binomial 0 1 2 3
0.064 0.288 0.432 0.216
```

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 $e^{-\lambda}\lambda^x/x!$.

```
   poisson=: ^@-@[ * ^ % !@]
   1.5 poisson 0 1 2 3 4
0.22313 0.334695 0.251021 0.125511 0.0470665
```

The geometric distribution gives the probability of first success in a sequence of binomial trials with constant probability of success. The probability of the first success occurring on the nth trial with probability p of success on a single trial is equal to $(1-p)^{n-1}p$, for n a positive integer.

```
   geometric=: [ * -.@[ ^ <:@]
   0.4 geometric 1 2 3 4 5 6
0.4 0.24 0.144 0.0864 0.05184 0.031104
```

The binomial distribution assumes a number of independent trials with the probability of success remaining constant throughout. The hypergeometric distribution assumes that this probability changes during the trials. As an example, if k balls are drawn at random without replacement from an urn containing m red balls and n black balls, where the red and black balls are considered to be Type A and Type not-A, respectively, it may be shown that the probability of drawing x red balls is

$${}^{m}C_{x}\,{}^{n}C_{k-x} / {}^{m+n}C_{k} \ ,$$

where $x = 0, 1, 2, ..., k$.

```
   hg=: 3 : 0
   :
   'm n k'=. x.
   x=. y.
   (x!m) * ((k-x)!n) % k!m+n
   )
   4 6 3 hg 0 1 2 3           NB. m = 4 (red), n = 6 (black), k = 3
0.166667 0.5 0.3 0.0333333

   NDISTN=: 3 : 0
   ndistn I y.
   )
   ndistn=: 3 : 0
   Const=. %:0.5p_1
   Const * ^--:*:y.
   )
   NDISTN 0 1 2 3
0 0.341345 0.47725 0.49865

   TDISTN=: 3 : 0
   :
   x.&tdistn I y.
   )
   tdistn=: 3 : 0
   :
   n=. x.
   Const=. (gamma -:>:n) % (gamma -:n) * %:n*1p1
   Const * (>:(*:y.)%n) ^ --:>:n
   )
   5 TDISTN 2.015 2.571 3.365
0.449997 0.475013 0.490001
```

```
   CSDISTN=: 3 : 0
   :
   x.&csdistn I y.
   )
   csdistn=: 3 : 0
   :
   n=. x.
   Const=. % (2^-:n) * gamma -:n
   Const * (y. ^ -:<:<:n) * ^ --:y.
   )
   10 CSDISTN 12.5 16 18.3
0.747015 0.900368 0.949891

   FDISTN=: 3 : 0
   :
   'm n' =. x.
   if. m=1 do.
      2 * n&tdistn I %:y.
      else.
      x.&fdistn I y.
   end.
   )
   fdistn=: 3 : 0
   :
   'm n'=. x.
   Const=. (gamma -:m+n) % (gamma -:m) * gamma -:n
   Const * (m^-:m) * (n^-:n) * (y.^-:<:<:m) * (n+m*y.)^--:m+n
   )
   5 20 FDISTN 2.16 2.71 3.29 4.1
0.900263 0.950012 0.975138 0.990169
   1 15 FDISTN 4.54 8.68
0.949932 0.989989
```

As an example of the use of the continuous distributions consider the last example of the previous section giving the analysis-of-variance table for a one-factor design with an unequal number of replications. The probability of getting an F-value greater than the `2.2` with numerator and denominator degrees of freedom of `2` and `14`, respectively, is given by the expression `2 14 FDISTN 2.2` which is `0.112326`. Since the 5%-value is 3.74, we may conclude there is no difference between founderies.

Random variables

In this section we shall discuss two primitive verbs in **J** for generating random non-negative integers

and then use them to develop verbs for the generation of uniform, normal and exponential random variables. We shall give a few simple examples of the simulation of rolling dice and finally give a simulation method of finding an estimate for the value of π.

We shall first introduce some constant verbs since two of them will be used in this section. The nineteen constant verbs `_9:`, . . . , `0:`, . . ., `9:` give the same constant value for an arbitrary argument. For example, the expressions `1: 5` and `1: 1 2 3` are both equal to `1`. A more meaningful example is the verb

```
odd=: 1: + 2: * i.
```

for successive odd integers, and, for example, `odd 5` is `1 3 5 7 9` and `odd _5` is `9 7 5 3 1`. Finally we introduce the constant verb infinity `_:` which has an infinite result for all arguments, and, for example, `_: 5` is `_` and so is `2 _: 5`. A constant verb with an arbitrary value may be defined very simply by using an infinite rank as, for example, `25"_` and `(25"_) 3.5`, say, is `25`.

We also note the monadic verb *double* `+:` which doubles its argument, and, for example, `+: 5` is `10`.

Random selection of non-negative integers is given by the monadic verb *roll* `?` which gives sampling with replacement, and `? n` gives a uniform random selection from the population `i. n`. For example, `? 10` could have any value between `0` and `9`, inclusive, and two successive values of `?10$6` could be

```
4 0 2 5 1 3 3 3 1 3
```

and

```
5 4 4 5 1 4 0 1 5 3 .
```

The dyadic verb *deal* `?` gives sampling without replacement and the expression `m ? n` is a list of `m` items chosen at random without replacement from the list `i. n`. For example, two possible values of `4 ? 6` could be `1 2 5 4` and `0 4 1 3`, and `10?10` which could have the value

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

is a random permutation of the first `10` non-negative integers.

The defined verb

```
Die=: [: >: [: ? [ $ 6:
```

simulates the throwing a die an arbitrary number of times as illustrated by the following examples:

```
   Die 10
1 5 3 4 2 1 5 5 6 3
   Die 10
4 5 1 1 4 5 1 3 1 3
   Die 12
1 5 3 4 2 1 5 5 6 3 4 5
```

The list `D` used at the beginning of the paper to illustrate the calculation of discrete frequencies was obtained in this manner.

A verb to simulate the rolling of two dice is given by

```
TwoDice=: [: <"1 [: Die ],2:
```

and, for example, `TwoDice 10` could be

```
┌─────┬─────┬─────┬─────┬─────┬─────┬─────┬─────┬─────┬─────┐
│6 4  │1 3  │6 2  │3 2  │4 6  │3 3  │5 3  │2 6  │1 4  │4 1  │
└─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┘
```

The sum of the faces occurring on each throw may be found by the expression

```
+/ EACH TwoDice 10
```

which for another simulation could give the result

```
3 8 9 7 3 4 8 10 6 11  .
```

The range for the sum occurring when two dice are thrown is given by

```
r=: >:>:i. 11  ,
```

or alternatively by `}. pos 12` where `}.` is the monadic verb *behead* which drops the first item of its argument, which has the value

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

Therefore the transposed frequency table for the sums when two dice are thrown `100` times is given by

```
|: r frtab +/ EACH TwoDice 100
```

and could have the value

```
2 3 4  5  6  7  8  9 10 11 12
2 5 6 10 16 20 15  8  9  7  2 .
```

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

```
rand=: (? % ]) @ ($&1e9)  ,
```

and, for example, `rand 5` is a five-item list which could have the value

```
0.903301 0.782992 0.746679 0.436426 0.00634169 ,
```

and `rand 3 4` would be a three-by-four table of random numbers.

As an example of the use of uniformly distributed random variables we give the verb

```
coords=: [: rand ],2:
```

which gives a specified number of random pairs of coordinates in a unit square lying in the first quadrant and with the lower left vertex at the origin. For example, `coords 4` could have the value

```
0.897656 0.909208
0.0605643 0.904653
0.504523 0.516292
0.319033 0.986642  .
```

This verb will be used at the end of the section in an example for estimating the value of π by random means.

Pairs of standard normal random variables X_1 and X_2 may be generated by the following algorithm:

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

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

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.

Following are the verbs `stdnrand` and `nrand` for generating standard normal variables and normal variables with an arbitrary mean and standard deviation, respectively, together with some examples of their use. The latter verb is ambivalent and gives gives standard normal variables when used monadically and normal variables with a mean and standard deviation specified by the left argument when used dyadically.

```
   stdnrand=: 3 : 0
   r=. i. 0
   while. y. > #r  do.
      whilst. S >: 1 do.
         V=: <:+:rand 2
         S=: +/ *: V
      end.
   r=. r, V * %: -+:(^.%])S
   end.
   y.{.r
   )
   nrand=: 3 : 0
   0 1&nrand y.
   :
   y.{., ({.x.) + ({:x.) * stdnrand y.
   )
   stdnrand 6
_1.44617 1.11674 _0.701165 _1.31714 1.24598 0.745155
   stdnrand 6
_0.638977 0.196838 0.57735 _0.341106 _0.36003 1.50886
   stdnrand 3
_0.135576 _1.59651 _1.34934
   nrand 5
1.12516 _0.285921 0.728642 _0.492181 _2.37745
   1 0.5 nrand 5
```

```
0.863109 0.0920709 0.83853 0.826178 1.15899
   (am,sd) 1 0.5 nrand 200
0.98917 0.522312
   (am,sd) 1 0.5 nrand 200
0.991909 0.510052
```

Random variables with the exponential distribution may be found by noting that if U is a uniformly 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=: [ * [: -@^. [: rand ]
```

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

```
   1.57809 1.77975 0.02637213 .
```

The area of the quadrant of a circle inscribed in a unit square is π/4. Therefore if a large number of points are selected at random within the square, the proportion lying within the inscribed quadrant may be used to estimate the value of π. It may be recalled that the verb

```
   coords=: [: rand ],2:
```

given earlier may be used to generate the random coordinates of points within a unit square with the lower left vertex at the origin, and that the four pairs of coordinates

```
   0.897656 0.909208
   0.0605643 0.904653
   0.504523 0.516292
   0.319033 0.986642
```

were given as an example. The row sums of the squares of these values would give the squares of the distances of the points from the origin. With the above table this value would be the list

```
   1.63245 0.822065 0.521101 1.07524
```

indicating that the second and third points lie within the quadrant of the circle. Thus the verb

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

will give the number of points lying within the quadrant where the argument is the total number of random points selected. Five evaluations of the expression `incircle 100` gave the values

```
   75 79 80 83 79
```

and thus estimates of π of

```
   4 * 75 79 80 83 79 % 100
```

or

```
   3 3.16 3.2 3.32 3.16  .
```

Finally the verb

```
   PI=: 4: * incircle % ]
```

will give an estimate of the value of π for an arbitrary number of points. Five evaluations of `PI 100000` gave `3.14112, 3.14596, 3.13996, 3.13556 and 3.13984.`

Chi-square

If we have a list of observed frequencies *obs* and the corresponding list *exp* of expected frequencies which would occur according to some hypothesis, then the chi-square statistic which may be used to test this hypothesis is given by

$\Sigma (obs - exp)^2 / exp$

with a number of degrees of freedom equal to one less than the number of frequency classes. As a simple example Hoel (1966) cites a breeding experiment that gave 120 magenta flowers with a green stigma, 48 magenta flowers with a red stigma, 36 red flowers with a green stigma, and 13 red flowers with a red stigma. Mendelian theory predicted that these flowers should be in the ratios 9:3:3:1. Therefore, the observed frequencies are

```
obs=: 120 48 36 13
```

and the expected frequencies, which may be calculated from the expression

```
217 * 9 3 3 1 % 16 ,
```

are

```
exp=. 122.063 40.6875 40.6875 13.5625 .
```

The value of chi-square is thus

```
+/ (*: obs - exp) % exp
```

which is `1.91244.` Since `3 CSDISTN 1.91` is `0.408`, there is a probability of approximately `0.6` of obtaining a larger value of chi-square if the hypothesis is true. Therefore there is no reason to doubt that Mendelian theory is applicable here.

The dyadic verb

```
chisq=: [: +/ ([: *: -) % [
```

where the left and right arguments give the expected and observed frequencies, respectively, may be used to find the value of chi-square. With the data of the previous paragraph we have that `exp chisq obs` is `1.91244`. However, the form of the verb we shall use is

```
chisq=: [: +/ [: , ([: *: -) % [
```

which will accommodate both one- and two-way frequency distributions. We note the monadic verb *ravel* `,` which converts its argument to a list, and, for example, if `W` is the table

```
62.3  58.5  44.6  50.3
63.4  50.4  45.0  46.7
64.5  46.1  62.6  50.3
```

introduced in a previous section, then `,W` is the list

```
62.3 58.5 44.6 50.3 63.4 50.4 45 46.7 64.5 46.1 62.6 50.3
```

with the items of `W` in row order.

As a second example let us use the expression

```
obs =. (i. 10) fr ? 200 $ 10
```

to give a frequency count of `200` random digits between `0` and `9`. One example of its use gave the list

```
23 21 24 18 23 19 24 19 15 14 .
```

Since the expected frequencies on the hypothesis of a random sequence of digits would all be equal to `20`, the value of chi-square is `20 chisq obs` which is `5.9`. Since `9 CSDISTN 2.8` is `0.25`, we may conclude that the `200` digits are from a uniform distribution.

The chi-square test may be used to test for independence in the factors of classification in a two-way contingency table. The expected frequencies on the assumption of independence 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 expected frequencies are given by the monadic verb

```
expfr=: (+/"1 */ +/) % [: +/ ,
```

whose argument is the table of observed frequencies. For sample data we shall use `Tab34` with the value

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

from Hoel (1966) in which `400` persons are classified both by level of education and marriage adjustment score. The expected frequencies are given by

```
expfr Tab34
```

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

```
(expfr Tab34) chisq Tab34
```

which is `19.94`. Since `6 CSDISTN 19.94` is `0.997`, we may conclude that the two variables of classification are related.

If the frequencies are small in a two-by-two table, then it has been suggested that the exact probabilities be calculated for the given table and for all other tables less likely to occur if the two variables of classification are independent. For example, with the three tables given by

```
T22=: (>2 5;3 3);(>1 6;4 2);>0 7;5 1
```

whose value is

```
┌─────┬─────┬─────┐
│2 5  │1 6  │0 7  │
│3 3  │4 2  │5 1  │
└─────┴─────┴─────┘
```

the first item is the given table and the second and third items represent the less likely tables on the assumption that the two criteria of classification are independent. For any table with frequencies a and b in the first row and c and d in the second row the associated probability may be shown to be

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

which may be rearranged as the hypergeometric distribution corresponding to c_1 red balls, c_2 black balls and a sample size of r_1. For example, the probability for the first table above is given by `5 8 7 hg 2` which is `0.32634`. Therefore we may define the monadic verb

```
chisq22=: (+/ , [:{.(+/"1)) hg [: {. ,
```

whose argument is a two-by-two table and whose result is the corresponding probability. The monadic verb *head* `{.` gives the first item of its argument, and, for example, `{. 1 2 3 4` is `1`. The probabilities for the above tables are given by

```
chisq22 EACH T22
```

with the value

```
0.32634 0.08159 0.00466 ,
```

and since the sum is `0.41259` we may assume that the two criteria of classification are independent.

As a final example of the chi-square test let us consider the simulation in the previous section in which we tossed two dice 100 times and tabulated the sums occurring and obtained the following frequency table with the sums in the first row and the observed frequencies in the second:

```
2 3 4  5  6  7  8  9 10 11 12
2 5 6 10 16 20 15  8  9  7  2
```

We would like to test if these results could be reasonably obtained with unbiased dice. We represent the observed frequencies as

```
obs=: 2 5 6 10 16 20 15 8 9 7 2
```

and then calculate the expected frequencies `exp` on the assumption that the dice are unbiased..

All possible sums that can be obtained when two dice are thrown are given in the table

```
+/~ p=: 1 2 3 4 5 6
```

which has the value

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

Equal values for the sums fall along the diagonals and may be grouped by the expression `</.+/~p`, where `/.` is the monadic verb *oblique*, to give the list

```
┌─┬───┬─────┬───────┬─────────┬───────────┬─────────┬───────┬────────┬─────┬──┐
│2│3 3│4 4 4│5 5 5 5│6 6 6 6 6│7 7 7 7 7 7│8 8 8 8 8│9 9 9 9│10 10 10│11 11│12│
└─┴───┴─────┴───────┴─────────┴───────────┴─────────┴───────┴────────┴─────┴──┘
```

The frequencies for the various sums are given by `# EACH </.+/~p` with the value

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

and the corresponding probabilities by

```
pr=: 36 %~ # EACH </.+/~p
```

which rounded to three decimal places are

```
0.028 0.056 0.083 0.111 0.139 0.167 0.139 0.111 0.083 0.056 0.028 .
```

Therefore, the expected frequencies are equal to

```
exp=. 100 * pr
```

which rounded to one decimal place are

```
2.8  5.6  8.3 11.1 13.9 16.7 13.9 11.1  8.3  5.6  2.8 .
```

The value of chi-square is given by `exp chisq obs` and is equal to `3.599`, and since

```
10 CSDISTN 3.599
```

is `0.0363705`, we may conclude that the simulated dice were unbiased.

Nonparametric methods

Nonparametric tests may be used in place of the more standard tests when the normality assumptions required by these latter tests are not satisfied. Since many of the calculations required in nonparametric tests require the ranks of the observations rather than the observations themselves, we shall first give verbs for calculating ranks, then verbs for calculating the rank correlation coefficient and for finding the number of runs in a sequence of observations. We shall begin our discussion by considering sorting the items in a list.

Sorting in non-decreasing order may be accomplished by the monadic verb *grade (up)* and the dyadic verb *sort*, both represented by `/:`. The monadic verb grades its argument giving the permutation which would sort the items of the argument in non-decreasing order. For example, for the list

```
v=: 4.5 2 6.1 3.7
```

the expression `/: v` is the list `1 3 0 2` giving the indices of the items of `v` beginning with the minimum and proceeding to the maximum. The dyadic verb sorts the left argument argument in the order specified by the grade of its right argument so that `v/:v` gives the list `2 3.7 4.5 6.1` of the items of `v` in non-decreasing order. This last expression may be written more simply as `/:~v` using the monadic adverb *reflex* which we have already seen. Thus we may define the utility verb

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

and `sort v` is `2 3.7 4.5 6.1`.

The monadic verbs *grade (down)* and *sort*, both represented by `\:`, are similar and sort in non-ascending order so that `\:v` is `2 0 3 1` and `\:~v` is `6.1 4.5 3.7 2`.

If we apply the grade verb twice, e.g., `/:/:v` which gives `2 0 3 1`, we shall obtain in zero-origin indexing the ranks of the items of `v` indicating that the first item is the third smallest, the second item is the smallest, the third is the largest, and the fourth item is the second smallest. The expression `>:/:/:v` gives `3 1 4 2`, the ranks in the more conventional one-origin indexing. To conveniently define a verb for unadjusted ranks we introduce the conjunction *power* `^:` which repeats its verb left argument a number of times specified by the right argument, and, for example, `(*:^:2) 5` is equivalent to `*: *: 5` or `625`. Thus we may define the verb

```
uranks=: >: @ /:^:2
```

for the ranks unadjusted for ties, and `uranks v` is `3 1 4 2`.

If there are ties in the observations, the ranks of equal observations are replaced by the arithmetic mean of their ranks neglecting ties. For example, for the list

```
v1=: 4.5 2 4.5 6.1 2 2 3.7  ,
```

the ranks unadjusted for ties are

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

while the adjusted ranks are

```
5.5 2 5.5 7 2 2 4  .
```

The adjusted ranks of the first and third observations are equal to `5.5`, the mean of the unadjusted ranks `5` and `6`, and those for the second, fifth and sixth observations are equal to `2` which is the mean of `1`, `2` and `3`.

A verb which gives the ranks adjusted for any ties follows quite readily from the symmetric table given by the expression `=/~` with a list argument. For example, since `v1` is the list

```
4.5 2 4.5 6.1 2 2 3.7  ,
```

the table `=/~ v1` has the value

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

and the `1`s in any column give the positions of equal items in `v1`. For example, the `1`s in the first and third rows of the first column show that the first and third items of `v1` are equal. The column sums are given by `+/=/~v1` and are equal to

```
2 3 2 1 3 3 1
```

and give the number of occurrences of each item in `v1`. Also the expression

```
(=/~ mp uranks) v1
```

which is equal to

```
11 6 11 7 6 6 4
```

gives the corresponding sums of the unadjusted ranks. Therefore, the verb

```
ranks=: (=/~ mp uranks) % [: +/ =/~
```

gives the ranks adjusted for ties, and `ranks v1` is

```
5.5 2 5.5 7 2 2 4 .
```

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`. Inverse ranks are given by the verb

```
invranks=: [: ranks - ,
```

and, for example, `invranks v` is `2 4 1 3`.

Spearman's rank correlation coefficient is the correlation coefficient computed from the ranks of the observations. Rather than give the customary expression involving the ranks, we shall give the verb

```
rcor=: cor&ranks
```

which calculates the rank correlation coefficient from the original observations. If we use the two lists

```
French=: 83 27 42 51 53 44 47 55 61 33
```

and

```
German=: 74 22 49 54 48 47 55 61 59 29
```

which give the marks of ten students in these two subjects 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`.

Before considering runs, we shall introduce two adverbs, the monadic *prefix* and the dyadic *infix* both represented by `\`, as we require the dyadic form in our discussion. The monadic prefix applies its verb left argument to each of the prefixes of its noun right argument, i.e., to the first item, the first two items, the first three items, etc. For example, since `pos 5` is the list of the first `5` positive integers, the prefixes are given by `<\pos 5` and are

```
┌─┬───┬─────┬───────┬─────────┐
│1│1 2│1 2 3│1 2 3 4│1 2 3 4 5│
└─┴───┴─────┴───────┴─────────┘
```

and `+/\ pos 5` gives the cumulative sums

```
1 3 6 10 15
```

and `*/\ pos 5` the cumulative products or factorials

```
1 2 6 24 120 .
```

The dyadic infix applies its verb argument to infixes of its right noun argument, overlapping infixes if the left noun argument is non-negative and non-overlapping of it is negative, as illustrated by the following examples:

```
   2 <\ pos 5
```

```
┌───┬───┬───┬───┐
│1 2│2 3│3 4│4 5│
└───┴───┴───┴───┘
```

```
  2 +/\ pos 5
3 5 7 9
  _2 <\ pos 5
```

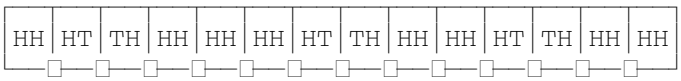
```
┌───┬───┬─┐
│1 2│3 4│5│
└───┴───┴─┘
```

```
  _2 +/\ pos 5
3 7 5
```

Now we may consider runs in a list of observations which are useful in determing the randomness of a sequence of observations. A run is defined as a sequence of consecutive identical observations preceded and followed by a different observation, or by no observation if the sequence begins or ends the list. For example, the list

```
HeadsTails=: 'HHTHHHHTHHHTHHH'
```

contains the seven runs `'HH'`, `'T'`, `'HHHH'`, `'T'`, `'HHH'`, `'T'` and `'HH'`. The expression `2 <\HeadsTails` is the list



of overlapping pairs, and `2 ~:/\HeadsTails`,

where `~:` is the dyadic verb *not-equal*, is the list

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

where the `1`s indicate the positions of pairs which are not the same. Thus we may be define the monadic verb

```
uneq=: 2: (~:/\) ]
```

whose argument is an arbitrary list and whose result is a list of `0`s and `1`s indicating equal and unequal overlapping pairs of items, respectively. Since the number of runs is one more than the number of unequal overlapping pairs, we may define the verb

```
runs=: [: >: [: +/ uneq
```

whose argument is an arbitrary list and whose result is the number of runs, and, for example, `runs HeadsTails` is `7`.

References

Bennett, C. A. and N. L. Franklin, 1954. *Statistical Analysis in Chemistry and the Chemical Industry*. John Wiley and Sons, Inc., New York.

Epstein, R. A., 1977. *The Theory of Gambling and Statistical Logic*. Academic Press, New York.

Eves, H., 1969. *An Introduction to the History of Mathematics. Third Edition*. Holt, Rinehart and Winston, New York.

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

Sprent, P. G., 1977. *Statistics in Action*. Penguin Books, Ltd., Harmondsworth, Middlesex.

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

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

Sternstein, M., 1994. *Statistics*. Barron's Educational Series, Inc., Hauppauge, N. Y.

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

Further examples

Common mean

A mean which is used in mathematical analysis in the study of elliptic functions, although it does not appear to have statistical applications, is the "common mean" or "arithmetic-geometric mean". It is defined as the limiting value of the arithmetic and geometric means and may be calculated as follows: Find the arithmetic and geometric means of the observations, then find the arithmetic and geometric means of these two means, and then the arithmetic and geometric means of these two means, ... continuing until the two means are equal. This common value is defined to be the common mean of the original observations. With the four-item list

```
   w=: 62.3 58.5 44.6 50.3
```

used previously we have the following example:

```
   (am,gm) w
53.925 53.4733
   (am,gm) (am,gm) w
53.6991 53.6987
   (am,gm) (am,gm) (am,gm) w
53.6989 53.6989
```

The common mean of `w` is thus `53.6989`.

To conveniently define the common mean we use the conjunction *power* `^:`. We note that an infinite power gives the limiting value of the application of the verb, and we have that `(*: ^:_) 5` is `_` and `(%: ^:_) 5` is `1`.

The above example for the calculation of the common mean may be given using the power conjunction as follows:

```
   (am,gm) ^:0 w
62.3 58.5 44.6 50.3
   (am,gm) ^:1 w
53.925 53.4733
   (am,gm) ^:2 w
53.6991 53.6987
   (am,gm) ^:3 w
53.6989 53.6989
```

These calculations may be given more concisely as

```
   (am,gm) ^:_ w
```

which is `53.6989 53.6989` as before. Finally using the monadic verb *head* `{.` which gives the first item of its list argument, we have that `{. (am,gm) ^:_ w` is `53.6989`. Therefore, a verb for the common mean is

```
   cm=: [: {. (am,gm)^:_
```

and `cm w` is `53.6989`.

One constant which has been given along with such constants as Archimedes constant π and Euler's constant *e* is the "ubiquitous constant" *u* defined as the common mean of 1 and 1/√ 2. In **J** this is given by `cm 1, %%:2` which is equal to `0.847213`. Finally the common mean is used in an iterative algorithm with quadratic convergence for the calculation of π in which the number of significant digits doubles on each iteration.

Coupon collector's problem

The coupon collector's problem is concerned with 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 for *n* coupons (or integers) may be shown to be "*n* times the sum of the reciprocals of the first *n* positive integers". If there are five prizes, say, a **J** expression for the expected sample size is

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

or equivalently

```
5 * +/ % pos 5 ,
```

which is equal to `11.4167`. Therefore we may define the verb

```
cc=: * [: +/ [: % pos  ,
```

and, for example, `cc 5` is `11.4167`, `cc 10` is `29.2897` and `cc 26` is `100.215`. We note that the verb `cc` consists of the fork `[: % pos` followed by the fork `[: +/ ([: % pos)` and finally by the hook `* ([: +/ [: % pos)`.

An explicit verb for simulating the coupon collector's problem is the following:

```
ccsize=: 3 : 0
:
m=. x.
n=. y.
r=. i. 0
while. m > #r do.
   CCsample=: i. 0
   while. n > # ~. CCsample do.
      CCsample=: CCsample, >:?n
   end.
   r=. r, #CCsample
end.
)
```

We note that the variables `m`, `n` and `r` are local to the definition, whereas the value `CCsample` which

gives the sample values for the last simulation is global. The expression `10 ccsize 5` gives the sample sizes for `10` simulations with `5` coupons, and could have the value

```
15 18 6 9 10 19 6 17 30 28
```

with `CCsample` being the the list

```
3 3 5 3 4 5 4 5 4 4 1 5 3 3 1 4 3 4 4 1 5 5 5 3 1 3 5 2
```

of `28` sample values of the last simulation. The sample for a single simulation may be obtained using a left argument of `1`, and, for example, `1 ccsize 5` could give the result `12` with a value for `CCsample` of

```
3 2 3 5 3 3 5 2 2 5 1 4 .
```

Finally, the expression

```
|: nubfrtab sort S=: 100 ccsize 5
```

gives a transposed nub frequency table of the sample sizes for `100` simulations for `5` coupons with the ungrouped and unsorted sample sizes in `S`. One value of this expression is

```
5 6  7  8 9 10 11 12 13 14 15 16 17 18 19 21 23 24 38
6 5 11 14 8 11  7  7  4  5  5  6  1  2  3  1  2  1  1
```

For this simulation we have `am S` equal to `11.97` where the expected sample size is approximately `11.4`.

A Windows form for the coupon collector's problem is given in Figure 4.2 of Appendix 4.

German army corps example

A classic example of the Poisson distribution given in many texts including Weaver (1963) is data giving the number of deaths which occurred from 1875 to 1894 in various German army corps due to kicks from horses. We shall construct a frequency distribution of the number of deaths, calculate the frequencies expected if the deaths are distributed in a Poisson distribution with the same mean, and finally use the chi-square distribution to compare the observed and expected frequencies.

The data are given as the list `H` which may be displayed conveniently by the expression `7 40$H` as

```
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 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 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 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 .
```

The number of observations is `#H` or `280`, the maximum number of deaths is `>./H` which is `4`, and the frequencies of `0` to `5` deaths are

```
0 1 2 3 4 5 fr 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 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  .
```

Since the last two frequencies are small, we shall group them and have the lists

```
obs1=: 144 91 32 11 2
```

and

```
exp1=: 139 97.3 34.1 7.9 1.6
```

for observed and expected frequencies. The value of chi-square is `exp1 chisq obs1` which is `2.03` so there is no significant departure of the observed data from the Poisson distribution.

Central limit theorem

A very important theorem in statistics is the Central Limit Theorem which states that the sample arithmetic mean, based on a random sample size of n from a population with mean μ and standard deviation σ, will possess an approximate normal distribution with mean μ and standard deviation $\sigma/\sqrt{n}$ with the approximation becoming increasingly good as n increases. This theorem is of great importance in estimation procedures and tests of significance. In this section we shall use some of the **J** verbs we have introduced previously to simulate repeated sampling from a given theoretical population and to examine the distribution of the resulting sample means.

In this example, taken from Hoel (1966), the population random variable x with the range 1, 2, ... , 6 has a probability density $p(x)$ given by the following table:

x	1	2	3	4	5	6
$p(x)$	0.25	0.25	0.20	0.15	0.10	0.05

The mean and standard deviation of this distribution may be found to be $\mu = 2.75$ and $\sigma = 1.48$, respectively. Now since the samples are of size 10, the sample mean will be distributed with mean 2.75 and standard deviation $1.48/\sqrt{10} = 0.47$. The simulation in the text consisted of 100 samples each of size 10 found by selecting a total of 1000 two-digit random numbers from a table of random numbers. A random number from 00 to 24 represented a value of the random variable of 1, a number from 25 to 49 represented a value of 2, etc. The 1000 values were arranged in groups of 10, the sample mean of each group calculated, the distribution of the 100 sample means tabulated, and the mean and standard deviation of the sample means calculated. The distribution of the means was seen to be a reasonable approximation

to the normal distribution with a mean and standard deviation close to those values predicted by theory.

The probabilities in the distribution constructed by Hoel may be represented by the list

```
p=: 0.25 0.25 0.20 0.15 0.10 0.05
```

and the range of the random variable by

```
   x=: 1 2 3 4 5 6  .
```

The population mean is thus

```
mu=: +/x * p
```

which has the value `2.75` and the population standard deviation is

```
sigma=: %: +/(+/p*x^2) - mu^2
```

which is `1.47902.` Therefore, the sample mean has a distribution with mean `2.75` and standard deviation

```
stdev=: sigma % %: 10
```

which is `0.467707`.

In the simulation the verb

```
Hdistn=: [: >: [: _1 24 49 69 84 94 99&io ?@$&100
```

gives sample values of a random variable distributed according to the required distribution. The expression `Hdistn 5` could have the value

```
1 4 2 3 1  ,
```

and `Hdistn 5 8` is a five-by-eight table whose items could be

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

The expression `am Hdistn 10 12` gives the arithmetic means of `12` samples of `10` items each and could have the value

```
 3 2.8 2.8 2.3 2.4 3.3 2.4 2.8 2.9 2.9 2.7 3  .
```

The variable

```
M1=: am Hdistn 10 100
```

gives the means of `100` samples of `10` items each. The expression

```
(<./,>./) M1
```

gives the minimum and maximum values of `1.6` and `4`, respectively. The class limits for a frequency classification are given by the arithmetic progression

```
ap 1.5 0.2 14
```

with values

```
1.5 1.7 1.9 2.1 2.3 2.5 2.7 2.9 3.1 3.3 3.5 3.7 3.9 4.1 .
```

The transposed grouped frequency table is given by

```
t1=: |: (ap 1.5 0.2 14) grfrtab M1
```

which has the value

```
1.6 1.8 2 2.2 2.4 2.6 2.8  3 3.2 3.4 3.6 3.8 4
  1   2 9   6  13  21  15 12   8   8   3   1 1 .
```

Let a second simulation be given by

```
M2=: am Hdistn 10 100
```

where `(<./,>./)M2 is 1.6 3.9`. Then a second frequency table is given by

```
t2=: |: (ap 1.5 0.2 14) grfrtab M2
```

which is

```
1.6 1.8 2 2.2 2.4 2.6 2.8  3 3.2 3.4 3.6 3.8 4
  2   0 6   9  10  17  16 18   8   9   2   3 0
```

Barcharts for the two simulations are given by

```
((0&{ barchart 1&{) t1);(0&{ barchart 1&{) t2  ,
```

where the dyadic verb *take* `{` is used to give the first and second rows of the transposed tables,.and are as follows:

```
┌──────────────────────┬──────────────────────┐
│1.6 *                 │1.6 **                │
│1.8 **                │1.8                   │
│  2 *********         │  2 ******            │
│2.2 ******            │2.2 *********         │
│2.4 *************     │2.4 **********        │
│2.6 ********************│2.6 ****************  │
│2.8 ***************   │2.8 ****************  │
│  3 ************      │  3 ******************│
│3.2 ********          │3.2 ********          │
│3.4 ********          │3.4 *********         │
│3.6 ***               │3.6 **                │
│3.8 *                 │3.8 ***               │
│  4 *                 │  4                   │
└──────────────────────┴──────────────────────┘
```

Throwing dice

In this section we shall give some further examples of the simulation of dice throwing and refer to two classical experiments in throwing dice. To begin we shall generalize the dice-throwing simulation to accommodate any number of dice with an arbitrary number of faces thrown any number of times with the dyadic verb

```
PolyDice=:  [: <"1 [: |: [: >: [: ? ] $ [
```

where the left argument gives the number of faces and the right argument is a two-item list giving the number of dice and the number of throws. Some examples of it use are the following:

```
6 PolyDice 1 12
```

```
┌─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┐
│4│2│3│4│6│1│1│4│4│5│4│1│
└─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┴─┘
```

```
4 PolyDice 3 8
```

```
┌─────┬─────┬─────┬─────┬─────┬─────┬─────┬─────┐
│2 1 4│2 3 4│1 4 2│3 1 3│2 1 3│2 3 4│1 3 4│4 3 4│
└─────┴─────┴─────┴─────┴─────┴─────┴─────┴─────┘
```

```
   8 PolyDice 2 5
```

5 8	5 1	4 4	5 7	8 4

```
   20 PolyDice 3 6
```

8 6 4	10 8 20	20 10 2	11 12 8	9 15 5	15 4 13

With real dice the number of different faces must correspond to one of the five regular polyhedra with four, six, eight, twelve or twenty faces, but we may simulate the throwing of any die either real or imaginary. For example, the expression

```
   2 PolyDice 1 20
```

which could have the result

2	2	1	1	2	2	1	1	2	1	2	1	1	1	1	2	1	2	2	1

would represent the simulation of throwing one two-sided die `20` times. It could also represent the results of tossing a coin with the `1`s and `2`s corresponding to heads and tails, respectively. Furthermore, the expression

```
    1 2  fr ; 2 PolyDice 1 20
```

would result in a two-item list giving the number of heads and tails occurring in the simulation. The monadic verb *raze ;* is used to convert the boxed list of the results to an unboxed list.

The English biologist W. F. R. Weldon (1860 - 1906) performed several dice-throwing experiments to illustrate his statistical arguments. In one he tabulated the results of rolling twelve dice 4096 times counting as a success the occurrence of 4, 5 or 6. The following shows the use of **J** to simulate this example:

```
NB. Roll 4 dice 8 times
   r=: 6 PolyDice 4 8
   r
```

6 4 5 4	2 5 6 6	5 4 5 3	2 4 4 4	6 2 5 1	3 1 3 1	6 6 2 2	3 3 2 4

```
NB. Occurrences of 4, 5 and 6 marked with 1s
   (e.&4 5 6 each) r
```

1 1 1 1	0 1 1 1	1 1 1 0	0 1 1 1	1 0 1 0	0 0 0 0	1 1 0 0	0 0 0 1

```
NB. Number of 4s, 5s and 6s on each roll
   +/ EACH (e.&4 5 6 each) r
4 3 3 3 2 0 2 1
NB. Frequencies of 4s, 5s and 6s
   0 1 2 3 4 5 fr +/ EACH (e.&4 5 6 each) r
1 1 2 3 1 0
```

```
NB. Range of frequencies
   i. 13
0 1 2 3 4 5 6 7 8 9 10 11 12
NB. Simulation of Weldon's experiment
   (i. 13) fr +/ EACH (e.&4 5 6 each) 6 PolyDice 12 4096
1 9 74 202 493 768 987 775 504 203 68 8 4
```

Another set of dice data which apparently is not as well known as Weldon's was generated by a Swiss scientist Rudolf Wolf who rolled a die 100 000 times and obtained the results 16 632, 17 700, 15 183, 14 393, 17 707 and 18 385 for the frequencies of occurrence of 1, 2, 3, 4, 5 and 6, respectively. This experiment may be simulated very simply by the expression

```
   1 2 3 4 5 6 fr ,>6 PolyDice 1 100000 ,
```

one execution of which gave the list

```
   16730 16597 16509 16875 16551 16738 .
```

The expected frequencies for `100000` throws of an unbiased die are `100000%6` or `16666.7` so the value of chi-square for Wolf's data is

```
   (100000%6) chisq 16632 17700 15183 14393 17707 18385
```

or `748.486`. Since `5 CSDISTN 748.5` is approximately `0.031`, we may conclude that the die used was certainly biased. On the other hand the value of chi-square for the simulated data is approximately `0.67` which is consistent with the hypothesis that the die is unbiased. In discussing Wolf's data Epstein (1977) states that it is reasonable to assume that a die of poor quality such as would have been manufactured in the nineteenth century would have developed a bias when rolled a large number of times. He then remarks that "the dice employed by major gambling casinos are generally machined to tolerances of 1/2000 inch, are of hard homogeneous material, and are rolled only a few hundred times on a soft green felt surface before being retired".

To simplify the simulation of throwing dice we give the dyadic verb `Dice` where the left argument gives the number of faces and the right argument gives the number of dice and the number of throws:

```
   Dice=: 3 : 0
   :
   Faces=. x.
   'Dice Throws'=. y.
   R=: +/ EACH Faces PolyDice Dice, Throws
   Range=: Dice }. i. >: Faces * Dice
   Nub=: sort ~. R
   RangeFreq=: Range fr R
   empty NubFreq=: Nub fr R
   )
```

The results are all given by global variables and represent the sums of the faces occurring and the range, nub and corresponding frequencies. We note the verb `empty` which causes its verb argument to give an empty result which will be convenient when we use `Dice` in a Windows form. The expression `4 Dice 2 10` corresponds to throwing `2` four-sided dice `10` times and for one simulation gave the following results:

```
   R
5 5 6 7 4 6 2 8 5 7
   Range
2 3 4 5 6 7 8
   Nub
2 4 5 6 7 8
   RangeFreq
1 0 1 3 2 2 1
   NubFreq
1 1 3 2 2 1
```

Figure 4.3 in Appendix 4 shows a Windows form for throwing one, two or three regular polyhedral dice either `10`, `50`, `100` or `200` times. The results give both the tabular and barchart representations of either the range or nub frequencies.

A few more numbers

In this last section we shall discuss three topics which have not been considered in previous sections: number systems, arithmetic with rational numbers, and extended precision integer arithmetic. First of all, however, we shall mention two primitive **J** verbs related to prime numbers.

The monadic verb *primes* `p:` with an integer argument `n` gives the *n*th prime. For example, `p: 4` is `11` and `p: i. 10` is `2 3 5 7 11 13 17 19 23 29`. The inverse of `p:` is the number of primes less than the argument, and `p:^:_1 i. 15` is

```
   0 0 0 1 2 2 3 3 4 4 4 4 5 5 6 .
```

The monadic verb *prime factors* `q:` gives the prime factors of its argument, and, for example, `q: 2772` is

```
   2 2 3 3 7 11
```

so that

```
   */2 2 3 3 7 11
```

is equal to `2772`. The dyadic verb *prime exponents*, also represented by `q:`, gives the exponents in the prime decomposition of the right argument to the number of factors given by the left argument. For example, `7 q: 2772` is `2 2 0 1 1 0 0` so that

```
   */2 3 5 7 11 13 17 ^ 2 2 0 1 1 0 0
```

is `2772`. An infinite left argument gives just a sufficient number of exponents so that `_ q: 2772` is `2 2 0 1 1`.

Number systems to arbitrary bases may be handled by the *base* verb `#.` for conversion from an arbitrary base to decimal and the *antibase* verb `#:` for conversion from decimal to an arbitrary base. The monadic forms of these verbs give conversion between binary and decimal, and the dyadic forms give conversion between a base specified by the left argument and decimal. The following simple examples illustrate their use:

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

```
29
   #: 29
1 1 1 0 1
   2 #. 1 1 1 0 1
29
   8 #. 3 7 5
253
   16 #. 10 6 15
2671
   2 2 2 2 2 #: 29
1 1 1 0 1
   2 2 2 #: 29
1 0 1
   8 8 8 #: 253
3 7 5
   16 16 16 #: 2671
10 6 15
```

A verb of considerable usefulness is

```
   tt=: [: #: [: i. 2: ^ ]
```

which gives table, called a truth table, of all possible arguments of a logical function for the number of variables specified by its argument. For example, the transposed truth table for three variables is given by the expression `|: tt 3` which has the value

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

An ordered truth table, which is similar but has the rows sorted by the number of `1`s, is given by

```
   ott=: [: (/: +/"1) tt ,
```

and `|: ott 3` is

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

The expression

```
   ,( }. ott 3) # 'ABC'
```

gives the labels

```
   C  B  A  BC AC AB ABC
```

for all the main effects and interactions in a three-factor factorial experiment. If we use the monadic verb *reverse* `|.` the expression

```
   ,(|."1 }. ott 3) # 'ABC'
```

gives the labels in the more familiar order

```
   A  B  C  AB AC BC ABC  .
```

The base specified by the left argument may be mixed, an especially useful mixed base being `24 60 60` for conversion between hours, minutes and seconds and the total number of seconds. For example, the expression

```
24 60 60 #. 5 25 10
```

is `19510` which is the number of seconds in `5` hours, `25` minutes and `10` seconds, and, conversely,

```
24 60 60 #: 19510
```

is `5 25 10`. Another example is

```
365 24 60 60 #. 1 0 0 0
```

which is `86400`, the number of seconds in one day.

As an example of the use a mixed-base number system suppose we have timed an event that began at 5:25:10 o'clock in the afternoon and ended about two hours later at 7:15:8 seconds, and we wish to find the elapsed time in hours, minutes and seconds. If

```
T=: 7 15  8; 5 25 10  ,
```

then the required elapsed time is given by the expression

```
24 60 60&#:-/24 60 60&#. EACH T
```

which has the value `1 49 58` representing `1` hour, `49` minutes and `58` seconds.

As we have used the derived verb `-/` in the last paragraph to find the elapsed time in seconds, this may be an appropriate occasion to examine the verb with an argument with an arbitrary number of items For the list

```
p=: 1 2 3 4  5 6
```

of the first `6` postive integers, we have that `+/p` is the sum `21` and `*/p 5` is the product `720`. However, `-/p` is `_3` which may be puzzling. To understand this result let us recall the precedence rule for verbs in **J**. In Appendix 1 we state that precedence 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. Three of the examples given there are `2 + 3 * 4` and `(3 * 4) + 2` each of which has the value `14` and `3 * 4 + 2` which is `18`. Now let us keep this rule in mind and give a step-by-step evaluation of the expression `-/p` as follows:

```
-/p
1 - 2 - 3 - 4 - 5 - 6
1 - 2 - 3 - 4 - _1
1 - 2 - 3 - 4 + 1
1 - 2 - 3 - 5
1 - 2 - _2
1 - 2 + 2
1 - 4
_3
```

We may also write this value as

```
(1 + 3 + 5) - (2 + 4 + 6)
```

or as

```
(1 - 2) + (3 - 4) + (5 - 6) .
```

In words the verb `-/` with a list argument gives the sum of the odd-indexed items minus the sum of the even-indexed items in one-origin indexing, or the first item minus the second plus the third item minus the fourth, etc. Such a sum is called an alternating sum.

As another example we shall look at some data on the lengths of program segments and commercial breaks during a television program picked almost at random from the innumerable inane sitcom series appearing on commercial television. The data are given by the list `Times`

```
┌─────┬──────┬─────┬──────┬──────┬──────┬──────┐
│9 0 8│9 2 19│9 5 0│9 13 27│9 16 32│9 26 41│9 29 25│9 30 0│
└─────┴──────┴─────┴──────┴──────┴──────┴──────┘
```

indicating that the program began at 9:00:08 p.m., the first commercial started at 9:02:19 and ended at 9:05:00, and so on with the program finishing at exactly 9:30. From these data we wish to find the total amount of program time in the nominal half-hour time slot. From the previous example involving only two times and the above discussion of alternating sums we can see that the expression

```
-/24 60 60&#. EACH Times
```

which has the value `_1282` gives the sum of the differences of successive pairs of times, i.e., the negative of the total program time. Thus the expression

```
24 60 60&#:-/24 60 60&#. EACH |. Times
```

which is the same as the expression used in the previous example except that the reversed list of times is used gives the total program time `0 21 22` in hours, minutes and seconds which is

```
21 + 22 % 60
```

or approximately `21.4` minutes.

Rational numbers are represented in **J** by the integer numerator and denominator separated by `r` and preceded optionally by a sign. For example, `4r5` is the rational number four-fifths which is represented conventionally as 4/5 and `_3r2` is negative three-halves. As a simple example of rational arithmetic let us calculate the expected sample size for the coupon collector's problem for a population of 5 different coupons. Using the results of a previous section, we see that this expectation is given by "5 times the sum of the reciprocals of the first 5 positive integers". The following sequence shows the step-by-step evaluation of this expression using rational arithmetic:

```
   1r1 + 1r2
3r2
   3r2 + 1r3
11r6
   11r6 + 1r4
25r12
   25r12 + 1r5
137r60
   5r1 * 137r60
137r12
```

We note that `137r12` is the rational representation of the decimal number `137%12` or `11.4167` which is the decimal value of the expression

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

Finally we note that the rational calculations may be performed more simply as

```
5 * +/ % 1 2 3 4 5r1
```

or even more simply still as `cc 5r1` where `cc` is the verb for expectations developed in the previous discussion of this problem, and each of these expressions has the value 137r12

Extended precision integer arithmetic may be performed exactly by suffixing at least one of the integer arguments with `x`. For example `!20` is `2.4329e18` while `!20x` is

```
2432902008176640000  ,
```

and `!52x`, the number of arrangements of a deck of cards, is

```
80658175170943878571660636856403766975289505440883277824000000000000
```

an integer with `68` digits. Also the number of bridge hands is

```
(!52x) % (!13x)^4
```

or

```
53644737765488792839237440000 .
```

The distinguished British astronomer Sir Arthur Eddington who died in 1944 once estimated that the total number of particles in the universe was

$3/2 \times 2^{256} \times 138$

which is approximately 2.4×10^{79}. (Different references give slightly different values for this estimate but all are of the same order of magnitude.) This expression may be computed exactly as

```
207 * 2^256x
```

which is the 80-digit integer

```
23968962472124452452679193896798396925626
68868257875967561677198896380178354667520   .
```

As an aside we refer to a remark by the nineteenth-century British economist and logician W. S. Jevons and quoted by Martin Gardner in one of his many books: “Can the reader say what two numbers multiplied together will produce the number 8,616,460,799? I think it is unlikely that anyone but myself will ever know; for they are two large prime numbers.” We may find the answer very simply from the expression

```
q: 8616460799x
```

which has the value

```
89681 96079 ,
```

and verify it by the multiplication

```
89681 * 96079x
```

to give `8616460799`.

We shall now consider two methods of calculating π, one using a very slowly converging infinite series and the second using a **J** primitive verb to give a value to an arbitrary number of digits. Finally we shall use the second method to arrange the digits of π in a "pi tree" with an arbitrary number of levels.

The first method is due to the seventeenth-century Scottish mathematician James Gregory who obtained the infinite series

$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + \hat{}\hat{}\hat{}$.

If we note that the infinite series is the alternating sum of the reciprocals of the postive odd integers, we have the following verb

```
pi=: 4: * [: -/ [: % [: odd ]  ,
```

and, for example, `pi 100` is `3.13159`. That the infinite series converges very slowly is shown by the fact that a thousand terms gives a value of `3.14059265` correct to only two decimals and a million terms gives the value `3.14159165` correct to only five places.

The monadic verb *pi times* `o.` gives the value of π times its argument, and, for example, `o. 1` is `3.14159`. Roger Hui, one of the developers of **J**, has given the wonderful expression

```
<.@o.10x^n
```

that gives π to `n` decimal places and for `n=: 10`, say, we have `31415926535`. From this expression we may derive the monadic verb

```
pi=: [: ": [: <.@o. 10x"_^]
```

which for an integer argument `n` gives a character string representing the truncated integer value of π to the power `n`. For example, `pi 10` is

```
31415926535  ,
```

`pi 11` is

```
314159265358  ,
```

and `pi 50` is

```
31415926535897932384626433832795028841971693993751 0 .
```

If `PI500=: pi 500`, we may use the expression

```
(10{.,:'3.'),. 10 50 $ }. PI500
```

to display π to `500` decimal places as

```
3.14159265358979323846264338327950288419716939937510
  58209749445923078164062862089986280348253421170679
  82148086513282306647093844609550582231725359408128
  48111745028410270193852110555964462294895493038196
  44288109756659334461284756482337867831652712019091
  45648566923460348610454326648213393607260249141273
  72458700660631558817488152092096282925409171536436
  78925903600113305305488204665213841469519415116094
  33057270365759591953092186117381932611793105118548
  07446237996274956735188575272489122793818301194912
```

A frequency table of the first `500` decimal digits of π is given by the expression

```
(i. 10),: '0123456789' fr }. PI500
```

and is

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

```
45 59 54 50 53 50 48 36 53 52  .
```

If we let the observed frequencies given as the second row in this table be the list

```
obs=: 45 59 54 50 53 50 48 36 53 52 ,
```

then the value of chi-square on the assumption that the digits are uniformly distributed is `50 chisq obs` which is `6.88`. Since `9 CSDISTN 6.88`, there is no reason to doubt the hypothesis of a uniform distribution.

As a final diversion we shall arrange the digits of π in a triangular arrangement with an arbitrary number of levels as illustrated by the following example:

```
   3
  141
 59265
3589793
```

This is a very simple task if we note that an arrangement of *n* levels is constructed from n^2 digits consisting of the digit *3* in the first level followed in the remaining levels by the first $n^2 - 1$ decimal digits of π. This is accomplished by the following **J** verbs which for a given number of levels first compute π to the required number of decimal places, then arrange the digits as a right-angled triangle, and finally as the required tree:

```
PiList=: [: pi [: <: [: *: ]
PiTriangle=: ([: #~ odd) ,/. PiList
PiTree=: ([: - [: i. -@]) |."0 1 PiTriangle
```

The expression `PiTree 4` gives the example at the beginning of the paragraph, and `PiTree 10` gives the decoration below the title on the first page of this paper.

Appendix 1. A brief summary of J

```
   3 + 5           NB. Plus
8
   2 * 3           NB. Times
6
   3 - 5           NB. Minus
_2
   15 % 6          NB. Divided by
2.5
   % 8             NB. Reciprocal
0.125
   2 + 3 * 4       NB. Precedence
14
   2 * 3 + 4
14
   (2 * 3) + 4
10
   4 + 2 * 3
10
   % 15 % 6        NB. Ambivalence
0.4
   2 | 0 1 2 3 4 5 NB. Residue
0 1 0 1 0 1
   6.5 <. 3        NB. Lesser of
3
   4 >. 10         NB. Larger of
10
   <: 8            NB. Decrement
7
   >: 3.14         NB. Increment
4.14
   2.3 + 5 + 3.5 + 6  NB. Sum
16.8
   +/2.3 5 3.5 6
16.8
   w=: 2.3 5 3.5 6
   +/w
16.8
   <./w            NB. Minimum
2.3
   >./w            NB. Maximum
6
   (<./,>./)w      NB. Min. and max.
2.3 6
   #w              NB. Tally
4
   (+/w) % #w      NB. Arithmetic mean
4.2
   (+/ % #) w
4.2
   am=: +/ % #
   am w
4.2
   am 2.3 5 3.5 6
4.2
   dev=: - am      NB. Deviations from
   dev w           NB.  mean
_1.9 0.8 _0.7 1.8
   i. 6            NB. Integers
0 1 2 3 4 5
   >: i. 6         NB. Positive
1 2 3 4 5 6        NB.  integers
   pos=: >: @ i.
   pos 6
1 2 3 4 5 6
   ei=: i. @ >:    NB. Extended
   ei 6            NB.  integers
0 1 2 3 4 5 6
   pos=: [: >: i.  NB. Pos. integers
   pos 6           NB.  using [:
1 2 3 4 5 6
   i. 3 4
0 1  2  3
4 5  6  7
8 9 10 11
   +/i. 3 4        NB. Col. sums
12 15 18 21
   +/"1 i. 3 4     NB. Row sums
6 22 38
```

- The standard ASCII character set is used.
- The terminology of English grammar is used rather than that of programming languages. Functions are referred to as *verbs*. Their arguments are considered as *nouns* and *pronouns* instead of constants and variables, although the use of these latter terms is quite common. Verbs may be modified by *adverbs* and joined by *conjunctions* to give additional verbs. For example, the verb `+/` is derived from the verb `+` *plus* by use of the adverb `/` *insert* to give the sum of the items of a list, and the conjunction *rank*, represented by `"`, in the expression `+/"1` gives the row sums of a two-dimensional array.
- Primitives, i.e., verbs, adverbs and conjunctions, are represented by a single character or a single character followed by either a period or a colon. For example, `>` is the verb *larger than*, and `6 > 3.5` is `1` and `2 > 7` is `0` indicating that the first relationship is true and the second false. The verb `>.` is *larger of* and gives the larger of its two arguments so that `6 >. 5` is `6`, while `>:` is *larger or equal* and `6 >: 5` is `1` as is `6 >: 6` but `2 >: 7` is `0`. In addition, the verbs `<./` and `>./` are similar to the verb `+/` and give the minimum and maximum, respectively, of their list arguments.
- Most verb symbols represent one function when used with one argument on the right and another function when used with arguments on the right and left. For example, the verbs *reciprocal* and *divided by* are represented by the symbol `%`, so that `% 8` is `0.125` and `15 % 6` is `2.5`. Both forms may be used in the same expression so that `% 15 % 6` is `0.4` and is interpreted as "the reciprocal of `15` divided by `6`". Functions with a single argument are termed *monadic*, and those with two *dyadic*. As another example, the verb `>:` with a single argument represents *increment* so that `>: 3.5` is `4.5`, but with two arguments represents *larger or equal*.
- 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, the expression `% 15 % 6` in the previous paragraph is "the reciprocal of (`15` divided by `6`)" rather than "(the reciprocal of `15`) divided by `6`". Likewise, `2 + 3 * 4` is `14` as is `(3 * 4) + 2` but `3 * 4 + 2` is `18`.
- Negative numbers are indicated by a preceding underbar `_` which is considered to be part of the number as is, for example, the decimal point. Also the decimal point is necessarily preceded by at least one digit so that, for example, two-fifths as a decimal fraction is represented as `0.4`.
- Nouns may be single items or *atoms*, one-dimensional arrays or *lis*ts, two-dimensional arrays or *tables*, or arrays of higher dimension or *reports*. Thus the expression `a + b` is a valid sum as long as `a` and `b` are compatible arrays.
- Verbs may be defined in a *functional* or *tacit* manner without explicit arguments appearing in their definition. For example, the monadic verb

  ```
  am=: +/ % #
  ```

  gives the arithmetic mean However, *explicit* verbs may be defined where the arguments are specified in the definition which may extend over several lines and involve control structures similar to those in conventional programming languages.
- Finally, we mention a construct of considerable usefulness known as a *fork*, an uninterrupted sequence of three verbs. The definition of the arithmetic mean given in the last paragraph is an almost mandatory example of a fork. A similar construct involving a sequence of two verbs is called a *hook*, an example being

  ```
  dev=: - am  ,
  ```

  where `am` is the arithmetic mean of the last paragraph, which gives deviations from the mean. The monadic verb *cap*, represented by `[:`, which caps the left branch of a fork, may often be used instead of the conjunction `@` *atop*, or informally *after*, and, for example, `pos=: >: @ i.` and `pos=: [: >: i.` are alternative definitions of a verb for positive integers.

Appendix 2. A J Statistical Pacakage

The statistical programs, written in **J**, version 4.06, are classified in the following categories: Frequency distributions and diagrams, averages, variability, summaries, correlation and regression, analysis of variance, probability distributions, random variables, chi-square, and nonparametric methods.

The documentation of almost all of the verbs has the following format:

`name` Left argument, if any (Integers `m`, `n`; integer or real `u`, `v` ;
Right argument lists `x`, `y`; table `t`)
Explicit result

Discrete frequencies

`fr` `x` Range
`y` (Integer obs.)
Frequencies over range

`frtab` `x` Range
`y` (Integer obs.)
Frequency table over range

`nubfr` -
`y` (Integer obs.)
Nub frequencies

`nubfrtab` -
`y` (Integer obs.)
Frequency table over nub

Grouped frequencies

`grfr` `x` (End points of classes)
`y` (Integer or real obs.)
Class frequencies

`grfrtab` `x` (End points of classes)
`y` (Integer or real obs.)
Frequency table with mid-points in 1st col. and freq. in 2nd.

Barcharts

`barchart` `x` (Range)
`y` (Frequencies)
Range in 1st col. and freq. as * in 2nd

Stem-and-leaf diagrams

`SLdiag` -
`y` (Integer)
Stem-and-leaf diagram

`SLfrtab` -
`y` (Integer)
Stem-and-leaf frequency table

Means

`am` -
`y`
Arithmetic mean of `y`

`gm` -
`y`
Geometric mean of `y`

`hm` -
`y`
Harmonic mean of `y`

Other averages

`median` -

`y`

Median of `y`

`mode` -

`y`

Mode of `y`

Variability

`var` -

`y`

Variance of `y`

`sd` -

`y`

Standard deviation of `y`

`Q1` -

`y`

First quartile of `y`

`Q2` -

`y`

Second quartile (median) of `y`

`Q3` -

`y`

Third quartile of `y`

Summaries

`five` -

`y`

Min., 1st, 2nd and 3rd quartiles and max.

`summary` -

`y`

Summary statistics (with labels) of `y`

Correlation and regression

`SR` `x`

`y`

Simple linear regression table with dep. var. `y` and indep. var. `x`

`cov` `x`

`y`

Covariance between `x` and `y`

`cor` `x`

`y`

Correlation coefficient between `x` and `y`

`covtab` -

List, each item of which is a list

Variance-covariance table of all pairs

`cortab` -

List, each item of which is a list

Correlation table of all pairs

`REG` -

List, each item of which is a list with last item dep. variable

Multiple linear regression

Analysis of variance

`ANOVA` [Character string giving model]

Table or higher-dimensional array with left argument giving specified terms. Default gives all terms, e.g., `ANOVA t` for a two-dimensional array is

```
'A B AB' ANOVA t.
```

Analysis-of-variance table

Probability distributions

`binomial n, x` (Number of trials and prob. of success in a single

trial)

`m` or `y` (Number of successes)

Probabilities

`poisson` `u` (Mean)

`n` or `y` (Number of successes)

Probabilities

`geometric` `p` (Probability of success in a single trial)

`n` or `y` (Number of trials)

Probabilities

`hg` `x` (3-item list giving no. in pop. of type A, no. of type not-A, sample size)

`n` or `y` (No. in sample of type A)

Hypergeometric probabilities

`NDISTN` -

`u` or `y`

Prob. density function values

`TDISTN` `m` (Degrees of freedom)

`u` or `y`

Prob. density function values

`CSDISTN` `m` (Degrees of freedom)

`u` or `y`

Prob. density function values

`FDISTN` `m`, `n` (Num. & denom. d.f.)

`u` or `y`

Prob. density function values

Typical expressions involving the last four verbs for continuous distributions which give cumulative probabilities are as follows:

```
NDISTN 0 1 2 3
5 TDISTN 2.015 2.571 3.365
10 CSDISTN 12.5 16 18.3
5 20 FDISTN 2.16 2.71 3.29 4.1
```

Verbs for the corresponding density functions for the continuous distributions are `ndistn`, `tdistn`, `csdistn` and `fdistn` and have the same syntax

Random variables

`rand` -

`n`, `x`

Uniformly distributed random numbers, e.g., `rand 3` is a `3`-item list, `rand 2 4` is `2` by `4` table

`nrand [u,v]`

`n`

`n` normal deviates with mean `u` and s.d. `v`. Default is st. normal

`exprand` `u`

`n`

Exponentially distributed random numbers with mean `m`

Chi-square

`expfr` -

`t` (Exp. freq.)

`t` (Obs. freq.)

`chisq` `x` or `t` (Exp. freq.)

`x` or `t` (Obs. freq.)

Chi-square

`chisq22` -

`t` (Obs. freq. for 2-by-2 table)

Probability

Nonparametric statistics

`uranks` -

`y`

Ranks unadjusted for ties

`ranks` -

`y`

Ranks with ties averaged

`invranks` -

`y`

Ranks in inverse order

`rcor` `x`

`y`

Rank correlation coefficient

`runs` -

`y`

Number of runs

Appendix 3. Graphics

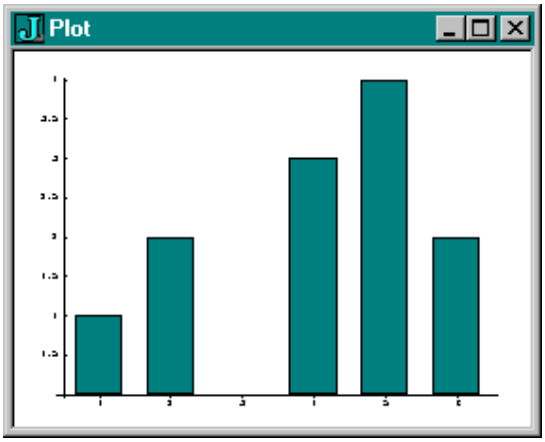


Fig. 3.1. Barchart for throwing 1 die.

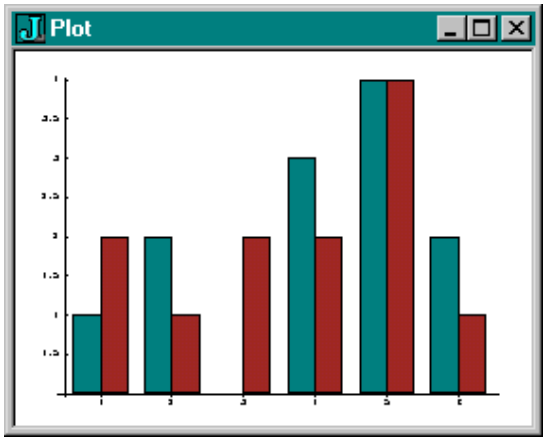


Fig 3.2. Barchart for throwing 2 dice.

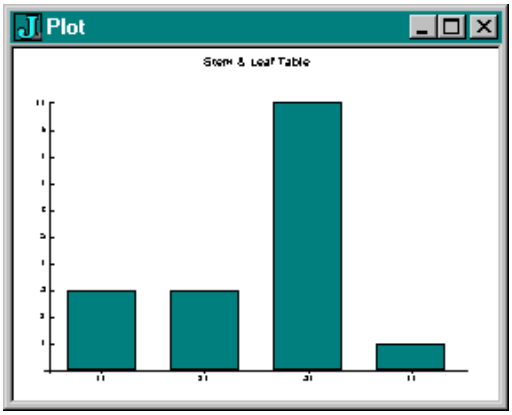


Fig. 3.3. Stem and Leaf Frequency Table

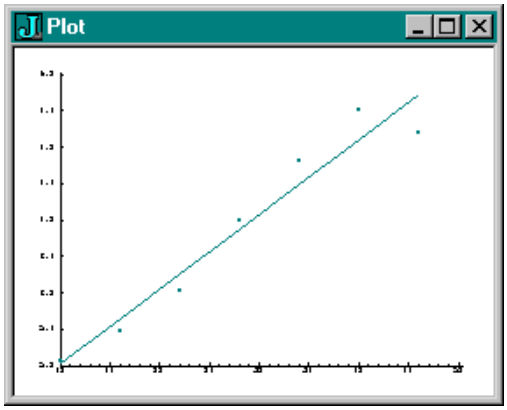


Fig 3.4. Linear regression of yield vs. water.

Appendix 4. Windows forms

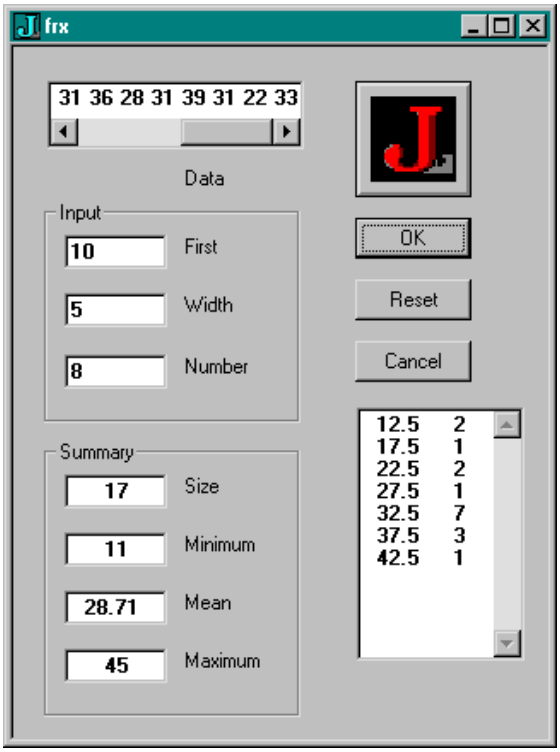


Figure 4.1. Summary statistics and grouped frequencies.

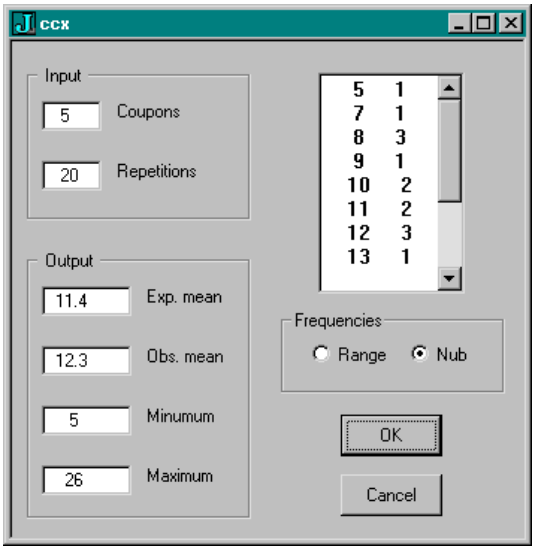


Figure 4.2. Coupon collector's problem.

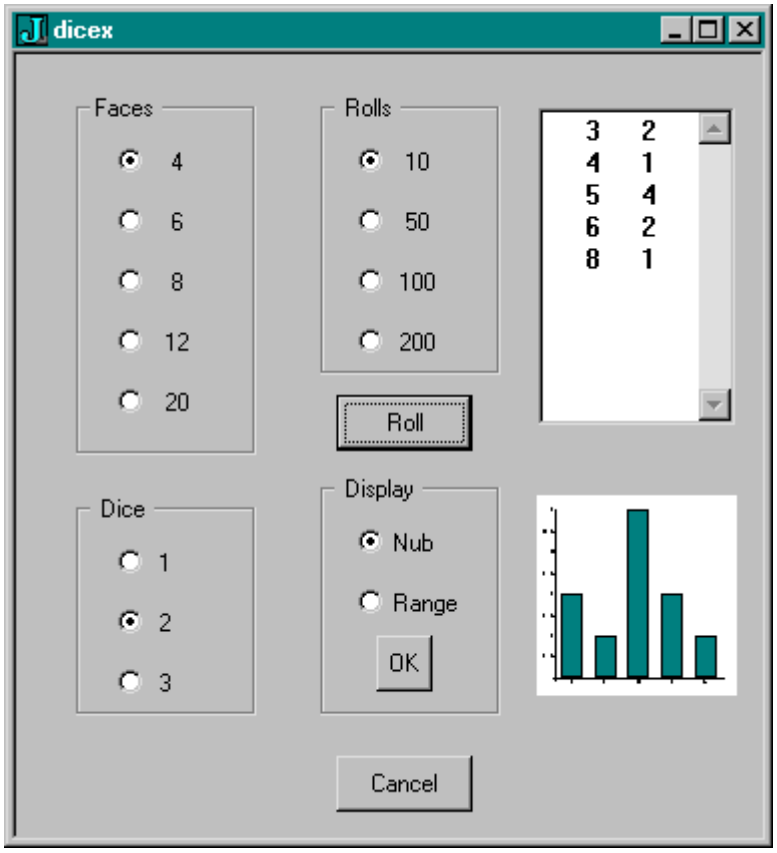


Figure 4.3. Dice throwing.

Appendix 5. J vocabulary

= Self-Classify • Equal	=. Is (Local)	=: Is (Global)
< Box • Less Than	<. Floor • Lesser Of (Min)	<: Decrement • Less Or Equal
> Open • Larger Than	>. Ceiling • Larger of (Max)	>: Increment • Larger Or Equal
_ Negative Sign / *Infinity*	_. *Indeterminate*	_: Infinity
+ Conjugate • Plus	+. Real / Imaginary • GCD (Or)	+: Double • Not-Or
* Signum • Times	*. Length/Angle • LCM (And)	*: Square • Not-And
- Negate • Minus	-. Not • Less	-: Halve • Match
% Reciprocal • Divide	%. Matrix Inverse • Matrix Divide	%: Square Root • Root
^ Exponential • Power	^. Natural Log • Logarithm	^: **Power**
$ Shape Of • Shape	$. Sparse	$: Self-Reference
~ ***Reflex • Passive*** / EVOKE	~. Nub •	~: Nub Sieve • Not-Equal
| Magnitude • Residue	|. Reverse • Rotate (Shift)	|: Transpose
. **Determinant • 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 • Antibase
! Factorial • Out Of	!. **Fit (Customize)**	!: **Foreign**
/ ***Insert • Table***	/. ***Oblique • Key***	/: Grade Up • Sort
\ ***Prefix • Infix***	\. ***Suffix • Outfix***	\: Grade Down • Sort
[Same • Left		[: Cap
] Same • Right		
{ Catalogue • From	{. Head • Take	{: Tail • {:: Map • Fetch
} ***Item Amend • Amend***	}. Behead • Drop	}: Curtail •
" **Rank**	". Do • Numbers	": Default Format • Format
` **Tie (Gerund)**		`: **Evoke Gerund**
@ **Atop**	@. **Agenda**	@: **At**
& **Bond** / **Compose**	&. &.: **Under (Dual)**	&: **Appose**
? Roll • Deal	?. Roll • Deal (fixed seed)	
a. *Alphabet*	a: *Ace* (Boxed Empty)	A. Anagram Index • Anagram
b. ***Boolean / Basic***	c. Characteristic Values	C. Cycle-Direct • Permute
d. **Derivative**	D. **Derivative**	D: **Secant Slope**
e. Raze In • Member (In)	E. • Member of Interval	f. ***Fix***
H. **Hypergeometric**	i. Integers • Index Of	i: Integers • Index Of Last
j. Imaginary • Complex	L. Level Of	L: **Level At**
m. n. Explicit Noun Args	NB. Comment	o. Pi Times • Circle Function
p. Polynomial	p: Primes •	q: Prime Factors • Prime Exponents
r. Angle • Polar	s: Symbol	S: **Spread**
t. ***Taylor Coefficient***	t: ***Weighted Taylor***	T. **Taylor Approximation**
u. v. Explicit Verb Args	u: Unicode	x. y. Explicit Arguments
x: Extended Precision	_9: to 9: Constant Functions	