*January 1999*

# J Companion for Statistical Calculations

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

**J** is used for the development of algorithms for a variety of statistical calculations including means, medians and quartiles, frequency tabulations, variances and covariances, regression analysis, graphical presentation, analysis of variance, random number generation and simulation, nonparametric tests, and probability distributions. Apart from the first two sections the organization of the paper is determined by the presentation of the statistical concepts with topics from J being introduced as required.

## Table of Contents



## Introduction

**J** is a general-purpose programming language developed by Kenneth Iverson, the originator of APL, and Roger Hui. It is intended to be a modern dialect of APL that will provide the simplicity and generality of APL while at the same time be readily and inexpensively available on a variety of computers and capable of being printed on standard printers. The language is fully described in *J Introduction and Dictionary*, Iverson (1998), which is an indispensible reference in learning and using the langauge. The version of **J** used in this paper is 4.01 for Windows 95/NT.

This paper may be considered to be a computational supplement to an introductory statistical text since the topics are discussed in an order similar to that which would be used in such a book with the elements of **J** being introduced as required. In other words **J** is

introduced and commented upon in the development of a real problem in the same way as a foreign language is presented in many classes and texts. A similar approach is used in the Berlitz books, as exemplified in the following quote from the introduction to a recent one: "Professor Berlitz's great innovation in the teaching of a foreign language was to modify the old practice of teaching grammar and vocabulary by rote, concentrating instead on the dynamic application of the living language from the moment a student begins his or her study." Furthermore, a comment on the cover of the same book says that it "… reinforces Japanese grammar and vocabulary skills with lively exercises and activities". Although we cannot promise any dynamic applications or lively exercises in this paper, we may at least say that we have approached the teaching of **J** in the way that we believe a natural language should be taught, and have illustrated the use of the language in the solution of some problems occurring in the real world.

Most sections of this paper begin with a discussion of some statistical calculation or group of related calculations and the introduction of any new **J** primitives that are required. These new primitives are then given in a box at the conclusion of the discussion. Further treatment of the new primitives and further examples usually conclude the section. Exceptions to this format are the following section in which the main features of **J** are illustrated with some very simple examples, a section giving some details on the Windows version of **J**, and a section on constants in **J**. This paper is an extensive revision of Smillie (1995), which it replaces together with Smillie (1996a, 1996b).

Appendix 1 gives a summary of the verbs derived in the paper. The script file containing all of the derived verbs and adverbs and utilities as well as the data is available by anonymous ftp at *ftp.cs.ualberta.ca* in the file *pub/smillie/jcomp.ijs*.

## The J language

In the following dialogue with the computer the **J** expressions we enter are indented three spaces whereas the computer responses begin at the left margin. Further more general comments are given later in the section.

```
   3 + 5              Plus
8
   3 – 5              Minus
_2
   2 * 3              Times
6
   15 % 6             Divided by
2.5
   % 2.5              Reciprocal
0.4
   % 15 % 6           Ambivalence
0.4
   2 * 3 + 4          Precedence
14
   (2 * 3) + 4
10
   5 > 6              Mnemonic one-
0                        or two-char.
   6 >. 5                primitives
6
   5 >: 5
1
   1 + 2 + 3          Sum
6
   +/1 2 3
6
   a=. 1 2 3          Lists and operations
   +/a                  on lists
6
   a % 2
0.5 1 1.5
   b=. 4 5 6
   a + b
5 7 9
   c=. a , 4 5        Append
   c
1 2 3 4 5
   +/c
15
   */c
120
   i. 5               Integers
0 1 2 3 4
   >: i. 5            Increment
1 2 3 4 5
   pos=: >: @ i.      Positive integers
   pos 5
1 2 3 4 5
   +/pos 5
15
   */pos 5
120
   ei=: i. @ >:       Extended integers
   ei 5
0 1 2 3 4 5
   +/ei 5
15
   */ei 5
0
```

| | |
|---|---|
| **x + y** | **x** *plus* **y** |
| **x – y** | **x** *minus* **y** |
| **x * y** | **x** *times* **y** |
| **x % y** | **x** *divided* by **y** |

| | |
|---|---|
| `% y` | *Reciprocal* of **y** |
| `x > y` | Is **x** *larger than* **y**? |
| `x >. y` | *Larger of* **x** and **y** |
| `x >: y` | Is **x** *larger or equal* to **y**? |
| `u / y` | ***Insert*** verb **u** between items of **y** |
| `x =. y` | *Is* (Local assignment of **y** to **x**) |
| `x , y` | **x** *append* **y** where axis 0 is lengthened |
| `i. n` | First **n** non-negative *integers* |
| `>: n` | *Increment* **n** *by* **1** |
| `x =: y` | *Is* (Global assignment of **y** to **x**) |
| `u @ v` | **u atop** (**after**) **v** |

The main characteristics of **J** are the following:

- 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* whose arguments are often called *nouns* and *pronouns* instead of constants and variables. Verbs may be modified by *adverbs* and, for example, the verb **+/** which gives the sum over a list is derived from the verb **+** *plus* by means of the adverb **/** *insert*. Also *conjunctions* allow the composition of verbs, and **@** is the conjunction *atop* (or *after*) which, for example in the defined verb **pos** applies the verb on the left after the verb on the right so that the verb might be read "increment (after generating) the non-negative integers".
- 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. Adverbs and conjunctions take precedence over verbs with the left argument being the entire verb phrase on the left.
- 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**. Constants may be written in exponential (scientific) notation, and, for example, **0.12345e3** and **12345e_2** each represents **123.45**.
- Most functions are ambivalent and represent one function when used with one argument and another function when used with two arguments. For example, **%** represents the monadic verb *reciprocal* and the dyadic verb *divided by*, and **/** represents the monadic adverb *insert* and the dyadic adverb *table*.
- Almost all primitives are represented by a single character or a single character followed by a period or a colon. For example, **>** represents the dyadic verb *larger than*, **>.** dyadic *larger of* and **>:** dyadic *larger or equal to*. Most primitives have both monadic and dyadic forms.
- Assignment is either local with the verb *is (local)* **=.** as in **x =. y**, or global with the verb *is (global)* **=:** as in **x =: y**.
- 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 or control structures, as, for example, **pos** which gives a list of positive integers. However, *explicit* verbs may be defined where the arguments are specified and where the definition may extend over several lines and involve control structures similar to those in conventional programming languages.

We may note the conventions used in the box in which new primitives are given. The primitives are given in **Bold Courier New**, and the names in Times New Roman with verb names in *Italic*, conjunctions in **Bold** and adverbs in ***Bold Italic***.

### Arithmetic mean

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

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

then the arithmetic mean is

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

which has the value **4.2**. In this expression the verb **+/**, which was given in the previous section, has been derived from the verb *plus* **+** by the adverb *insert* **/**, and gives the sum of the items of **w**, and **%** is the verb *divided by* which was also given in the previous section. The monadic verb *tally* **#** gives the number of items in its argument.

The expression for the mean may be written more simply as

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

so we may define the verb

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

and **am w** is **4.2**.

A sequence of three verbs such as occurs in the definition of the verb **am** is known as a *fork*. It is used in conventional mathematical notation, where, for example, $(f_1+f_2)x$ represents the sum $f_1(x)+f_2(x)$. The fork for the arithmetic mean is monadic as it has a single argument. In general, if **f**, **g** and **h** are verbs and **y** is a noun, then the fork

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

is equal to

**(f y) g (h y)** .

| | |
|---|---|
| **# y** | number of items (*tally*) in **y** |

Forks may also be dyadic with two arguments, and, for example,

**7 (+ * -) 3**

is equal to **40**, and is equivalent to

**(7 + 3) * (7 - 3)** .

In general, the dyadic fork

**x(f g h)y**

is equal to

**(x f y) g (x h y)** .

An uninterrupted sequence of an arbitrary number of verbs is called a *train* which may be evaluated by repeated resolution into zero or more forks followed by a final fork, or a *hook* which will be introduced later. For example, the sequence of verbs **d e f g h** represents the fork **d e (f g h)** where the verb on the right **f g h** is itself a fork.

### Geometric and harmonic means

While the arithmetic mean is the most commonly used average, two other averages - the geometric and harmonic means are discussed in some statiistics texts. The first may be used for averaging ratios and for finding an average percentage change, and the second for averaging data given as rates such as miles per hour. We shall introduce them here mainly for the insight their definition may provide for the **J** language.

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

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

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

The harmonic mean is the reciprocal of the arithmetic mean of the reciprocals of the observations. It is given by the verb

**hm=: % @ am @: %** ,

and **hm w** is **3.6793**. The conjunction *at* **@:** is used so that the sum is applied over the list of reciprocals rather than to each item in the list.

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

**means=: am , gm , hm**

which is a fork, and **means w** is equal to

**4.2 3.94211 3.6793**.

| | |
|---|---|
| **u @: v** | **u at** (**after**) **v** with infinite ranks |
| **x %: y** | **x** *root* of **y** |
| **x u~ y** | ***pass*** (**y u x**) |

Let us introduce the dyadic adverb *pass* **~** which interchanges the arguments of the verb. For example, the expression **5 %~ 14** is equivalent to **14 % 5** and is equal to **2.8** so that **%~** may be considered as "divided into". Then the geometric mean may be expressed in a similar manner to the arithmetic mean as

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

### Frequencies I - Range

We shall define verbs which give the frequency distribution for discrete data, i.e., data whose range is restricted to the non-negative integers **0**, **1**, **2**, ... . We shall need the 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=. i. 5** and **q=. i. 6**, then the expression **p+/q** is equal to

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

and gives the first five rows and six columns of an addition table. This table may be given also in the convenient form

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

by the expression

**p + table q**

where **table** is a general utility adverb, the details of which need not concern us, which may be used for producing bordered tables. (Appendix 6 gives the definition of all of the utility verbs and adverbs which are used in this paper.)

As an example of constructing a frequency distribution, suppose the list

**D=: 4 5 1 4 3 6 5 4 6 4 6 1**

represents the results of throwing a die **12** times, and let us consider tabulating the frequency of occurrence of each of the six faces. If we represent the range of faces by the list

**r=. 1 2 3 4 5 6** ,

then the expression **r=/D**, where **=** is the dyadic verb *equal*, has the value

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

where the first row shows that a **1** occurred on the third and twelfth toss, the second row that no **2**'s occurred, etc. The row sums, given by

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

are **2 0 1 4 2 3** and give the required frequencies. The row summation **+/"1**, in contrast to column summation **+/**, over a table shows the use of the conjunction *rank* **"** which will be discussed later in this section.

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 given above. 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
```

where the dyadic verb *left* **[** gives its left argument and the dyadic verb *stitch* **,.** joins its arguments in a table so that **r frtab D** is the table

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

A frequency table with two rows rather than two columns may be shown 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
2 0 1 4 2 3
```

The verb **fr** may also be defined as

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

where the monadic verb *cap* **[:** which has the effect of passing no left argument to the fork often allows a verb to be defined with fewer parentheses.

| | |
|---|---|
| **x = y** | Is **x** *equal* **y**? |
| **u"n y** | **u** with **rank n** on **y** |
| **x ,. y** | **x** *stitched* to **y** with axis 1 lengthened |
| **|: y** | *Transpose* order of axes of **y** |
| **[: f g** | ***cap*** of left branch of fork |
| **x [ y** | *Left* argument **x** |
| **x ] y** | *Right* argument **y** |
| **x ; y** | **x** *link* **y** with boxing if necessary |

We may note that the verb **frtab** uses a dyadic fork so that **r frtab D** is eqivalent to

```
(r [ D) ,. r fr D
```

In addition to the verb **[** which gives its left argument there is the dyadic verb *right* **]** which gives its right argument, so that, for example, **3[4** is **3** and **3]4** is **4**.

We have already noted that nouns may be single atoms, lists, tables, or higher-dimensional dimensional arrays sometimes called reports. The dimension of a noun is its *rank*, and atoms, lists and tables are of rank 0, 1 and 2, respectively. The last *k* axes of a noun determine a *rank k-cell* or more simply a *k-cell* of the noun. For example, if **t=. i. 3 4** is the table

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

then the 0-cells are the atoms **0**, **1**, **2**, ... , the 1-cells are the rows, and the 2-cell is **t** itself. For an array of rank r the (r-1)-cells are known as the *(major*) *items*.

The rank conjunction " is used to apply a verb, primitive or defined, to each of the k-cells of its argument so that in the expression **u"k y** the verb **u** is applied to each **k**-cell of the argument **y**. For example, **+/"1 t** is the list **6 22 38** of row sums, and **+/"2 t** is the list **12 15 18 21** of column sums. These results may be summarized by the expression

```
() ; +/"1 ; +/"2) t
```

which has the value

| | | |
|---|---|---|
| 0 1 2 3<br>4 5 6 7<br>8 9 10 11 | 6 22 38 | 12 15 18 21 |

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

### Frequencies II - Nub

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. For this purpose we intoduce the monadic verb *nub* **~.** which selects the distinct items from its list argument. For example, if **a** is the list **1 1 0 3 1 3 1**, then **~. a** is **1 0 3**. The

monadic verb *self-classify* **=** gives the distribution table which relates the items of its argument to the nub of the argument, and, for example, **= a** is

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

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

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

to give the list of frequencies so that **nubfr a** is **4 1 2** which means that **a** has four **1**s, one **0** and two **3**s. A frequency table for the nub is given by

```
nubtab=:  ~. ,. nubfr
```

and **nubtab D**, where **D** is the dice data of the last section, is

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

A table with the items of the nub in sorted order may be given very simply if we introduce the utility verb **sort** which sorts the items of its argument in ascending order. The required sorted frequency table is given by **nubtab sort D** and is

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

The two frequency tables just given may be displayed together very simply. We first note that

```
D; sort D
```

is the two-item list

| 4 5 1 4 3 ... | 1 1 3 4 4 ... |
|---|---|

with **D** as the first item and **sort D** as the second item. Then the desired tables are given by the expression

```
nubtab each D;sort D
```

which has the value

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

.

The utility adverb **each** applies the verb on the left to each item of its right argument.

**~. y** *nub* (distinct items) of **y**

**= y** *self-classify* items of nub of **y**

We may define the verb

```
bnubtab=: ~. ; "0 +/"1 @ =
```

where the two columns are joined with link **;** rather than stitch **,.** to give a boxed frequency table, and, for example, **bnubtab D** is

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

.

This verb may be used to give a frequency table for character data, so that, for example, if

```
t=. 'toronto' ,
```

then **bnubtab t** is

| | |
|---|---|
| t | 2 |
| o | 3 |
| r | 1 |
| n | 1 |

.

The verb **sort** may be used to sort character as well as numeric data, and **sort t** is **nooortt** and

```
bnubtab sort t
```

is

| | |
|---|---|
| n | 1 |
| o | 3 |
| r | 1 |
| t | 2 |

.

### J for Windows

In this section we shall make a few remarks about using **J** in a Windows environment. A complete discussion is given in the *User Manual* (Burke, 1998) which, as is all of the documentation from Iverson Software Incorporated, available either in printed form or on-line.

The Windows implementation of **J** combines the language interpreter with a user interface called the

session manager which allows several document windows, either execution or script windows, to be open at one time although only one may be active at a time. An execution window allows expressions to be evaluated and objects - nouns, verbs, etc. - to be defined as they are entered. A script window allows expressions and objects to be defined for later execution. Script and execution windows are ordinary Windows files which may be saved and used later on. The usual Windows edit controls for copying and pasting to and from the Clipboard, resizing, etc. are available. In a typical **J** development session the execution window is used for testing expressions and verbs which then may be copied and pasted to the appropriate script file and saved.

Script files may be - and, indeed, should be - annotated using the monadic verb *comment* **NB.** whose argument which is the remainder of the line on which it appears is ignored. For example, the definition of the arithmetic mean could be annotated either as

```
NB. Arithmetic mean
am=: +/ % #
```

or as

```
am=: +/ % #  NB. Arithmetic mean
```

Execution windows have the file extension of **.ijx** and are named **1.ijx**, **2.ijx**, etc. Loading **J** creates a new execution window for the session, and by default **1.ijx** is used unless it already exists when the next available name is used. Script windows have the extension **.ijs** and are given the names **1.ijs**, **2.ijs**, etc. However, script files that are to be used in more than one session should be given meaningful names as, for example, **jcomp.ijs** which is the name of the script file for this paper. An active script file may be run by the command **Run/Window** or alternatively **Ctrl+W**.

Global definitions using **=:** should be used in scripts since local definitions made with **=.** are no longer available after a script has been loaded with **Run/Window** and **Run/File...** . In this paper local definitions will be used only for temporary variables.

During a session the one-page **J** Vocabulary of primitive verbs, adverbs and conjunctions is available by pressing **F1**. In addition, context-sensitive help is available through **Ctrl-F1**. For example, if the cursor is immediately to the left of the monadic verb tally **#** in the definition of the arithmetic mean appearing in either a script or execution window and **Ctrl-F1** is pressed, then the page in the *Dictionary* is given in which the monadic and dyadic verbs tally and copy which are represented by **#** are defined. Buttons are displayed at the top of the page for scrolling backwards and forwards through the *Dictionary* from that page.

There are over three dozen on-line tutorial laboratory sessions on a variety of elementary and advanced topics available in **Studio/Labs...** . These may be recommended to both newcomers to **J** and to those who are interested in the new features in **J4.01**. There are also a number of demonstration examples available in **Studio/Demos...** .

A large number of scripts are available to support a variety of applications and may be loaded directly into an execution window or indirectly by including the appropriate command in a script file. For example, the command

```
load 'dates'
```

will load the date utilities script which has verbs for a variety of calendar calculations. A number of scripts with the suffix **lib** are automatically loaded at the beginning of a **J** session. These include **stdlib** and **winlib** which provide a large number of general-purpose verbs and also verbs for interacting with the Windows environment.

There is **J** plotting package named Plot which requires utilities made available by the command

```
load 'plot'
```

.

This package provides the verb **pd** which handles all calls to Plot and the verb **plot** which will handle most simple plots. The syntax of **plot** is

```
opt plot data
```

where **data** gives the data to be plotted and **opt** gives the plotting options. In this section we shall give only a few very simple examples of the use of Plot.

In a previous section we gave the variable **D** whose value is

```
4 5 1 4 3 6 5 4 6 4 6 1
```

representing the results of throwing a die **12** times and found that the frequencies of occurrence of the faces **1**, **2**, .. were

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

which we will refer to as **Throw1**. Then a bar chart of the frequencies may be found from the expression

```
'bar' plot Throw2
```

which is shown in Figure 0a in Appendix 2. We note that the default values along the horizontal axis are the positive integers from **1** to **6**.

Now consider throwing the die another **12** times and tabulating the frequencies which could be

```
2 3 2 3 0 2
```

which we will represent by **Throw2**. A table **T** with these frequencies in the rows is

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

which could be given by the expression

```
T=. |: Throw1,.Throw2
```
.

The following sequence of expressions will give the grouped bar chart with both sets of frequencies shown in Figure 0b and illustrate the use of the plotting verb **pd**:

```
pd 'new'
pd 'type bar'
pd 'ytic 1 0'
pd 'title Dice Throwing'
pd T
pd 'show'
```

An alternative method of preparing graphics is to use the graphics facilities of a software package such as MS Works. The data are prepared in **J**, copied to the Clipboard, pasted into an MS Works spreadsheet, and then processed appropriately. Also data made available in a spreadsheet may be passed to **J** by the converse process. The two utility verbs **CLIPwrite** and **CLIPread** are available for these operations. Although the details of these verbs need not concern us, we might remark that they require the verbs **charsub** from the script **strings.js**, **chop** from **misc.js**, and **clipfmt** from **format.js,** and that these files may be loaded into the execution window by the command

```
load 'strings misc format'
```
.

One of the functions of these verbs is to convert between the underbar _ used to represent negative numbers in **J** and the negative sign of conventional notation.

Let us now see how **J** could be used with MS Works to give the barchart for the two sets of frequencies. The expression

```
T=.  r,. Throw1,. Throw2
```
,

where

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

gives the range of faces, is a three-column table with the range in the first column and the two sets of frequencies in the second and third columns. The expression

```
CLIPwrite T
```

will copy the table **T** into the Clipboard, and also give an explicit result of **42**, the number of characters read, which may be ignored. The barchart prepared by MS Works is shown in Figure 0c.

The converse verb **CLIPread** will paste the Clipboard contents into the execution window. This verb requires an argument, **''**, say, which is ignored.

### Frequencies III - Continuous

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

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

and we wish to find the number of items in each of the intervals defined by the two-item lists

**1 3**, **3 5**, **5 7**, **7 9** and **9 11** .

The resulting frequency classification is the list **0 3 2 3 2** indicating that there are no items of **b** in the first interval **1 3**, three in the second interval **3 5**, etc.

To simplify the calculations we shall introduce a generalization of the dyadic verb *index of* **i.** which gives the index of first occurrence of each item in the right argument in the left argument. For example,

```
1 2 3 2  i. 2 5 3
```

is equal to **1 4 2** where the second item **4** indicates that the second item of the right argument does not occur in the left argument. The generalized utility verb is **io**, "Interval Of", and gives the interval of occurrence of each item in the right argument in the intervals specified by the left argument. For example,

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

is the list **2 3 1** which indicates that **5.2** occurs in the third interval **5 7**, **8.6** occurs in the fourth interval **7 9**, and **3.4** occurs in the second interval. For convenience, we shall let

```
a=. 1 3 5 7 9 11
```

be the list defining the intervals.

Now consider the dyadic verb

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

whose left argument is the list defining the intervals of classification and right argument is the list of data to be classified. The monadic verb *shape of* **$** gives the shape of its argument which for a list is the number of items. The monadic verb *decrement* **<:** subtracts **1** from its argument, and, for example, **<:3.5** is **2.5**. Now for the lists **a** and **b** defined above, we see that **a cfr b** is equivalent to

```
(i.<:$ a) fr a io b
```

so that the left argument of **fr** is

```
0 1 2 3 4
```

and the right argument is

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

and the problem is one of the classification of discrete data. Finally we define the dyadic verb

```
cfrtab=: midpts@[ ,. cfr
```

to give the desired frequency table. The utility verb **midpts** gives the two-term moving averages of its list argument, and, for example, **midpts a** is

```
2 4 6 8 10
```
.

The frequency table for the example used in this section is given by **a cfrtab b** and is

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

| | |
|---|---|
| **x i. y** | *Index of* first occurrence of **y** in **x** |
| **$ y** | *Shape of* **y** |
| **<: y** | *Decrement* **y** by **1** |

### Frequencies IV - Multidimensional

In order to conveniently tabulate in more than one dimension we shall need the monadic verb *catalog* **{** which is a generalization of the Cartesian product. As a simple example the expression **{1 2;3 4 5** has the value

| | | |
|---|---|---|
| 1 3 | 1 4 | 1 5 |
| 2 3 | 2 4 | 2 5 |

which is an array of shape **2 3** whose items are the two-item lists formed by selecting the first item from the list **1 2** and the second item from the list **3 4 5**.

Now consider a two-way classification with the ranges along the axes given by the two-item list

```
0 1 ; 0 1 2
```

which has the value

| | |
|---|---|
| 0 1 | 0 1 2 |

.

Now the joint range is given by

```
{ 0 1; 0 1 2
```

which is

| | | |
|---|---|---|
| 0 0 | 0 1 | 0 2 |
| 1 0 | 1 1 | 1 2 |

.

If some sample data are given by

```
d2=. 0 1;1 1;1 2;0 1;0 0;1 2;
   0 0;1 0;0 1  ,
```

then expression

```
({0 1; 0 1 2) =/ d2
```

will have the value

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

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

which is an array of dimension **2 3 9**. Summation along the last axis by the verb **+/"1** will give the required two-way table of frequencies

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

The calculations discussed in the last paragraph may be accomplished by the dyadic verb

```
FR=: [: +/"1 {@[ =/ ]
```

where the left argument is a list giving the ranges for each of the axes and the right argument the list of pairs of values to be classified. Thus the expression

```
(0 1;0 1 2) FR d2
```

will give the frequency distribution shown at the end of the last paragraph. As another example, if **y** is a list of triples whose items are either **0** or **1**, then the expression

```
(0 1;0 1;0 1) FR y
```

will give an array of rank **2 2 2** whose items give the frequency of occurrence of the triples in **y**.

| | |
|---|---|
| **{ y** | *catalogue* (Cartesian product) of items of **y** |

### Barcharts

Instead of displaying a numerical frequency table we may wish to represent the frequencies graphically in the form of a barchart. This may be done very simply but we first introduce two new primitives, the dyadic verb *copy* **#** and the conjunction *bond* **&**. The verb **#** 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 conjunction **&** may be used to bind an argument to a dyadic verb As an example, the verb

```
GST=: 0.07&*
```

which could also be defined as

```
GST=: *&0.07
```

multiplies its argument by **0.07** and may be used in tax calculations, and, for example, **GST 2.99** is **0.2093**. As another example, the expression **#&'*'** is a monadic verb which replicates the symbol ***** a specified number of times, and

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

is the array

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

We may now define the verb

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

to construct the bars for a barchart. The required barchart is given by the verb

```
barchart=: (": EACH @ [) ,.
           [: ' '&,. bars  ,
```

where the monadic verb *default format* **":** converts its argument to a character array. For example, with the dice data used previously we have that

```
r barchart D
```

has the value

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

| | |
|---|---|
| **x # y** | *Copy* items in **y** according to **x** |
| **n&g** | **Bond n** to dyad **g** as fixed left argument, |
| **g&n** | and fixed right argument |
| **": y** | *Default format* of **y** |
| **u ~ y** | *Reflex* (**y u y**) |
| **\|. y** | *Reverse* order of items of **y** |
| **< y** | *Box* **y** (Atomic encode) |
| **> y** | *Open* **y** (Unbox) |
| **x -: y** | **x** *match* **y**? |

The monadic adverb *reflex* **~** may be used to provide the preceding verb with equal left and right arguments. With its use, for example, the expression

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

used earlier in the section may be simplied to **#~ i. 4**.

A vertical barchart may be constructed very simply by the verb

```
vbarchart=: [: |. [: |:
            [: '^'&,.bars  ,
```

where the monadic verb *reverse* **|.** reverses the order of the items of its argument, and, for example, **|. i. 3** is **2 1 0** and **|. i. 3 4** is

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

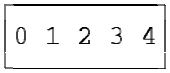
A vertical barchart for the dice data is given by

```
r vbarchart D
```

and is equal to

```
   *
   * *
*  ***
* ****
^^^^^^ .
```

Finally let us introduce open and boxed arrays which we shall begin to use in the next section. The expression **i. 5** gives the five-item list **0 1 2 3 4**, whereas the expression **<i. 5**, where **<** is the monadic verb *box*, gives the atom



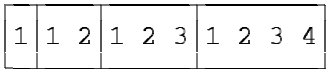

.

The verb inverse to box is *open* >, and, for example, **><i. 5** is equal to **i. 5**.

Two utility adverbs which are useful in dealing with boxed arrays are **each**, which was introduced earlier, and **EACH**. Each of these adverbs performs the operation given on the left on each item of the argument on the right, and the first preserves the boxing while the second does not. For example,

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

is the list

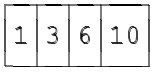



,

and

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

is



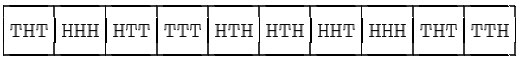

and

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

is

```
1 3 6 10  .
```

Two similar utility verbs which will be used only in the last example of the paper will be introduced here. They are **EACHRIGHT** and **EACHLEFT** and their use will be illustrated by a simple example. Suppose **c** is the list




representing the results of tossing three coins **10** times. We wish to find how many times heads and tails alternate, i.e., we wish to count the total number of occurrences of the sequences **HTH** and **THT**. Now the expression

```
'HTH' -: EACHRIGHT c  ,
```

where **-:** is the dyadic verb *match* which gives **1** if its arguments match and **0** otherwise, has the value

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

showing that the fifth and sixth tosses yield **HTH**, and

```
+/'HTH' -: EACHRIGHT c
```

is **2**, the number of occurrences of **HTH**. Similarly

```
'THT' -: EACHRIGHT c
```

is

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

and

```
+/'THT' -: EACHRIGHT c
```

is **2**, the number of occurrences of **THT**. These calculations may be given by the single expression

```
+/+/('HTH';'THT') -: EACHLEFT
                     EACHRIGHT c
```

which has the value **4**, the total number of occurrences of **HTH** or **THT**.

### Stem-and-leaf diagrams

Discrete data may be summarized by a stem-and-leaf diagram in which 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. Furthermore, for each group the integer quotient multiplied by **10**, or stem, is displayed once for the corresponding **10**-residues, or 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&|
```

,

where the monadic verb *floor* **<.** gives the largest integer less than or equal to its argument and the dyadic verb *residue* **|** gives the integer remainder when the right argument is divided by the left. For example, **10 % 7** is **1.42857** and **<. 10 % 7** is **1**, and **7|10** is **3**. Therefore, the diagram at the end of the last paragraph is given by the expression

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

.

We will also use 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** given by the list

```
4 5 1 4 3 6 5 4 6 4 6 1
```

the expression **D > 3** is the list

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

where the **1**s give the positions of the items of **D** which are greater than **3**. Then

```
(D > 3) </. D
```

is the two-item list

| | |
|---|---|
| 4 5 4 6 5 4 6 4 6 | 1 3 1 |

where the first item gives those items of **D** which are greater than **3** and whose second item are those items of **D** which are less than or equal to **3**. The expression

```
(D > 3) #/. D
```

which is equal to **9 3** gives the number of items satisfying each of these relationships.

If **u** is the ten-item list

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

we have that **stem u** is

```
20 10 30 30 10 10 20 20 20 30
```

and **leaf u** is

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

.

The expression

```
(stem u) </. leaf u
```

gives the grouping of the leaves

| | | |
|---|---|---|
| 2 8 1 5 | 4 9 6 | 2 0 1 |

corresponding to the stems

```
20 10 30
```

which are given by the expression

```
~. stem u
```

.

Therefore, we may define the monadic verb

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

which gives the stem-and-leaf diagram for an arbitrary list of data. The expression

```
SLtab u
```

gives the diagram

| | |
|---|---|
| 20 | 2 8 1 5 |
| 10 | 4 9 6 |
| 30 | 2 0 1 |

while

```
SLtab sort u
```

gives the more conventional diagram

| | |
|---|---|
| 10 | 4 6 9 |
| 20 | 1 2 5 8 |
| 30 | 0 1 2 |

with the stems in sorted order.

| | |
|---|---|
| **<. y** | *Floor* of **y** |
| **x \| y** | *Residue* when **x** divides **y** |
| **x u/. y** | ***Key*** (group) **y** by **x** and apply **u** |

A table of stem frequencies rather than stem values is given by

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

and

```
stemfrtab sort u
```

is equal to

```
10 3
20 4
30 3
```

.

The key adverb may be used to give the simple definition

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

for the nub frequencies, and **`fr D`** is **`4 2 2 1 3`**.

### Median and quartiles

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
```

given in the last section, 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 *decrem* 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 to 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 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. Therefore, the first quartile is simply the median of those items less than the median, or

```
median u #~ (median u) > u
```

which is **`19`**. Since the median is the second quartile, we may define the synonym

```
Q2=: median
```

and then define a verb for the first quartile as

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

,

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

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

,

and **`Q3 u`** is **`30`**.

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

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

where the dyadic verb lesser of **`<.`** gives the lesser of its arguments, and thus **`<./`** and **`>./`** give the minimum and maximum items, respectively, of their arguments, and **`five u`** is the five-item list **`14 19 23.5 30 32`**.

| | |
|---|---|
| **`-: y`** | *Halve* **`y`** |
| **`>. y`** | *Ceiling* of **`y`** |
| **`x { y`** | *From* (Item at position **`x`** in **`y`**) |
| **`x <. y`** | *Lesser* of **`x`** and **`y`** |

### Mode

The mode is defined as that item which occurs most frequently, and, for example, for the list **`D`** of dice data which has the value

```
4 5 1 4 3 6 5 4 6 4 6 1
```

the mode is **`4`**. The mode may be found very simply using one of the frequency verbs defined previously.

First we shall define the verb

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

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

```
imax D
```

,

where **`D`** is the list

```
4 5 1 4 3 6 5 4 6 4 6 1
```

of the results of throwing a die **`12`** times, is the list **`5 8 10`** which gives the indices of the maximum item **`6`**. 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, **`D e. 6`** is the list **`0 0 0 0 0 1 0 0 1 0 1 0`** .

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

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

and **mode D** is **4**. 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**.

| | |
|---|---|
| **x e. y** | Is **x** a *member* of **y**? |

### An example

In this section we shall use some of the statistical methods we have developed in previous sections to analyze some data, taken from Sprent (1988), on a company's computer downtime for May 1984. These data are given in Appendix 3 and are represented in **J** as a table **T12** with **95** rows and **4** columns where the rows represent individual breakdowns and the columns represent date, time of breakdown, equipment involved, and duration in minutes, respectively. The type of equipment is coded as follows: **0** No breakdown, **1** CPU, **2** Disk Drive, **3** Graph Plotter, **4** Printer, **5** Service and **6** Tape Reader. The first five rows of the table are

```
1  842 4  49
2 1035 2  18
2 1529 1 123
2 1735 1   3
3    0 0   0
```

which show, for example, that on May 1 there was a Printer breakdown of 49 minutes starting at 8:42, on May 2 there were three breakdowns the first being a Disk Drive breakdown of 18 minutes starting at 10:35 and the second and third being CPU breakdowns beginning at 15:29 and 17:35, and on May 3 there were no breakdowns.

Individual columns of this table may be easily selected, and

```
X=. 3 {"1 T12
```

gives the durations

```
49 18 123 3 0 87 21 7 24 11 ...
```

in the last column, and

```
E=. 2 {"1 T12
```

gives the type of failures

```
4 2 1 1 0 6 4 4 4 4 ...
```

in the third column. The dyadic verb *from* **{** selects from its right argument the item specified by the left argument. The rank conjunction is used to select columns rather than rows of **T12**.

Since a day without an equipment failure is represented by an item in **E** equal to **0**, the corresponding items of the durations list **X** must be removed. This may be accomplished by the expression

```
T=. (E > 0) # X
```

so that **T** has the value

```
49 18 123 3 87 21 7 24 11 19 ...
```

.

The items of **T** may be displayed conveniently as a table with **10** rows and **9** columns by the expression **10 9$T** which has the value

```
 49  18 123   3  87  21   7 24 11
 19 243  38  11  18  32 101  6 32
 27  41   3  25  19 242 122 15  7
  6   2  18  42  24   7   9 14 10
 30  23 141   7 102  83  29 11 15
  7   6   8 121   4  23  42 18 31
 25  42   6 339  19  27  12 11  3
  5   9   3  22  17  20   5  3 18
142  51  18  41   9   9  27 14 33
128  34  18 232 179 143 181  6 14
```

The dyadic verb *shape* **$** reshapes the right argument according to the left argument, and is used here to display the list **T** as a table with **10** rows and **9** columns.

Some very simple **J** expressions may be used to provide summary information about these data. For example, the number of breakdowns is **#T** or **90**, the shortest breakdown is **<./T** or **2** minutes, the longest is **>./T** or **339** minutes, the average time is **am T** or **44.9** minutes, and the median time is **Q2 T** or **19** minutes.

The number of breakdowns longer than one hour, say, is **+/T > 60** or **17** since the expression **T > 60** is equal to

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

where the **1**s correspond to breakdowns longer than one hour. The percentage of these breakdowns is thus

```
100 * (+/T > 60) % #T
```

which has the value **18.89**.

Some indication of the distribution of the breakdown times is given by a stem table which may be found by the expression

```
stemfrtab sort T
```

, a table with **15** rows, which may be conveniently displayed by

```
(8&{.; _7&{.) stemfrtab sort T
```

as

```
  0 24 | 120 4
 10 22 | 140 3
 20 13 | 170 1
 30  7 | 180 1
 40  6 | 230 1
 50  1 | 240 2
 80  2 | 330 1
100  2 |
```

.

A more conventional frequency distribution is given

in the barchart given as Figure 1 in Appendix 2 and has been produced by the classification

```
(ap _0.5 10 35) cfrtab T
```

where **ap** is a general utility verb for arithmetic progressions so that the left argument is the **35**-term arithmetic progression

```
_0.5 9.5 19.5 29.5  … .
```

A frequency table for equipment failures is given by

```
t=. (pos 6) frtab E
```

and the frequencies by **f=. 1{"1 t**. Therefore, the expression

```
5.0 5.0 7.1 ": t,. 100 *
              (+/ %~ ]) 1 {"1 t
```

gives the following three-column table with the type of failure in the first column, frequencies in the second, and percentages in the third:

```
1   19   21.1
2   13   14.4
3    9   10.0
4   29   32.2
5    2    2.2
6   18   20.0 .
```

This shows that there are **19** CPU failures, **13** Disk Drive failures, etc.

A stem-and-leaf diagram for the Printer is

| | |
|---|---|
| 0 | 2 3 6 6 6 7 7 7 7 9 9 |
| 10 | 0 1 4 5 8 9 9 |
| 20 | 1 3 4 5 7 |
| 30 | 0 |
| 40 | 2 2 9 |
| 100 | 2 |
| 140 | 1 |

and is given by

```
SLtab sort P
```

where

```
P=. (E = 4) # X
```

is the list of Printer failures. Thus, there were three failures of **100** minutes or more, and these accounted for

```
100 * (+/(P >: 100) # P) % +/P
```

or **34.7** percent of the Printer downtime.

It may be of interest to see how the number of breakdowns is distributed throughout the week. The dates are given by

```
Date=. 0 {"1 T12
```

which has the value

```
1 2 2 2 3 4 4 4 4 5 5 6 7 8 ... .
```

Now May 1, 1984 was a Tuesday, and since a common convention is to number the days of the week with Sunday as **0**, Monday as **1**, etc., the day numbers are given by

```
Day=. 7 | Date + 1
```

which is

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

and their range by **r=. i. 7** which is

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

Therefore, the frequencies of breakdowns for the days of the week are

```
f=. r fr Day
```

which has the value

```
16 9 23 20 5 9 13
```

so that there were **16** breakdowns on Sundays, **9** on Mondays, etc. The frequency of occurrence of each day of the week is given by

```
n=. (i. 7) fr 7 | 2 + i. 31
```

which is equal to

```
4 4 5 5 5 4 4
```

which means that for May 1984 there were four Sundays, Mondays, Fridays and Saturdays, and five Tuesdays, Wednesdays and Thursdays. Therefore the average number of breakdowns on each day of the week is **f % n** or

```
4 2.25 4.6 4 1 2.25 3.25 .
```

These results may be summarized in the table

```
(5.0 5.0 7.2) ": r ,. f ,. f % n
```

which is equal to

```
0   16   4.00
1    9   2.25
2   23   4.60
3   20   4.00
4    5   1.00
5    9   2.25
6   13   3.25
```

where the first column gives the day number, the second the number of breakdowns, and the third the average number of breakdowns.

| | |
|---|---|
| **x $ y** | Shape **x** *reshape* of **y** |
| **x { y** | Item at position **x** *from* **y** |
| **x {. y** | Size **x** *take* of **y** |
| **x }. y** | Size **x** *drop* of **y** |
| **{. y** | *Head* of **y** (**1 {. y**) |
| **{: y** | *Tail* of **y** (**_1 {. y**) |
| **}.** y | *Behead* **y** (**1 }. y**) |
| **}: y** | *Curtail* **y** (**_1 }. y**) |

Only the first three of these primitives were introduced in this section; the others have been included here to give a more complete list of selection verbs. A negative left argument for **{.** or **}.** selects items from the right end of the right argument so that, for example,

```
(3&{. ; _4&{.) i. 7
```

is

| 0 1 2 | 3 4 5 6 |
|---|---|

,

and

```
(3&}. ; _4&}.) i. 7
```

is

| 3 4 5 6 | 0 1 2 |
|---|---|

.

The monadic verbs in the list give a convenient means of taking or dropping the first or last items of a list.

### Another example

In this section we shall analyze some data on the lengths of program segments and commercial breaks during the showing of a movie on television. In order to conveniently handle the starting and finishing times of the intervals we shall introduce verbs for converting numbers represented in any base to a decimal base.

The verb represented by **#.** has both a monadic and dyadic definition. The monadic form, *base 2*, gives the decimal value of its argument considered as a binary representation, and, for example, **#. 1 1 0 1** is **13**. The dyadic form, *base*, allows conversion from an arbitrary base to decimal and **8 #. 1 2 3** is **83**, the decimal equivalent of the octal number **1 2 3**. The base may be mixed, and, for example,

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

is **19510** which is the number of seconds in **5** hours, **25** minutes and **10** seconds. We shall define the verb

```
tosecs=: 24 60 60&#.
```

to convert from hours, minutes and seconds to seconds, and, for example,

```
tosecs EACH 5 25 10; 5 40 30; 6 0 0
```

is **19510 20430 21600**.

The data we shall analyze represent the beginning and ends of the program intervals and commercial breaks given in hours, minutes and seconds for the James Bond movie *The Spy Who Loved Me*. The data are given as an **18**-item list **T1** which may be displayed by the expression **6 3 $ T1** as

| | | |
|---|---|---|
| 8 51 30 | 9 11 38 | 9 15 45 |
| 9 33 52 | 9 38 20 | 9 50 0 |
| 9 54 10 | 10 5 5 | 10 9 6 |
| 10 27 41 | 10 31 55 | 10 41 5 |
| 10 45 1 | 10 56 47 | 11 0 10 |
| 11 11 58 | 11 16 53 | 11 29 52 |

For example, the movie began at 8:51:30 pm, the first program interval lasted until 9:11:38, the first commercial break until 9:15:45, and so on. These times may be converted to seconds by the expression

```
tosecs EACH T1
```

which is equal to

```
31890 33098 33345 34432 34700 . . .
```

Now the lengths in minutes of the intervals and commercial breaks are given by

```
d=. 60 %~ diff tosecs EACH T1
```

, where the general utility verb **diff** gives first differences, and is equal to

```
20.1333 4.11667 18.1167 4.46667
    11.6667  4.16667 . . .
```

. Since the lengths of the program intervals and commercial breaks have even-numbered and odd-numbered indices, respectively, they may be selected by their 2-residues by the expression

```
'p c'=. (2|i.#d)</.d
```

, where **p** and **c** are the lists of lengths of the intervals and breaks. The values of **p** and **c** rounded to one decimal place are

```
20.1 18.1 11.7 10.9 18.6 9.2 11.8
   1.8 13.0
```

and

```
4.1 4.5 4.2 4.0 4.2 3.9 3.4 4.9
```

, respectively. The total times for programs and commercials are **+/p** or **125.1** minutes and **+/c** or **33.2** minutes, respectively, and the percentage of time for commercials is

```
100 * (+/c) % +/d
```

or approximately **21**.

A similar analysis for another James Bond movie, *Dr. No*, gave the times

```
24.1 22.7 11.4 14.0 9.7 9.0 7.8
                      16.3
```

for program intervals, and

```
4.7 5.1 4.2 4.0 4.2 3.8 3.8
```

for commercial breaks which results again in approximately **21** percent of the time being taken up with commercials.

The monadic verb *do* **".** executes its literal argument, and, for example, the expression **". '3+5'** has the numeric value **8**. This verb may be used together with the dyadic format verb **":** which has been already introduced to round an array of numbers to an arbitrary number of decimal places. For example,

```
". 8.3": 0 1 2 3 % 3
```

is equal to the numeric list

```
0 0.333 0.667 1
```

. The lists **p** and **c** given above were displayed to one

decimal place by the expressions **".6.1":p** and **".6.1":c**, respectively.

| | |
|---|---|
| **#. y** | *Base* **2** of **y** |
| **x #. y** | *Base* **x** of **y** |
| **". y** | *Do* literal **y** |
| **#: y** | *Antibase* **2** of **y** |
| **x #: y** | *Antibase* **x** of **y** |

The verb represented by **#:** is the converse of the base verb. The monadic form, *antibase 2*, gives the binary representation of its argument, and, for example, **#: 13 is 1 1 0 1**, the binary representation of the decimal number **13**. The dyadic form, *antibase*, gives the representation of the right argument to the base given by the left argument. For example, **8 8 8 #: 123** is **1 7 3**, the octal representation of the decimal number **123**, and **24 60 60 #: 19510** is **5 25 10**, the number of hours, minutes and seconds in **19510** seconds.

A verb of considerable usefulness which uses the monadic antibase 2 verb is

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

which gives a truth table, i.e., all possible arguments for a logical function of an arbitrary number of variables. For example, **tt each 0 1 2 3** gives the following four truth tables:

```
┌─┬─┬─────┬─────┐
│0│0│0 0  │0 0 0│
│ │1│0 1  │0 0 1│
│ │ │1 0  │0 1 0│
│ │ │1 1  │0 1 1│
│ │ │     │1 0 0│
│ │ │     │1 0 1│
│ │ │     │1 1 0│
│ │ │     │1 1 1│
└─┴─┴─────┴─────┘
```

A partial truth table, given by

**ptt=: ([ e.~ +/"1 @ tt@]) # tt@]** ,

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

```
2 3 ptt 3
```

is

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

### Variance

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 square root of the variance is known as the standard deviation.

The deviations of the items of the list

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

from the mean, which is equal to **4.2**, are given by

```
w - am w
```

or

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

The above 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 which has been mentioned earlier in connection with forks and trains. Thus we may define the verb

```
dev=: - am
```

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

```
ssp=: +/ @ (*&dev)~
```

when used monadically with a single argument gives the sum of squares of deviations from the mean, and **ssp w** is **7.98**. Thus the variance is given by

```
var=: ssp % <:@#
```

and **var w** is **2.66**.

Finally, the standard deviation is

```
sd=: %: @ var ,
```

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

| | |
|---|---|
| **%: y** | *Square root* of **y** |
| **u @: v** | **A**t (Same as **@** except ranks are infinite) |
| ***: y** | Square of **y** |

The sum of squares of deviations from the mean may also be defined as

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

where **@:** is the conjunction *at* which is required so that the sum **+/** is applied over the entire list of deviations rather than to each item of the list.

The monadic hook **(g h)y** is equal to **y g (h y)**, and the dyadic hook **x(g h)y** is **x g (h y)**. The verb for deviations from the mean may also be written as

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

which is a fork followed by a hook.

The following verbs represent definitions using forks of the verbs in this section except **ssp**:

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

### Summary statistics

Some of the statistics discussed in the previous sections may be combined in the following explicit monadic verb **summary** to calculate and display the statistics with suitable labels for a given list argument:

```
summary=: 3 : 0
r=.  'Sample size      ',5.0 ": #y.
r=. r,: 'Minimum          ', 8.3 ": <./y.
r=. r, 'Maximum          ',8.3": >./y.
r=. r, 'Arithmetic mean  ',8.3": am y.
r=. r, 'Variance         ',8.3": var y.
r=. r, 'Standard deviation',8.3": sd y.
r=. r, 'First quartile   ',8.3": Q1 y.
r=. r, 'Median           ',8.3": Q2 y.
r=. r, 'Third quartile   ',8.3": Q3 y.
r=. r, 'Geometric mean   ',8.3": gm 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.**.

For the dice data **D** we have that **summary D** is

```
Sample size         12
Minimum              1.000
Maximum              6.000
Arithmetic mean      4.083
Variance             2.992
Standard deviation   1.730
First quartile       1.000
Median               4.000
Third quartile       6.000
Geometric mean       3.560
```

| | |
|---|---|
| **x ,: y** | *Laminate* **x** to **y** giving a 2-item array |
| **x ": y** | *Format* **y** according to **x** |

To illustrate some of the main features of explicit definition we shall define the following three very simple verbs **f1**, **f2** and **f3**:

```
f1=: 3 : 0  f2=: 3 : 0  f3=: 3 : 0
% y.        :           % 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** is divide so that **15 f2 6** is **2.5**. The verb **f3** is ambivalent so that it gives the reciprocal when used monadically and the quotient when used dyadically, i.e., **f3 2.5** is **0.4** and **15 f3 6 is 2.5**.

The first line in each definition gives the name and specifies the definition of a verb. 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.

The verbs **frtab** and **nubtab** introduced earlier for a frequency table over a specified range and for a frequency table over the nub may be combined into the single verb

```
frtable=: 3 : 0
nubtab y.
:
x. frtab y.
)
```

Therefore for the dice data **D** and the range **r**, which is equal to the list **1 2 3 4 5 6**, the expression

```
frtable D
```

is equivalent to

```
nubtab D
```

and the expression

```
r frtable D
```

is equivalent to

```
r frtab D  .
```

### Covariance and correlation

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

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

and

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

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

The sum of products of the deviations of two sets of observations from their means is given by the dyadic use of the ambivalent verb

```
ssp=: +/ @ (*&dev)~
```

which has been used monadically in an earlier section to give the sum of squares of observations on a single variable, and **x1 ssp y1** is **103.68**. The covariance is given by

```
cov=: ssp % <:@#@]
```

and **x1 cov y1** is **17.28**, and the correlation coefficient by

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

and **x1 cor y1** is **0.9724083**.

Variance-covariance and correlation tables for an arbitrary number of variables are given very simply by the verbs

```
covtab=: cov Each /~
```

and

```
cortab=: cor Each /~
```

respectively. As an example we shall use the following data from Searle (1966) which represent six observations on each of three variables given as the three-item list **d**:

```
d=. 1 4 2 2 1 3;0 6 4 3 1 5;
         10 17 13 14 12 15
```

The variance-covariance table is given by

```
10.5": covtab d
```

which is

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

where the items on the main diagonal represent the variances of the three variables and the off-diagonal items represent the covariances. The correlation table is given by

```
10.5": cortab d
```

which is

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

### Linear regression

To introduce a simple regression example we shall use the data

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

and

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

on crop yield and the amount of water applied in the previous section. Then

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

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

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

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 **y1**. 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 general 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 verb **SR** given below finds the regression coefficients and some of the statistics required to test for the significance of the linear regression. 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 so that it is available outside of the definition of **SR**. The following is the definition of **SR**:

```
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
)
```

The expression **x1 SR y1** 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**, giving the values of the independent variable and the observed and estimated values of the dependent variable. Figure 2 in Appendix 2 shows the observed data and the regression line.

| | |
|---|---|
| **x %. y** | *matrix divide* |

The monadic explicit verb **REG** given in Appendix 4 provides for linear regression models with an arbitrary number of independent variables For the above data the expression **REG x1;y1** gives the following results:

```
          Var.     Coeff.      S.E.       t
           0     3.99429     0.35656   11.20
           1     0.10286     0.01104    9.32

Source     D.F.    S.S.        M.S.       F
```

```
Regression    1    10.66423    10.66423  86.87
Error         5     0.61377     0.12275
Total         6    11.27800

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

If there was an additional independent variable, **x2**, say, then the argument for **REG** would be **x1;x2;y1**.

### Analysis of variance I - Sums of squares

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

The marginal sums over one or more axes in an arbitrary array are given by the general utility verb **msum** whose left argument gives the axis or axes being summed over and whose right argument is the array. The verb **mnum** with the same syntax gives the number of items entering into the sum. For example, if **D2** is the array

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

, then **0 1 msum D2** gives the column sums **15 16 15 17** and **1 0 msum D2** the row sums **22 24 17**, and **0 1 mnum D2** or **3** and **1 0 mnum D2** or **4** are the numbers of items being summed to give each of these margins. Also **0 0 msum x** is the sum **63** of the **0 0 mnum x** or **12** items in the array. Finally, the expression **1 1 msum D2** is the array **D2** with each item being the sum of **1 1 nsum D2** or **1** item.

The signed weighted sum of squares is given by the verb

```
T=. msgn@[ * mnum %~
   [: +/ [: , [: *: msum
```

where the arguments are the same as for the two verbs discussed in the last paragraph. The verb

```
msgn=. [: _1&^ ~:/
```

gives the sign which is positive or negative according as the left argument of **T** has an even or odd number of **1**s. We note the dyadic verb *not-equal* **~:** and its use in the derived verb **~:/** whose result is **0** or **1** according as the number of **1**s in its logical list argument is even or odd. We also note the dyadic verb *power* **^** so that the expression **_1&^** gives negative one raised to the power of its right argument. (The sign is used in the verb **AOV** introduced in the next paragraph in the calculation of the sums of squares for the main effects and interactions.) The weighted sums of squares, given by **0 1 T D2**, **1 0 T D2**, **0 0 T D2** and **1 1 T D2**, are **_331.667**, **_337.25**, **330.75** and **383**, respectively.

The ambivalent verb **AOV** given in Appendix 4 may be used for analysis-of-variance calculations for factorial experiments with an arbitrary number of factors. The right argument gives the array of data and the left argument, if there is one, the main effect and interaction terms for which the degrees of freedom, sums of squares and mean squares are required. If the left argument is omitted, all main effect and interaction terms are given. The array **D3** whose value is

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

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

and whose shape is **$D3** or **2 4 6** represents the results of a two-factor experiment with **2** levels of the first factor, **4** levels of the second factor and **6** replications. The following are two examples of the use of **AOV** with these data:

```
   AOV D3
A         1   252.08333   252.08333
B         3    69.16667    23.05556
C         5    74.66667    14.93333
AB        3    10.41667     3.47222
AC        5    51.91667    10.38333
BC       15   309.83333    20.65556
ABC      15  1495.58333    99.70556
Total    47  2263.66667

   'A B AB C' AOV D3
A           1   252.08333   252.08333
B           3    69.16667    23.05556
```

```
AB          3     10.41667      3.47222
C           5     74.66667     14.93333
Error      35   1857.33333     53.06667
Total      47   2263.66667
```

The analysis-of-variance table with the degrees of freedom, sums of squares and mean squares is given as a three-column numeric table as the global variable **AOVtable**.

| | |
|---|---|
| **x ~: y** | Is **x** *not-equal* to **y**? |
| **x ^ y** | **x** to *power* **y** |
| **x I } y** | **y** ***amended*** at **I** by **x** |

As a convenience for assembling and modifying data in an array of arbitrary dimensions for the verb **AOV** we shall give the ambivalent verb **AOVa** whose right argument is a list of data each with the corresponding indices in the array and whose left argument, if there is one, is the array being modified. If there is no left argument, then the right argument gives all of the items in the array. The result is the required array. First we shall introduce the dyadic adverb *amend* **}** which allows specified items of an array to be modified.

As a first example, suppose that the second and fourth items, i.e., those items with indices **1** and **3**, in the list **w** which has the value

```
2.3 5 3.5 6 ,
```

are incorrect. Then they may be replaced by the correct values, **5.2** and **6.1**, say, by the statement

```
w=. 5.2 6.1 (1 3) } w
```

so that **w** now has the value

```
2.3 5.2 3.5 6.1 .
```

As another example, the expression

```
'C' 0 } 'cat'
```

is equal to **Cat**. Finally for the table **D2** which has the value

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

the expression

```
D2x=. 12 10 (1 0;2 2) } D2
```

will give the modified array **D2x** with the value

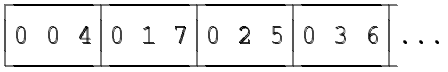
```
 4 7  5 6
12 4  3 8
 2 5 10 3
```

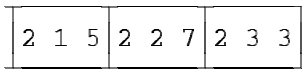
and leave the value of **D2** unchanged.

Now define the ambivalent verb

```
AOVa=. 3 : 0
(0 $~ >: >./ }: EACH y.) AOVa y.
:
({:EACH y.)(}:each y.)}x.
)
```

where the right argument is a list of array indices and corresponding values of the array items. When used monadically, **AOVa** gives the array specified by the right argument with any missing items given as **0**s in the result. When used dyadically, the list right argument specifies changes in the array right argument. For example, the list




gives all of the items of the array **D2**, and **AOVa D2a** is the array **D2**. The statement

```
D2x=. D2 AOVa 1 0 12;2 2 10
```

defines **D2x** given in the last paragraph. Finally we might note that the expression

```
AOVa 1 0 12;2 2 10
```

gives the array

```
 0 0  0
12 0  0
 0 0 10 .
```

### Analysis of variance II - Orthogonal contrasts

We shall consider briefly the partition of a main effect or interaction sum of squares into orthogonal linear, quadratic, cubic, etc. components each representing a single degree of freedom. As a first example we shall consider the following data, given here as the table **D2b**, from Steel and Torrie (1960) which give the seed yield of soybeans in bushels per acre for six replications of each of five row spacings:

```
33.6 31.1 33.0 28.4 31.4
37.1 34.5 29.5 29.9 28.3
34.1 30.5 29.2 31.6 28.9
34.6 32.7 30.7 32.3 28.6
35.4 30.7 30.7 28.1 29.6
36.1 30.3 27.9 26.9 33.4
```

The treatment sum of squares may be found to be **125.66133** with **4** degrees of freedom giving a mean square of **31.41533** which is significant at the 1% level. It is this sum of squares which we shall partition into linear, quadratic, cubic and quartic components each with a single degree of freedom.

The orthogonal polynomial coefficients required for from **2** to **6** treatments are given by the items of the five-item list **OP**. Those required for the **5** treatments in this example are given by

```
p=. >3{OP
```

which is equal to

```
 2 _1  0  1 2
 2 _1 _2 _1 2
_1  2  0 _2 1
 1 _4  6 _4 1
```

where those in the first row are for the linear component, those in the second row for the quadratic component, etc. We shall require the sums of squares of the rows of this array which are given by

```
   k=. +/"1 *: p
```

and are equal to

```
   10 14 10 70 .
```

The treatment sums are given by

```
   t=. 0 1 msum D2b
```

and are

```
   210.9 189.8 181 177.2 180.2 .
```

Now the weighted treatment sums are given by

```
   q=. p mp t
```

where **mp** is the utility verb for the matrix product, and are

```
   _74 53.2 _5.5 9.1 .
```

Finally the single-degree-of-freedom components are given by

```
   (*: q) % 6*k
```

where the multiplier **6** is the number of replications, and are

```
   91.2667 33.6933 0.504167 0.197167 .
```

We note that their sum is **125.661** as would expected.

A similar calculation may be used to partition the sums of squares for interaction terms provided that the appropriate direct product of orthogonal coefficient tables is used. As an example consider partitioning the three degrees of freedom of the AB-interaction sum of squares which was equal to **10.41667** of the example of the previous section. Since there are two levels of the first factor and four of the second, the two tables of orthogonal polynomial coefficients are given by

```
   'm2 m4'=. > each (0&{ ; 2&{) OP
```

so that **m2** is

```
   _1 1
```

where **$m2** is **1 2**, and **m4** is

```
   _3 _1  1 3
    1 _1 _1 1
   _1  3 _3 1
```

where **$m4** is **3 4**. The direct product of **m2** and **m4** is

```
   p=. m2 dp m4 ,
```

where **dp** is the utility verb for direct product, and **p** has the value

```
    3  1 _1 _3 _3 _1  1 3
   _1  1  1 _1  1 _1 _1 1
    1 _3  3 _1 _1  3 _3 1 .
```

The following sequence of calculations will give the desired partition of sum of squares into single-degree-of-freedom components:

```
   k=. +/"1 *: p
   k
40 8 40
   t=. 1 1 0 msum D3
   t
 83 68  81 75
107 94 118 98
   t=. ,t
   t
83 68 81 75 107 94 118 98
   q=. p mp t
   q
8 _16 _34
   (*: q) % 6*k
0.266667 5.33333 4.81667
   +/(*: q) % 6*k
10.4167
```

### Analysis of variance III - Unequal numbers

In some experiments the number of replications may not be the same for all treatments. This may be because of the loss of data or to the lack of an appropriate design. In this section we shall give a verb for a one-way analysis of variance with unequal numbers for the various treatments.

We shall use 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.
   SStr=. (+/ wsqs EACH D) -
                        CT=. wsqs ;D
   SStot=. (+/ *: ;D) - CT
   SSerr=. SStot - SStr
   SS1=: SStr,SSerr,SStot
   DFerr=. (DFtot=. <:#;D) -
                        DFtr=. <:#D
   DF1=: DFtr,DFerr,DFtot
   MS1=: (SStr,SSerr) % DFtr,DFerr
   SS1,.DF1,.MS1,0
   )
```

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, and mean squares.

As an example we shall use the following data given by the variable **D2c** from Bennett and Franklin (1954) giving observations from an experiment with three different treatments:

| | |
|---|---|
| 84 60 40 47 34 | 67 92 95 40 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 98 | 60 | 59 | 108 | 86 | 46 | 93 | 100 |

The analysis of varaince given by **aov1 D2c** is given by the following table:

```
2329.1  2 1164.55
7398.67 14 528.476
9727.76 16       0
```

The verb **aov1** may be used, of course, for a one-way analysis of variance with equal treatment numbers, and for the sample data **D2** given previously we would have that **aov1 <"1 D2** would be equal to

```
  6.5  2    3.25
45.75  9 5.08333
52.25 11       0
```

which agrees with the results

```
A        2     6.50000     3.25000
B        3     0.91667     0.30556
AB       6    44.83333     7.47222
Total   11    52.25000
```

which are obtained with the expression **AOV D2**.

### Probability

The sample space for a random experiment is a representation of the various outcomes of the experiment. For example, if a coin is tossed once and the occurrence of a head or tail is observed, then the sample space could be represented by the symbols *H* and *T*. If the coin is tossed twice, then the sample space could be represented by *HH*, *HT*, *TH* and *TT*. If the random experiment consisted in rolling two dice and finding the sum of the two numbers which appeared, then the sample space could be represented by the integers *2, 3, …, 12*.

Now if the two-item list **c=. 'TH'** represents the sample space for tossing one coin, then the sample space for tossing two coins is given by the expression **{c;c** which has the value

| | |
|---|---|
| TT | TH |
| HT | HH |

A more convenient representation as a list is given by **,{c;c**, where **,** is the mondic verb *ravel*, and is

| | | | |
|---|---|---|---|
| TT | TH | HT | HH |

Similarly, the sample space for tossing three coins is given by **,{c;c;c** and is

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TTT | TTH | THT | THH | HTT | HTH | HHT | HHH |

We may use the expression

```
'H'&= each ,{c;c;c
```

to give the equivalent numerical representation

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |

The expression

```
+/ EACH 'H'&= each ,{c;c;c
```

has the value

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

which gives the number of heads which occur for each of the eight possible outcomes. Now suppose that the coin is slightly biased so that the probability of a head on a single toss is **0.6** and the probability of a tail is **0.4**. Then the corresponding probabilities for the outcomes are given by

```
*/ EACH ,{p;p;p=. 0.4 0.6
```

which is equal to

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

Thus the probability of **3** tails is **0.064**, **2** tails followed by **1** head is **0.096**, etc. We note that

```
+/*/ EACH ,{p;p;p
```

is equal to **1**.

These results may be summarized very simply. If the range of the number of heads is given by the list

```
heads=. 0 1 2 3 ,
```

then the frequency of the number of heads is

```
num=. heads fr +/EACH
              'H'&=each,{c;c;c
```

which has the value **1 3 3 1**. Now we may find the probabilities associated with the frequencies by

```
prob=. key +//. */ EACH ,{p;p;p
```

which is equal to

```
0.064 0.288 0.432 0.216 ,
```

where

```
key=. +/EACH'H'&=each,{c;c;c
```

is the list

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

of the number of heads associated with each item in the sample space. Finally the expression

```
heads,.num,.prob
```

gives the table

```
0 1 0.064
1 3 0.288
2 3 0.432
3 1 0.216 .
```

Such a distribution with a fixed number of trials and a constant probability of a success in each trial is known as a binomial distribution which will be discussed further in the next section.

The items in the second column in the table at the end of the last paragraph are known as binomial coefficients since they occur as the coefficients in the expansion of $(x+y)^n$. They may be generated very simply by the dyadic verb *out of* **!** which gives combinations, and, for example, **3!5** or **10** is the number of combinations of **5** things taken **3** at a time. The monadic verb

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

where **ei** is a general utility verb which gives the non-negative extended integers up to its argument, gives the binomial coefficients for a specified argument. For example, **ei 3** is the list **0 1 2 3**, and **bc 3** is **1 3 3 1**. Binomial coefficients are often shown in a triangular display known as Pascal's triangle, which may be given by the verb

```
pascal=: !~/~@ei
```

and, for example, **pascal 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
```

Now consider drawing two balls with replacement from an urn which contains three red, two green and three black balls. We shall find the probability that at least one of the balls is red. The sample space is given by **,{;~'rgb'** which is

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| rr | rg | rb | gr | gg | gb | br | bg | bb |

The corresponding list of probabilities, say **pr**, is

```
*/ EACH ,{p;p=. 3 2 5 % 10
```

and is equal to

```
0.09 0.06 0.15 0.06 0.04 0.1 0.15
           0.1 0.25
```

and **+/pr** is **1**. Now the expression

```
+./ EACH 'r'&= each ,{;~'rgb'
```

say **m**, where **+.** is the dyadic verb *or* which gives the logical sum of its arguments, is the list

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

where the **1**s indicate those items in the sample space containing at least one **1**. Therefore, the probabilities associated with the items in the sample space corresponding to at least one red ball being drawn are given by **m # pr** which is the list

```
0.09 0.06 0.15 0.06 0.15
```

and the required probability of drawing at least one ball is **+/m # pr** or **0.51**.

The definition of the dyadic verb **+.** introduced in the last paragraph for the logical sum of its logical arguments, i.e., with values restricted to **0** and **1**, may be given in the form of a table by the expression

```
+. table ~ 0 1
```

whose value is

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

Thus the logical sum is **1** if at least one of the arguments is **1** and is **0** if both are **0**. The dyadic verb *and* ***.** gives the logical product of its arguments whose result is **1** if both arguments are **1** and **0** otherwise. Its definition is given by the expression

```
*. table ~0 1
```

and is

| | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

(For arguments not restricted to the values **0** and **1** these two verbs represent greatest common divisor and least common multiple, respectively. However these definitions need not concern us here.) Finally we shall introduce the monadic verb *not* **-.** which for a logical argument gives its negation and thus **-. 0 1** is **1 0**. For an argument which is a probability the result is the complementary probability, and, for example, **-. 0.75** is **0.25**.

| | |
|---|---|
| **, y** | *Ravel* items of **y** |
| **x ! y** | *Out of* (Combinations) |
| **x +. y** | *Or* (Logical sum) |
| **x *. y** | *And* (Logical product) |
| **-. y** | *Not* |

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

dice as there are with one die. We shall conclude this section with a brief look at this problem, the solution of which by Pascal is said to mark the beginnings of the theory of probability.

If we let **f=. 1 2 3 4 5 6** represent the six faces on a die. then the sample space for throwing a die four times is

```
S=. ,{f;f;f;f
```

which is a list with **#S** or **1296** items, the first four of which are

| | | | |
|---|---|---|---|
| 1 1 1 1 | 1 1 1 2 | 1 1 1 3 | 1 1 1 4 |

.

Now the expression

```
+./ EACH 6&= each S
```

gives a list of the same length with **1**s corresponding to those items of **S** having at least one **6**. For example, the first **15** items of this list are

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

where the **1**s in the sixth and twelfth positions correspond to the two items **1 1 1 6** and **1 1 2 6** of **S**. The total number of items in **S** having at least one **6** is

```
+/ +./ EACH 6&= each S
```

or **671**. Therefore, the probability of getting at least one six in four rolls of a single die is **671%1296** or **0.5177469**. This supports the conjecture that it is reasonable to bet even money on rolling at least one 6 in four rolls of a single die.

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

```
PR=: [: -. -.@[ ^ ]
```

where the left argument gives the probability of success in a single of trial and the right argument is the number of trials. For example, the list of probabilities of at least one success in rolling a single die **1**, **2**, **3** or **4** times is

```
(%6) PR 1 2 3 4
```

which has the value

```
0.166667 0.305556 0.421296
   0.517747 .
```

The probabilities of getting at least one double **6** on **24** and **25** rolls of two dice are

```
(%36) PR 24 25 ,
```

and are

```
0.491404 0.505532,
```

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

### Discrete probability distributions

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

```
binomial =: 3 : 0
:
'n p'=. x.
x=. y.
(x!n) * (p^x) * (-.p)^n-x
) .
```

The binomial probabilities found in a previous section with **3** trials and probability **0.6** success in a single trial may be found by the expression

```
3 0.6 binomial 0 1 2 3
```

and are

```
0.064 0.288 0.432 0.216 .
```

The binomial distribution assumes a number of independent trials with the probability of success remaining constant throughout. The hypergeometric distribution, however, 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, it may be shown that the probability of drawing *x* red balls is

${}^mC_x \, {}^nC_{k-x} / {}^{m+n}C_k$ ,

where $x = 0, 1, 2, ..., k$. A corresponding verb in **J** is

```
hg=: 3 : 0
:
'm n k'=. x.
x=. y.
(x!m) * ((k-x)!n) % k!m+n
)
```

For example, if there are **4** red balls and **6** black balls and **3** balls are drawn without replacement, then the probabilities for **0**, **1**, **2** and **3** red balls are given by

```
4 6 3 hg 0 1 2 3
```

and are

```
0.1666667 0.5 0.3 0.03333333 .
```

The Poisson distribution is another common discrete distribution and 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 $\lambda$, then the probability of *x* successes, where *x* is a non-negative integer, is $e^{-\lambda}\lambda^x/x!$. The probability density function is

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

where **!** is the monadic verb *factorial* and **^** is the monadic verb *exponential* and the dyadic verb *power*, and, for example, the first five probabilities for a

distribution with mean **1.5** are given by the expression

```
1.5 poisson 0 1 2 3 4
```

which has the value rounded to five decimal places of

```
0.22313 0.33470 0.25102 0.12551
                0.04707 .
```

The probability of first success in a sequence of binomial trials is given by the geometric distribution. Since the probability of the first success occurring on the *n*th trial with probability *p* of success in a single trial is equal to $(1-p)^{n-1}p$, for *n* a positive integer, we may define

```
geometric=: [ * -.@[ ^ <:@] .
```

For example, for a probability **0.4** of success on a single trial the first six probabilities are given by

```
0.4 geometric 1 2 3 4 5 6
```

and are equal to

```
0.4 0.24 0.144 0.0864 0.05184
                0.031104 .
```

The expression

```
+/\0.4 geometric 1 2 3 4 5 6 ,
```

where **** is the monadic adverb *prefix*, gives the cumulative sums

```
0.4 0.64 0.784 0.8704 0.92224
                0.953344
```

of these probabilities, and we see that the probability is approximately **0.95** that the first success will occur at least by the sixth trial.

The prefix adverb 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 .
```

Of course, these factorials are given more simply as **!pos 5**.

The dyadic adverb *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
```

| | |
|---|---|
| **! y** | *Factorial* of **y** |
| **^ y** | *Exponential* (base e) of **y** |
| **u\ y** | Apply **u** to ***prefixes*** of **y** |
| **x u\ y** | Apply **u** to ***infixes*** of length **x** of **y** |

Kendall and Stewart (1963) gives the the following frequencies of occurrence of 0, 1, 2, ..., 13 cards of a certain suit in 3400 hands of whist:

35 290 696 937 851 444 115 21 11 0 0 0 0 0

The expected frequencies are those given by a hypergeometric distribution with $m = 13$, $n = 39$ and $k = 13$, and are thus equal to

```
3400 * 13 39 13 hg i. 14
```

which rounded to one decimal place are

```
43.5 272.2 700.0 973.5 811.3 424.0
   141.3 30.0 4.0 0.3 0.0 0.0 0.0
                          0.0
```

The authors state that the agreement between the observed and expected values is reasonably good.

The classic example of the Poisson distribution, given in Weaver (1963) and in many other texts, is the number of deaths which occurred from 1875 to 1894 in various German army corps due to kicks from horses, and are given by the list **h** which may be displayed by the expression **1":14 20$h** as

```
02210011030210010101
00020302000111020310
00020200110021100200
00011120200010121000
01011110000100001100
00002100100101111110
00102002001131110300
10100010110020021020
10001001000010001101
00000211102110120100
00110102020000213011
00002401301111213131
11211304010321021100
01000001011000220000 .
```

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, inclusive, 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,

inclusive, is given by **`0.7 poisson i. 6`** which is

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

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

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

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

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

which has the value

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

As an example of the Poisson approximation to the binomial distribution consider some data from Feller (1950) which give the number of occurrences of the pair (7,7) among the first 5000 pairs (10 000 digits) of a random number table. The observed frequency distribution is given in the first two columns of the table

```
0   41    36.6    36.8
1   34    37.0    36.8
2   16    18.5    18.4
3    8     6.1     6.1
4    0     1.5     1.5
5    1     0.3     0.3
6    0     0.0     0.1
7    0     0.0     0.0
8    0     0.0     0.0
9    0     0.0     0.0
```

and shows that in **`41`** groups the pair did not occur, in **`34`** groups the pair occurred once, etc. Now the occurrence of the pair (7,7) may be considered as a sequence of **`100`** binomial trials with probability **`0.01`** of success in a single trial. Therefore, the binomial probalilities for **`0`**, **`1`**, …, **`9`** successes are given by

**`100 0.01 binomial i. 10`** ,

and the expected binomial frequencies shown in the third column are **`100`** times these probabilities. The frequencies expected by a Poisson approximation are

**`100 * 1 poisson i. 10`**

and are given in the last column.

### Three problems

In this section we shall discuss three well-known problems in probability. The first is the problem of coincidences in random events such as drawing numbered balls from an urn or dealing cards; the second is the birthday problem of two or more persons in a group having the same birthdays; and the third is the coupon collector's problem of collecting a complete set of coupons or prizes which are included one at a time in some product. We shall be concerned only with the computational aspects of these problems. The theoretical derivations, which are not that difficult, may be found in many textbooks on probability theory, for example, Feller (1950).

The problem of coincidences was first proposed and solved by the French mathematician Pierre de Montmort in 1708. One way of describing it is in terms of an urn model with an urn containing $n$ balls numbered 1, 2, …, $n$, respectively. If the balls are drawn one at a time at random without replacement, what is the probability that no ball is drawn in the order of its number? In other words, what is the probability that the ball numbered 1 is not drawn on the first draw, and the ball numbered 2 is not drawn on the second draw, and so on? Montmort showed that the required probability for $n \geq 2$ balls is the sum of the first $n-1$ terms of the series

$1/2! - 1/3! + 1/4! - 1/5! + \ldots$ .

For example, the probability of no coincidences for two balls is 1/2! or 0.5, the probability of no coincidences with three balls is 1/2! – 1/3! or 0.33333, and the probability of no coincidences with four balls is 1/2! – 1/3! + 1/4! or 0.37500.

We may also define the probability of no coincidences for $n \geq 0$ balls as the sum of the first $n+1$ terms of the series

$1/0! - 1/1! + 1/2! - 1/3! + 1/4! - 1/5! + \ldots$ .

Therefore, the probability of no coincidences with 0 balls is the first term or 1 which is reasonable, and the probability for 1 ball is the sum of the first two terms or 0 which is also reasonable. The probabilities for 2 or more balls are the same as with the previous formula since the first two terms contribute zero to the corresponding sums. This series is the expansion of $1/e$ which is approximately 0.3678794 so that the probability of no coincidences approaches this value asymptotically as the number of balls increases.

The sum of a series such as that in the last paragraph in which the signs of the terms are alternately + and – is called an alternating sum. Therefore, the probability of no coincidences with $n \geq 0$ balls can be defined simply as "the alternating sum of the reciprocals of the factorials of the first $n+1$ non-negative integers". Since the alternating sum is given in **J** by the derived verb **`-/`**, an expression for this probability is given by **`-/%!i.>:n`** for a non-negative value of **`n`**. Therefore, we may define the verb

**`coinc=: [: -/ [: % [: ! ei`** ,

where **`ei`** is the general utility verb for extended integers, for the probability of no coincidences, and, for example, **`coinc 5`** is **`0.3666667`**, the probability for **`5`** balls.

Also we have that **coinc EACH ei 10** gives the probabilities

```
1 0 0.5 0.3333333 0.375 0.3666667
   0.3680556 0.3678571 0.3678819
   0.3678792 0.3678795
```

for from **0** to **10** balls, inclusive. We may see from this list that the probability converges quite rapidly to the limiting value of **^_1** or **0.3678794**. Indeed, the probability of no coincidences with four balls differs from the probability of no coincidences with ten balls by about two per cent. We shall discuss the alternating sum **-/** further at the end of this section.

The birthday problem is to find the probability that for a group of *n* persons there at least two persons who share the same birthday. It is relatively simple to show that this probability is given by

$1 - (365 \cdot 364 \cdots (365 - n + 1))/365^n$

so that a corresponding verb may be written as

```
bd=: [: -. ([: */ [: 365&- i.) %
                          365&^  .
```

For example,

```
bd EACH 5 10 15 20
```

is

```
0.027 0.117 0.253 0.411 .
```

Since **bd 22** is **0.478** and **bd 23** is **0.507**, we see that if there are twenty-three persons in a room, there is a better than even chance that at least two persons have a birthday on the same day. At the end of this section we shall discuss the calculation of these probabilities for large values of *n* and give a more accurate verb for these calculations.

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 may be shown to be "*n* times the sum of the reciprocals of the first *n* positive integers". For example, if there are 5 prizes, then the average number of purchases is 5 times the sum 1 plus 1/2 plus 1/3 plus 1/4 plus 1/5, or 5 times 2.28333, or approximately 11.4. Therefore, on the average 11 or 12 purchases are required to collect all 5 prizes. These calculations may be represented in **J** by the expression

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

which has the value **11.4167**. In general, if there are **n** prizes, where **n** is a non-negative integer, the expression for the average number of purchases is

```
n * +/ % pos n
```

which for **n** equal to **10**, say, is approximately **29.3**.

A verb to give the expectation for an arbitrary number of prizes is given by

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

and, for example, **cc 5** is **11.4167** and **cc 10** is **29.2897**. We note the very reasonable results that **cc 1** is **1** and **cc 0** is **0** since if there is only one prize it is obtained with the first purchase and if there are no prizes there is nothing to purchase.

Now let us define the lists

```
rows=. 5 * i. 10
```

and

```
cols=. pos 5 ,
```

which will have the values

```
0 5 10 15 20 25 30 35 40 45
```

and

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

respectively. Then the expression

```
rows (".@(6.0&":)@cc@+) table cols
```

will give the following table with the expected number of purchases rounded to the nearest integer for **1** to **50** prizes, inclusive:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 1 | 3 | 6 | 8 | 11 |
| 5 | 15 | 18 | 22 | 25 | 29 |
| 10 | 33 | 37 | 41 | 46 | 50 |
| 15 | 54 | 58 | 63 | 67 | 72 |
| 20 | 77 | 81 | 86 | 91 | 95 |
| 25 | 100 | 105 | 110 | 115 | 120 |
| 30 | 125 | 130 | 135 | 140 | 145 |
| 35 | 150 | 155 | 161 | 166 | 171 |
| 40 | 176 | 182 | 187 | 192 | 198 |
| 45 | 203 | 209 | 214 | 219 | 225 |

| | |
|---|---|
| _ | *Negative sign/Infinity* |
| __ | *Negative infinity* |
| _. | *Indeterminate* |
| _: | *Infinity* (constant) |

We first used the insert adverb **/** in the Introduction to derive the verb **+/** to give the sum of its list argument, and, for example, **+/pos 7** is

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

which, by the definition of insert, is

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

or **15**. In this section we defined the alternating sum **-/** for the evaluation of the series in Montmort's problem. Since the value of any specific alternating sum may at

first be puzzling, e.g., **-/pos 5** has the value **3**, as it depends on the precedence rules in **J**, we shall give the following step-by-step evaluation of this expression:

```
-/pos 5
-/1 2 3 4 5
1 - 2 - 3 - 4 - 5
1 - 2 - 3 - _1
1 - 2 - 3 + 1
1 - 2 - 4
1 - _2
1 + 2
3
```

The alternating sum is also given as the sum of the even-indexed items minus the sum of the odd-indexed items, so that the value of **-/pos 5** is

```
(1 + 3 + 5) – (2 + 4)
```

which is **3**.

The alternating sum could have been used in the earlier example with the lengths of program intervals and commercial breaks to give a meaningful although possibly not very interesting result. For example, in the first set of data the value of the list **d** of interval lengths in minutes was

```
20.1333 4.11667 18.1167 4.46667
      11.6667  4.16667 . . .
```

with the items being alternately program intervals and commercial breaks. Therefore the alternating sum **-/d** which has the value **91.9** would be the excess time in minutes of total program time over total commercial time. It could also be found using the previous computations as **(+/p) -+/c**.

For an increasingly large group of persons the probability of at least two persons sharing the same birthday gets closer and closer to 1. For example, for 50, 75 and 100 persons these probabilities are 0.9703736, 0.9997199 and 0.9999997, respectively, so from practical considerations it is almost a certainty that in any group of fifty or more persons at least two will have the same birthday. The expression for the probability is valid for any number of persons not greater than 365 and so should the verb **bd** which is based upon it. Therefore, the expression **bd n** should give valid results, albeit indistinguishable from **1**, for increasingly large values of **n**. This is indeed true until we evaluate **bd 125** which has the value **_.** which is an indication of an indeterminate result. The reason for this is not difficult to see since the verb **bd** requires the evaluation of a numerator

```
*/365 – i. n
```

and a denominator

```
365^n
```

where **n** is the value of the argument. Now for **n** equal to **124** the numerator is **2.5607e307**, a very large number indeed, and for **n** equal to **125** the numerator is _ which is the representation of infinity. Thus it is reasonable that the value of **bd 125** is indeterminate. This indeterminacy may be removed by expressing the quotient as

```
*/1 - 365&(%~)i. n
```

or more simply as

```
*/-. 365&(%~)i. n
```

so that that there is an item-by-item division by **365** before the multiplication is performed. Thus we may define the verb

```
bd1=: [: -. [: */[: -.
                    [: 365&(%~) i.
```

and, for example,

```
bd1 EACH 10 20 50 150
```

gives the reasonable results

```
0.1169482 0.4114384 0.9703736 1
```

We may note that __ represents negative infinity so that **2 % 0** is _ and **_2 % 0** is __. However, we have that **0 % 0** is **0**. Finally, for completeness we introduce the constant verb infinity **_:** which has an infinite result for all arguments, and, for example, **_: 5** is _ and so is **2 _: 5**.

### J constants

The only specific remarks we have made in this paper concerning constants were that the negative sign is represented by the underbar and that the decimal point must be preceded by at least one digit. In this section we shall discuss briefly for the sake of completeness the other forms of constants that are available in **J** although only some of those discussed in the following two paragraphs will be used in the remainder of the paper. Further information may be found in the *Introduction and Dictionary*.

The constant function **1:** is one of the nineteen constant verbs **_9:**, . . . , **0:**, . . ., **9:** which 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**. As a more meaningful example the verb

```
odds=. 1&+@(2&*@i.)
```

for odd numbers may be written more simply as

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

where with either definition we have, for example, that **odds 5** is **1 3 5 7 9**.

Numbers based on π and Euler's constant are given by **p** and **x**, respectively. For example, **1p1** is **3.14159**, **2p1** is **6.28319**, and **0.5p_1** is **0.15915** which is *1/2π*. Also **1x1** is **2.71828**, **2x1** is **5.43656**, **1x_1** is **0.367879**. Stirling's

approximation to *n!*, $\sqrt{2\pi} \times n^n \times e^{-n}$ , may be represented as

```
(2p1^0.5) * (n ^ 0.5+n) * (1x1)^-n
```

and, for example, for **n** equal to **8** is **39902.4** compared to the correct value of **!8** which is **40320**.

Extended precision integer constants may be entered as a sequence of decimal digits terminated by an **x**. For example, the number of arrangements of the **52** cards in a deck of cards is **!52** which in exponential notation is **8.06582e67** may be calculated exactly as **!52x**, an integer with **68** digits which may be displayed by the expression

```
2 34 $ ": !52x
```

as

```
80658175170943878571660636856403766975289505440883277824000000000000
```

As another example, the number of bridge hands is

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

or

```
53644737765488792839237440000 .
```

The monadic verb *extended precision* **x:** has an integer argument and gives an extended integer result, and, for example,

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

is also equal to

```
53644737765488816000000000000 .
```

A rational number may be represented by the integer values of the numerator and denominator separated by **r** and preceded by an optional sign, and is stored with a relatively prime numerator and denominator and a positive denominator. For example, **3r4** and **9r12** are both stored as **3r4** and represent **0.75**. As another example, the expected sample size in the coupon collector's problem with **5** coupons is

```
5 * 1 + 1r2 + 1r3 + 1r4 +1r5
```

which has the value **137r12** which is **137%12** or **11.4167**. The verb **x:** also may be used dyadically, and **2 x: y** gives two extended integers of the numerator and denominator of **y**. For example,

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

is **137r12**. The expression **1 x: y** is equivalent to **x: y**.

The letter **j** is used to represent the real and imaginary parts of a complex number as, for example, **3j4** which represents the imaginary number with real part **3** and imaginary part **4**. The expression

```
3j4 * 5j6
```

is equal to **_9j38** which is an imaginary number with real part **_9** and imaginary part **38**.

Finally, base value representations up to base **35** are given by **b**, where **a** to **z** represent **10** to **35**. For example, **2b10101** is **21**, **8b123** is **83**, **16babc** is **2748.**

| | |
|---|---|
| **1: y** | *One* (**1** for all **y**) |
| **2: y** | *Two* (**2** for all **y**) |
| **[x] x: y** | *Extended precision* |

In Chapter 4, "Factorial Oddities" of Gardner (1978) there is an interesting discussion of tree factorials defined as those factorials whose digits may be arranged in a triangle with one digit in the first row, three in the second, five in the third, etc. The idea originated with the thought that such a display of a large factorial resembled a Christmas tree and could be used on a Christmas card. Gardner's article gives the twenty tree factorials between seven and one thousand, and displays in triangular form 105! shown below and the 1156-digit 508!.

The **10** factorials between **0** and **9**, inclusive, are given by **!i. 10** and are equal to

```
1 1 2 6 24 120 720 5040 40320
   362880
```

and we see that the first four are trivially tree factorials as each consists of a single digit and so is **!7** or **5040** which may be displayed as

```
 5
040  .
```

Further investigation of tree factorials requires the use of extended integer arithmetic. We may note that the number of digits in a tree factorial is the sum of the first so many odd integers and is thus a perfect square.

Let us first define the monadic verb

```
isSQ=: 0: = 1: | %:
```

which is **1** or **0** according as its argument is or is not a perfect square. Also let us define

```
isTF=: [: isSQ [: # [: ": !
```

which indicates whether the factorial of its argument is a tree factorial, and

```
TF=: isTF EACH # ]
```

which gives the tree factorials of its argument. For example,

```
TF EACH i. 10
```

is

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

and **TF i. 10** is **0 1 2 3 7**. Therefore, the **20** tree factorials shown in Gardner's article are given by

```
TF x: 7}.i. 1000
```

and are

```
7 12 18 32 59 81 105 132 228 265
   284 304 367 389 435 483 508
   697 726 944 .
```

The following sequence of calculations shows how a

tree factorial may be represented in triangular form using **!18** as an example:

```
   !18x
6402373705728000
   ":!18x
6402373705728000
   #":!18x
16

   (#~ 1 3 5 7) </. ":!18x
```

| | | | |
|---|---|---|---|
| 6 | 402 | 37370 | 5728000 |

```
   >(#~ 1 3 5 7) </. ":!18x
6
402
37370
5728000

NB. For brevity let
NB.   t=. (#~ 1 3 5 7) </. ":!18x

   7{. each t
```

| | | | |
|---|---|---|---|
| 6 | 402 | 37370 | 5728000 |

```
   _3 _2 _1 0 |. each 7{. each t
```

| | | | |
|---|---|---|---|
| 6 | 402 | 37370 | 5728000 |

```
   >_3 _2 _1 0 |. each 7{. each t
   6
  402
 37370
5728000
```

Similar calculations to those given in the last paragraph give that **!105x** may be represented as

```
                1
               081
              39675
             8240290
            900504101
           30580032964
          9720646107774
        902579144176636
       57322653190990515
     3326984536526808240
    339776398934872029657
  9938729078134368160972 8
 000000000000000000000000
```

### Continuous probability distributions

In determining probabilities for continuous distributions we shall make use of the *integral* or *anti-derivative* adverb **I** developed by Iverson (1993) such that the expression **f I x** gives the area under the graph of **f** from **0** to **x**. The details of the verbs and one conjunction necessary for its implementation need not concern us. As an example, consider the verb

**circ=: [: %: 1: - *:** ,

where **1:** is one of the *constant* verbs which always give the designated value for any argument, which defines a unit circle with centre at the origin so that we have, say,

**circ 0 1 2 3 4 % 4**

equal to

**1 0.9682458 0.8660254 0.6614378 0** .

Now the area bounded by the circle in the first quadrant is **circ I 1** which is equal to **0.78509** and is an estimate of $\pi/4$ or **0.78540**.

We shall define explicit verbs, given at the end of the section, which give probability density functions for the normal, Student's *t*, chi-square and *F* distributions. For each verb the right argument gives the value or values of the independent variable and the left argument for the last three distributions give the parameters which are commonly designated *n* for the *t* and chi-square distributions, and *m* and *n* for the *F*-distribution. In three of the definitions we shall need the verb

**Gamma=: !@<:**

for the Gamma function, and, for example, **Gamma 6** is **120** and **Gamma 1.5** is **0.88623**.

The following are examples of the use of the four distribution verbs with the adverb **I** to give cumulative probabilities:

```
    ndistn I 0 1 2 3
0 0.3413447 0.4772499 0.4986501

    5&tdistn I 2.015 2.571 3.365
0.4499969 0.4750127 0.4900008

    10&csdistn I 12.5 16 18.3
0.7470147 0.9003676 0.949891

    5 20&fdistn I 2.16 2.71 3.29 4.1
0.9002631 0.9500118 0.975138 0.990169

ndistn=: 3 : 0
Const=. % 0.5p_1
Const * ^--:*:y.
)

tdistn=: 3 : 0
:
n=. x.
Const=. (Gamma -:>:n) % (Gamma -:n) *
                         %:n*1p1
```

```
Const * (>:(*:y.)%n) ^ --:>:n
)

csdistn=: 3 : 0
:
n=. x.
Const=. % (2^-:n) * Gamma -:n
Const * (y. ^ -:<:<:n) * ^ --:y.
)

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
)
```

### Simulation I - Discrete variables

A useful verb for simulation is the monadic *roll* **?** which gives sampling with replacement, and **? y** gives a uniform random selection from the population **i. y**. As an example, if

**t=. ? 6 6 6 6 6**

or

**t=. ? 5$6**,

then **t** could have the value **3 0 5 3 2**, and **>: t**, which would then be equal to **4 1 6 4 3**, could represent the results of rolling one die **5** times.

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

**Die=: >: @ ? @ $&6**

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

**1 1 2 6 6 6 1 3 5 6 2 5** .

The verb

**Dice=: -@[ <\ Die @ ***

could be used to simulate the rolling of an arbitrary number of dice an arbitrary number of times, and, for example, **5 Dice 3**, which could have the value

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

would represent the rolling of **5** dice **3** times. Also the verb

**SumDice=: +/EACH @ Dice**

gives the sums for each of the rolls, and **5 SumDice 10** gives the sums occurring on **10** rolls of **5** dice and could have the value

**11 22 19 14 18 22 17 24 19 18** .

Finally the expression

**|: (2+i.11) frtab 2 SumDice 100**

gives the transposed frequency table of the sums occurring when **2** dice are rolled **100** times and could have the value

**2 3 4 5 6 7 8 9 10 11 12**
**5 3 10 12 16 20 7 13 6 7 1** .

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

**Coins=: [: {&'HT' [: ? ] $ 2:**

gives the results of tossing a coin an arbitrary number of times, and, for example, **Coins 10** could be

**THHTHTTTHH** .

The verb

**Heads=: [: +/ [: ? , $ 2:**

gives the number of heads which appear when a number of coins are repeatedly tossed, and, for example, **5 Heads 10** could have the value

**4 3 4 4 1 4 4 2 2 3** .

The verb

**CoinTest=: ([: +/\ Heads) %**
**[: +/\ $~**

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

**100 CoinTest 5**

gives a five-item list whose items give this ratio for **100**, **200**, ..., **500** tosses, and could have the value, rounded to three places of decimals, of

**0.470 0.475 0.497 0.498 0.518** .

The dyadic verb deal **?** gives sampling without replacement and the expression **x ? y** is a list of **x** items chosen without replacement from the list **i. y**, and, for example, two possible values of **4 ? 10** are **6 4 0 7** and **5 3 7 2**. If the cards in a deck of playing cards are numbered sequentially from **0** to **51**, inclusive, then the expression **13 ? 52** could represent dealing **13** cards from a shuffled deck, and, for example, **13 ? 52** could have the value

**22 10 48 51 24 2 11 20 4 16 6 34 31**

Now suppose that the numbers **0**, **4**, **8**, … correspond to hearts, **1**, **5**, **9**, … to diamonds, **2**, **6**, **10**, … to spades, and **3**, **7**, **11**, … to clubs. Therefore, hearts, diamonds, spades and clubs may be represented by **0**, **1**, **2** and **3**, respectively, the **4**-residues of these numbers. The values on any card may be given by **1** plus the **13**-residues of the card number with **11** corresponding to Jack, **12** to Queen and **13** to King. Therefore, the list, **t** say, of **13** cards given above may be represented by the expression

**(], >:@(13&|) ,: 4&|) t** ,

where **,:** is the dyadic verb *laminate*, as

**22 10 48 51 24 2 11 20 4 16 6 34 31**

```
10 11 10 13 12 3 12  8 5  4 7  9  6
 2  2  0  3  0 2  3  0 0  0 2  2  3
```

Thus the first card is the ten of spades, the second the Jack of spades, the third the ten of hearts, etc.

Since the initial value, or random seed, for the random number generator is changed each time either of the above roll or deal verbs are used, calculations involving these verbs cannot be duplicated. Use of the alternate verb **?.** does not change the seed as is illustrated by the following examples:

```
((? ; ?.)5$10) , 3(? ; ?.)10
```

| | | |
|---|---|---|
| 9 3 9 1 8 | 1 7 4 5 2 | 1 0 3 | 8 2 4 |

```
((? ; ?.)5$10) , 3(? ; ?.)10
```

| | | |
|---|---|---|
| 4 0 8 5 4 | 1 7 4 5 2 | 0 1 9 | 8 2 4 |

```
((? ; ?.)5$10) , 3(? ; ?.)10
```

| | | |
|---|---|---|
| 9 9 9 7 5 | 1 7 4 5 2 | 1 7 8 | 8 2 4 |

| | |
|---|---|
| **? y** | *Roll* (Random selection from **i. y**) |
| **x ? y** | *Deal* (random selection without rep.) |
| **x ,: y** | **x** *laminated* to **y** |
| **?. y** | *Roll* with constant seed |
| **x ?. y** | *Deal* with constant seed |

The English biologist W. F. R. Weldon (1860 - 1906) performed several dice-rolling 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 entire experiment may be simulated very simply with the single expression

```
(ei 12) fr +/ EACH
   (e.&4 5 6 each) 12 Dice 4096
```

The results of one execution of this expression were

```
2 11 75 204 476 824 871 842 496 201
            73 16 5
```

giving the frequencies of **0**, **1**, **2**, ..., **12** occurrences of the required number of faces.

Another set of dice data which is apparently 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 Die 100000
```

, one execution of which gave the list

```
16700 16556  16604 16825 16684
      16631  .
```

In the game of craps a player rolls two dice. The player wins if the sum of the two faces is either 7 or 11, and loses if the sum is 2, 3 or 12. If any other sum occurs, the player continues to roll until either the sum which appeared on the first roll occurs which is a win, or a 7 occurs which is a loss. The probability that the player will win may be calculated to be 0.4929 so that it is favourable in the long run to bet against the player.

The explicit verb **Craps** simulates one game of craps.

```
Craps=: 3 : 0
CRAPS=: ,p=. 2 SumDice 1
if. -. p e. 7 11 2 3 12 do.
   whilst. -. (({:CRAPS) e. 7,p)
                          do.
      CRAPS=: CRAPS, 2 SumDice 1
   end.
end.
if. 1 = $CRAPS do.
   r=. p e. 7 11
else.
   r=. p = {:CRAPS
end.
)
```

We note that the verb is monadic and requires a right argument of **''** which is ignored, and returns a result of **1** or **0** according as the simulation gives a winning or losing game. The sequence of sums is available in the global variable **CRAPS**. The following are the results of several simulations:

```
   Craps ''
1
   CRAPS
9 10 10 5 6 11 12 6 9
   Craps ''
0
   CRAPS
5 11 8 11 6 4 6 9 10 9 6 7
   Craps ''
1
   CRAPS
9 6 9
   Craps ''
1
   CRAPS
7
```

The monadic verb **CrapsProb** uses **Craps** to simulate a number of games of craps specified by the argument and to estimate the probability of success in a single game, given as the number of successes divided by the total number of games:

```
CrapsProb=: 3 : 0
n=. s=. 0
```

```
while. n < y. do.
   s=. s + Craps ''
   n=. >:n
end.
s % y.
)
```

Four estimates of the probability of success in a single game based on **1000** simulations each were **0.51**, **0.493**, **0.481** and **0.499**.

We shall conclude this section with a brief discussion of a problem known as the gambler's ruin. Suppose that a person with a capital of *d* dollars plays a series of games in which the probability of success in a single game is *p* against another person or a gambling house with capital *D* dollars. We wish to find the probability *R* of the gambler's ultimate ruin. We shall only give expressions for *R*, omitting the details of their derivation which may be found in Weaver (1963).

If the game is fair to both parties, i.e., if *p* is 0.5, then it may be shown that the probability of the gambler's ruin is equal to *D/(d + D)*. Suppose now we calculate these probabilities for persons who each have 10 dollars against an opponent who has 10, 50, 100 and 1000 dollars. Therefore, if

```
d=. 10
```

and

```
D=. 10 50 100 1000
```

, then

```
D%d+D
```

is to three decimal places

```
0.500 0.833 0.909 0.990
```

.

If the game is not fair, i.e., if *p* is not equal to *0.5*, then the expression for the gambler's ruin may be shown to be

$$R = ( (q/p)^C - (q/p)^d ) / ( (q/p)^C - 1 ) ,$$

where $C = d + D$ and $q = 1 - p$. A corresponding verb in **J** is

```
Ruin=: 3 : 0
:
q=. -. p=. x.
'd D'=. y.
C=. d + D
(((q%p)^C)-(q%p)^d) % _1+(q%p)^C
)
```

where the left argument is the probability and the right argument is a two-item list giving the values of **d** and **D**. As an example if a person with **5** dollars plays a series of games with probability of success **0.45** in a single game against an opponent with **25** dollars, the probability that the person will be ruined is **0.45 Ruin 5 25** or **0.99579**.

Finally since the probability of success in a single game of craps is **0.4929**, the probabilities of ruin for the values of **d** and **D** used above are given by the expression

```
6.0 6.0 10.5": (],. .4929&Ruin"1@])
                        d,"1 0 D
```

which gives the table

```
10    10   0.57053
10    50   0.92695
10   100   0.98489
10  1000   1.00000
```

.

### Simulation II - Continuous variables

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

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

, and, for example, **rand 3** is a three-item list which could have the value

```
0.129889 0.04406748 0.6422329
```

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

Pairs of standard normal random variables 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

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

which will be independently distributed standard normal variables.

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

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

The expression **stdnmlrand 3** could give the following table representing three pairs of independent

standard normal variables:

```
  2.03104   0.544758
 _1.46607 0.06222853
0.4249043   _2.22973
```

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 $\mu$ and standard deviation $\sigma$. The ambivalent verb **normal** generates a list of normal variables whose number is given by the right argument. When used monadically the variables have a standard normal distribution and when used dyadically have a mean and standard deviation specified by the left argument:

```
nmlrand=: 3 : 0
0 1&nmlrand y.
:
y.{., ({.x.) + ({:x.) *
                stdnmlrand >.-:y.
)
```

The following are some examples of its use:

```
   nmlrand 4
_1.60573 _0.1939038 0.6280351 1.18397
   1 0.5 nmlrand 3
1.22567 1.89995 1.51459
   (am,sd) 1 0.5 nmlrand 200
0.9367325 0.5098191
```

Finally 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 $\mu$X has an exponential distribution with mean $\mu$. 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 .
```

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

The coordinates of random points within the square are given by

```
coords=: rand@,&2
```

and **coords 3** could have the value

```
0.4642448 0.5619059
0.9523642 0.3856299
0.2824959 0.9092284
```

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

```
0.5312614 1.05571 0.9065003
```

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

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

will give the number of points within or on the circle. Five evaluations of the expression **incircle 100** gave the values **83**, **80**, **76**, **85** and **83**. Finally the verb

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

will give an estimate of the value of $\pi$ for an arbitrary number of points. Five evaluations of **PI 10000** gave **3.1364**, **3.14**, **3.146**, **3.1328** and **3.1572** for estimates of the value of $\pi$.

### Simulation III - Central limit theorem

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

It may be shown that the sum $T$ occurring when $n$ dice are rolled has mean $\mu = 7n/2$ and standard deviation $\sigma = \sqrt{35n/12}$. Thus according to the central limit theorem the distribution of the standardized sum $(T - \mu)/\sigma$ approaches that of the standard normal distribution as $n$ increases.

The rolling of the dice will be simulated by the verb **SumDice** where, it will be recalled, the left argument is the number of dice and the right argument is the number of rolls. The resulting sums for each simulation will be standardized by the verb

```
stdize=: () - 3.5&*@[) %
              [: %: 2.91667&*@[
```

where the left argument is the number of dice and the right argument is the list of sums to be standardized. The standardized sums are then tabulated in a frequency table by the verb **cfrtab** whose left argument is the list

```
r=. ap _3.25 0.5 14
```

which is the arithmetic progression

```
_3.25 _2.75 _2.25 … 2.25 2.75 3.25 .
```

Thus the standardized sums are grouped in thirteen

intervals of width one-half standard deviation centred at the origin. For two dice these calculations may be performed by the expression

```
t2=. r cfrtab 2 stdize 2 SumDice 100
```

which is a two-column table with the midpoints of the intervals in the first column and corresponding frequencies in the second. If the results of simulations for five and ten dice are given by the variables **t5** and **t10**, then

```
t2 ,. ({:"1 t5) ,. {:"1 t10
```

is a four-column frequency table with the midpoints of the intervals of classification in the first column and the frequencies for the three simulations in the remaining columns. The results of one such simulation are given in Figure 3 in Appendix 2.

### Simulation IV – Sampling

In this section we shall consider very briefly the sampling distribution of the arithmetic mean. It is known that the sampling distribution of the sample mean based on a random sample of size $n$ from a distribution with mean $\mu$ and standard deviation $\sigma$ has an approximate normal distribution with mean $\mu$ and standard deviation $\sigma/\sqrt{n}$. Furthermore, the distribution of the sample mean is often approximately normal even for populations whose distributions depart noticeably from the normal and for quite small values on $n$.

We shall use the following data from Hoel (1966) on the hours worked in a week by **100** workmen given as the list **Hours**:

```
36 36 32 35 41 32 41 30 22 32
27 35 35 10 29 41 45 45 30 39
45 33 23 28 27 43 44 31 34 33
36 28 31 39 29 42 43 31 28 39
33 18 25 36 45 45 24 37 52 26
23 23 38 37 38 42 40 42 40 40
42 40 34 37 34 36 40 33 40 20
10 23 15 28 28 32 28 37 37 44
25 36 26 40 40 39 39 41 33 39
40 38 39 16 39 38 41 41 28 27
```

These data have a mean of **am Hours** or **34.1** and a standard deviation of **sd Hours** or **8.0**. These values will be used for the population mean and standard deviation, respectively. The frequency distribution given by the expression

```
(ap 7.5 5 10) cfrtab Hours
```

is

```
10  2
15  2
20  3
25 12
30 18
35 22
40 31
45  9
50  1
```

which shows a significant departure from normality.

In the first simulation we shall take **75** samples of size **5** with replacement from the population of **100** hours and find the mean of each sample, and the mean and standard deviation of these means. The **75** samples may be generated very simply as a table with **75** rows and **5** columns by the expression

```
(5 ? 75$100){Hours
```

and the means found by applying the verb **am"1** to this table giving the list, **m**, say, of sample means. The mean and standard deviation of the sample means are given by **(am,sd)m**. Finally the distribution of the sample means is given by the expression

```
(ap 24.5 1 21) cfrtab m
```

which gives a frequency classification with **20** unit intervals centred at the values **25**, **26**, …, **44**. These calculations may be summarized in the single expression

```
(ap 24.5 1 21) cfrtab m=. am"1
                    (5?75$100){Hours
```

The expression **(<./,>./)m** gives a check that all of the means have been included in the distribution. One such simulation gave the sample means lying between **25.2** and **42.4** with a mean of **33.8** and a standard deviation of **3.41**. The standard deviation expected from a sample of size **5** is **8%%:5** or **3.58**.

This simulation was repeated with samples of size **10** and gave means between **28.6** and **40.3** with a mean of **34.1** and a standard deviation of **2.34** which may be compared with an expected standard deviation of **2.53**. The frequency distributions for the two simulations are given in Figure 4 in Appendix 2.

### One more example

In the Lotto 6-49 lottery a person purchases a ticket for one dollar and selects six integers at random without replacement from the integers between 1 and 49, inclusive. The payout in the weekly draw depends on the number of matches and is zero for 2 or fewer matches and 10, 75, 2500 or 100000 dollars for 3, 4, 5 or 6 matches, respectively. We shall find the probabilities for each of the prizes, assess the game in light of the gambler's ruin, and finally simulate this game for an arbitrary number of purchases and compute the cumulative winnings.

Without loss of generality we may assume that the winning integers are 1, 2, ..., 6, and that the number of matches is the number of occurrences of these integers in the six integers which are drawn. Therefore, the probabilities for the various number of prizes are given by the hypergeometric distribution with parameters 6, 43 and

6 corresponding to the number of winning integers, the number of remaining integers and the number of integers drawn, respectively. Therefore, the probabilities are given by the monadic verb

```
Lotto=: 6 43 6&hg
```

and the probabilities are given by

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

which to five decimal places are

```
0.43596 0.41302 0.13238 0.01765
          0.00097 0.00002 0.00000
```

(The last probability is given by **Lotto 6** which is **7.15112e_8**.) If the payouts are given by the list

```
PayOut=: 0 0 0 10 75 2500 1000000
```

and if for convenience we let the number of matches be given by the list

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

then the expectations in cents for each number of matches is

```
5.1":100 * PayOut * Lotto M
```

or

```
0.0 0.0 0.0 17.7 7.3 4.6 7.2
```

and the expectation for one game is

```
5.1":100 * +/ PayOut * Lotto M
```

or **36.7** cents.

The probability of not winning anything in a single game is

```
+/Lotto 0 1 2
```

or **0.98136** and the probability of winning something is

```
+/Lotto 3 4 5 6
```

or **0.01864**. Thus if we make the most unlikely assumption that each of the player and the house starts with ten dollars, then we find the probability of the player's ultimate ruin to be

```
0.01864 Ruin 10 10
```

or **1**. (We may compare this with the same probability for the game of craps which is **0.4929 Ruin 10 10** or **0.5705**.)

The verb

```
LottoNums=: [: >: 6: ? $&49
```

simulates the random selection of integers, and, for example, **LottoNums 4** could have the value

```
43 12 26 22 36 42
16 49  4 29 22  8
48 46 23 16 45 28
46 28 15 19  4  7
```

which is an array of shape **4 6** representing the numbers associated with the purchase of **4** tickets. The expression **LottoNums 2 3** could have the value

```
24 44 17 27 14  4
12 36 45 13 31 17
12  1 30 35 48 13

13 17 44 48  6 33
28 12 25 36 33 29
41 25  5 47 43 46
```

an array of shape **2 3 6** which could represent the purchase of **2** tickets on each of **3** occasions. The dyadic verb

```
LottoSim=: [ {~ [: +/"1
      [: e.&1 2 3 4 5 6 LottoNums@]
```

where the right argument gives the purchases and the left argument gives the payouts which have been given previously by the list **PayOut**. The expression

```
PayOut LottoSim 4
```

gives the payout for each of **4** purchases and almost certainly has the value

```
0 0 0 0
```

while the expression

```
PayOut LottoSim 2 3
```

gives the payout for each of **2** tickets purchased on **3** occasions and again would almost certainly be

```
0 0 0
0 0 0
```

In order to get some non-zero results let us consider the purchase of **2** tickets a week for each week for **25** years, and find the cumulative yearly winnings. The simulation may be done by the expression

```
PayOut LottoSim 104 25
```

which gives a table with **104** rows and **25** columns with the columns giving the individual winnings during each of the years. The column sums

```
+/PayOut LottoSim 104 25
```

give the yearly winnings, and

```
20 10 10 0 10 30 10 20 30 10 30 10
   20 20 10 20 10 0 20 30 0 20 20
   10 0
```

is a typical value. Therefore, the cumulative column sums

```
+/\+/PayOut LottoSim 104 25
```

give the desired cumulative yearly winnings, and

```
10 20 40 60 80 110 110 160 170 190
   275 295 315 315 335 355 365 385
   385 415 425 435 495 505 505
```

is another typical value. We note for this last simulation that an investment of **104 * 25** or **2600** dollars gave a return of **505** dollars.

### Chi-square distribution

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
```

or **1.91244**. If we use the verb **csdistn** given in an earlier section, we see that **3&csdistn I 1.91** is **0.408** which means that 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 observed and expected frequencies, respectively, may be used to find the value of chi-square. With the data of the previous paragraph we have that **obs chisq exp** is **1.91244**. We shall generalize this verb later in the section to accommodate two-way frequency distributions.

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 a list **obs** equal to

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

.

The expected frequencies on the hypothesis of a random sequence of digits would be

```
exp=. 20 20 20 20 20 20 20 20 20 20
```

so that a value of chi-square would be **obs chisq exp** which is **5.9**. Since **9&csdistn 5.9** is **0.25**, we may conclude that the **200** digits are from a uniform distribution. Since all of the expected frequencies are equal to **20**, we could have calculated the value of chi-square as **obs chisq 20**.

As another example of the chi-square test for a list of observations consider the simulation of Weldon's dice data considered earlier in which twelve dice were rolled 4096 times and the occurrence of a 4, 5 and 6 was considered a success. The simulation given there resulted in the observed values

```
obs=. 2 11 75 204 476 824 871 842
            496 201 73 16 5
```

for **0**, **1**, **2**, ..., **12** successes. Since the first and last observed frequencies are comparatively small, we shall combine them with the second and second last, respectively. This may be done with the adverb key so that if

```
obs1=. k +//. obs
```

where

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

,

then **obs1** is

```
13 75 204 476 824 871 842 496 201
          73 21
```

.

Since the expected frequencies are given by a binomial distribution with probability **0.5** on a single trial, we have that

```
exp=. 4096 * 12 0.5 binomial i. 13
```

which has the value

```
1 12 66 220 495 792 924 792 495 220
                    66 12 1
```

.

The combined expected frequencies are thus

```
exp1=. k +//. exp
```

which is

```
13 66 220 495 792 924 792 495 220
          66 13
```

.

The value of chi-square may be found to be **17.9182**, and since **10&csdistn I 17.9** is **0.943** we may just accept the hypothesis of a binomial distribution for this simulation.

It is interesting to note that Wolf's data on rolling a die 100 000 times gave a value of chi-square of 784.5 from which we may conclude that the die he used was certainly biased. On the other hand the simulated data for the same experiment gave a value of 2.6 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".

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 $i$th row and $j$th 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. Now if we modify the verb **chisq** so that it becomes

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

it may be used for both one- and two-way classifications. The expected frequencies are given by the monadic verb

```
ExpFrTab=: (+/"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

```
ExpFrTab 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

```
() chisq ExpFrTab) Tab34
```

which is **19.94**. 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 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<br>3 3 | 1 6<br>4 2 | 0 7<br>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 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.

### Nonparametric statistics - Preliminaries

Since in many nonparametric tests the original observations are replaced by their ranks, we shall first discuss the computation of ranks in **J**. We shall also define a verb that gives combinations of the non-negative integers taken so many at a time as it will be used in one of the tests discussed here.

The monadic verb *grade up* **/:** grades any argument **v** to give a permutation list so that **(/:v){v** sorts **v** in ascending order. For example, if

```
v=. 4.5 2 6.1 3.7
```

then **/: v** is

```
1 3 0 2
```

so that the third item is the smallest, the first is the second smallest, the fourth is the third smallest, and the second is the largest, and

```
1 3 0 2 { 4.5 2 6.1 3.7
```

is the sorted list **2 3.7 4.5 6.1**. Now if we grade up this permutation list, i.e., **/:/:v** giving **2 0 3 1**, we shall obtain, in zero-origin indexing, the ranks of the items in **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. Thus we may define the verb

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

for the ranks unadjusted for ties, and **uranks v** is **3 1 4 2**. The conjunction *power* **^:** is used to repeat the verb left argument a number of times specified by the right argument, and, for example, **(*:^:2) 5** is equivalent to ***: *: 5** and is equal to **625**.

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

```
v1=. 2 3.7 2 4.5 6.1 2 4.5
```

then **uranks v1** is **1 4 2 5 7 3 6**, and the ranks of the first, third and sixth observations are replaced by the average of the ranks **1**, **2** and **3** or **2**, and similarly the ranks of the fourth and seventh observations are replaced by **5.5**. Thus the ranks of **v1** adjusted for ties are given by the list

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

We shall derive the verb **ranks** later in this section for finding ranks adjusted for ties.

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`**.

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

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

gives a table of the integers **`i. n`** taken **`m`** at a time, and

```
(2&comb;3&comb) 4
```

is

```
┌─────┬─────┐
│0 1  │0 1 2│
│0 2  │0 1 3│
│0 3  │0 2 3│
│1 2  │1 2 3│
│1 3  │     │
│2 3  │     │
└─────┴─────┘
```

.

Finally we introduce the monadic verb *magnitude* **`|`**, and, for example, **`|2.5 0 _2`** is **`2.5 0 2`**.

| | |
|---|---|
| **`/: y`** | *Grade up* **`y`** |
| **`u ^:n y`** | **Power** (Iterate **`u n`** times on **`y`**) |
| **`\| y`** | *Magnitude* of **`y`** |
| **`x /: y`** | *Sort* (*Up*) **`x`** according to grade up of **`y`** |
| **`\: y`** | *Grade down* |
| **`x \: y`** | *Sort* (*Down*) **`x`** accord. to grade down of **`y`** |

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

```
nranks=: (= mp uranks) %
                    [: +/"1 =    ,
```

where **`mp`** is the general utility verb matrix product, and **`nranks v1`** is **`2 4 5.5 7`**. The ranks adjusted for ties are given by

```
ranks=:  [: , |:@= # nranks
```

and **`ranks v1`** is **`2 4 2 5.5 7 2 5.5`**.

The dyadic verb *sort* (*up*) **`/:`** sorts the left argument according to the grade up of its right argument, and, for example,

```
'toronto' /: 0 1 0 1 0 0 1
```

is

```
trntooo
```

.

Also **`y/:y`** or **`/:~y`** sorts **`y`**, and

```
/:~ 4.5 2 6.1 3.7
```

is **`2 3.7 4.5 6.1`**. Indeed the general utility verb **`sort`** is defined as

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

.

The monadic *grade down* and dyadic *sort* (*down*) verbs **`\:`** are similar, and, for example, **`\:~ v`** is

**`6.1 4.5 3.7 2`**.

An ordered truth table is a truth table with the rows sorted according to the number of **`1`**s and is given by

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

, and, for example, **`ott each 0 1 2 3`** is

```
┌─┬─┬─┬─┬─────┐
│0│0│0 0│0 0 0│
│ │1│0 1│0 0 1│
│ │ │1 0│0 1 0│
│ │ │1 1│1 0 0│
│ │ │   │0 1 1│
│ │ │   │1 0 1│
│ │ │   │1 1 0│
│ │ │   │1 1 1│
└─┴─┴─┴─┴─────┘
```

.

We shall conclude this section with a brief look at the riffle shuffle which is also known as the faro shuffle or the weave shuffle. In this shuffle the deck of cards is divided into two decks and the cards put down, one at a time, alternately from the two decks. If the number of cards in the full deck is even, then the deck must be divided evenly into the two decks. If the number is odd, then the deck must be divided as evenly as possible with one deck having just one more card than the other deck and the first card put down must be from the larger half.

For an even deck if the first card in the first half is put down first, then the two cards previously at the ends will be at the ends of the shuffled deck. For example, if there are ten cards represented as *abcdeFGHIJ*, then the shuffled deck is *aFbGcHdIeJ*. Such a shuffle is called an "out-shuffle" since the first and last cards in the unshuffled deck will remain the end cards in the shuffled deck. If the first card to be put down is from the second half of the deck, then the former first and last cards go into the shuffled deck and become the second cards from either end. In the ten-card example the shuffled deck is *FaGbHcIdJe*. This shuffle is called an "in-shuffle". If the cards in an odd deck are divided so that the first half contains the extra card, then the shuffle is an out-shuffle, and if the second half contains the extra card, the shuffle is an in-shuffle. For example, if the deck is *abcdeFGHI*, then the in-shuffle and the out-shuffle produce *eaFbGcHdI* and *aFbGcHdIe*, respectively.

In **J** the riffle shuffle may be given very simply by a sort produced by the appropriate use of the grade up and grade down verbs. For example, if

```
x=. 'abcdeFGHIJ'
```

, then

```
x/:/:10$0 1
```
gives
```
aFbGcHdIeJ
```
which is an out-shuffle, and
```
x/:\:10$0 1
```
is
```
FaGbHcIdJe
```
which is an in-shuffle. Similarly, if
```
y=.'abcdeFGHI'
```
then
```
y/:/:9$0 1
```
is the out-shuffle
```
aFbGcHdIe
```
and
```
y/:\:9$0 1
```
is the in-shuffle
```
eaFbGcHdI
```
.
Verbs for the out-shuffle and in-shuffle are given by
```
outshuffle=: ] /: [: /: #@] $ 0:,1:
```
and
```
inshuffle=: ] /: [: /: #@] $ 1:,0:
```
, respectively, and, for example,
```
outshuffle x
```
is
```
aFbGcHdIeJ
```
, and
```
inshuffle y
```
is
```
eaFbGcHdI
```
.

We may note a couple of interesting features of the **J** expressions for the riffle shuffle. If the list of **0**s and **1**s is changed to a list of **1**s and **0**s, then an out-shuffle is changed to an in-shuffle and an in-shuffle is changed to an out-shuffle. For example,
```
x/:/:10$1 0
```
is
```
FaGbHcIdJe
```
, and
```
x/:\:10$1 0
```
is
```
aFbGcHdIeJ
```
.
If the second grade up from the right is changed to a grade down, then the order of the final shuffle is reversed, and, for example,
```
x\:/:10$0 1
```
is
```
JeIdHcGbFa
```
, and
```
x\:\:10$0 1
```
is
```
eJdIcHbGaF
```
.
The same observations apply, of course, to decks with an odd number of cards.

### Nonparametric statistics - Three examples

The first nonparametric test is Spearman's rank correlation coefficient which 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=: ranks@[ cor ranks@]
```
which calculates the rank correlation coefficient from the original observations. If we use the two lists **French** and **German** with the values
```
83 27 42 51 53 44 47 55 61 33
```
and
```
74 22 49 54 48 47 55 61 59 29
```
which give the marks of ten students in these two subjects and which have been taken from Sprent (1981), we find that
```
French rcor German
```
has the value **0.879**, rounded to three decimal places, while the correlation coefficient found from the original marks by
```
French cor German
```
is equal to **0.925**.

The second nonparametric test we shall consider is the Wilcoxon two-sample test for determining any difference between two sets of observations. For sample data we shall use the data
```
ht=. 7.9 6.5 17.8 4.2 13.2
```
and
```
hc=. 5.6 0.4 6.7 1.2 2.9
```
taken from Hodges and Lehman (1964) representing the number of years ten patients survived a heart attack, only the first five of whom had received a specific treatment. We wish to determine if the treatment significantly prolonged the lives of the patients who received it.

The ranks of the combined observations are
```
r=. ranks ht,hc
```
which is equal to
```
8 6 10 4 9 5 1 7 2 3
```
where the first five ranks refer to treated patients and the last five to control patients. The sums of these two groups are **+/5{.r** or **37** and **+/_5{.r** or **18**, respectively. We wish to find the probability that the sum of the five ranks for the control patients would be less than or equal to the observed value of **18** on the assumption that the treatment is of no effect, There are **5!10** or **252** combinations of the ranks **1**, **2**, ..., **10** taken **5** at a time and they are given by
```
>: 5 comb 10
```
which is a table with **252** rows and **5** columns. The row sums are
```
+/"1 >: 5 comb 10
```
and the number of these sums less than or equal to **18** is

```
+/ 18 >: +/"1 >: 5 comb 10 .
```
Thus the probability that the sum of the control ranks is less than or equal to **18** is **7%252** or **0.028** and we may conclude that the treatment was effective in prolonging life.

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

```
s1=. 42 51 31 61 44 55 48
```
and also a new lotion where the burn measurements are
```
s2=. 38 53 36 52 33 49 36 .
```
To estimate the efficacy of the new lotion we shall first find the differences of the pairs of measurements, or **s1 - s2**, which is
```
4 _2 _5 9 11 6 12 ,
```
and then the ranks of the absolute value of these differences or
```
ranks |s1 - s2
```
which is
```
2 1 3 5 6 4 7 .
```
The sum of the ranks corresponding to negative differences, which indicate that there is more burning with the new lotion, is **4** so we should find the probability that of the **2^7** or **128** ways of assigning signs to the ranks the sums associated with negative ranks would be less than or equal to this number. If we use an argument similar to that used for the Wilcoxon two-sample test, we see that the number of ways is
```
+/ 4 >: +/"1 (tt 7) # >: i. 7
```
which is **7**. Thus the probability is **7%128** or approximately **0.055** which casts some doubt at the 5% level on the hypothesis that the new lotion is an improvement.

### Nonparametric statistics - Runs

Tossing a coin fifteen times, say, could produce the sequence
```
HHTHHHHTHHHTHHH
```
of heads and tails. (This sequence could be obtained from the expression **Coins 15**, where the verb **Coins** has been introduced earlier.) One measure of the randomness of this sequence is given by the total number of consecutive runs of either heads or tails which occur. The above sequence begins with the run **HH** of two heads, is followed by the run **T** of a single tail, and then by the run **HHHH** of four heads, and so on. The total number of runs is seven. If the number of runs lies between reasonable limits which may be calculated but which need not concern us here, then we may conclude that the sequence of heads and tails is random and that the coin is unbiased.

In this section we shall give **J** verbs for finding both the number of runs in a given sequence and the distribution of the lengths of the runs.

We shall first introduce two verbs which perform operations on successive overlapping pairs of items in a list. If **c** is the above list of heads and tails, then **2 <\c** is the list

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| HH | HT | TH | HH | HH | HH | HT | ... |

of overlapping pairs, and **2 ~:/\c** 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
```
noteq=: 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. Similarly we may define the monadic verb
```
diff=: 2: (-~/\) ]
```
which gives first differences, i.e., second item minus the first, third item minus the second, etc., and which has been introduced earlier as a general utility verb, and, for example,
```
diff 5 8 10 9 15
```
is the list **3 2 _1 6**.

From the above discussion we may see that the number of runs in a sequence is one more than the number of unequal overlapping pairs. Therefore, we may define the verb
```
nruns=: [: >: +/ @ noteq
```
whose argument is an arbitrary list and whose result is the number of runs, and, for example, **nruns c** is **7**.

For the list **c** we have **noteq c** is the list
```
0 1 1 0 0 0 1 1 0 0 1 1 0 0
```
so that **1, (noteq c),1** is
```
1 0 1 1 0 0 0 1 1 0 0 1 1 0 0 1 .
```
Since **c** is the list
```
HHTHHHHTHHHTHHH ,
```
we see that the **1**'s in the second list of **0**s and **1**s mark the beginnings and ends of the runs in **c**. The indices of these end points is the list
```
0 2 3 7 8 11 12 15 ,
```
the first differences of which is the list,
```
2 1 4 1 3 1 3 ,
```
giving the lengths of the runs. Therefore, we may define the verb
```
lruns=: [: diff [: iones 1: ,
              noteq , 1:
```
which gives the lengths of the runs, where **iones** is a utility verb giving the indices of the **1**s in its logical argument. The verb

```
lrunstab=: nubtab @ sort @ lruns
```
gives the nub frequency table of the lengths of the runs in sorted order. For example, **lruns c** is
```
2 1 4 1 3 1 3
```
and **lrunstab c** is
```
1 3
2 1
3 2
4 1 .
```

The number of different characters need not be limited to two, and, for example,
```
(nruns;lruns) 'aabbbbacacccbddbbbb'
```
is the two-item list

| | |
|---|---|
| 9 | 2 4 1 1 1 3 1 2 4 |

where the first item is the number of runs and the second is their lengths.

The following verb gives the frequency distribution of maximum run length of heads or tails where the number of coins tossed is given by the left argument and the number of repetitions by the right argument:
```
CoinRuns=: 3 : 0
:
max=. i. 0
while. y. > $max do.
   max=. max, >./lruns, Coins x.
end.
(ei >./ max) frtab max
)
```
The following are the frequency distributions obtained by four simulations resulting from the expression **200 CoinRuns 1000**, i.e., tossing **200** coins **1000** times:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | 0 | 2 | 0 | 2 | 0 | 2 | 0 |
| 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 |
| 4 | 1 | 4 | 1 | 4 | 0 | 4 | 0 |
| 5 | 35 | 5 | 36 | 5 | 28 | 5 | 46 |
| 6 | 173 | 6 | 173 | 6 | 172 | 6 | 171 |
| 7 | 277 | 7 | 242 | 7 | 256 | 7 | 277 |
| 8 | 207 | 8 | 206 | 8 | 245 | 8 | 208 |
| 9 | 157 | 9 | 157 | 9 | 130 | 9 | 136 |
| 10 | 66 | 10 | 93 | 10 | 88 | 10 | 78 |
| 11 | 39 | 11 | 50 | 11 | 41 | 11 | 36 |
| 12 | 18 | 12 | 21 | 12 | 18 | 12 | 23 |
| 13 | 17 | 13 | 12 | 13 | 10 | 13 | 12 |
| 14 | 6 | 14 | 4 | 14 | 6 | 14 | 6 |
| 15 | 2 | 15 | 2 | 15 | 1 | 15 | 4 |
| 16 | 1 | 16 | 1 | 16 | 1 | 16 | 0 |
| 17 | 1 | 17 | 0 | 17 | 2 | 17 | 0 |
| | 18 | 2 | 18 | 2 | 18 | 1 | |
| | | | | | 19 | 1 | |
| | | | | | 20 | 0 | |
| | | | | | 21 | 1 | |

In the discussion of the median earlier in this paper we introduced the list **u**
```
22 14 32 30 19 16 28 21 25 31 ,
```
whose median was **23.5**. If we are interested in determining if the sampled population, whatever it may be, changed with time, we could examine how the sequence of observations fluctuated about the median. One indication is given by the number of runs above and below the median. Since
```
u > median u
```
is equal to
```
0 0 1 1 0 0 1 0 1 1 ,
```
where the **0**s would represent items of **u** less than or equal to the median and the **1**s values greater than the median. The number of runs is given by
```
nruns u > median u
```
which is **6**, a number within limits for the randomness of the list **u** of observations.

### Markoff chains

We shall introduce Markoff chains by means of a simple variation on the well-known random walk example. Suppose that an insect, a ladybug, say, alights at random on one of several locations in a network and then moves at random from one location to another. All of the locations are the same except for a few which are traps from which the ladybug cannot escape. Given some probabilities about the ladybug's initial position and its movements from location to location, we wish to give some overall description of its behaviour.

Let us assume there are six locations or states, $S_0$, $S_1$, ..., $S_5$, in the network and that the first and last are traps from which there is no escape and that the ladybug lands initially on one of the remaining four states and then moves at random from one state to another. Let the list **a** whose value is
```
0 0.25 0.25 0.25 0.25 0
```
give the initial probabilities so that the ladybug lands on one of the four states $S_1$, $S_2$, $S_3$ and $S_4$ with equal probability. Also let the table **LB** given by
```
1.0  0.0  0.0  0.0  0.0  0.0
0.2  0.0  0.3  0.0  0.5  0.0
0.0  0.0  0.0  0.3  0.6  0.1
0.0  0.0  0.0  0.0  0.7  0.3
0.0  0.3  0.6  0.1  0.0  0.0
0.0  0.0  0.0  0.0  0.0  1.0
```
give the matrix of transition probabilities of moving from

one state to another. Thus, for example, if the ladybug is at $S_1$, the probability it will move to $S_0$ on the next step and be trapped is **0.2**, and the probabilities that it will move to $S_2$ and $S_4$ are **0.3** and **0.5**, respectively. We note that **+/a** is equal to **1** and the row sums **+/"1 LB** are **1 1 1 1 1 1**.

A sequence of random events such as described in the above example is called a Markov chain. It is defined by a matrix of constant one-step transition probabilities of moving from one state to another and by a list of initial probabilities for each of the states.

The expression **a mp LB**, where **mp** is the utility verb for matrix product, is equal to

```
0.05 0.075 0.225 0.1 0.45 0.1
```

and gives the probabilities that the ladybug is in each of the various states after one step. If the initial probabilities change, then these probabilities change, and, for example, if **a** has the value

```
0 0.5 0 0.5 0 0 ,
```

then **a mp LB is equal to**

```
0.1 0 0.15 0 0.6 0.15 .
```

Regardless of the value of **a**, we have that **+/a mp LB** is equal to **1** since after one step the ladybug obviously has to be in one of the states.

It may be seen that the matrix product **LB mp LB** which has the value

```
1.00 0.00 0.00 0.00 0.00 0.00
0.20 0.15 0.30 0.14 0.18 0.03
0.00 0.18 0.36 0.06 0.21 0.19
0.00 0.21 0.42 0.07 0.00 0.30
0.06 0.00 0.09 0.18 0.58 0.09
0.00 0.00 0.00 0.00 0.00 1.00
```

will give the two-step probabilities of moving about the network. For example, consider the second row

```
0.20 0.15 0.30 0.14 0.18 0.03
```

which gives the probabilities of all of the two-step moves from state $S_1$. The first probability **0.2** is the probability of moving from $S_1$ to $S_0$ on the first step and staying there, the second probability **0.15** is the probability of moving with probability **0.5** from $S_1$ to $S_4$ on the first step times the probability **0.3** of moving from $S_4$ to $S_1$ on the second step, etc. Higher powers of **LB** may be similarly interpreted, and, for example,

```
LB mp LB mp LB
```

gives the three-step probabilities of moving about the network. Of course, we have that the row sums of any power of **LB** are all equal to unity, and, for example, **+/"1 LB mp LB** is **1 1 1 1 1 1**.

A state of the Markov chain from which is impossible to leave is called an absorbing state. A state which is not an absorbing state is called a nonabsorbing state. In the above example, $S_0$ and $S_5$ are absorbing states, and $S_1$, $S_2$, $S_3$ and $S_4$ are nonabsorbing states. It is convenient to permute the rows and columns of the transition matrix so that those rows and columns corresponding to the absorbing states form an identity matrix in the top left corner. The transition matrix is then said to be in canonical form. Thus for the transition matrix of the above example we have

```
P=. (;~0 5 1 2 3 4) subtab LB,
```

where **subtab** is a utility verb for giving arbitrary subtables of a table, so that **P** is equal to

```
1.0  0.0  0.0  0.0  0.0  0.0
0.0  1.0  0.0  0.0  0.0  0.0
0.2  0.0  0.0  0.3  0.0  0.5
0.0  0.1  0.0  0.0  0.3  0.6
0.0  0.3  0.0  0.0  0.0  0.7
0.0  0.0  0.3  0.6  0.1  0.0 .
```

Two matrices of interest are

```
Q=. (;~2 3 4 5) subtab P
```

which is equal to

```
0.0  0.3  0.0  0.5
0.0  0.0  0.3  0.6
0.0  0.0  0.0  0.7
0.3  0.6  0.1  0.0
```

and

```
R=. (2 3 4 5;0 1) subtab P
```

which is equal to

```
0.2  0.0
0.0  0.1
0.0  0.3
0.0  0.0 .
```

We shall see that we may use the matrices **Q** and **R** to give some interesting information about the behaviour of the system, i.e., the movement of the ladybug, described by the Markov chain.

First of all, we shall define what is known as the fundamental matrix for the chain as

```
N=. %. (id 4) - Q
```

where **id** is the utility verb for the identity matrix so that **id 4** is the identity matrix of order **4**. The items of **N** which is equal to

```
2.00814 2.61872 1.12166 3.36047
1.09905 3.52782 1.42469 3.66350
0.94980 2.18453 1.97196 3.16599
1.35685 3.12076 1.38851 4.52284
```

give the average number of times that the ladybug is in each nonabsorbing state for each initial nonabsorbing state. For example, if the ladybug starts in state $S_1$, it is in state $S_1$ an average of approximately **2** times before being absorbed, in state $S_2$ approximately **2.6** times before being absorbed, etc. The column sums **+/N** which are equal to

```
5.41384 11.4518 5.90683 14.7128
```

give the average number of steps in each state before absorption, and, for example, the ladybug is in state $S_2$ approximately **11** times before being absorbed. Finally, the product **N mp R**, which has the value

```
0.40163 0.59837
0.21981 0.78019
0.18996 0.81004
0.27137 0.72863
```

gives the probability of absorption in each of the absorbing states given the initial state. For example, if the ladybug starts in $S_1$, the probability of being absorbed in $S_0$ is approximately **0.4** and in state $S_5$ approximately **0.6**.

As another example of a Markov chain consider two urns which contain a total of *n* balls numbered *1, 2, ..., n*. A number is chosen at random between *1* and *n*, and the ball with this number is transferred from the urn it is in to the other urn. This process is repeated a large number of times. Since it is more likely that a ball in the urn with the larger number of balls will be picked, it is intuitive that there will be flow of balls from the urn with the larger number of balls to the urn with the lesser number. We are interested in investigating the limiting distribution of the balls.

This process may be represented as a Markov chain with *n+1* states $S_0$, $S_1$, ..., $S_n$ where the state $S_i$ corresponds to their being *i* balls in the second urn. The transition probabilities are as follows:

$p_{ij} = i/n, \quad j = i - 1,$
$= 1 - i/n, \quad j = i + 1,$
$= 0, \quad$ otherwise,

for $i,j = 0, 1, 2, ..., n$. The list *a* with *n+1* items gives the initial distribution of the balls. As an example for ten balls the matrix of transition probabilities **U** is

```
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.1 0.0 0.9 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.2 0.0 0.8 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.3 0.0 0.7 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.4 0.0 0.6 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.5 0.0 0.5 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.6 0.0 0.4 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.7 0.0 0.3 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.8 0.0 0.2 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.9 0.0 0.1
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
```

If all of the balls were initially in the first urn, then the list **a** would have the value

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

The expression

```
a mp U (mp ^:N) U
```

will give the probability distribution of the balls between the two urns after **N+1** steps, and, for example, for **N** equal to **49** we have

```
0 0.019533 0 0.234386 0 0.492187 0
          0.234364 0 0.0195295 0 .
```

The average number of balls in the first urn after **N+1** steps is equal to

```
(|.i.11) mp a mp U (mp ^: N) U  ,
```

and we find that after **1**, **2**, **5**, **10** and **50** steps the average numbers of balls in the first urn are **9**, **8.2**, **6.6384**, **5.53687** and **5.00007**, respectively. It may be shown that the limiting distribution of the balls in the two urns is binomial. In other words, regardless of the number of balls in the first urn, after a large number of random replacements the distribution of balls is the same as if they had been distributed at random with each ball having the probability of **0.5** of being in either urn.

Finally we may simulate this urn model by the following monadic verb **urn** whose right argument is the three-item list **n s t**, where **n** is the number of balls initially in the first urn and **s*t** is the total number of random drawings and whose result is a list with **t+1** items given the proportion of balls in the first urn initially and after each **s** drawings:

```
urn=: 3 : 0
'n s t'=. y.
r=. +/A=. n$1
while. t >: #r do.
   r=. r, +/A=. A ~: 2|(i. n) fr
                  ?s#n
end.
r=. r % n
)
```

For example, **urn 10000 2000 10** gives the results after each of **10** sequences of **2000** drawings with **10000** balls in the first urn initially, and in one instance had the value

```
1 0.8374 0.729 0.6634 0.6124 0.5762
   0.547 0.5268 0.5184 0.5126
   0.5092
```

### Moving averages

We shall discuss smoothing a series by moving averages using quarterly data expressed as hundreds, given in Sprent (1988), on new houses completed in both the private and public sectors in Scotland from 1981 to 1983:

| | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| 1981 Private | | 29 | 28 | 27 |
| 1981 Public | | 24 | 20 | 19 |
| 1982 Private | 25 | 31 | 29 | 29 |
| 1982 Private | 13 | 12 | 10 | 13 |
| 1983 Public | 30 | 33 | 32 | |
| 1983 Private | 10 | 11 | 12 | |

These data are given as the two-item list **HS** with the private sector data as the first item and the public sector data as the second:

| 29 28 27 25 ... | 24 20 19 13 ... |
|---|---|

We shall illustrate smoothing with moving averages using the public sector data represented by **H**, where **H=. >{:HS**, which has the value

**24 20 19 13 12 . . .** .

The successive triples used for a three-term moving average are given by the expression **3<\H** which is

| 24 20 19 | 20 19 13 | 19 13 12 | . . . |
|---|---|---|---|

The corresponding moving averages, given by **3 am\H** and rounded to one decimal place, are

**21 17.3 14.7 11.7 . . .** .

A more satisfactory five-term weighted moving average may be found from the expression

**5 (1 2 2 2 1&AM)\H**

where **AM** is the verb

**AM=: ([: +/ *) % +/@[**

for a weighted mean with arbitrary weights, which gives the values

**17.5 14.75 12.75 . . .** .

Figure 5 in Appendix 2 gives the original data with the five-term moving averages shown as dotted lines.

### Windows programming

**J** is available on a number of computers and operating systems including Windows 95/NT, Macintosh and UNIX systems with the same core **J** language on all machines. Windows systems combine the language with a large collection of utilities for simplifying the exchange of objects with other Windows applications and with an application development environment for the construction of graphical interfaces for **J** applications. All of the usual controls such as push buttons, scroll bars, edit boxes, etc. are available as well as controls for Dynamic Data Exchange (DDE), Object Linking and Embedding (OLE) and Visual BASIC Extended (VBX).

The Windows form given in Figure 6 in Appendix 2 simulates the rolling of one, two or three four- or six-sided dice up to 200 times and displays the frequency distribution of the sums appearing on the rolls. The programming for the form is given in Appendix 5.

### A final example

As a final example we shall give an analysis of the one-armed bandit type of slot machine as described in Weaver (1963). The machine has three dials, each with 20 symbols from the selection cherries, oranges, lemons, plums, bells and bars with a typical configuration having the following numbers of symbols:

| | Dial 1 | Dial 2 | Dial 3 |
|---|---|---|---|
| Cherries | 7 | 7 | 0 |
| Oranges | 3 | 6 | 7 |
| Lemons | 3 | 0 | 4 |
| Plums | 5 | 1 | 5 |
| Bells | 1 | 3 | 3 |
| Bars | 1 | 3 | 1 |

The payoff in dollars for each play is given by the following table:

| Combination | | | Amt. | Prob. |
|---|---|---|---|---|
| Bar | Bar | Bar | 62 | 0.000375 |
| Bell | Bell | Bell | 18 | 0.001125 |
| Bell | Bell | Bar | 18 | 0.000375 |
| Plum | Plum | Plum | 14 | 0.003125 |
| Plum | Plum | Bar | 14 | 0.000625 |
| Orange | Orange | Orange | 10 | 0.015750 |
| Orange | Orange | Bar | 10 | 0.002250 |
| Cherry | Cherry | Lemon | 5 | 0.024500 |
| Cherry | Cherry | Bell | 5 | 0.018375 |
| Cherry | Cherry | * | 3 | 0.079625 |
| Any other combination | | | 0 | 0.853875 |

* Not a Lemon or Bell

The probabilities may be calculated very simply. First of all we note that the total number of combinations of selections on the three dials is **20 * 20 * 20** or **8000**. The number of ways of obtaining three bars is **1 * 3 * 1** or **3** so that the probability of winning the jackpot of **62** dollars is **3 % 8000** or **0.000375**. Similarly the probability of obtaining three bells and winning **18** dollars is **1 * 3 * 3 % 8000** or **0.001125**. The probabilitities for the first nine winning combinations may be calculated simply if we define the table

```
T=: >1 3 1;1 3 3;1 3 1;5 1 5;
    5 1 1;3 6 7;3 6 1;7 7 4;7 7 3
```

where the rows give the numbers of winning selections on each of the three dials for each of the combinations. Therefore, the numbers of combinations are ***/"1 T** or

**3 9 3 25 5 126 18 196 147**

and the probabilities are **(*/"1 T) % 8000** or

**0.000375 0.001125 0.000375 0.003125**
**0.000625 0.01575 0.00225 0.0245**
**0.018375** .

The last winning combination consists of one of the selections cherry-cherry-orange, cherry-cherry-plum or cherry-cherry-bar. Thus if we define the table

**T1=: >7 7 7;7 7 5;7 7 1** ,

the total number of ways of winning is **+/*/"1 T1** or **637**, and the corresponding probability is

**(+/*/"1 T1) % 8000**

or **0.079625**. The probability of winning a non-zero amount on any one play is the sum of all of these individual probabilities or **0.146125**. For convenience

we have listed these probabilities as the last column in the second table given above. The complementary probability of not winning anything is **-.0.146125** or **0.853875**.

If we let the probabilities be given by the list

```
Prob=: 0.000375 0.001125 0.000375
   0.003125 0.000625 0.01575
   0.00225 0.0245 0.018375 0.079625
   0.853875
```

and the payout by

```
WinAmt=: 62 18 18 14 14 10 10 5 5 3
                          0 ,
```

then the expected value of a game is

```
+/WinAmt * Prob
```

which has the value **0.736**. This means that for every dollar spent on the game the player can expect a return of about **74** cents.

Another method of calculating the probabilities is by use of the sample space representing the **8000** possible combinations of the three dials. Let us represent cherries, oranges, lemons, plums, bells and bars by **c**, **o**, **l**, **p**, **b** and **r**, respectively. Then the three dials may be represented by the lists

```
Dial1=: 'ccccccooolllpppppbr' ,
Dial2=: 'ccccccoooooopbbbrrr',
```

and

```
Dial3=: 'ooooooollllpppppbbbr'
```

which, of course, do not necessarily represent the sequences in which the symbols appear on the dials. If we let

```
Dials=: Dial1;Dial2;Dial3 ,
```

then the sample space may be represented by

```
S=: ,Dials
```

which is a list of **8000** items each of which is a three-item list giving one of the possible combinations of the dials. Now let **W1** be the list

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| rrr | bbb | bbr | ppp | ppr | ooo | oor | ccl | ccb |

of the first nine winning combinations, and **W2** be the list

| | | |
|---|---|---|
| cco | ccp | ccr |

.

of the combinations which together make up the last method of winning. The expression

```
'rrr' -: EACHRIGHT S
```

gives a list of **8000** items equal to **0** or **1** with the **1**s indicating those items of **S** equal to **rrr**, and thus

```
+/'rrr' -: EACHRIGHT S
```

which is equal to **3** gives the number of combinations giving the jackpot. The number of combinations for each of the first nine winning selections is given by

```
+/W1 -: EACHLEFT EACHRIGHT S
```

which is equal to

```
3 9 3 25 5 126 18 196 147 .
```

Similarly the total number of combinations for the last method of winning is

```
+/+/W2 -: EACHLEFT EACHRIGHT S
```

which is **637**. The corresponding probabilities may be calculated by dividing each of these numbers for the nine cases by **$S** or **8000**. The total number of items in the sample space giving a win is

```
+/+/(W1,W2) -: EACHLEFT EACHRIGHT S
```

which is **1169**. The number not giving a win is simply the number of items remaining in the sample space when the winning items are removed, and is **$S -. W1,W2** or **6831**. We shall define

```
WinComb=: W1,W2
```

which is the **12**-item list

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| rrr | bbb | bbr | ppp | ppr | ooo | oor | ccl |

| | | | |
|---|---|---|---|
| ccb | cco | ccp | ccr |

.

Let us now modify the lists **Prob** and **Amt** for the probabilities and corresponding amounts won by replacing the second last items in the lists, i.e., the values for the combined probabilities and amounts for cherry-cherry-orange, cherry-cherry-plum or cherry-cherry-bar, by the individual probabilities and amounts for these combinations. The three probabilities are given by

```
(+/W2 -: EACHLEFT EACHRIGHT S)%8000
```

which is equal to

```
0.042875 0.030625 0.006125 ,
```

and each of the amounts is **3**. Therefore, the probabilities and amounts are now given by the **13**-item lists

```
Prob=: 0.000375 0.001125 0.000375
   0.003125 0.000625 0.01575
   0.00225 0.0245 0.018375 0.042875
   0.030625 0.006125 0.853875
```

and

```
WinAmt=: 62 18 18 14 14 10 10 5 5
                       3 3 3 0 ,
```

respectively.

We shall begin a simulation of the slot machine by defining the dyadic verb

```
OnePlay=. (([:>[:1&{[) i. [:<] {
   EACH [:>[:0&{[) { [:>[:2&{[  ,
```

the details of which need not concern us, which gives the amount won for any specified combination of symbols. The left argument is the following three-item list: three-item list of dial configurations such as the list **Dials** given above; *n*-item list of winning combinations such as **WinComb**; and an (*n+1*)-item list of winning amounts with the last item being zero for all losing combinations such as **WinAmt**. The right argument is a three-item list

of indices for selecting a combination of symbols, one from each of the dials. For example, if we let

```
X=: Dials;WinComb;WinAmt
```

then

```
X OnePlay 19 18 19
```

is **62**,

```
X OnePlay 15 13 19
```

is **14**, and

```
X OnePlay 12 16 14
```

is **0**. These three expressions may be combined into the single expression

```
X OnePlay EACHRIGHT 19 18 19;
                    15 13 19;12 16 14
```

whose value is

```
62 14 0
```

.

The dyadic verb

```
SM1=. [ OnePlay EACHRIGHT
      [: <"1 [: ? (],3:) $ 20"1
```

simulates an arbitrary number of plays where the left argument is the same as for **OnePlay** and whose right argument is the number of plays. For example, **X SM1 15** may have the value

```
0 0 0 3 0 0 0 0 3 0 14 0 0 0 0
```

. Now a list of the unique amounts won is given by

```
AmtNub=. ~. WinAmt
```

which has the value

```
62 18 14 10 5 3 0
```

. Therefore, the observed frequencies for the amounts won in **8000** simulations may be found from the expression

```
s=. AmtNub fr X SM1 8000
```

where **fr** is the verb for frequencies over a specified range which was given earlier in the paper. Three evaluations of this expression gave the following values for **s**:

```
3 12 28 147 347 665 6798
2 13 29 132 319 666 6839
1 14 20 146 327 613 6879
```

The average amount won for every dollar spent is equal to

```
(s mp AmtNub) % 8000
```

, where **mp** gives the matrix product, which for the three values of **s** just given is equal to **0.74925**, **0.709625** and **0.691**, respectively. These values are consistent with a return of about **74** cents for every dollar invested which was calculated above from the probabilitities of winning.

The corresponding theoretical frequencies may be found very simply by finding the probabilities corresponding to the nub amounts and then multiplying by **8000** as shown by the following calculations:

```
ProbNub=. (=Amt) mp Prob
ProbNub
0.000375 0.0015 0.00375 0.018 0.042875
                0.079625 0.853875
8000 * ProbNub
3 12 30 144 343 637 6831
```

It is seen that the sample values of **s** given in the last paragraph are consistent with these values.

The dyadic verb

```
SM2=. 3 : 0
:
X=. x.
'Capital NumPlays NumReps'=. 3{.y.,1
R=. i. 0
while. NumReps > $R do.
   Total=. Capital
   Num=. 0
   while. (Num < NumPlays) *. (Total > 0) do.
      Total=. Total + <: X OnePlay ?3$20
      Num=. >:Num
   end.
   R=. R,<Num,Total
end.
)
```

allows simulations starting with a given initial capital and continuing for a given number of plays or until the capital is exhausted, whichever occurs first. The left argument is the same three-item list as the left argument in **SM1**, and the right argument is a three-item list giving the initial capital, the number of plays, and the number of repetitions. (If the number of repetitions is omitted, it is assumed to be **1**.) The result is a list of two-item lists giving for each repetition the number of plays and the final capital. For example,

```
X SM2 5 10 7
```

represents an initial capital of **5** dollars, a maximum of **10** plays and **7** repetitions and could give

| | | | | | | |
|---|---|---|---|---|---|---|
| 5 0 | 8 0 | 5 0 | 10 8 | 10 22 | 10 9 | 5 0 |

while

```
X SM2 5 10
```

represents a single repetition with an initial capital of **5** dollars and a maximum of **10** plays and could give

| |
|---|
| 10 17 |

.

Two other simulations are

```
X SM2 25 10 5
```

which gave

| | | | | |
|---|---|---|---|---|
| 10 18 | 10 36 | 10 18 | 10 18 | 10 25 |

and

```
X SM2 25 _ 5
```

| | | | | |
|---|---|---|---|---|
| 83 0 | 205 0 | 79 0 | 49 0 | 201 0 |

.

This latter result is of interest as it shows that one is certain to lose all of one's capital by playing a sufficient number of games.

A Windows form is given in Appendix 5 for playing an arbitrary number of games. The following variables are defined so that that upper-case letters can be used to represent the symbols:

```
DIAL1=: 'CCCCCCCOOOLLLPPPPPBR'
DIAL2=: 'CCCCCCCOOOOOOPBBBRRR'
DIAL3=: 'OOOOOOOLLLLPPPPPBBBR'
DIALS=: DIAL1;DIAL2;DIAL3
WINCOMB=: ;:'RRR BBB BBR PPP PPR
      OOO OOR CCL CCB CCO CCP CCR'
```

An example of the use of this form is given in Figure 7 of Appendix 2.


### Acknowledgements

Cliff Reiter read an early draft of this paper and made many helpful comments. Roger Hui from time to time gave valuable advice as he always has with my work. I am very grateful to these two persons for their help and encouragement and also to the many other persons in the **J** commmunity whom I have met or corresponded with.


### References


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

Burke, Chris, 1998. ***J*** *User Manual*. Iverson Software, Inc., Toronto.

Burke, Chris et al, 1998. *Phrases*. Iverson Software, Inc., Toronto.

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

Feller, William, 1950. Introduction to Proability Theory. Volume One. John Wiley & Sons, Inc., New York.

Gardner, M, 1977. *Mathematical Carnival*. Vintage Books, New York.

Gardner, M., 1978. *Mathematical Magic Show*. Vintage Books, New York.

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

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

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

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

Iverson, K. E., 1998. ***J*** *Introduction and Dictionary*. Iverson Software Inc., Toronto.

Kac, Mark and Stanislaw Ulam, 1971. *Mathematics and Logic*. Penguin Books Ltd., Harmondsworth, Middlesex.

Kemeny, J. G., A. Schleifer, J. L. Snell and G. L. Thompson, 1962. *Finite Mathematics with Business Applications*. Prentice-Hall, Inc., N. J.

Kendall, M. G. and A. Stewart, 1963. *The Advanced Theory of Statistics, Vol. 1, Second edition*. Hafner Publishing Company, New York.

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

Reiter, Clifford A., 1995. *Fractals Visualization with J*. Iverson Software Inc., Toronto.

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

Selby, S. M., 1971. *Standard Mathematical Tables. Nineteenth Edition.* The Chemical Rubber Company, Cleveland, Ohio.

Smillie, Keith, 1995. *Some Notes on Introducing J with Statistical Examples.* Revised Edition, 51 pp. (Unpublished).

Smillie, Keith 1996a. *Teaching with J - An Example from Statistics*, 8 pp. (Unpublished)

Smillie, Keith, 1996b. "Understanding data with J" *J User Conference Proceedings*, Toronto, Ontario, June 24 - 25, 25 pp.

Sprent, P., 1981. *Quick Staistics.* Penguin Books, Ltd., Harmonsworth, Middlesex.

Sprent, P., 1988. *Understanding Data.* Penguin Books, Ltd., Harmonsworth, Middlesex.

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

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

## Appendix 1. Summary of verbs

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

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

### Summarization

**am** -
**y**
Arithmetic mean of **y**

**gm** -
**y**
Geometric mean of **y**

**hm** -
**y**
Harmonic mean of **y**

**var** -
**y**
Variance of **y**

**sd** -
**y**
Standard deviation of **y**

**median** -
**y**
Median of **y**

**Q1** -
**y**
First quartile of **y**

**Q2** -
**y**
Second quartile of **y**

**Q3** -
**y**
Third quartile of **y**

**five** -
**y**
Min., first, second and third quartiles and max. of **y**

**summary** -
**y**
Summary statistics (with labels) of **y**

**mode** -
**y**
Mode of **y**

### Frequencies

**fr** **x** Range
**y** (Integer)
Frequencies over range of **y**

**frtab** **x** Range
**y** (Integer)
Frequency table over range of **y**

**nubfr** -
**y** (Integer)
Nub frequencies of **y**

**nubtab** -
**y** (Integer)
Frequency table over nub of **y**

**alphafrtab** -
**y** (Character)
Frequency table over nub of **y**

**cfr** **x** (End points of classes)
**y** (Integer or real)
Class frequencies

**cfrtab** **x** (As in **cfr**)
**y** (Integer)
Frequency table with mid-points in 1st col. and frequencies in 2nd

**FR** **x** (List of pairs giving ranges for each axis)
**y** (Integer pairs)
Two-way freq. table over range of **y**

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

**vbarchart** **x** (Range)
**y (**Frequencies)
Frequencies as * given vertically with no range

**SLtab** **-**
**y** (Integer)
Frequency table of stems

**stemtab** -
**y** (Integer)
Stem-and-leaf table

### Correlation and regression

**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

**SR** **x**
**y**
Simple linear regression with dep. var. **y** and indep. var. **x**

**REG** -
List, each item of which is a list with last item dep. variable
Multiple linear regression

### Analysis of variance

**AOV** [**x**]
Table or higher-dimensional array
AOV table with **x** giving specified terms (Default gives all terms, e.g., **AOV t** is **'A B AB' AOV t**)

**aov1** -
List, each item of which gives observations for one level in one-way AOV with unequal subclass numbers
AOV table

### Chi-square

**ExpFrTab** -
**t** (Obs. freq.)
**t** (Exp. freq.)

**chisq** **x** or **t** (Obs. freq.)
**x** or **t** (Exp. freq.)
Chi-square

**chisq22** -
**t** (Obs. freq. 2-by-2 table)
Probability

### Nonparametric statistics

**uranks** -
**y**
Ranks of items of **y** unadjusted for ties

**ranks** -
**y**
Ranks with ties averaged

**invranks** -
**y**
Ranks in inverse order

**rcor** **x**
**y**
Rank correlation coefficient between **x** and **y**

**runs** -
**y**
Number of runs

### Simulation

**Die** -
**n**
Results of rolling die **n** times

**Dice** **m**
**n**
Results of rolling **m** dice **n** times as an **n**-item list with **m** items in each item

**SumDice** **m**
**n**
**n**-item list of sums

**Heads** **m**
**n**
**n**-item list of no. of heads

**rand** -
**n**, **x**, …
Uniformly distributed random numbers,
e.g., **rand 3** is a **3**-item list,
**rand 2 4** is **2** by **4** table

**nmlrand** [**u**,**v**]
**n**
**n** normal deviates with mean **u** and
s.d. **v**. Default is standard normal

**exprand** **m**
**n**
Exponentially distributed random
numbers with mean **m**

### Probability distributions

**binomial** **n, p** (Number of trials and prob. of
success in a single trial)
**m** or **y** (Number of successes)
Probabilities

**poisson** **m** (Mean)
**n** or **y** (Number of successes)
Probabilities

**geometric** **p** (Probability of success in a single
trial)
**n** or **y** (Number of trials)
Probabilities

**hgeometric** **x** (3-item list giving no. in population
of type A, no. of type not-A,
sample size)
**n**, **x** (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

**chisq** **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

Cumulative probabilities for the last four distributions may be found by means of the integral adverb **I**, Iverson (1993), and typical uses are as follows:

```
ndistn I 0 1 2 3
5&tdistn I 2.015 2.571 3.365
10&csdistn I 12.5 16 18.3
5 20&fdistn I 2.16 2.71 3.29 4.1
```

## Appendix 2. Figures

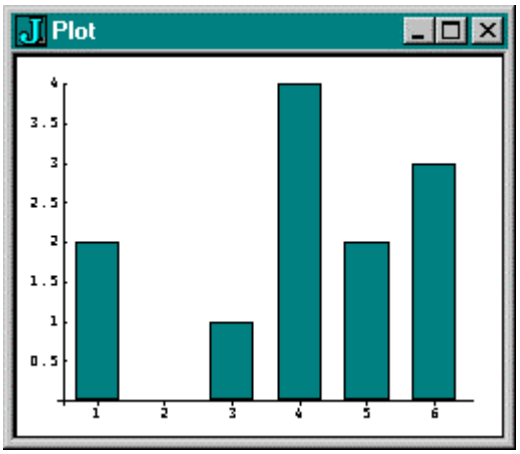



Figure 0a. Dice throwing

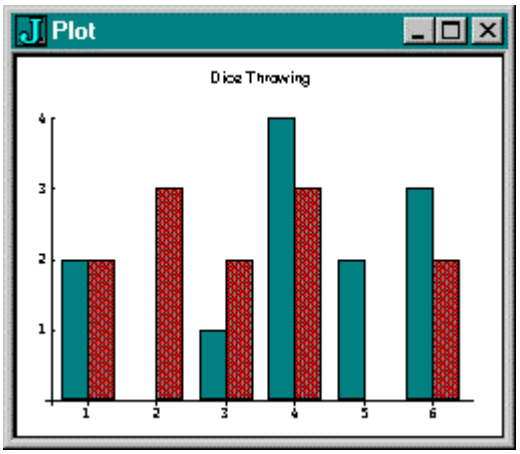



Figure 0b. Dice throwing

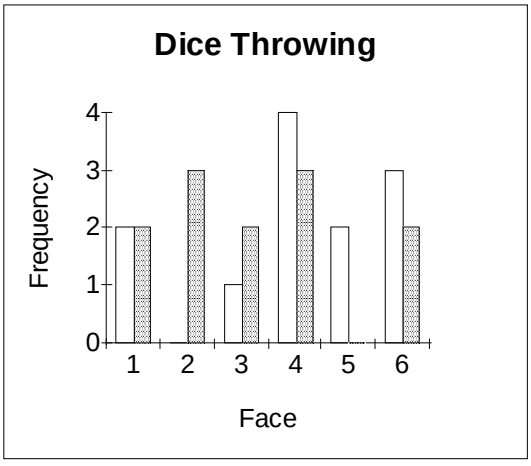



Figure 0c. Dice throwing

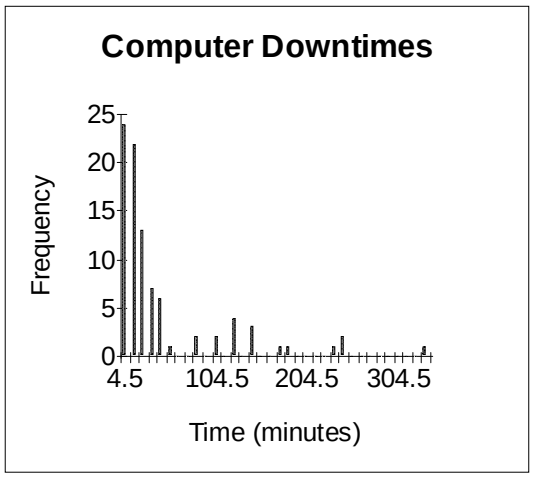



Figure 1. Computer downtimes

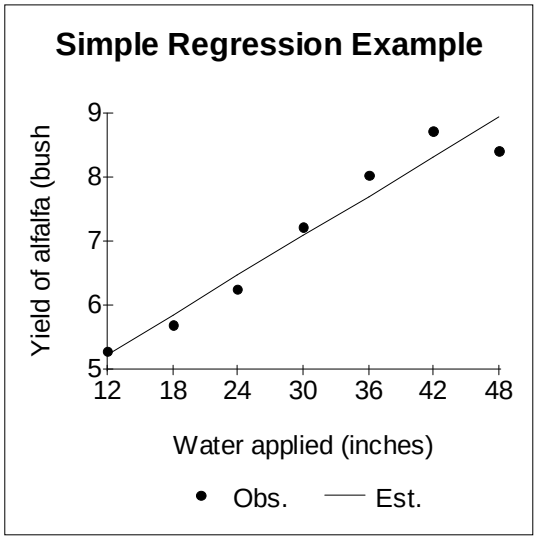



Figure 2. Simple regression example

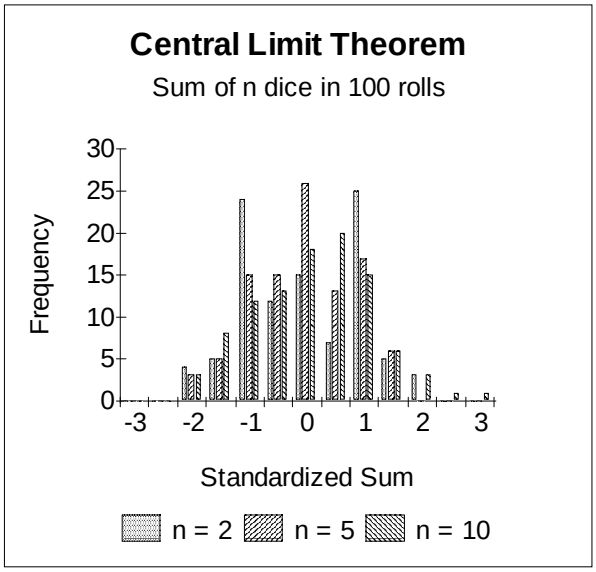



Figure 3. Central limit theorem

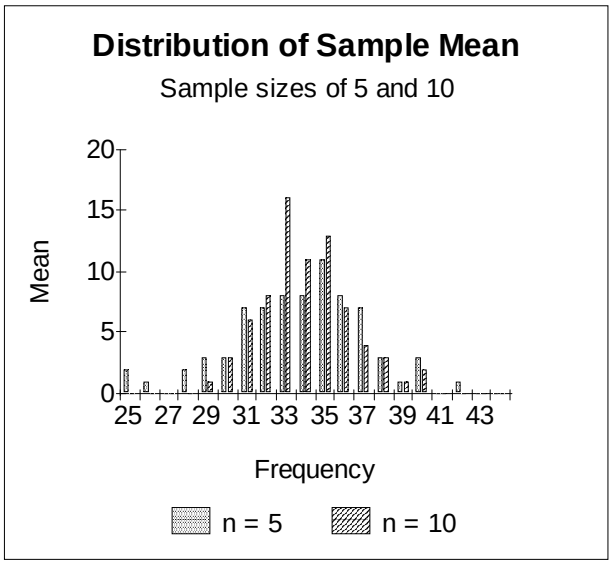



Figure 4. Distribution of sample mean

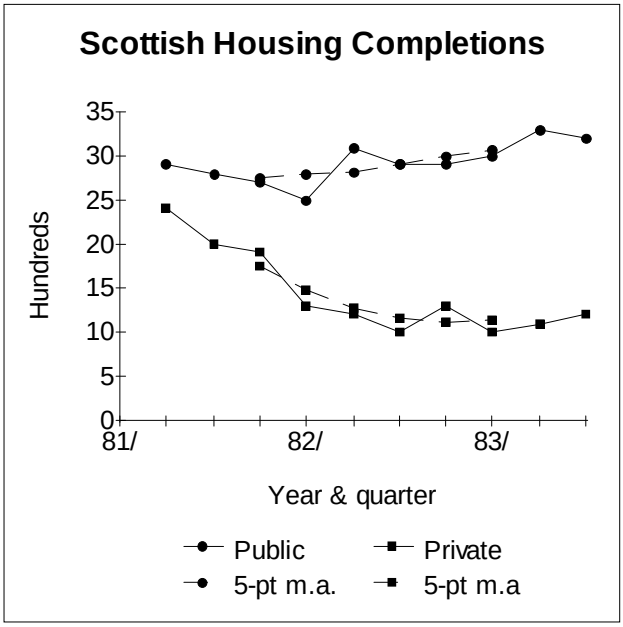



Figure 5. Moving averages

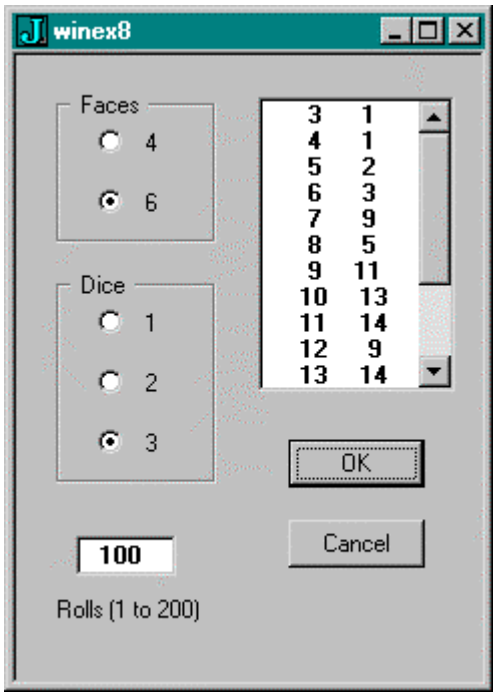



Figure 6. Windows form for dice rolling

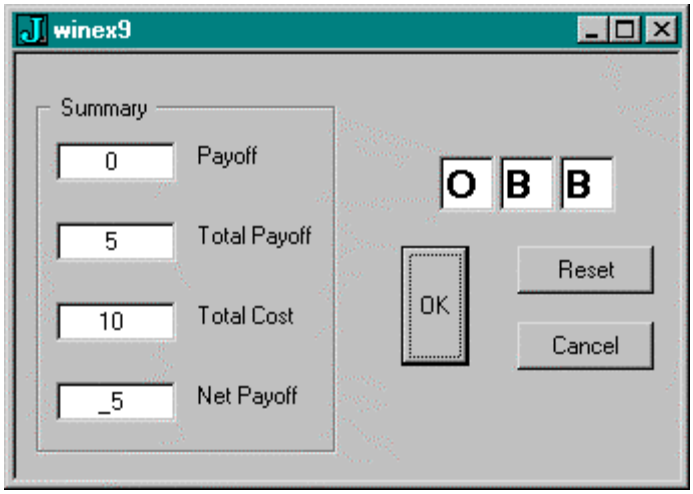



Figure 7. One-armed bandit

## Appendix 3. Table of computer breakdowns

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 842 | 4 | 49 | 13 | 1516 | 4 | 42 | 23 | 931 | 6 | 12 |
| 2 | 1035 | 2 | 18 | 14 | 1759 | 2 | 24 | 23 | 952 | 6 | 11 |
| 2 | 1529 | 1 | 123 | 15 | 803 | 4 | 7 | 23 | 1005 | 6 | 3 |
| 2 | 1735 | 1 | 3 | 15 | 817 | 4 | 9 | 23 | 1017 | 6 | 5 |
| 3 | 0 | 0 | 0 | 15 | 831 | 4 | 14 | 23 | 1024 | 6 | 9 |
| 4 | 1109 | 6 | 87 | 15 | 847 | 4 | 10 | 23 | 1038 | 6 | 3 |
| 4 | 1907 | 4 | 21 | 15 | 905 | 4 | 30 | 23 | 1402 | 1 | 22 |
| 4 | 2033 | 4 | 7 | 15 | 942 | 4 | 23 | 23 | 1823 | 1 | 17 |
| 4 | 2051 | 4 | 24 | 15 | 1009 | 4 | 141 | 24 | 0 | 0 | 0 |
| 5 | 835 | 4 | 11 | 15 | 1240 | 4 | 7 | 25 | 2129 | 3 | 20 |
| 5 | 922 | 4 | 19 | 15 | 1249 | 4 | 102 | 26 | 1340 | 6 | 5 |
| 6 | 900 | 5 | 243 | 15 | 1947 | 1 | 83 | 26 | 1545 | 6 | 3 |
| 7 | 1525 | 3 | 38 | 16 | 1429 | 3 | 29 | 26 | 1550 | 6 | 18 |
| 8 | 947 | 1 | 11 | 17 | 0 | 0 | 0 | 27 | 955 | 1 | 142 |
| 8 | 1010 | 1 | 18 | 18 | 0 | 0 | 0 | 27 | 1235 | 1 | 51 |
| 9 | 1432 | 2 | 32 | 19 | 1033 | 2 | 11 | 27 | 1707 | 2 | 18 |
| 9 | 1527 | 1 | 101 | 19 | 1102 | 2 | 15 | 27 | 1729 | 2 | 41 |
| 9 | 1712 | 1 | 6 | 19 | 1221 | 2 | 7 | 27 | 1944 | 4 | 9 |
| 9 | 1809 | 6 | 32 | 19 | 1234 | 2 | 6 | 28 | 1137 | 3 | 9 |
| 9 | 1931 | 4 | 27 | 19 | 1247 | 2 | 8 | 28 | 1430 | 6 | 27 |
| 9 | 2122 | 1 | 41 | 19 | 1342 | 2 | 121 | 28 | 1504 | 6 | 14 |
| 10 | 805 | 4 | 3 | 19 | 1829 | 3 | 4 | 29 | 913 | 3 | 33 |
| 11 | 1331 | 4 | 25 | 20 | 1328 | 6 | 23 | 29 | 952 | 1 | 128 |
| 11 | 1447 | 4 | 19 | 20 | 1700 | 3 | 42 | 29 | 1244 | 1 | 34 |
| 11 | 1702 | 6 | 242 | 20 | 1748 | 3 | 18 | 29 | 1359 | 6 | 18 |
| 12 | 0 | 0 | 0 | 21 | 904 | 6 | 31 | 29 | 1505 | 1 | 232 |
| 13 | 1022 | 1 | 122 | 21 | 1227 | 2 | 25 | 29 | 1901 | 1 | 179 |
| 13 | 1403 | 4 | 15 | 21 | 1513 | 4 | 42 | 29 | 2206 | 1 | 143 |
| 13 | 1422 | 4 | 7 | 21 | 1907 | 4 | 6 | 30 | 801 | 1 | 181 |
| 13 | 1433 | 4 | 6 | 22 | 800 | 5 | 339 | 30 | 1212 | 4 | 6 |
| 13 | 1445 | 4 | 2 | 22 | 1926 | 2 | 19 | 31 | 2105 | 6 | 14 |
| 13 | 1451 | 4 | 18 | 22 | 2240 | 3 | 27 | | | | |

The columns represent date, time of breakdown, type of breakdown and length in minutes. The type of equipment is coded as follows: **0** No breakdown, **1** CPU, **2** Disk Drive, **3** Graph Plotter, **4** Printer, **5** Service and **6** Tape Reader.

## Appendix 4. REG and AOV verbs

### Multiple linear regression

```
NB. Requires am

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

### Analysis of variance

```
NB. Previously defined verbs
msum=: 4 : '+/^:(+/-.x.)(/:x.)|:y.'          NB. Marginal sums
mnum=: [: */ -.@[ # $@]                      NB. Marginal numbers
msgn=: _1:^~:/                               NB. Marginal signs
T=: msgn@[ * mnum %~ [: +/ [: , [: *: msum   NB. Signed weighted SS
tt=: #: @ i. @ (2&^)                        NB. Truth table
ott=: (/: +/"1) @ tt                         NB. Ordered truth table

NB. Test data
D2=: >4 7 5 6;9 4 3 8;2 5 7 3
   D2
4 7 5 6
9 4 3 8
2 5 7 3

NB. Expand columns
expand=: [ #^:_1"1 ]
   0 1 0 1 0 0 expand i. 4 2
0 0 0 1 0 0
0 2 0 3 0 0
0 4 0 5 0 0
0 6 0 7 0 0
```

```
NB. All terms for arbitrary array
AllTerms=: [: |."1 [: }. ott@#@$
   AllTerms D2
1 0
0 1
1 1

NB. Subterms for given term
SubTerms=: [: |. ] expand [: tt +/
   SubTerms 0 1 0 1 0
0 1 0 1 0
0 1 0 0 0
0 0 0 1 0
0 0 0 0 0

NB. Single sum of squares
SS1=: [: | [: +/ SubTerms@[ T"1 _ ]
   1 0 SS1 D2
6.5

NB. All sums of squares
SS=: [ SS1"1 _ ]
   (>1 0;0 1;1 1) SS D2
6.5 0.9166667 44.8333

NB. All degrees of freedom
DF=: [: */"1[: |@<: [ #"2 $@]
   (>1 0;0 1;1 1) DF D2
2 3 6

NB. Factors A, B, ...
Factors=: [: 'ABCDEF'&({.~) #@$
   Factors D2
AB

NB. All labels for main terms, two-factor terms, etc.
AllLabels=: AllTerms # Factors
   AllLabels D2
A
B
AB
```

```
NB. Analysis-of-variance
AOV=: 3 : 0
(AllLabels y.) AOV y.
:
Terms=. (Factors Data=. y.)e."1 >Labels=. ;:,x.
AOVtable=: Terms (DF,. SS) Data
Total=. (<:*/$Data), (+/*:,Data) - (*:+/,Data)%*/$Data
Error=. Total - +/AOVtable
if. {.Error > 0 do.
   AOVtable=: AOVtable,Error
   Labels=. Labels,<'Error  '
end.
AOVtable=: (AOVtable,. %~/"1 AOVtable), Total,0
r=. ((5 12.5 12.5)":}:AOVtable), (5 12.5":}:{:AOVtable),12$' '
(>Labels,<'Total'),.r
)

   AOV D2
A        2     6.50000     3.25000
B        3     0.91667     0.30556
AB       6    44.83333     7.47222
Total   11    52.25000

   'A AB' AOV D2
A          2     6.50000     3.25000
AB         6    44.83333     7.47222
Error      3     0.91667     0.30556
Total     11    52.25000
```

```
NB. *** Rolling 1, 2 or 3 dice with 4 or 6 faces ***

NB. Requires pos, each, EACH, fr, frtab, WDtable

WinEx8=: 0 : 0
pc winex8;
xywh 68 96 34 12;cc ok button;cn "OK";
xywh 68 116 34 12;cc cancel button;cn "Cancel";
xywh 10 9 40 38;cc frame0 groupbox;cn " Faces ";
xywh 20 15 20 14;cc rb0 radiobutton;cn "  4";
xywh 20 30 20 14;cc rb1 radiobutton group;cn "  6";
xywh 10 54 40 53;cc frame1 groupbox;cn " Dice ";
xywh 20 60 20 14;cc rb2 radiobutton;cn "  1";
xywh 20 75 20 14;cc rb3 radiobutton group;cn "  2";
xywh 20 90 20 14;cc rb4 radiobutton group;cn "  3";
xywh 15 120 25 10;cc rolls edit;
xywh 10 135 40 15;cc rollsname static;cn "Rolls (1 to 200)";
xywh 60 10 50 75;cc frtable listbox ws_border ws_vscroll;
pas 6 6;pcenter;
rem form end;
)

winex8_run=: 3 : 0
wd WinEx8
wd 'set rolls *',' 0 '
wd 'setfont rolls "MS Sans Serif" 11 bold;'
wd 'setfont frtable "MS Sans Serif" 11 bold;'
NB. DICEtab=:i. 0 2
wd 'pshow;'
)

winex8_ok_button=: 3 : 0
R=: ". rolls
R=. R * R e. pos 200
NB. wd 'set frtable *',''
if. R > 0 do.
   F=. ('1' = rb0,rb1) # 4 6
   D=. ('1' = rb2,rb3,rb4) #  1 2 3
   Range=. (D-1) }. pos F*D
   Sums =. +/ EACH >: each ? each R$<D$F
   DICEtab=: Range frtab Sums
   wd 'set frtable *',6.0 WDtable DICEtab
end.
)

winex8_cancel_button=: 3 : 0
wd 'pclose;'
)
```

```
NB. *** One-armed bandit ***

NB. Requires EACH, DIAL1, DIAL2, DIAL3, DIALS, WINCOMB, WinAmt

WINEX9=: 0 : 0
pc winex9;
xywh 105 25 15 15;cc symbol1 edit ws_border;
xywh 120 25 15 15;cc symbol2 edit ws_border;
xywh 135 25 15 15;cc symbol3 edit ws_border;
xywh 5 10 75 90;cc output groupbox;cn " Summary ";
xywh 10 22 30 10;cc pay edit;
xywh 45 22 25 10;cc payname static;cn "Payoff";
xywh 10 42 30 10;cc totpay edit;
xywh 45 42 30 10;cc totpayname static;cn "Total Payoff";
xywh 10 62 30 10;cc totcost edit;
xywh 45 62 25 10;cc totcostname static;cn "Total Cost";
xywh 10 82 30 10;cc netpay edit;
xywh 45 82 25 10;cc netpayname static;cn "Net Payoff";
xywh 96 48 17 30;cc ok button;cn "OK";
xywh 125 48 34 12;cc reset button;cn "Reset";
xywh 125 67 34 12;cc cancel button;cn "Cancel";
pas 6 6;pcenter;
rem form end;
)

Reset=: 3 : 0
Pay=: 0
TotPay=: 0
TotCost=: 0
NetPay=: 0
wd 'set pay *',8.0&":Pay
wd 'set totpay *',8.0&":TotPay
wd 'set totcost *',8.0&":TotCost
wd 'set netpay *',8.0&":NetPay
wd 'set symbol1 *',' '
wd 'set symbol2 *',' '
wd 'set symbol3 *',' '
)

winex9_run=: 3 : 0
wd WINEX9
Reset ''
wd 'setfont symbol1 "MS Sans Serif" 18 bold;'
wd 'setfont symbol2 "MS Sans Serif" 18 bold;'
wd 'setfont symbol3 "MS Sans Serif" 18 bold;'
wd 'pshow;'
)

winex9_close=: 3 : 0
wd'pclose'
)

winex9_reset_button=: 3 : 0
Reset ''
)
```

```
winex9_cancel_button=: 3 : 0
winex9_close''
)

winex9_ok_button=: 3 : 0
'S1 S2 S3'=: S=: (? 3$20) { EACH DIALS
wd 'set symbol1 *',S1
wd 'set symbol2 *',S2
wd 'set symbol3 *',S3
Pay=: ((>WINCOMB) i. S) { WinAmt
TotPay=: TotPay + Pay
TotCost=: >:TotCost
NetPay=: TotPay - TotCost
wd 'set pay *',8.0&":Pay
wd 'set totpay *',8.0&":TotPay
wd 'set totcost *',8.0&":TotCost
wd 'set netpay *',8.0&":NetPay
)
```

## Appendix 6. Utilities

```
NB. General utilities
ap=: {.@]+1&{@]*i.@{:@]      NB. Arithmetic progression
by=: ' '&;@,.@[,.]           NB. By (format)
dp=: ($@[ * $@]) $ [: , 0 2 1 3&|: @ (*/) NB. Direct product
diff=: 2: (-~/\) ]           NB. First differences
EACH=: &>                    NB. Open each
each=: &.>                   NB. Boxed each
ei=: i.@>:                   NB. Extended integers
id=: =@i.                    NB. Identity matrix
io=: [:<:[:+/[</]            NB. Index of for intervals
iones=: +/{.\:               NB. Indices of all 1s
midpts=: [:-:2:+/\]          NB. Mid points
mnum=: [: */ -.@[ # $@]      NB. Marginal numbers
mp=: +/ . *                  NB. Matrix product
msum=: 4 : '+/^:(+/-.x.)(/:x.)|:y.' NB. Marginal sums
over=: ({.,.@;}.)@":@,       NB. Over (format)
pos=: >:@i.                  NB. Positive integers
rand=: (?%])@($&1e9)         NB. Random integers in (0,1)
sort=: /:~                   NB. Sort up
subtab=: <@[ { ]             NB. Sub table
table=: 1 : '[by]over x./'   NB. Table adverb

NB. Input/Output utilities
hfu=: '_-'&charsub           NB. Hyphen from underbar
ufh=: '-_'&charsub           NB. Underbar from hyphen
Jfmt=: (".&>)@chop@toJ       NB. J format
CLIPwrite=: wdclipwrite@hfu@clipfmt
CLIPread=: Jfmt@ufh@wdclipread

NB. Integration utilities
W=: +:(#.|.@])"0 1|:@(0:,%.@(^/~@i)%>:@i=: i.@>:@+:)
ew=: +/@(-@(<:@(#@])*i.@[)|."0 1]{.~>:@([*<:@(#@]))
EW=: ([ ew W@]) f.
ai=: 2 : 0
ai=: +/@(x.&space * x.&[ * y.@(x.&grid))"0
)
grid=: space *i.@#@[
space=: ] % <:@#@[
I=: (4 EW 5) ai

NB. Windows utilities
WDtable=: 3 : 0    NB. Display table
:
;(<"1 (x. ": y.)), each LF
)
```