Basic Use of S

Examples

The **prod** function can be used to compute factorials. For example 10!.

```
> prod(1:10)
[1] 3628800
```

Beware that very large factorials can't be represented with finite precision arithmetic (200! is too large).

```
> prod(1:200)
[1] Inf
```

The following computation shows how to compute binomial coefficients.

```
> prod(1:10) / prod(1:3, 1:7)
[1] 120
```

The Recycling Rule

- Operations are carried out on vectors according to an *element recycling rule*.
- Under this rule, when vectors of different lengths are combined in an arithmetic operation,
  - The shorter vector is first enlarged to match the length of the longer vector by recycling its elements.
  - The vectors are then combined element by element.

Example

Consider adding the vectors **1:3** and **1:6**.

$$\begin{bmatrix}1\\2\\3\end{bmatrix} + \begin{bmatrix}1\\2\\3\\4\\5\\6\end{bmatrix} \xrightarrow{recycle} \begin{bmatrix}1\\2\\3\\1\\2\\3\end{bmatrix} + \begin{bmatrix}1\\2\\3\\4\\5\\6\end{bmatrix} \xrightarrow{add} \begin{bmatrix}2\\4\\6\\5\\7\\9\end{bmatrix}$$

This is how the result appears when computed with S.

```
> 1:3 + 1:6
[1] 2 4 6 5 7 9
```

Cumulative Summaries

In addition to the simple summary functions above, there are also cumulative versions called **cumsum**, **cumprod**, **cummax**, and **cummin**, which produce a vector which consists of the summary computed for; the first element, first two elements, the first three elements and so on.

```
> cumsum(1:10)
 [1]  1  3  6 10 15 21 28 36 45 55

> cumprod(1:10)
 [1]       1       2       6      24
 [5]     120     720    5040   40320
 [9]  362880 3628800
```

Summary Operations on Vectors

The following functions return single-value numerical summaries for the values in a vector or collection of vectors.

sum(x)	the sum of the elements of **x**
prod(x)	the product of the elements of **x**
min(x)	the minimum of the elements of **x**
max(x)	the maximum of the elements of **x**

The **range** function computes a two value summary.

```
> range(1:10)
[1]  1 10
```

Statistical Summaries

There are a number of summaries which compute quantities of statistical interest. The most important of these are:

mean(x)	the mean of the elements of **x**,
median(x)	the median of the elements of **x**,
var(x)	the variance of the elements of **x**.

In addition to these, R also has

sd(x)	the s.d. of the elements of **x**.

Sample Quantiles

The **quantile** function computes summaries based on the percentiles of the values in its argument. The simplest use of **quantile** computes the median, upper and lower quartiles, and extremes.

```
> quantile(1:10)
   0%   25%   50%   75%  100%
 1.00  3.25  5.50  7.75 10.00
```

An optional second argument to **quantile** can be used to specify a different set of quantiles.

```
> quantile(1:10, 0:10/10)
  0%  10%  20%  30%  40%  50%  60%  70%
```

Type Coercion

It is possible to combine numbers and character strings into a single vector using "**c**". In this case, the numbers are first converted to strings.

```
> c(greeting, 100)
[1] "hello, world" "100"
```

This an example of automatic *type coercion*.

Arithmetic on a mixture of strings and numbers does not work, however.

```
> "100" + 10
Error in "100" + 10 : non-numeric argument
                      to binary operator
```

```
 1.0  1.9  2.8  3.7  4.6  5.5  6.4  7.3
 80%  90% 100%
 8.2  9.1 10.0
```

Naming Vector Elements

Character strings can be used to name the individual elements of a vector.

```
> x <- 1:5
> x
[1] 1 2 3 4 5
> l <- c("a","b","c","d","e")
> l
[1] "a" "b" "c" "d" "e"
> names(x) <- l
> x
a b c d e
1 2 3 4 5
```

Character Strings

Any value in quotes is regarded by R as being a character string.

```
> greeting <- "hello, world"
> greeting
[1] "hello, world"
```

This is true, even when the value appears to be numeric.

```
> number <- "21"
> number
[1] "21"
```

Character strings can be combined to form vectors.

When a named vector prints, the names of the elements are printed above the elements and the indexing information at the start of a line is dropped.

```
> c(greeting, number)
[1] "hello, world" "21"
```

Names and Subsetting

Names are preserved during subsetting,

```
> x[4:5]
d e
4 5
```

and it is possible to extract subsets by using the names as subscripts.

```
> x[c("a","d")]
a d
1 4
```

Matrices

In addition to vectors, S has a wide range of data structures.

```
> A <- matrix(1:6, nrow = 3, ncol = 2)
```

creates a 3×2 matrix. The value can be viewed as follows.

```
> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Notice that the elements are inserted into the matrix in *column major order*.

Diagonal Matrices

The function **diag** can be used to create diagonal matrices and to extract the diagonals from matrices.

```
> A <- diag(1:3)
> A
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    2    0
[3,]    0    0    3

> diag(A)
[1] 1 2 3
```

Row-Major Order

By default, values are inserted into matrices in column major order. It is also possible to specify that the matrix be filled in row major order.

```
> B <- matrix(1:6, nrow = 3, ncol = 2,
+             byrow = TRUE)

> B
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Special Matrix Forms

There is a special form of call to **diag** which can be used to create identity matrices.

```
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

It is also easy to construct a block matrix of ones.

```
> matrix(1, nrow = 3, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    1    1
```

Dimensioning Information

Matrix dimensions can be obtained in a number of ways.

```
> A <- matrix(1:6, nrow = 3, ncol = 2)

> row(A)
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3

> ncol(A)
[1] 2

> dim(A)
```

```
[2,]    1    1    1
[3,]    1    1    1
```

```
[1] 3 2
```

Matrix Arithmetic

The standard arithmetic operations are all defined for matrices, and take place elementwise.

```
> A <- matrix(1:6, nrow = 3, ncol = 2)
> B <- matrix(1:6, nrow = 3, ncol = 2,
+             byrow = TRUE)
> A + B
     [,1] [,2]
[1,]    2    6
[2,]    5    9
[3,]    8   12
```

In particular, note that **A * B** is the elementwise product of **A** and **B**, not the matrix product.

Other Matrix Operations

The function **t** computes the transpose of its argument.

```
> t(A)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Matrix multiplication can be performed with the **%*%** binary operator.

```
> t(A) %*% B
     [,1] [,2]
[1,]   22   28
[2,]   49   64
```

Regression

The general linear model can be written in matrix form as

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon.$$

The least-squares estimates of the parameters are given by

$$\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

and the residuals by

$$\hat{\varepsilon} = \mathbf{y} - \mathbf{X}\hat{\beta}.$$

The estimated dispersion matrix of $\hat{\beta}$ is

$$\mathcal{D}[\hat{\beta}] = \frac{\hat{\varepsilon}'\hat{\varepsilon}}{n-p}(\mathbf{X}'\mathbf{X})^{-1}.$$

Solving Linear Equations

The **solve** function an be used to solve systems of linear equations. For example the linear system

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

can be solved as follows

```
> A <- matrix(c(1, 3, 2, 4), ncol = 2)
> b <- c(1, 1)
> solve(A, b)
[1] -1  1
```

Regression In S

The equations for regression analysis can be translated directly into S statements.

```
> n <- nrow(x)
> p <- ncol(x)
> betahat <- solve(t(x) %*% x, t(x) %*% y)
> epsilonhat <- y - x %*% betahat
> sigmahat2 <- sum(epsilonhat^2) / (n - p)
> D <- sigmahat2 * solve(t(x) %*% x)
```

Warning: This is a direct coding of the mathematical equations associated with regression analysis. It is **not** the best way of computing the results.

Matrix Inversion

When **solve** is called with single argument, it computes the matrix inverse that argument.

```
> solve(A)
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

You do, however, need to remember that computers only work to a finite precision and that all computations are subject to roundoff error.

```
> A %*% solve(A)
     [,1]         [,2]
```

Matrix Decompositions

There are many S functions which support computations on matrices. The most useful are:

eigen	eigenvalues and eigenvectors
svd	singular-value decomposition
qr	QR decomposition

```
[1,]    1 1.110223e-16
[2,]    0 1.000000e+00
```

Subsetting

It is possible to extract submatrices from matrices.

```
> A <- matrix(1:6, ncol=3)
> A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> A[1,2]
[1] 3

> A[1:2,1:2]
     [,1] [,2]
[1,]    1    3
```

```
[2,]    2    4
```

Example: Zeroing A Lower Triangle

row and **col** can be used together with logical subscripting to operate on upper and lower triangles of matrices.

```
> A <- matrix(1:9, ncol=3)
> A[row(A) > col(A)] <- 0
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    0    5    8
[3,]    0    0    9
```

Subsetting Shorthands

An empty subscript field is equivalent to the full range of possible subscripts. For example:

```
> A[,2:3]
     [,1] [,2]
[1,]    3    5
[2,]    4    6
```

extracts the second and third columns of **A** and

```
> A[2,]
[1] 2 4 6
```

extracts the second row.

Row and Column Labeling

Matrices can be made rather more useful by using row and column labels.

```
> A <- matrix(1:6, nrow=3)
> dimnames(A) <-
+       list(c("sex","drugs","rock&roll"),
+            c("this", "that"))
```

The primary benefit of labeling can be seen when the matrix is printed.

```
> A
           this that
sex           1    4
drugs         2    5
rock&roll     3    6
```

Subscripting matrices by using logical indices and row or column labels is also possible, but tends to be far less common.

Row and Column Indices

The two functions **row** and **column** return matrices containing the index for each row an column.

```
> A <- matrix(1:6, ncol=3)

> row(A)
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    2    2    2

> col(A)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
```

Labeling The Labels

In R, but not S, it is possible to “label the labels.”

```
> A <- matrix(1:6, nrow=3)
> dimnames(A) <-
+       list(what=c("sex","drugs","rock&roll"),
+            which=c("this", "that"))
> A
           which
what        this that
  sex          1    4
  drugs        2    5
  rock&roll    3    6
```

Multiway Arrays

Multiway arrays generalise the notion of matrices.

- Arrays are created with the **`array`** function and their subsetting and labeling methods parallel those of matrices.
- The only major difference is that the notion of transpose must be generalized.
- The function **`aperm`** provides such a generalized transpose operation.