Simple Computing in S-Plus

Mathematical computing is an integral part of any data analysis and therefore `S-Plus` has several capabilities for carrying out different kinds of computations. This section of these notes introduces the reader to several useful facts about mathematical computing in `S-Plus`.

Arithmetic Operations and Elementary Functions

Basic arithmetic operations can be carried out by using the well known operators `+`, `-`, `*`, `/`.

```
> 7+3
[1] 10
> 15-19
[1] -4
> 4*67
[1] 268
> 56/9
[1] 6.222222
```

The operator `^` is useful for exponentiation and root extraction.

```
> 2^6
[1] 64
> 2^(1/3)
[1] 1.259921
```

Some other important functions are `%/%` (integer divide operator), `%%` (modulo operator), `abs()` (absolute value function), `floor()` (greatest integer function), and `ceiling()` (next integer function).

```
> 27%/%3.4
[1] 7
> 27%%3.4
[1] 3.2
> 7*3.4+3.2
[1] 27
```

```
> abs(-10.56)
[1] 10.56
> floor(5.6)
[1] 5
> ceiling(5.6)
[1] 6
```

Here is how you can use these commands when operating in vectors and matrices.

```
> x <- c(1,4,7)
> y <- c(2,4,6,4,6,10)
> A <- matrix(c(2,3,4,5,6,7,1,2,3), nrow=3)
> A
     [,1] [,2] [,3]
[1,]    2    5    1
[2,]    3    6    2
[3,]    4    7    3
> B <- rbind(c(0,0,1), c(2,4,5), c(1,4,2))
> B
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    2    4    5
[3,]    1    4    2
> A*B
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    6   24   10
[3,]    4   28    6
> x+y
[1]  3  8 13  5 10 17
> A/y
          [,1] [,2] [,3]
[1,] 1.0000000 1.25  0.5
[2,] 0.7500000 1.00  0.5
[3,] 0.6666667 0.70  0.5
Warning messages:
  Length of longer object is not a multiple
  of the length of the shorter object in: A/y
> A%*%B   #matrix multiplication
     [,1] [,2] [,3]
[1,]   11   24   29
[2,]   14   32   37
[3,]   17   40   45
> z <- c(2,3,1)
> z%*%x   #vector dot product
     [,1]
[1,]   21
```

Most calculations on vectors or matrices are carried out *element by element* provided that the matrices have the same dimension. Foe vectors, if the one vector is shorter than the other, then the shorter vector is repeated cyclically to match the length of the longer vector. Mathematical operations on combination of vector and matrices have usually unexpected results.

Some well known built in functions in `S-plus` are `sqrt`, `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `exp`, `log`, `log10`, `gamma`, `lgamma`. They act element by element to their arguments.

```
> log(x)
[1] 0.000000 1.386294 1.945910
> log(x, base=2)   #logaritm to base 2
[1] 0.000000 2.000000 2.807355
> cos(A)
           [,1]      [,2]       [,3]
[1,] -0.4161468 0.2836622  0.5403023
[2,] -0.9899925 0.9601703 -0.4161468
[3,] -0.6536436 0.7539023 -0.9899925
> atan(A)
         [,1]     [,2]      [,3]
[1,] 1.107149 1.373401 0.7853982
[2,] 1.249046 1.405648 1.1071487
[3,] 1.325818 1.428899 1.2490458
> exp(y)
[1]     7.389056    54.598150   403.428793    54.598150   403.428793 22026.465795
```

Vector and Matrix Computations

The following function refers to the computation of a vector norm:

$$|\mathbf{x}| = \left(\sum_{i=1}^{n} x_i^p \right)^{1/p}$$

```
> vecnorm(x)  # Eucliden norm
[1] 8.124038
> vecnorm(x, p=1)
[1] 12
> vecnorm(x, p=Inf)
[1] 7
```

Here are some useful functions for matrix manipulations.

```
> t(A)    # transpose of a matrix
     [,1] [,2] [,3]
[1,]    2    3    4
[2,]    5    6    7
```

```
[3,]    1    2    3
> diag(A)   # extract the diagonal
[1] 2 6 3
> sum(diag(A))   # trace of a matrix
[1] 11
> X <- diag(c(1,2,3,4)) # create a diagonal matrix
> X
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    2    0    0
[3,]    0    0    3    0
[4,]    0    0    0    4
> I <-  diag(4) # create an identity matrix
> I
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
> eigen(A)    # compute eigenvalues and eigenvectors of a matrix
$values:
[1]  1.072015e+001  2.798467e-001 -1.887379e-015

$vectors:
           [,1]      [,2]       [,3]
[1,] -0.4902022 -2.332769 -0.7817656
[2,] -0.6806916  0.239993  0.1954414
[3,] -0.8711809  2.812755  0.5863242
> prod(eigen(A)$values)   # determinant
[1] -5.662137e-015
```

You can also use the functions `kronecker` (for a Kronecker product of two matrices), `qr` (for the QR decomposition), `svd` (for the singular value decomposition) and `chol` (for the Choleski decomposition).

Linear Systems of Equations

To solve a system of equations, like $A\mathbf{x} = \mathbf{y}$ it is convenient to define the matrix A and then use the `solve` function to get the solution, provided that there exists. For example, consider

$$\begin{aligned} 2x + 3y &= 13 \\ x - 2y &= -4 \end{aligned}$$

Then

```
> A <- rbind( c(2,3), c(1,-2))
```

```
> A
     [,1] [,2]
[1,]    2    3
[2,]    1   -2
> solve(A, c(13,-4))
[1] 2 3
> solve(A)  # getting the inverse
          [,1]       [,2]
[1,] 0.2857143  0.4285714
[2,] 0.1428571 -0.2857143
> solve(rbind(c(1,2), c(2,4))) # getting the inverse of a singular matrix
Error in solve.qr(a): apparently singular matrix
```

More functions related to matrix computations can be found in the library `matrix` which can be called with `library(matrix)`.

Random Numbers

There are many functions available for random number generation and probability calculations including outcomes related to the most common distributions. Each of these functions has a name beginning with on of the following four one-letter codes indicating the type of function.

`r`: Random number generator.

`p`: Probability function ($F(x) = P[X \leq x]$).

`d`: Density function ($f(x)$).

`q`: Quantile function ($F^{-1}(x)$).

The following table lists the most important distributions in S-plus.

`beta`	Beta Distribution
`binom`	Binomial Distribution
`chisq`	Chi–square Distribution
`gamma`	Gamma Disribution
`lnorm`	Lognormal Distribution
`norm`	Normal Distribution
`pois`	Poisson Distribution
`t`	t Distribution
`unif`	Uniform Distribution

Here are some examples of how you can use these functions

```
> x
[1] 1 4 7
```

```
> pnorm(x)
[1] 0.8413447 0.9999683 1.0000000
> pnorm(x, mean=2, sd=2)
[1] 0.3085375 0.8413447 0.9937903
> dnorm(x)
[1] 2.419707e-001 1.338302e-004 9.134720e-012
> qchisq(c(0.90,0.95,0.99), 2)
[1] 4.605170 5.991465 9.210340
> runif(30, -10, 10)
 [1]  9.213183280  8.749200171 -9.117961340  5.292370273  4.117153846  0.071010422
 [7]  8.572964445  6.805462409  0.942033436 -0.243897894 -2.020305358 -4.729607552
[13]  8.518492579 -1.429708684  9.201227576 -4.033246860  1.544312881 -0.227093771
[19] -6.805263087 -6.349458788 -5.736339493 -4.680284206  4.654475767  2.877361933
[25]  7.949797967 -0.003705453  1.538872020  8.116273358 -9.711499065  4.931
```

Some Other Useful Functions

There are several other useful functions that can be used for computing but we will not examine all of them in detail. Notably, we mention the function `integrate` which can be used to compute the integral of a real valued function over a given interval, the function `diff` which returns the nth difference of lag k for a set of data and the function `fft` which gives the fast Fourier transform of a data set.

Here is an example of the `stepfun` which computes a left-continuous step function from (x,y) points.

```
 > x <- seq(1,10, length=8)
> y <- x^{2}
> stepfun(x,y)
$x:
 [1]  1.000000  2.285714  2.285714  3.571429  3.571429  4.857143  4.857143  6.142857
 [9]  6.142857  7.428571  7.428571  8.714286  8.714286 10.000000 10.000000

$y:
 [1]   1.00000   1.00000   5.22449   5.22449  12.75510  12.75510  23.59184  23.59184
 [9]  37.73469  37.73469  55.18367  55.18367  75.93878  75.93878 100.00000

> plot(stepfun(x,y), type="l")
```

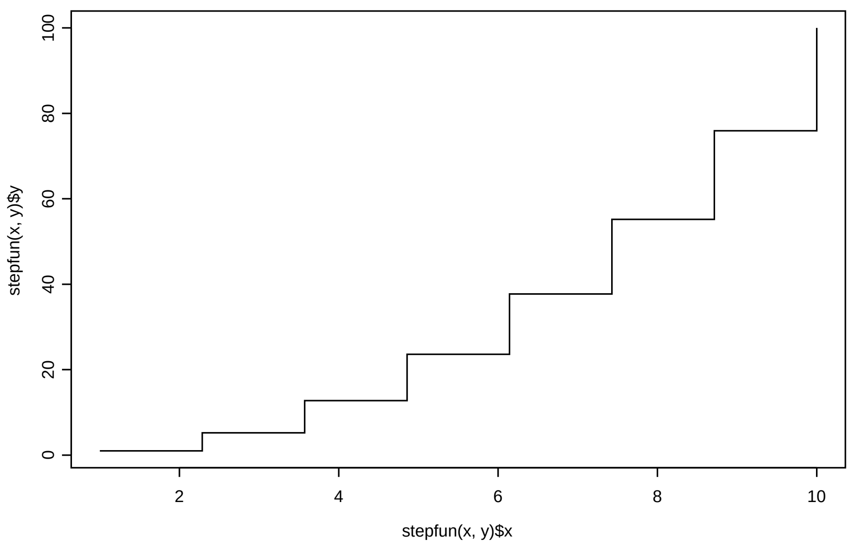

Figure 1: Output of the stepfun function.