Part A Simulation and Statistical Programming – Practical Sheet 1

Arithmetic operations

Q. Create the following variables and check they are correct.

Variable name	Value	Answer
x1	$5 \times 4 + 6 \div 4 - 1.2$	20.3
x2	$3^3 - 1$	26
x3	$e^{3x6-15} - 2$	18.08554
x4	564 mod 17	3
x5	$\cos(\pi/6)$	0.8660254
x6	$\sqrt{81}$	9
x7	$\log_{10}(67)$	1.826075
x8	$\log_e(-1)$	NaN
x9	$x1 + x2 - x3 \div x4$	40.27149

```
x1 = 5 * 4 + 6 / 4 - 1.2
x2 = 3^3 - 1
x3 = exp(3*6-15) -2
x4 = 564 %% 17
x5 = cos(pi/6)
x6 = sqrt(81)
x7 = log10(67)
x8 = log(-1)
x9 = x1 +x2 - x3 / x4
```

Vectors and sequences

To create a vector containing a sequence of integers from a to b we can use `a:b`

Q. Create a vector v0 containing the integers from -10 to 10.

```
v0 = -10:10
```

The R function `seq` can be used to create more complex sequences.

For example,

```
seq(from = 0, to = 3, by = 1)
```

creates the vector (0 1 2 3)

The same can be achieved using

```
seq(0, 3, 1)
```

Here the numbers 0, 3 and 1 are assigned to the arguments `from`, `to` and `by` using *positional matching*.

Q. Use the R function seq to create the following sequences

Name	Sequence
v1	1,2,3,4,5,6,7,8,9,10
v2	3,6,9,12,15,18,21,24,27,30
v3	1,4,7,10
v4	1, 2.5, 4, 5.5, 7

```
v1 = seq(1, 10, 1)
v2 = seq(3, 30, 3)
v3 = seq(1, 10, 3)
v4 = seq(1, 7, 1.5)
```

Q. Apply the functions `min()`, `max()` and `length()` to these vectors.

```
> min(v1);max(v1);length(v1)
[1] 1
[1] 10
[1] 10
```

The function `rep()` can also be used to generate sequences

```
rep(1, 5)
```

creates the vector (1 1 1 1 1)

Q. Use `rep()` and `seq()` as needed, create the vectors

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

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

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

```
> rep(0:4, rep(4,5))
 [1] 0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4
> rep(1:4, 5)
```

```
 [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
> rep(0:4, rep(4,5))+rep(1:4, 5)
 [1] 1 2 3 4 2 3 4 5 3 4 5 6 4 5 6 7 5 6 7 8
```

Extracting elements from vectors : to extract the *k*th element of the vector `v1` we use

```
v1[k]
```

Q. Calculate the sum of the 5^{th} element of v1 and 7^{th} element of v2

```
> v1[5]+v2[7]
[1] 26
```

We can use this same method to exclude certain elements from vector.

Q. Look at what the following commands do

```
v1[-5]
```

```
v1[-c(1:6)]
```

Q. Using only v1, v2, v3, v4 create new vectors containing the following sequences

Name	Sequence
v5	3 12 21 30
v6	4 8 12 16 20 24 28 32 36 40
v7	$\cos(\pi i/3)$ for $i \in \{1,2,\ldots,10\}$
v8	$e^i - 3i$ for $i \in \{1,2,\ldots,10\}$
v9	$3i \pmod 7$ for $i \in \{1,2,\ldots,10\}$

```
v4 = 3 * v3
v6 = 4 * v1
v7 = cos(v1 * pi / 3)
v8 = exp(v1) - 3*v1
v9 = (3*v1) %% 7
```

Q. Create two new vectors, called v10 and v11, that contain the sorted elements of v9 in increasing and decreasing order respectively.

Q. Create vectors, called v12 and v13, that contain the following first names and surnames

(Mary, Sam, Beth, George, Helen, Nick, Tracy, David, Jill, Fred)

(Stern, Trill, Matthews, Cray, Beal, Simpson, Deal, Hunter, Wetherby, Sims)

```
v12= c("Mary", "Sam", "Beth", "George", "Helen", "Nick",
"Tracy", "David","Jill", "Fred")

v13 = c("Stern", "Trill", "Matthews", "Cray", "Beal",
"Simpson", "Deal", "Hunter", "Wetherby", "Sims")
```

Q. Use the `paste()` function to create a vector which contains the full names of the individuals?

```
paste(v12, v13)
```

Logical Operators

Q. Use the logical operators and the `which()` function to determine

1. which elements of v9 are greater than 3
2. which elements of v9 are between than 2 and 4 inclusive
3. which elements of v9 are greater than 4 or divisible by 2

```
> which(v9 > 3)
[1] 2 4 6 9
> which(v9 >= 2 & v9 <= 4)
[1]  1  3  6  8 10
> which(v9 > 4 | (v9 %% 2) == 0)
[1]  2  3  4  6  7  9 10
```

Simulation

Q. We saw in lectures that if u~U[0,1] and X = - (1/r) log(u) then X~Exp(r). Write a one-line simulator for an exponential rv X~Exp(0.5). Simulate 1000 Exp(0.5) r.v. and check the mean is about 2 (you may find `runif()` and `mean()` useful).

```
> u=runif(n=1000); X=-2*log(u);
> X[1:5]
[1] 2.4079761 0.3474278 1.3791364 0.5259934 2.4905842
> mean(X)
[1] 2.043599
```

Q. We saw in lectures thatat if u~U[0,1] and we set X=ceiling(log(u)/log(1-p)) then X~Geom(p). Simulate 1000 Geom(1/2) r.v. and check the mean is about 2. Note that `ceiling()` is an R command.

```
> u=runif(1000); p=1/2
```

```
> X=ceiling(log(u)/log(1-p))
> X[1:5]
[1] 2 1 3 2 3
> mean(X)
[1] 2.004
```

Q. In lectures we saw we could sample a discrete distribution p=(p_1,p_2,…p_m) by simulating u~U[0,1] (`u=runif(1))` and looking for the smallest x in {1,2,…,m} such that

$$u < p\_1 + p\_2 + \ldots + p\_x$$

See if you can use this to simulate X~p where p=(0.1,0.2,0.3,0.4). Do it (a) 'by hand' just using R as a calculator and (b) see if you can automate it using the `cumsum(), which(), min()` commands.

```
> p=c(0.1,0.2,0.3,0.4)
> ps=cumsum(p)
> ps
[1] 0.1 0.3 0.6 1.0
> u=runif(1)
> u
[1] 0.5046117
> #3 is smallest x such that sum(p[1:x])>u
> (u<ps)
[1] FALSE FALSE  TRUE  TRUE
> which(u<ps)
[1] 3 4
> min(which(u<ps))
[1] 3
> #can repreat this to sample p in one line
> min(which(runif(1)<cumsum(p)))
[1] 2
> min(which(runif(1)<cumsum(p)))
[1] 3
> min(which(runif(1)<cumsum(p)))
[1] 1
> # our X~p are 3,2,3,1
```

Matrices

Matrices can be constructed from vectors using the `matrix()` function.

Q. Type in the command

```
m1 = matrix(v1, nrow = 2, ncol = 5, byrow = TRUE)
```

Q. What does m1 look like?

Q. What happens if you increase or decrease the number of rows and columns of the matrix m1?

```
> m1 = matrix(v1, nrow = 2, ncol = 5, byrow = TRUE)
> m1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10

> m1 = matrix(v1, nrow = 3, ncol = 5, byrow = TRUE)
Warning message:
In matrix(v1, nrow = 3, ncol = 5, byrow = TRUE) :
  data length [10] is not a sub-multiple or multiple of the
number of rows [3]
> m1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]    1    2    3    4    5

> m1 = matrix(v1, nrow = 2, ncol = 4, byrow = TRUE)
Warning message:
In matrix(v1, nrow = 2, ncol = 4, byrow = TRUE) :
  data length [10] is not a sub-multiple or multiple of the
number of columns [4]
> m1
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
```

In the above command we set the argument `byrow = TRUE` which specifies that the elements of the vector v1 are to be placed into the matrix starting with the first row, then the second row etc. The default value of this argument is FALSE, thus if we simply omit this part of the command we get a matrix which has been filled by column. Type in the command to try this

```
m2 = matrix(v1, nrow = 2, ncol = 5)
```

Like vectors, matrices can be manipulated as if they were variables. To square all the elements and add 3 enter the command and check this is what has happened

```
m1^2 + 3
```

We can also extract elements in a similar way to vectors. For example, to extract the element of m2 in the 2nd row and 3rd column we can use

```
m1[2, 3]
```

Alternatively, submatrices can be extracted. For example, to extract the 4th and 5th columns of the matrix we would use

```
m1[, 4:5]
```

The transpose of a matrix can be calculated using the `t()` function.

```
m3 = t(m1)
```

Matrices can be multipled together using the `%*%` operator.

Q. What do you get when you type `m1 %*% m2`

Q. What do you get when you type `m1 %*% m3`

```
> m1 %*% m2
Error in m1 %*% m2 : non-conformable arguments
> m1 %*% m3
     [,1] [,2]
[1,]   55  130
[2,]  130  330
```

Matrices can also be constructed by combining vectors or other matrices together using the `cbind()` and `rbind()` functions.

For example

```
m4 = cbind(v1, v2)
```

creates a 10x2 matrix with v1 as the first column and v2 as the second column.

```
m5 = rbind(v1, v2)
```

creates a 2x10 matrix with v1 as the first row and v2 as the second row.

Q. Create a 10x10 matrix (m6) of integers from 1 to 100, filled one row at a time. Create a matrix (m7) with v2 as the first column and v1 as the second column. Multiply m6 by m7. Square all the elements of the matrix and subtract 400. What is the element in the 5th row and 2nd column? (**ans. 6681825**)

```
m6 = matrix(1:100, 10, 10, byrow = T)
m7 = cbind(v7, v1)
m = m6 %*% m7
```

```
m = m^2 -400
m[5,2]
```

Arrays

The function `array()` can be used to construct higher dimensional arrays

```
a1 = array(1:1000, dim = c(5,8,25))
```

creates an array with dimensions 5x8x25 containing the integers from 1 to 1000.

Q. By examining the elements of the array determine how R places the integers into the array.

```
> a1[1:3, 1:3, 1:3]
, , 1

     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13

, , 2

     [,1] [,2] [,3]
[1,]   41   46   51
[2,]   42   47   52
[3,]   43   48   53

, , 3

     [,1] [,2] [,3]
[1,]   81   86   91
[2,]   82   87   92
[3,]   83   88   93
```

Factors

Factors are variables which can only take one of a finite set of discrete values. The `cut()` command is very useful in converting numeric vectors into a factor. For example,

```
f1 = cut(v5, c(0, 10, 20, 30, 40, 50))
```

Creates a factor with 5 levels : (0,10] (10,20] (20,30] (30,40] (40,50]

Lists

Often we would like to store data of several different types and sizes in one object. This can be achieved using a list.

To create a list that include the vectors v1 and v4 and the matrix m1 enter the command and look at the result

```
l1 = list(v1 = v1, v4 = v4, m1 = m1)
```

Components of a list can be accessed by name using the $ symbol. For example, the component v1 can be extracted by entering the command

```
l1$v1
```

To access the 2nd component of the list (without reference to its name) enter the command

```
l1[2]
```

This returns a new list with one component. An alternative would be to use

```
l1[[2]]
```

This returns the vector that is the second component of the list l1.

Data Frames

Most datasets are stored in R as data frames and many R functions take data frames as input. Data frames are like matrices but the columns can be of different types from each other. The R function `data.frame()` is used to create data frames.

For example,

```
d1 = data.frame(firstname = v11, surname = v12, age = f1)
```

Keeping track of objects

Use `ls()` to see all the R objects you have created so far.

Use the function `mode()` to look at the type of each of these objects.