Statistical Programming — Worksheet 1

Bits with an asterisk (*) should be tackled after completing the other exercises.

1. **Sequences.** Generate the following sequences using `rep()`, `seq()` and arithmetic:

   (a) $1, 3, 5, 7, \dots, 21$.

   (b) $1, 10, 100, \dots, 10^9$.

   (c) $0, 1, 2, 3, 0, \dots, 3, 0, 1, 2, 3$ [with each entry appearing 6 times]

   (d) $0, 0, 0, 1, 1, 1, 2, \dots, 4, 4, 4$.

   (e)* $50, 47, 44, \dots, 14, 11$.

   (f)* $1, 2, 5, 10, 20, 50, 100, \dots, 5 \times 10^4$.

   Can any of your answers be simplified using recycling?

2. **Arithmetic.** Create a vector containing the following sequences:

   (a) $\cos\left(\frac{\pi n}{3}\right)$, for $n = 0, \dots, 10$.

   (b) $1, 9, 98, 997, \dots, 999994$.

   (c) $e^n - 3n$, for $n = 0, \dots, 10$.

   (d)* $3n \mod 7$, for $n = 0, \dots, 10$.

   Let
   $$S_n = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{2i-1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + \frac{(-1)^{n+1}}{2n-1}.$$
   You will recall that $\lim_n S_n = \pi/4$.

   (e) Evaluate $4S_{10}$, $4S_{100}$ and $4S_{1000}$. *[Hint: use the `sum()` function.]*

   (f) Create a vector with entries $S_i - \frac{\pi}{4}$, for $i = 1, \dots, 1000$. *[Hint: try creating the vector with entries S_i first; the function `cumsum()` may be useful.]*

3. **Subsetting**

   Create a vector `x` of normal random variables as follows:

```
> set.seed(123)
> x <- rnorm(100)
```

   The `set.seed()` fixes the random number generator so that we all obtain the same `x`; changing the argument `123` to something else will give different results. This is useful for replication.

   Give commands to select a vector containing:

   (a) the 25th, 50th and 75th elements;

   (b) the first 25 elements;

   (c) all elements except those from the 31st to the 40th.

   Recall the logical operators `|`, `&` and `!`. Give commands to select:

(d) all values larger than 1.5 (how many are there?);

(e) what about the entries that are either > 1.5 or < -1?

4. **Monte Carlo Integration.** Now let's try some simple examples related to what you've studied in lectures. Suppose we have $Z \sim N(0, 1)$ and want to estimate $\theta = \mathbb{E}\phi(Z)$: we can generate a large number of independent normals, $Z_1, \dots, Z_n$ and then look at the sample mean:
$$\frac{1}{n}\sum_{i=1}^{n} \phi(Z_i).$$
Let's try this for $\phi(x) = x^4$; generate $n = 10\,000$ standard normal random variables in a vector called `Z`.

   Now, find the sample mean of Z^4, and have a look at the values you get. Try the `summary()` and `hist()` functions to help you understand the data:

```
> mean(Z^4)
> summary(Z^4)
> hist(Z^4, breaks = 250)  # notice how skewed this is!
```

   Using the central limit theorem we also know that a $(1-\alpha)$-confidence interval is given by
$$\hat{\theta}_n \pm c_\alpha \frac{S_{\phi(Z)}}{\sqrt{n}}.$$
where
$$S^2_{\phi(Z)} \equiv \frac{1}{n-1}\sum_{i=1}^{n}(\phi(Z_i) - \hat{\theta}_n)^2$$
Calculate $S^2_{\phi(Z)}$ using the mean and sum functions. Check that using `var(Z^4)` gives the same answer.

   You can get the quantiles of a normal distribution using `qnorm()`. For example:

```
> qnorm(0.975)

## [1] 1.959964
```

   Use this function with your work above to obtain a 99% confidence interval for the value of $\mathbb{E}\phi(Z)$.

5. **Records.*** Let $X_1, X_2, \dots$ be independent and identically distributed continuous random variables. Call i a **record** if $X_i > X_j$ for all $j < i$ (trivially including $i = 1$). Let R_t be the index of the tth such record.

   Suppose we have a vector `x` and want to find the indices that correspond to records. Using the `cummax()` function with `which()` and `==`, work out commands to give you a vector of the incides of records.

   Thinking about the inversion method of random variables, can you explain why the distribution of R_t does not depend upon the distribution of the X_is?

Statistical Programming Worksheet 2

By the end of the practical you should feel confident writing and calling functions, and using `if()`, `for()` and `while()` constructions.

You should complete and understand questions 1–6 for next time.

1. Review

(a) Let $t = 2$: create a vector with $(i + 1)$th entry $\frac{e^{-t}t^i}{i!}$ for $i = 0, \dots, 10$ (you might want to use the function `factorial()` for this).

(b) Write a function with arguments `t` and `n` that evaulates $\sum_{i=0}^{n} \frac{e^{-t}t^i}{i!}$.

(c) Write your function again using a `for()` loop. Do not use vectors, or the `sum()` function. Check it gives the same answers as (b).

2. Solving a Quadratic. Write a function with three arguments `a`, `b` and `c`, that returns the **real** roots of the equation $ax^2 + bx + c = 0$, if any. Your function should behave well if $a = 0$ and return an empty vector when there are no real roots.

3. Sieve of Eratosthenes. The Sieve of Eratosthenes is a method for finding all the prime numbers less than some specified n. Here is an outline of the algorithm:

- Create a vector `x` of integers from 2 to n, and an empty vector `p`.
- Given `x`, append the first element (say `z`) to `p`; then remove any multiples of `z` (including `z` itself) from `x`.
- Stop when `x` is empty, and return `p`.

Write a function to implement this of Eratosthenes. It should take one argument `n`, and return all the primes up to `n`.

Try it for $n = 10^3, 10^4, 10^5$. [Optional: can you see any way to speed this procedure up?]

4. Random Walks. Write a function `rndwlk`, with an argument `k`, that simulates a symmetric random walk (see lecture), stopping when the walk reaches k (or $-k$). After stopping it should return the entire walk.

Try calling `plot(rndwlk(10))` a few times to see how it looks.

5. Simulating Discrete Distributions. In lectures you've seen that we can sample X from a discrete distribution on $\{1, \dots, k\}$ as follows: let $p_i = P(X = i)$. Then:

- generate $U \sim \text{Unif}[0, 1]$;
- set $X = \min\{i \mid \sum_{j=1}^{i} p_j \geq U\}$.

Write a function that, given `p` containing $(p_1, \dots, p_k)$ can simulate X from this distribution. You may find the functions `cumsum()` and `which()` useful.

Modify your function so that it takes an argument `n`, and produces a vector of `n` i.i.d. values from the distribution p. Comment on how you could check that your function worked as expected.

6. **Rejection Sampling**. We will write an R function to simulate $X \sim N(0,1)$ using rejection sampling with the double exponential proposal. That is, from a random variable Y with density

$$f_Y(y) = \exp(-|y|), \qquad y \in \mathbb{R}.$$

   (i) Write a function to simulate n i.i.d. values of Y. *[Hint: you might want to start thinking about how to simulate an exponential random variable.]*

   (ii) Write a function implementing rejection for X. The algorithm is:

   1. simulate $Y \sim \exp(-|x|)$ and $U \sim U(0,1)$;
   2. if $U < \exp(-Y^2/2 + |Y| - 1/2)$ accept $X = Y$ and stop. Otherwise repeat 1.

   [Hint: you can do this using a while statement. You should call the function you wrote in (a) to simulate Y. Your function should have no inputs, and return the simulated value of X.]

   (iii) Test your rejection sampler by simulating 1000 samples and checking they are normal using the `qqnorm()` function.

7. **Double for() Loop**. Using two `for()` loops, write a function with an argument `n`, which constructs the $n \times n$ matrix with entries $a_{ij} = i - j$.

8. **Moving Averages**

   (a) Write a function to calculate the moving averages of length 3 of a vector $(x_1, \ldots, x_n)^T$. That is, it should return a vector $(z_1, \ldots, z_{n-2})^T$, where

$$z_i = \frac{1}{3}\left(x_i + x_{i+1} + x_{i+2}\right), \qquad i = 1, \ldots, n-2.$$

   Call this function `ma3()`.

   (b) Write a function which takes two arguments, `x` and `k`, and calculates the moving average of `x` of length `k`. [Use a `for()` loop.]

   (c) How does your function behave if k is larger than (or equal to) the length of x? You can tell it to return an error in this case by using the `stop()` function. Do so.

   (d) How does your function behave if $k = 1$? What should it do? Fix it if necessary.

Statistical Programming — Worksheet 3

1. Cystic Fibrosis dataset

On the class website you will find the file `cystfibr.txt` which contains a set of measurements on a set of individuals with cystic fibrosis. Save the file locally onto your computer.

(a) Take a look at this file using a text editor like Wordpad or Notepad. Read the data into R using `read.table()`; call the resulting dataframe `cf`.
Check that the data have been formatted correctly by typing:

```
> head(cf)
```

(b) Calculate the following

(i) the number of individuals in the dataset;
(ii) the number of variables measured on each individual;
(iii) the names of the variables measured on each individual;
(iv) the mean, median, standard deviation and range of each of the variables (use `apply()`);

(c) Create the following data frames and for each one calculate the mean of each of the variables

(i) a new data frame containing just individuals older than 15
(ii) a new data frame containing just individuals with bmp in the interval [70,90]
(iii) a new data frame containing just individuals with either FEV1 greater than 30 or RV greater than 300.

2. Data Frames

Load the `MASS` library and take a look at the dataset called `survey`.

```
> library(MASS)
> head(survey)
```

You can look at the documentation:

```
> ?survey
```

and get a brief summary of each variable:

```
> summary(survey)
```

(a) Find the mean pulse rate of the students. What goes wrong here?

The vector of pulses stored in `survey` contains some entries which are labelled `NA`. This is used in `R` to represent **missing data**.

(b) Try looking at the documentation for `mean()` to see how to get around this.
(c) The ages are recorded as fractions representing a number of months. Change that columns of the data frame so that it contains whole years (the `floor()` command may be useful).

(d) Find the mean pulse rate for students under 20.

Subsetting. Suppose I want to obtain the records of students who are over 190 cm tall. Since data frames allow me to subset just like a matrix, I might just think of typing:

```
> survey[survey$Height > 190, ]
```

What goes wrong here? Try instead the following:

```
> subset(survey, Height > 190)
```

The subset function ignores missing values, which is usually the behaviour we would prefer. We can also select only some of the fields of the data frame if we prefer:

```
> subset(survey, Height > 190, select = c("Pulse", "Clap"))
```

Recall that the `&` operator does a point-wise logical 'and' comparison.

```
> subset(survey, (Pulse > 70) & (Smoke == "Heavy"))
```

Similarly, `|` is for 'or', and `!` for 'not'.

```
> with(survey, (Pulse > 70) | (Pulse < 45))
> !(survey$Age > 30)
```

(e) Find the mean age of students who write with their right hand.

(f) What proportion of left handers do not clap with their left hand on top?

(g) Using the `plot()` command, plot the pulse of the subjects against their age.
Try subtracting 10 from the age and taking the logarithm (using the function `log()`), to obtain a slightly clearer picture.

3. Factors

Take a look at the `birthwt` data from the `MASS` package.

(a) How is race stored in these data? Is this sensible?

(b) Turn this into a factor with level names as indicated in the documentation. To do this, look at the documentation for the function `factor()`.

(c) Use the `table()` command to count the number of babies of each race.

(d) Try the following command:

```
> tab = with(birthwt, table(smoke, low))
```

What do the results in `tab` suggest?

4. *Gaussian Random Walk. Write a function which simulates a Gaussian random walk with `n` steps. (In other words, $X_0 = 0$ and $X_i - X_{i-1} \sim N(0,1)$ independently.)

Generate and plot the walk for `n = 1000` (use `type="l"`).

Try to vectorise your code (if it is already vectorised, try constructing a naïve version with a `for()` loop). Compare the speed of the vectorised and unvectorised versions by generating a random walk of length 50,000

Statistical Programming Worksheet 4

1. Cholesky Decomposition.

(a) Write a function with argument `n` to generate a random symmetric $n \times n$-positive definite matrix. To do this:

- generate an $n \times n$ matrix C whose entries are independent normal random variables;
- return CC^T.

Check your matrices are positive definite using the `eigen()` function.

(b) Implement the recursive Cholesky decomposition algorithm from the lecture.

(c) Test it using your function for generating positive definite matrices, and by comparing the answers to `chol()`.

(d) Create a function which takes a vector `mu` and a symmetric positive definite matrix `Sigma` and uses them to generate a multivariate normal vector $N_n(\mu, \Sigma)$. Your function should check that `Sigma` is positive definite using `eigen()` and symmetric using `isSymmetric()`.

2. Sorting. Here is an algorithm called 'Quicksort' for sorting the objects in a vector.

Function:	sort a vector x
Input:	vector x of length n
Output:	a vector $Q(x)$ containing entries of x arranged in ascending order

```
1. if $n \leq 1$ return $x$;
2. pick an arbitrary 'pivot' element $i \leq n$;
3. let $z = (x_j \mid x_j < x_i)$ and $y = (x_j \mid x_j > x_i)$;
4. let $z' = Q(z)$ and $y' = Q(y)$; [i.e. call the algorithm on the smaller vectors]
5. let $x'$ be the entries in $x$ not used in $y$ or $z$; [i.e. any entries equal to $x_i$]
6. return $(z', x', y')$.
```

(a) Implement the algorithm in R, and test it on some random numbers.

(b) What is the complexity if x_i is always the smallest element?

(c) Show that, if the pivot x_i is the median element on each call, that the complexity is at most $O(n \log_2(n))$.

3. Back Solving. Here is a recursive algorithm to solve $Ax = b$ where A is an upper triangular matrix, using back substitution.

```
Function:  solve $Ax = b$ for $x$ by back-substitution
Input:     $n \times n$ upper triangular matrix $A$ and vector $b$ of length $n$
Output:    vector $x$ of length $n$ solving $Ax = b$

1. If $n = 1$ return $x = b/A$;
2. create a vector $x$ of length $n$;
3. set $x_n = b_n/A_{nn}$;
4. set $b' = b_{1:(n-1)} - A_{[1:(n-1),n]}x_n$;
5. set $A' = A_{[1:(n-1),1:(n-1)]}$;
6. solve $A'x' = b'$ for $x'$ by back-substitution ;
7. set $x_{[1:(n-1)]} = x'$;
8. return $x$.
```

(a) Implement this algorithm as a recursive function in R. Your function should take as input an upper triangular $n \times n$ matrix A and return a solution x satisfying $Ax = b$.

(b) For $n = 10$, create an $n \times n$ upper triangular matrix A and a vector b of length n. Check the solution from your function against `backsolve()` and `solve()`.

Statistical Programming Worksheet 5

1. **Polynomials.** Find the coefficients of the quintic polynomial $f(x)$ for which

$$(f(2), f(3), f(4), f(5), f(6), f(7))^T = (3, 2, 1, 6, 95, 538)^T$$

[Hint: set the problem up as a system of linear equations and use solve()].

2. **Linear Models.** The data in `speed.txt` (available on the course website) give the times in seconds recorded by the winners in the finals of various men's running events (200, 400, 800 and 1500 metres) at each of the 21 Olympic Games from 1900 to 2004, along with the heights above sea level of the different venues.

   Read in the data and take a look at it. Calculate the average speed in metres per second of the winner of each race, and add this to the dataframe.

   Fit the following models to the dataset

   (i) speed against $\log_2$(distance).

   (ii) speed against $\log_2$(distance), year and $\log_2$(altitude).

   Fit model (i) twice using (a) the `lm()` command, and (b) using `qr.solve()`.

3. **Gram-Schmidt**. One method for obtaining the QR decomposition of an $n \times p$ matrix A is to use the Gram-Schmidt process. The algorithm for this is below.

```
Input  : $n \times p$ matrix $A$.
Output: Orthogonal $n \times p$ matrix $Q$ and upper triangular
        $p \times p$ matrix $R$ such that $A = QR$.
for $k \in \{1, \ldots, p\}$ do
    Set $r_{kk} = \sqrt{\sum_j a_{jk}^2}$;
    Set $q_{[1:n,k]} = A_{[1:n,k]}/r_{kk}$;
    for $i \in \{k+1, \ldots, p\}$ do
        $r_{ki} = \sum_{j=1}^{n} a_{ji} q_{jk}$;
        $a_{[1:n,i]} = a_{[1:n,i]} - r_{ki} q_{[1:n,k]}$;
    end
end
```

   (a) Implement the algorithm as a function in R.

   (b) Combine your answer to (a) and to Question 3 from Sheet 4 (recall it implemented a back-solving method) to construct a function for solving linear models from scratch. Apply the function to the data in question 2.

   (c) What is the complexity of this algorithm?

Statistical Programming Worksheet 6

1. **Numbers.** Prof. Goldschmidt has 200 square tiles with which to decorate a wall of the kitchen in the new Department of Statistics building. 20 of the tiles are red, 30 blue, and the rest are white. Write down a formula for the number of patterns she can create.

   How many digits does this number have?

   How many digits does 1000! have?

2. **Metropolis Hastings.** Suppose that $X_1, \dots, X_n \overset{\text{i.i.d.}}{\sim} \text{Gamma}(\alpha, \beta)$, and let α and β have independent Exponential(1) priors.

   (a) Write a function to evaluate the log-posterior of α and β given data $\boldsymbol{x}$. The function should have arguments `x`, `alpha` and `beta`.

   (b) Write a function to perform a single Metropolis-Hastings step to explore the posterior above. Use a proposal

   $$\alpha' = \alpha + \sigma Z_1 \qquad \beta' = \beta + \sigma Z_2$$

   for Z_1, Z_2 independent standard normals (i.e. $q(\alpha' \mid \alpha) \sim N(\alpha, \sigma^2)$.) It should take as arguments `x`, `alpha`, `beta` and `sigma`.

   (c) Write a function to run the Metropolis-Hastings algorithm for N steps and return an $N \times 2$ matrix of the parameter values. It should take as input the data `x`, number of steps `N`, starting values `alpha` and `beta`, and proposal standard deviation `sigma`.

   (d) The file `airpol.txt` contains daily PM2.5 readings taken from various measuring stations around Seattle during 2015. Read in the data as a vector and plot it in a histogram.

   ```
   x <- scan("airpol.txt")  # note use of scan(), not read.table()
   hist(x, breaks = 100, freq = FALSE)
   ```

   Model the data as i.i.d. Gamma distributed observations using the model above. Run your Metropolis-Hastings algorithm for 5,000 steps with starting point $\alpha = 1$, $\beta = 1$. Plot your output with `plot()` and investigate different values of $\sigma \in \{0.01, 0.02, 0.05\}$.

   (e) Find the posterior means for α and β. Plot the density of the corresponding Gamma distribution over the histogram of the data.

3. Image Reconstruction.[*] Let the $n \times n$ matrix $Y = (y_{ij})$ of ± 1s follow the distribution of the Ising model with parameter θ, so that

$$\pi(Y) \propto \exp\left\{\theta \sum_{(i,j)\sim(i'j')} y_{ij}y_{i'j'}\right\}$$

where $(i,j) \sim (i'j')$ if either $i = i' \pm 1$ and $j = j'$, or vice versa (i.e. they differ by exactly one column or one row, but not both).

(a) Let $\widetilde{Y} = Y$ except that $\widetilde{y}_{ij} = 1 - y_{ij}$ (so they are equal except for a single entry). Show that

$$\log \pi(\widetilde{Y}) - \log \pi(Y) = \theta(d_{i,j} - 2a_{i,j})$$

where $d_{i,j}$ is the number of pixels adjacent to i,j, and $a_{i,j}$ is the number of adjacent pixels which have the same value as y_{ij}.

We will construct a Metropolis-Hasting algorithm to target π.

(b) First, look at the function `mh_step()` in the file `MHcode.R` on the website. The function performs one M-H step by proposing to flip `Y[r,c]`.
Complete the function by replacing the questions marks with code to calculate $\log \alpha$. Comment the code to show you understand what the rest of the function is doing.

(c) Now create a function with arguments `n`, `N` and `theta` which creates an $n \times n$ matrix with random entries 0 or 1, and then performs N M-H steps by calling `mh_step()`. When finished, it should return the state of the chain.

(d) Run the function for $n = 50$ and values $\theta = 0.2, 0.5, 0.8$ (you'll probably need $N > 10^5$ to get reasonable convergence). You can plot your solution using the `image()` function:

```
> out <- mh_ising(50, theta=0.5, N=1e5)
> image(out)
```

(e) Consider an $n \times n$ matrix $X = (x_{ij})$ of independent Bernoulli random variables, where

$$P(x_{ij} = 1) = \begin{cases} 1-p & \text{if } y_{ij} = 0 \\ p & \text{if } y_{ij} = 1 \end{cases}$$

for an unknown matrix of numbers $Y = (y_{ij})$. Defining $\widetilde{Y}$ as in (a), show that

$$\log L(\widetilde{Y}; X) - \log L(Y; X) = \begin{cases} +\log \frac{p}{1-p} & \text{if } y_{ij} \neq x_{ij} \\ -\log \frac{p}{1-p} & \text{if } y_{ij} = x_{ij} \end{cases},$$

where $L(Y; X)$ is the likelihood for the unknown parameter Y given X.

(f) Read in the data and look at it:

```
X <- as.matrix(read.table("image_noisy.txt"))
image(X)
```

Modify your previous M-H functions to accept a matrix X of data as an argument, and to include the change in the likelihood in your acceptance ratio α. Have the function return the estimated posterior mean of the chain (i.e. the average position of each pixel over the iterations).
Run the chain for a million iterations, setting $p = \frac{2}{3}$ and $\theta = 0.8$, and plot the results.