**COURSEWORK**

**ST2195 – Programming for Data Science**

EMFSS Bsc Data Science and Business Analytics

Univerisity of London

**Joana Dias Maia**

August 2025

Índice

[PART 1 3](#_Toc204002251)

[Exercise a 3](#_Toc204002252)

[Exercise B 4](#_Toc204002253)

# PART 1

## Exercise A

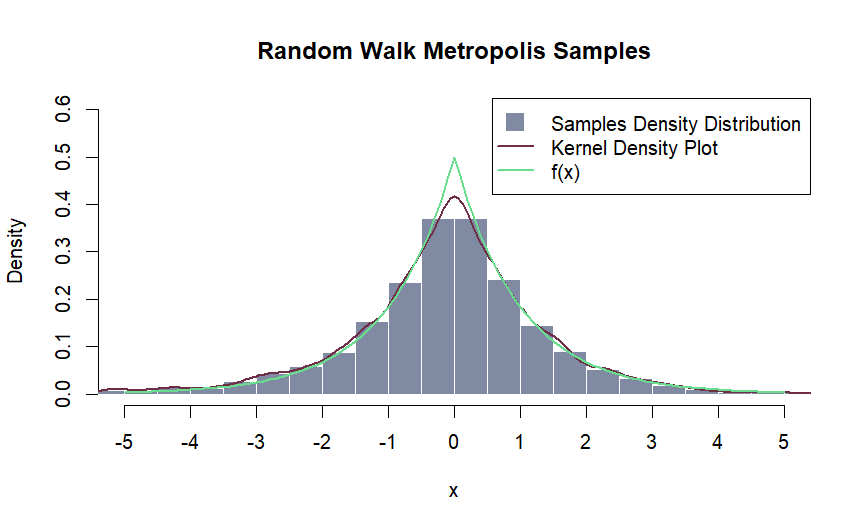
In this exercise we are implementing the Random Walk Metropolis algorithm to simulate samples from the given distribution (f(x), stated as pdf in the code).

The number of iterations (N=10000) and standard deviation (s=1) were given in the instructions. For the initial value of x (stated as x0 in the code) I chose 0, so that this chain begins with the mode of the distribution, and, hopefully, improve the symmetry of the generated samples.

A function, called metropolis, with parameters x0, N and s was created to run this algorithm in scenarios where we can change these three parameters and still work. Inisde this function:

* To implement the algorithm a vector (R)/array(Python) x with dimension N was created. This vector is a “list” with N ordered positions not filled yet, in which we will add values later. The first value was instructed to be x0;
* For the remaining values, from the 2nd position until the Nth position of the vector/array (from 2 to N in R, and 1 to N-1 in Python), a loop was constructed (going one “spot after the other” in order: position 2, position 3 …, position N). For example, the current position in the loop is i, where a random number (random\_number in the code) was drawn from a normal distribution with standard deviation 1 (s) and mean was the value from the previous position in the vector/array (that is i-1);
* From this random number, a formula was applied to calculate r, using the pdf with this number and the previous value in the vector/array (position i-1);
* With an “if/else” statement, we checked if u (a random number from a uniform distribution) was smaller than r. In this case the random number was added to the vector in the i position, and if that was not the case the previous value in the vector/array (i-1) was duplicated into the i position. In the end of the function, the x vector/array was returned.

This metropolis function was used to get the samples, given the N, s and x0 stated in the beginning and then the kernel density, that is the estimated probability density function from the sampled values.



The standard mean from the generated samples was -0,0943715 and standard deviation was 1,481383, in R and in Python the standard mean was 0.13218160322129116 and standard deviation 1.4309633105043644.

## Exercise B

For exercise B, to assess the convergence of the Marvok chains generated by the algorithm we applied the R\_hat to compare between-chain (B) and within-chain (W) variances across multiple cains, considering a threshold of <1.05.

The number of iterations (N=2000) and standard deviation (s=0.001) and the number of chains (J=4) were given in the instructions. For the initial value of x (stated as x0 in the code) four different values were considered, 0,1,2 and 3 to represent the initial value of each chain.

In the code we firstly defined a function (calculate\_r\_hat) with parameter chains – a list of vectors (R)/arrays(Python) of the same length:

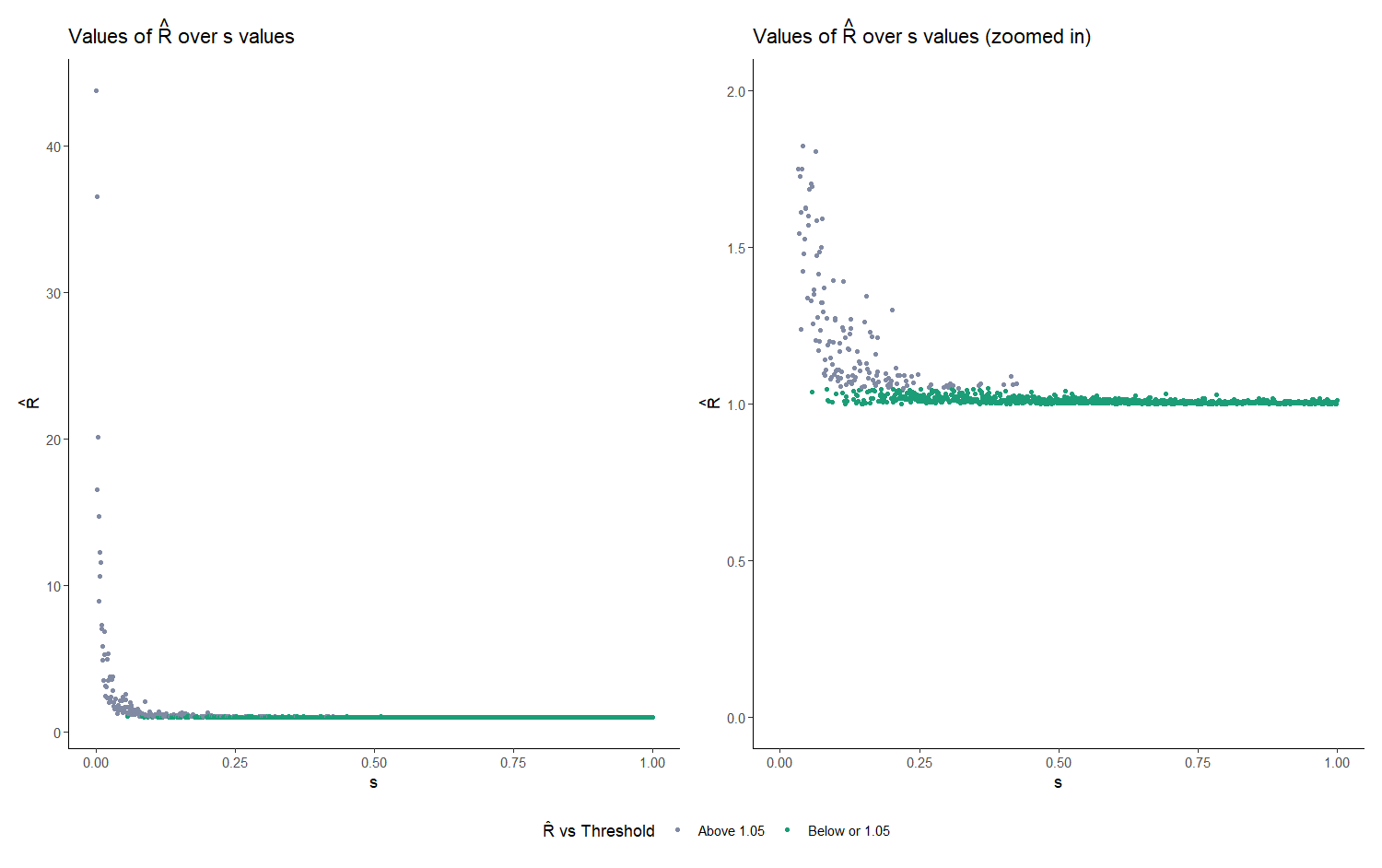
* In order to make it usable with different number and lengths chains, the value of J was obtain with the length of chains (which returns the number of vectors) and N was obtain with the length of one of the vectors/arrays (which returns the number of iterations);
* Two other vectors/arrays were created: means and variances with the mean and variance, respectively, of each vector/array (so 4 means and 4 variances);
* With the values calculated previously, we then obtained the values of W (mean of the variances), M (the mean of the means), and B (the mean of (means – M) squared));
* Given the formula of R\_hat (square root of (B+W)/W) this function returned the obtained value of R\_hat.

After defining this function the exercise was ready to be completed. The function created in exercise A, metropolis, was used to create the chains: we ran the function four times with the four defined values of x0, in order to create four different vectors/arrays that were placed in the list called chains. With the chains ready, the value of R\_hat, considering s=0.001, was obtained by calling the calculate\_r\_hat function. The r\_hat obtained in R was 71.1329 and in Python was 56.8187.

After this, we considered a sequence of s values (s\_values) from 0.001 to 1, increasing each time by 0.001. A loop was ran, where for each value of s in the sequence, the exercise was repeated (with metropolis function obtain the chains, and with calculate\_r\_hat obtain the r\_hat, add r\_hat to a new vector/array called R\_hats, for each value of s in the sequence).

A dataframe was created with columns s (with the s\_values) and R\_hat (with the R\_hats). Additionaly an extra column was then created, where each row had possible values of “Above 1.05” (if R\_hat > 1.05) or “Below or 1.05” (when that criteria was not met).

With this dataframe it was possible to create a scatter plot of the values of R\_hat over the values of s, in which a dot was purple if above 1.05 or green if below or equal to 1.05:



# PART 2