**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](#_Toc204095040)

[Exercise A 3](#_Toc204095041)

[Exercise B 4](#_Toc204095042)

[PART 2 5](#_Toc204095043)

[Exercise A 5](#_Toc204095044)

[Exercise B 6](#_Toc204095045)

# 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.

Uma imagem com texto, diagrama, Gráfico, captura de ecrã

Os conteúdos gerados por IA podem estar incorretos.

Figure 1 - Random Walk Metropolis Histogram and Kernel Density Plot, and f(x)

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:

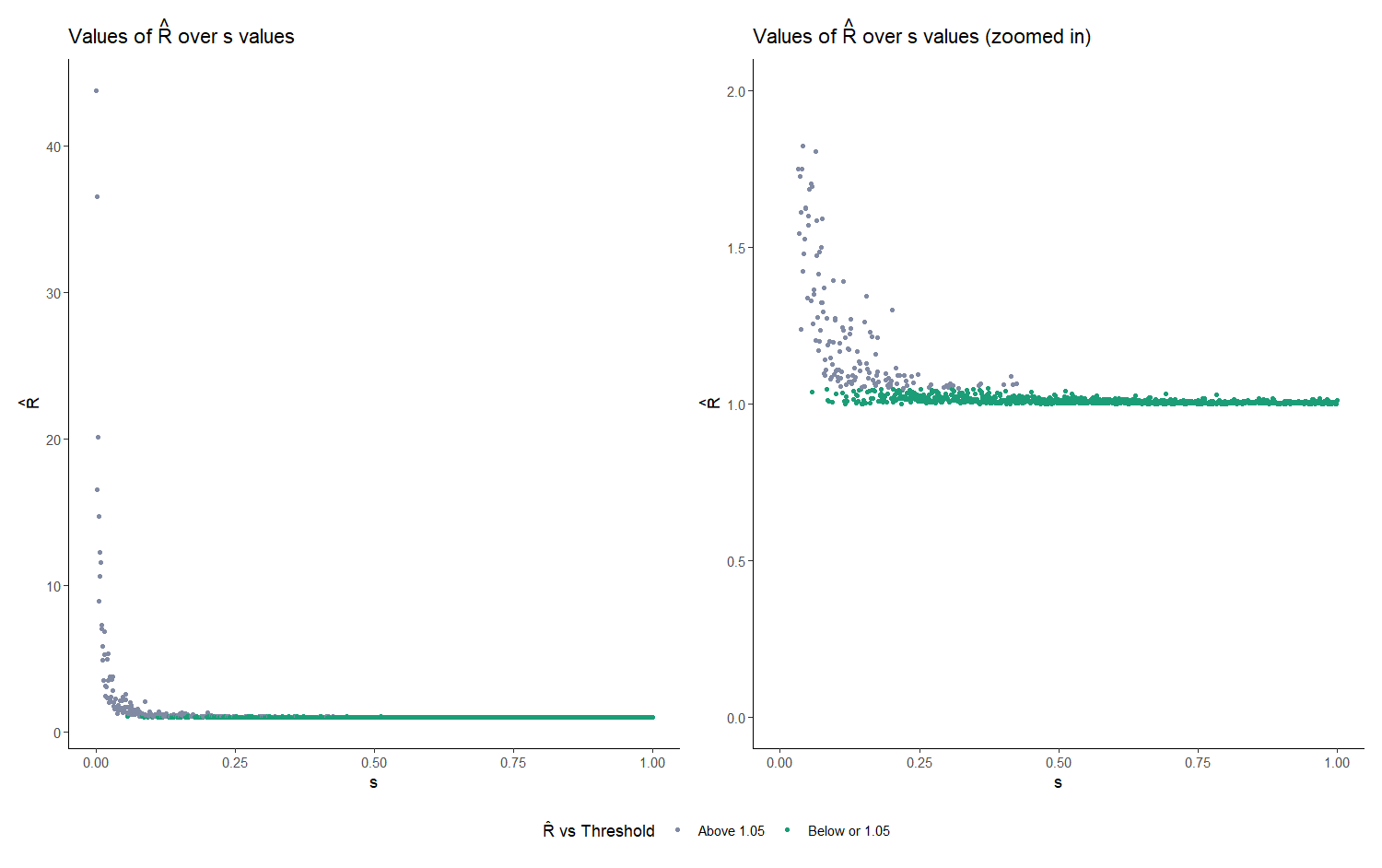


Figure 2 - Scatter Plot of R\_hat over s values, across all values of R\_hat (left) and zoomed in to values of R\_hat between 0 and 2 (right)

# PART 2

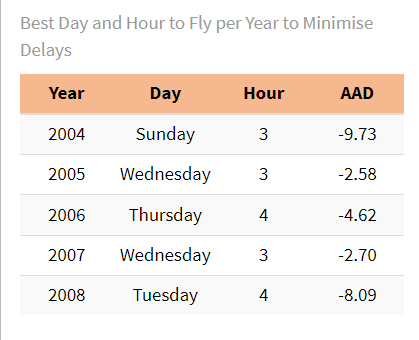
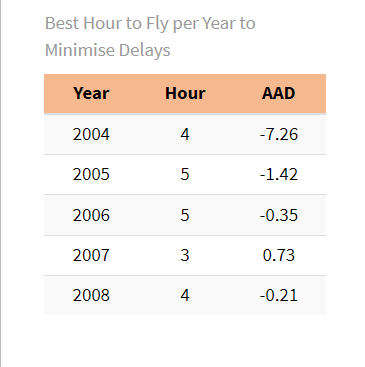
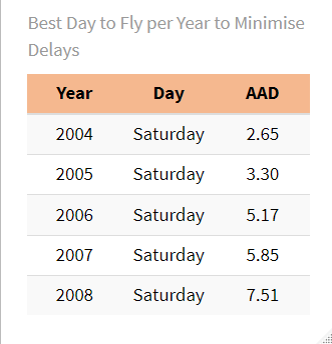
To handle the large airline dataset, I created a local SQLite database (airlines) using the RSQLite and dplyr packages in R. First, I ensured any previous version of the database was removed to avoid duplicate records. In Python, chunksize was used due to the heavy size of the tables. Additionally in R, a reference to those was use for dyplr. Before importing the dataframes into the database, all columns headers were changed to lowercase to facilitate working the data. The selected five consecutive years for the exercise were 2004 to 2008. All the ontime dataframes were loaded independently and then merged before writing the tables in the database. Additional tables for airports, carriers, and planes were also imported.

## Exercise A

Instead of considering the actual departure time (deptime), it was considered the scheduled departure time (crsdeptime), because the variable deptime had 608611 missings, and crsdeptime had none. For this question, it was not took into account if there were diverted or cancelled flights, as all of this had missing arrival delay so were excluded automatically. It was considered that the hours and days of the week with least mean arrival delay were the best moments to minimise delays. In order to make the output table more readable the days of the week, which were in a numeric format, an additional variable was created (weekday\_label, such that 1 = Monday, 2 = Tuesday, and so on. Additionally, the times come in a format of hour and minutes (e.g. 630 is 06:30), so to better understand the hours, the format was changed to HHMM and then extracted only the hour, discarding the minutes in a new variable (hour).

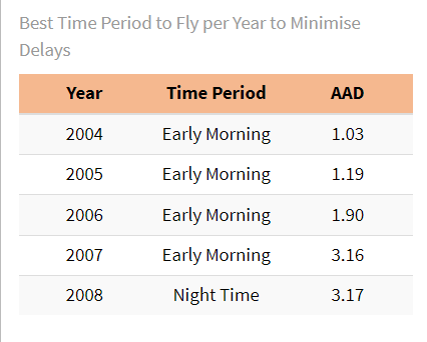
From Table 1, we can see that overall, across the five years examined, Saturdays had the least average arrival delay (AAD). Considering the best hour per year, it was 4am in 2004 and 2008, 5am in 2005 and 2006, and 3 am in 2007. If we took both the day of the week and hour of the day to select the best time to minimise delays was Sundays at 3 am in 2004, Wednesdays at 3 am in 2005, Thursdays at 4 am in 2006, Wednesday at 3 am in 2007 and Tuesday at 4 am in 2008. It is interesting to note that considering both day and hour, we keep the same hour or close to it to when considering only the hour, however Saturday never comes up.

Table 1 - Best time to Fly to Minimise Delays per Year, considering Day of the Week, Hour of the Day, or both



Considering that there are 24 hours in a day and that all of the years gave us times between 3 and 5am, it was then considered the option of grouping them into categories: Night Time (23:00 – 5:59), Early Morning (6:00 – 9:59), Late Morning (10:00 – 11:59), Early Afternoon (12:00 – 13:59), Afternoon (14:00 – 16:59), Evening (17:00 – 19:59) and Late Evening (20:00 – 22:59). From this time period groups, according to Table 2, in we can see that in all the years analysed the best time to minimise delays was in the Early Morning, except for 2008, when it was best at Night Time. Taking into account both the day of the week and the time period we see that across the years analysed the best time to minimise delays was Sundays in the Early Mornings, except for 2008 when it was best to travel on Wednesdays at Night Time.

Table 2 - Best time to Fly to Minimise Delays per Year, considering Day of the Week, Time Period of the Day, or both

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Os conteúdos gerados por IA podem estar incorretos.

## Exercise B

Firstly, to answer the question it was needed to know the age of the airplane. So, firstly we had to inner join the planes df onto ontime by the talinum (here it was great to lowercase everything prior, so now the column name to join is the same). It is important to note that, because ontime already has a variable year (2004-2008), this changed to year.x and the year of the make of the airplane changed to year.y after merging into ontime (year.x was then changed back to year, and year.y kept as is).

Before proceeding further, I noted that year.y was imported as character and not numeric, so, before converting the format, I checked, beside 4 characters (represent the year) what other values were there that imposed this format. There were two options: “” and “None”. Converting them to integers would automatically force NA on those. 8.39% of year.y was missing (n= 2121237), which is only a small part of the data given this is such a large dataset. However, there was a possibility that this missingness was not on random (maybe older planes were missing, because the database was created later?). All of the missings appeared when model of the airplane was also missing or on the model 1121, which corresponded to only one talinum: N382AA. Given such detailed information (model 1121 and talinum N382AA), a quick websearch was enough to get more information about the aircraft: The Rockwell 1211 Jet Commander, manufactured by Aero Commander, was produced and sold to costumers between 1965 and 1968. In order to reduce missings the year 1966 was introduced in all of the missings year.y with model 1121. The percentage of missing was still 8.39% (n=2121237). Lastly we checked for differences in arrival delay between the present and missing year.y: In 0.80% of missing year the arrival delay was missing, the mean average delay was 10.3 minutes, the median -1 minutes and standard deviation 40.9 minutes. In 0.91% of present year the arrival delay was missing, the mean average delay was 8.74 minutes (slightly lower), the median was -1 minutes (same as missing year) and standard deviation 36.5 minutes (slightly lower). Even though a t test showed that the difference in average arrival delay was significative (p<2.2e-16), which suggests that year.y is not MCAR, for now those rows were still dropped to ensure that airplane age estimates are based on reliable data and not introduce bias from uncertain airplanes info.

To calculate the airplane age I subtracted the year.y (year of make of the airplane) to year (year of the flight). We were left with 0.906% of missing arrival delays in the database (n=209778). There was a possibility that this missings influence the results particularly if the percentage was higher: for example created a kind of “survivorship bias”: I might be excluding a particular “profile” of the age of this airplanes (for example older planes would be cancelled or diverted more often, so would not have arrival delay - this group would be underrepresented in the analysis and that the only few older airplanes left might actually be the only ones great and actually have fewer delays than newer models). However, given that it was such a small percentage (<1%), it was decided to just ignore this rows and drop them.

After deleting all of the missings, it was time to understand if there were any “strange” values, for example outliers.

Table 3 Summary Statistics for Airplane Age, Year of Manufacture, and Arrival Delay

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Os conteúdos gerados por IA podem estar incorretos.From the summary statistics in Table 3, it was clear that some values didn’t make sense. We had negative airplane ages (which needed to be excluded) and year of manufacture 0 (which also needed to be excluded). From the arrival delays, not much could be said about veracity. We can see that those extremities are clearly outliers, given the quartiles, however a delay of 2598 minutes, is almost 48h – which seems reasonable in case of a strike.

After excluding this values, extreme values were checked again visually, with the help of histograms and boxplots: