ST2195 COURSEWORK PART 1

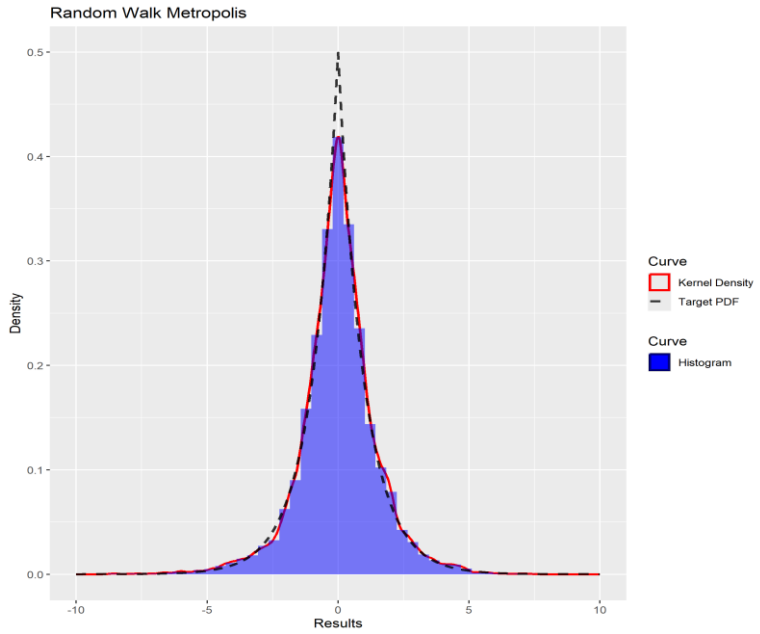
RAHUL PANT

UOL ID: 240725969

SIM ID: 1216470

Table of Contents

PART A



The results we obtained are Monte Carlo Mean -> 0.0117 (to 4dp and Monte Carlo SD -> 1.4727 (to 4dp)

The objective of part one is to simulate random numbers and run them through the Random Walk Metropolis (RWM) algorithm and then compute the mean and Standard deviation (SD). We then must compare them to the True Mean and True SD of the f(x) curve. The comparison will allow us to know whether our Random Walk Metropolis (RWM) algorithm is running correctly

We start by plotting the output of the RWM in the form of a histogram and kernel density plot (KDE). The histogram illustrates the frequency of data values falling within the intervals set, while the KDE presents the density estimates of the data values in form of a curve.

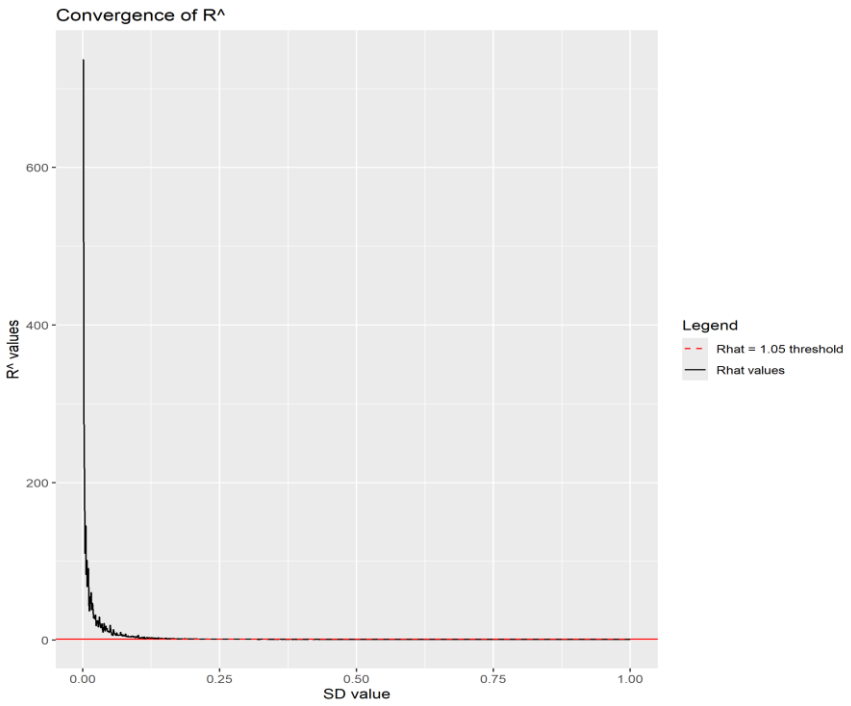
From the diagram generated, we can see that the KDE and true f(x) curves share the same symmetry. The Monte Carlo mean we got was 0.0117 (to 4dp) which is close to the theoretical mean of 0. The distribution is symmetric and centered at 0 hence generating a sample mean near 0 indicates that our chain is exploring the distribution correctly. However, we cannot achieve an exact 0 because of the limit put on the number of samples. As we collect more samples, we are more likely to capture the full range of possible values, hence allowing the KDE to achieve a better approximation of the true density. With more data we are also able to decrease the variance of KDE as with more samples we can reduce the random fluctuations, allowing the estimates we generate to have less noise.

The Monte Carlo Standard Deviation we got was 1.4727 (to 4dp) which is reasonably close to the sqrt (2) ≈ 1.4142, which is the theoretical SD. The SD captures the spread of

the distribution. It tells us that the samples are on average, 1.47 units away from the mean. We can tell the chain is capturing the correct spread as it is close to sqrt (2).

Refer to Appendix A for Python

PART B




```r
index <- which(s_value == 0.001)  # Find index where s = 0.001
Rhat <-R[index]  # Get the corresponding R-hat value

print(paste("The Rhat for the random walk Metropolis algorithm with N = 2000, s = 0.001 and J = 4 is", round(Rhat, digits = 2)))
```

```
[1] "The Rhat for the random walk Metropolis algorithm with N = 2000, s = 0.001 and J = 4 is 737.26"
```

The Rhat value that we got for N = 2000, s = 0.001 and J = 4 was 737.26. The Rhat value informs us how well the 4 chains are mixing (if they have converged). As the question stated, the value of Rhat close to 1 indicates convergence, and its usually desired for Rhat value to be less than 1.05. The cause of the high Rhat value is due to the small s that was said to be set. We can interpret s as the standard deviation (SD) of the proposal distribution, which controls the step size in the RWM function. When we set the s value to 0.001, the steps taken from each initial point that we set are small the chains stay in those regions for a longer time instead of spreading out and mixing well. Hence the chains take longer to converge, explaining a large Rhat value.

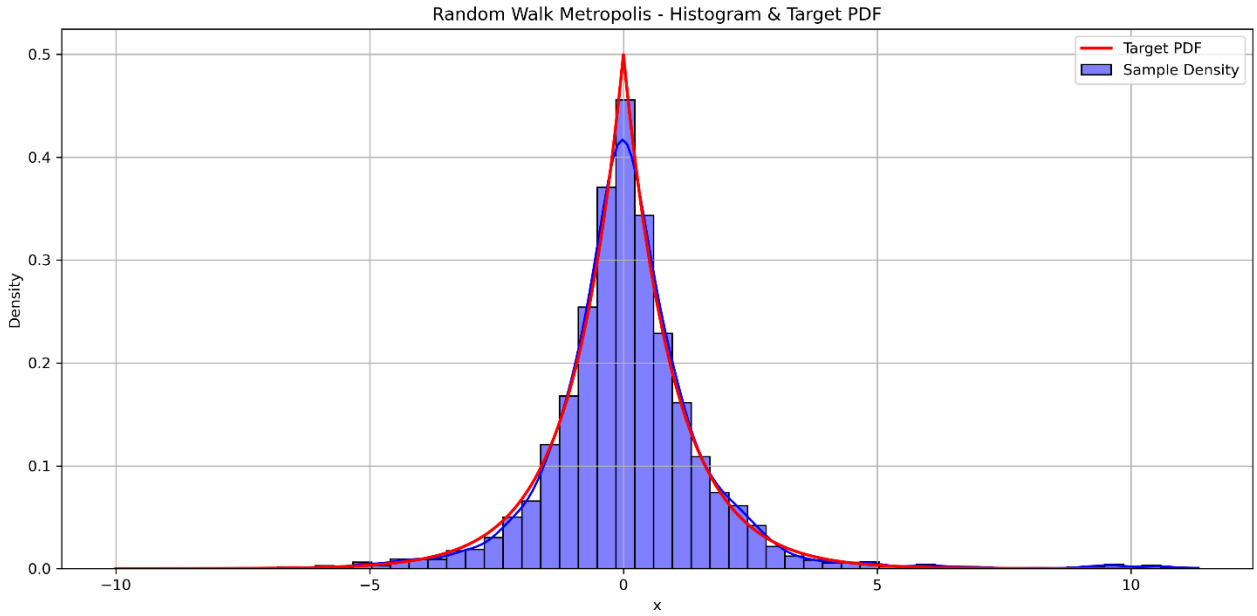
However, it is important to note that as the s_value increases, the step size in the proposal distribution also increases, which allows the chain to take lager steps, exploring a wider region. By being able to move more freely between the different areas they can mix better and are not be fixed to a localized area. We can confirm this as we get an r hat value of 1.013643 when the s is set to 1 proving that the chains eventually converge.

Refer to Appendix B for Python

REFERENCES

- R rnorm() Function - Scaler Topics. Scaler Topics. https://www.scaler.com/topics/r-rnorm/
- Generating a sequence in R using seq() function. DigitalOcean. https://www.digitalocean.com/community/tutorials/generating-a-sequence-in-r
- Indexing and iteration. https://www.stat.cmu.edu/~ryantibs/statcomp/lectures/iteration.html
- numpy.linspace — NumPy v2.1 Manual. https://numpy.org/doc/2.1/reference/generated/numpy.linspace.html
- seaborn.histplot — seaborn 0.13.2 documentation. https://seaborn.pydata.org/generated/seaborn.histplot.html

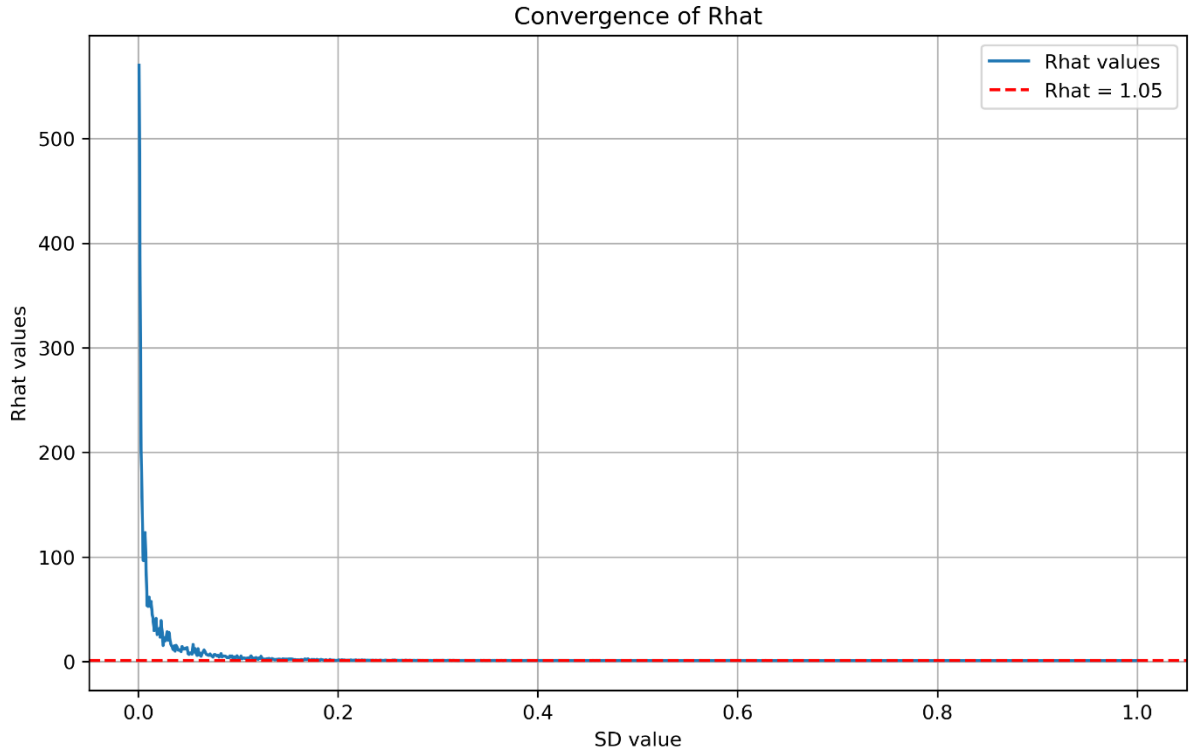
APPENDIX A




```
sample_mean = r_sample_df["Results"].mean()
print("The mean of the generated samples is", sample_mean)
sample_sd = r_sample_df["Results"].std()
print("The standard deviation of the generated samples is", sample_sd)
```

```
The mean of the generated samples is 0.10086085573189417
The standard deviation of the generated samples is 1.5887575767294413
```

APPENDIX B



[51]:

```
index = np.where(s_values == 0.001)[0][0]

# Get the corresponding R̂ value
Rhat = R[index]


print(f"The Rhat for the random walk Metropolis algorithm with N = {N}, s = 0.001 and J = {J} is {Rhat}")
```

```
The Rhat for the random walk Metropolis algorithm with N = 2000, s = 0.001 and J = 4 is 569.9785779835122
```