Problem 1

Question:

Remember from last week we discussed that skewness and kurtosis functions in statistical packages are often biased. Is your function biased? Prove or disprove your hypothesis.

Answer:

For this problem, I used the methods *skew* and *kurtosis* from the python package *scipy.stats*. Even though there's a parameter called *bias* of which when set to *False* can correct the statistical bias of skew and kurtosis calculation, I still used its biased version in default. However, my tested results show that the calculations from *skew* and *kurtosis are statistically* significantly biased for smaller sample sizes but not for the larger ones.

In this test, the null hypothesis is formed as both the *skew* and *kurtosis* functions are unbiased. Here I tested for three test sample sizes—10, 100, 1000—to see if a smaller sample size would generate a biased result and a larger sample size would not. For each individual in the sample, 100 observations are generated from standard normal distribution, and their skewness and kurtosis values are calculated using the aforementioned functions and stored lists. Eventually, a t-test was performed on each list of skewness and kurtosis using the *ttest_1samp* method from the *scipy.stats* module. The test results are recorded in the following table.

Table 1: Test Statistics for Biased of Skew and Kurtosis Functions

Sample Size	Skew		Kurtosis	
	t-stats	p-value	t-stats	p-value
10	5.68	0.0003	-5.28	0.0005
100	-1.39	0.17	-2.64	0.009
1000	0.7	0.48	-3.96	0.113

The t-stats and p-values for the test on *skew* indicate that we fail to reject the null hypothesis when the sample sizes are 100 and 1000 but can reject the null when the sample size is 10. Thus for a smaller sample size, we can have the 95% confidence to say that the *skew* function is biased. For the *kurtosis* function, it is biased for 10 and 100 sample sizes at the 95% confidence level. As a result, the *skew* and *kurtosis* functions are biased for small sample sizes, but a larger sample size can offset this biasedness.

Problem 2

Question:

Fit the data in problem2.csv using OLS and calculate the error vector. Look at its distribution. How well does it fit the assumption of normally distributed errors?

Fit the data using MLE given the assumption of normality. Then fit the MLE using the assumption of a T distribution of the errors. Which is the best fit?

What are the fitted parameters of each and how do they compare? What does this tell us about the breaking of the normality assumption in regards to expected values in this case?

Answer:

There are three parts of the problem and are solved in terms. To fit the data in OLS and study the errors, the model called *LinearRegression* is used, which is in the *linear_model* module under *sklearn* package in python. The regressed model is in such form:

$$y = 0.12 + 0.605x.$$

This regression model fits the data pretty well, by capturing the trends between x and y (Figure 1). Figure 2 and 3 shows the scatter plot and distribution of the error terms. The errors are not perfectly normally distributed but still are close. They satisfy the mean zero assumption pretty well (Figure 2) . Even though the errors are not exactly normal, with the highest bar in histogram not in the middle, it fits the normal distribution in a close form.

Figure 1: Regression fit

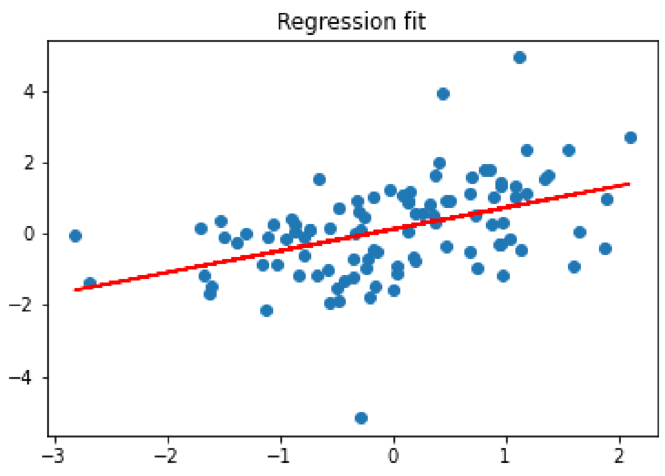


Figure 2: Error scatters

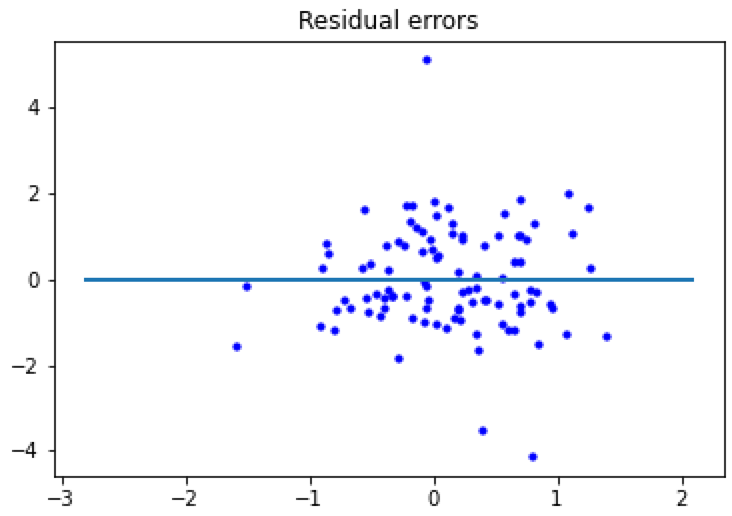
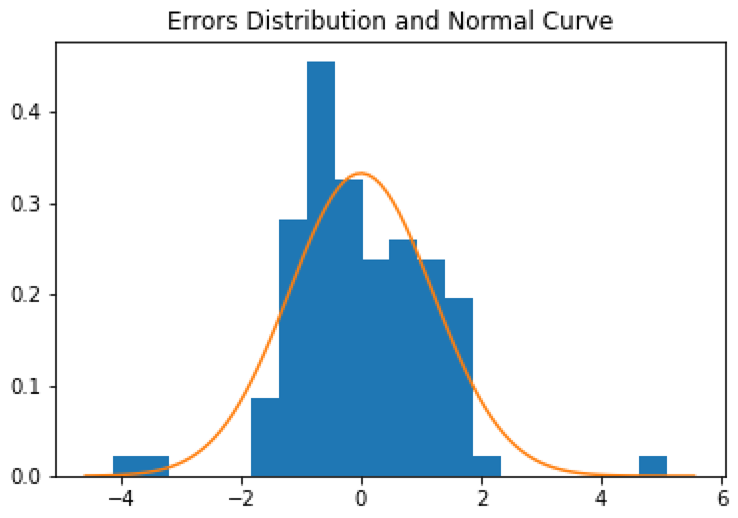


Figure 3: Errors Distribution and Normal Curve



At last, a shapiro-wilk test is performed to test the normality of the error terms. The tested p-value is 0.00015, from which the null hypothesis is rejected, and it can be concluded that there's not enough confidence to say that the errors satisfy the normality assumption.

For fitting the data using MLE, I first construct the log-likelihood function under the assumption of both normal distribution and T distribution respectively using the *stats.norm* and *stats.t* method in python. Then, the *minimize* function under *scipy.optimize* module is used to optimize the log-likelihood function and find the MLE. The results from MLE fitting are as follows:

Table 2: Results MLE Fitting

	Normal Distribution	T Distribution
Intercept	0.12	0.07
Beta	0.605	0.56
R-square	0.1946	0.1674
AIC	323.984	315.031
BIC	329.194	320.241

The results show that under the assumption of normality, the coefficients of the model fitted using MLE are the same as the OLS.

To compare the fitted parameters and methods, I calculated the R-square and information criterion such as AIC and BIC. The lower AIC and BIC scores from the model fitted under T distribution indicates that this one is a better fit. Therefore, although the OLS usually gives good results, the assumption may be hard to satisfy. As a result, we can loosen the assumption and use the MLE fitting to obtain a better fit.

Problem 3

Question:

Simulate AR(1) through AR(3) and MA(1) through MA(3) processes. Compare their ACF and PACF graphs. How do the graphs help us to identify the type and order of each process?

Answer:

The AR and MA processes are simulated using the *ArmaProcess* method under the *statsmodels* package in python, and the ACF and PACF graphs at the end of this report.

From the graphs we can see that the ACF of AR(p) models decays exponentially, while the PACF cuts off after the lag parameter p. The pattern for MA(p) is the opposite, white the ACF of MA(p) cuts off after the lag p and the ACP decays exponentially.

Figure 4: AR(1)

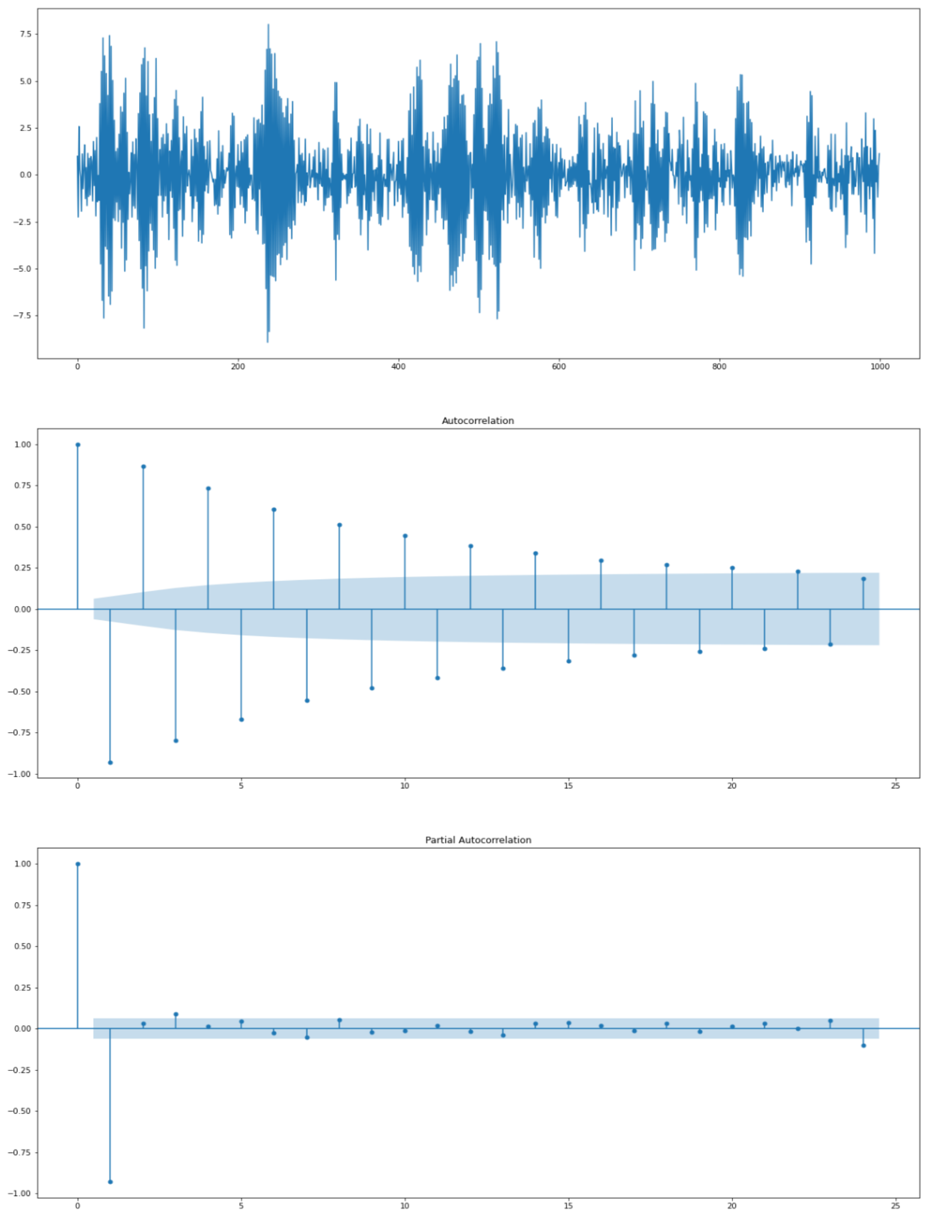


Figure 5: AR(2)

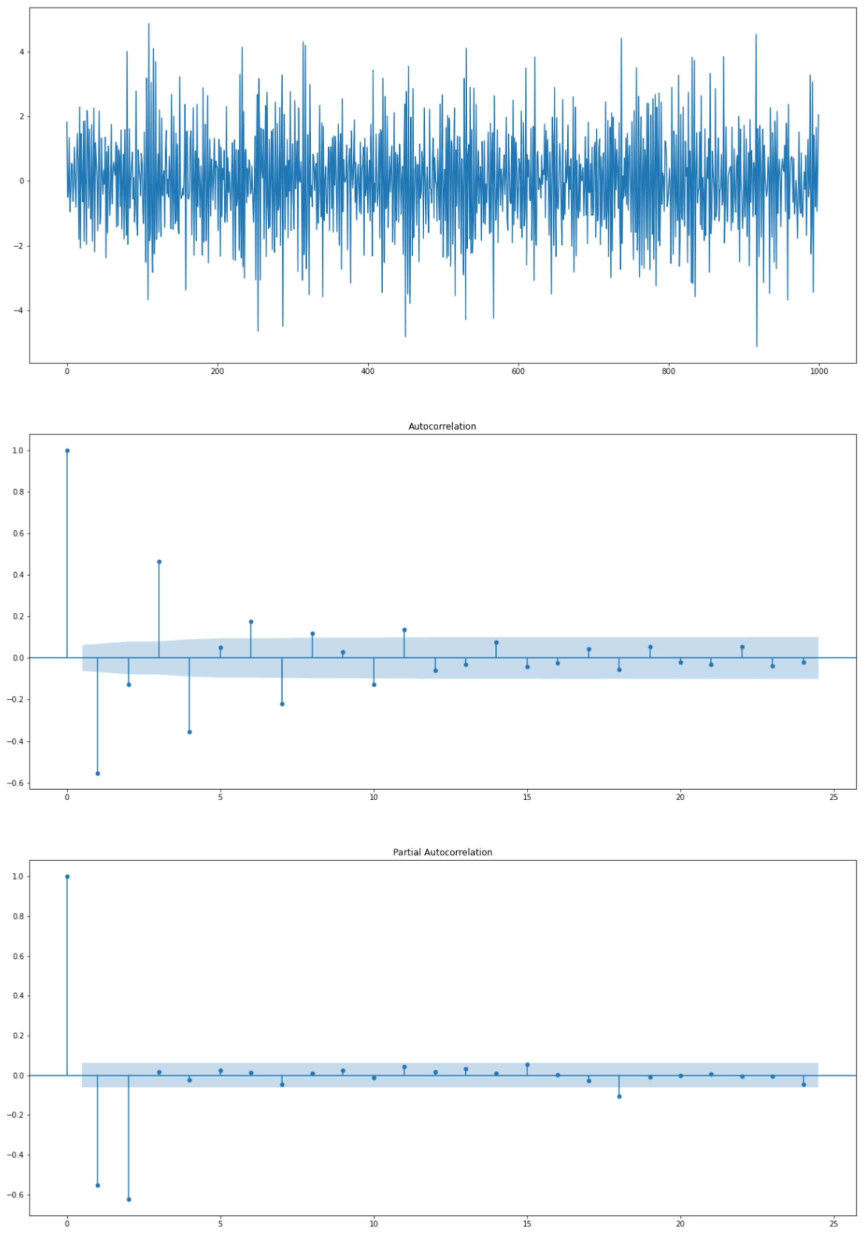


Figure 6: AR(3)

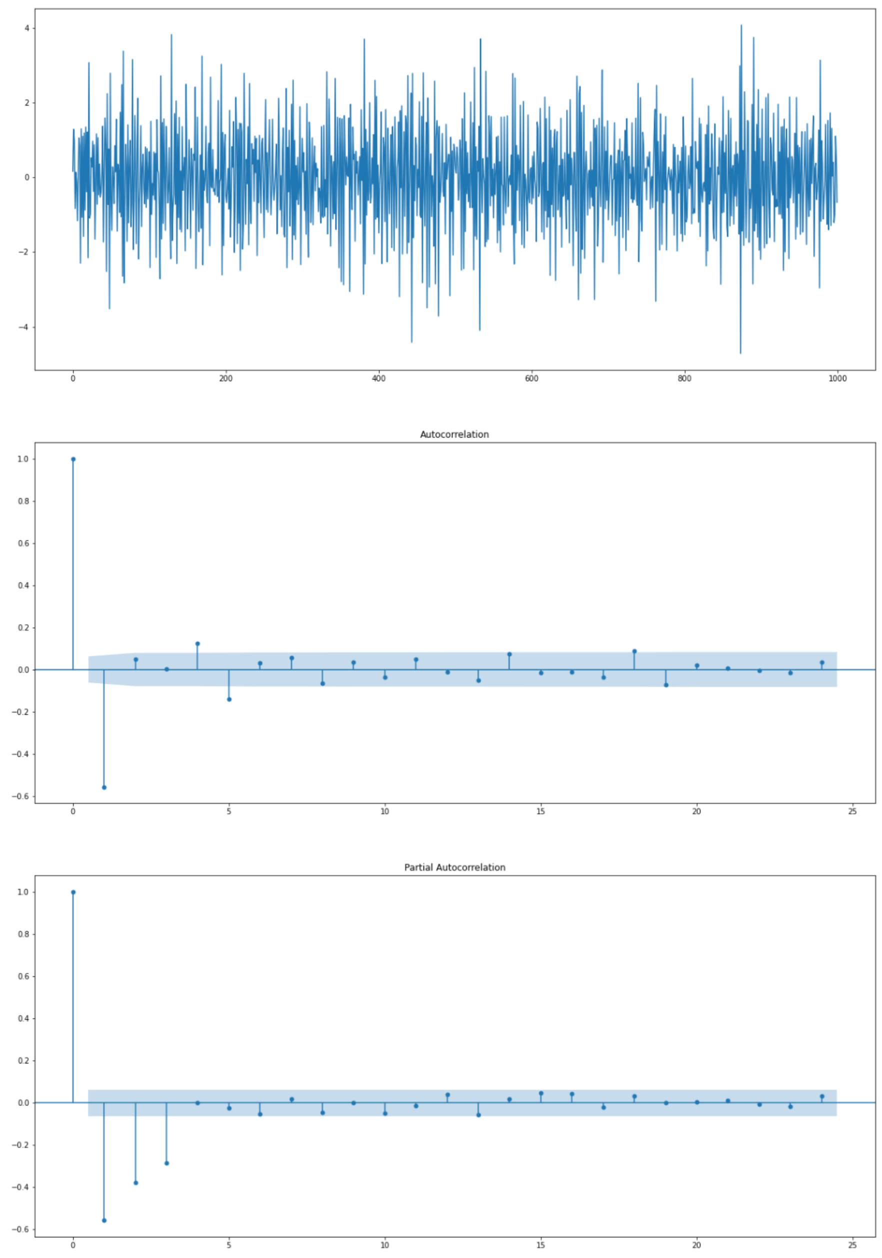


Figure 7: MA(1)

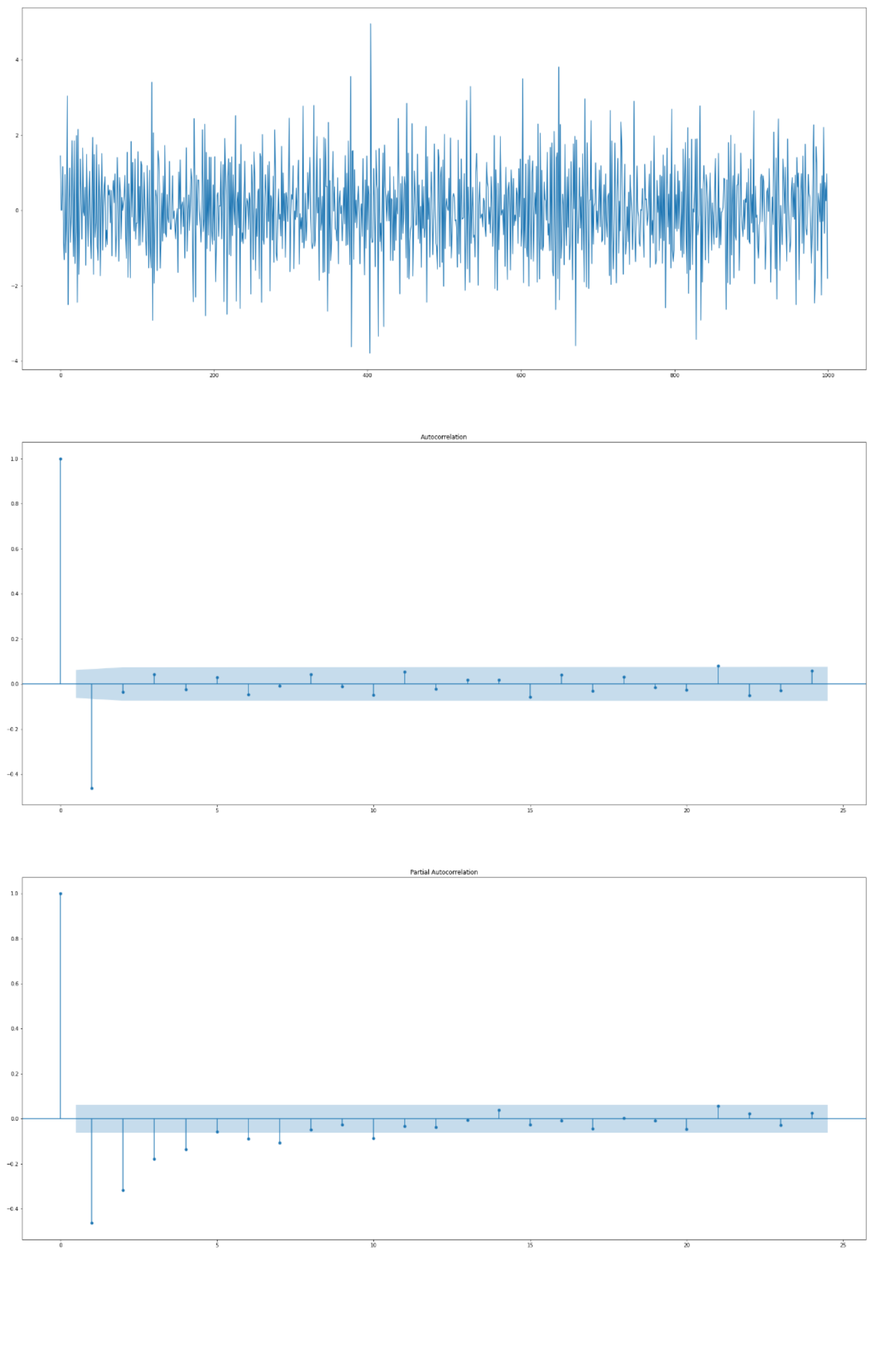


Figure 8: MA(2)

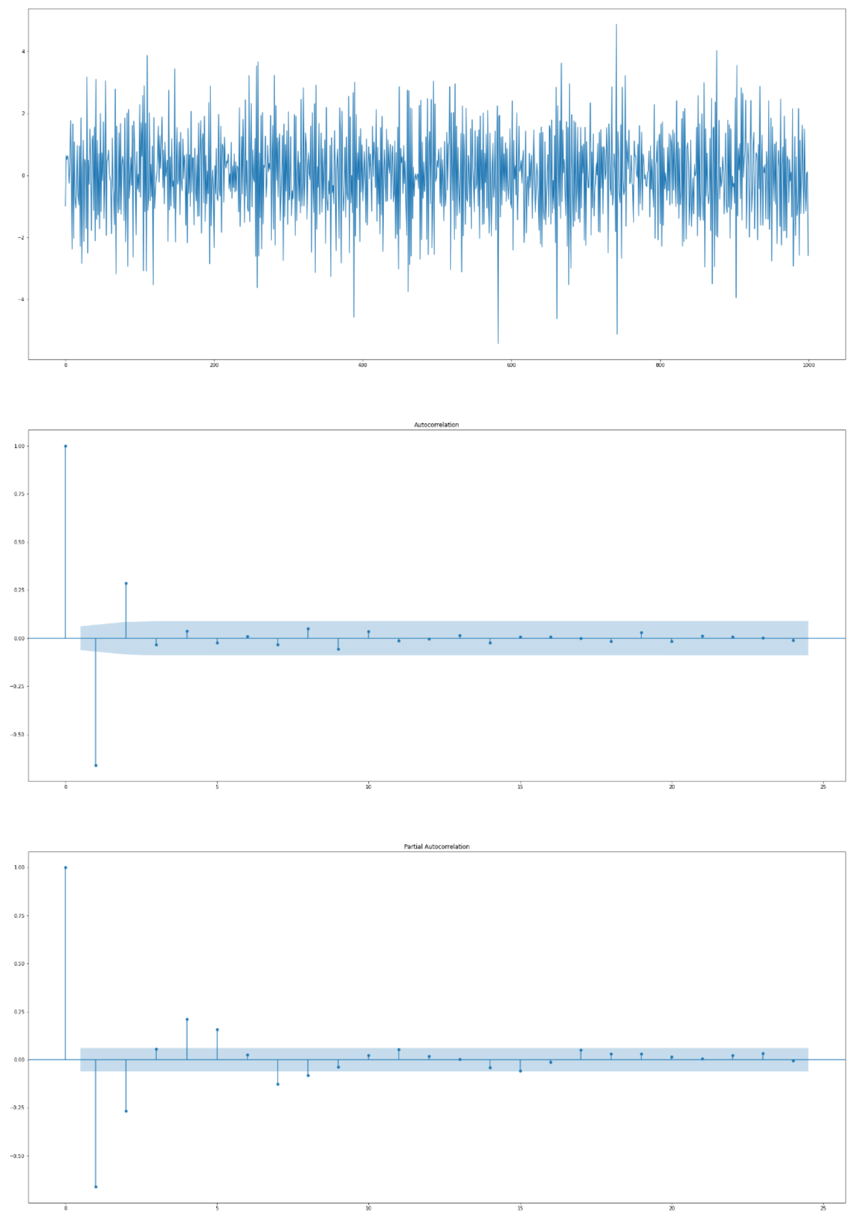
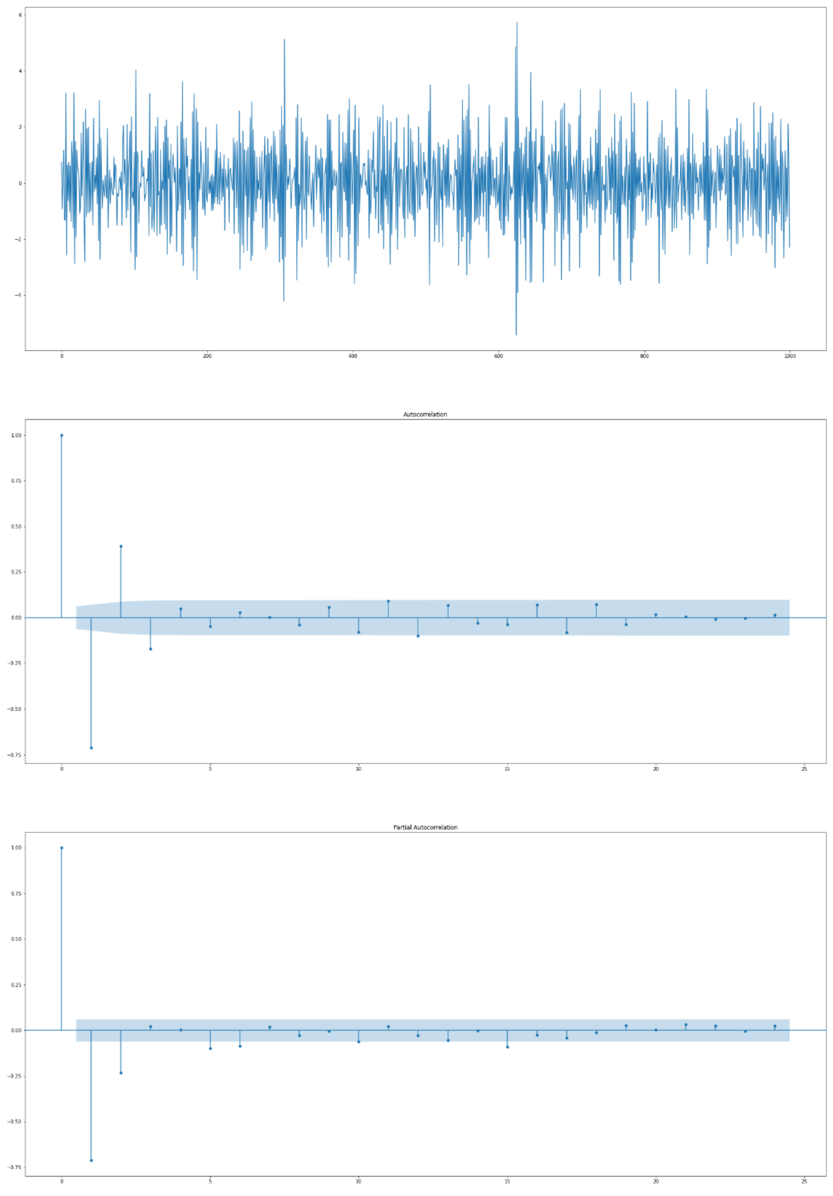


Figure 9: MA(3)


Autocorrelation
Partial Autocorrelation