Problem 1

Is skewness and kurtosis functions in python's packages biased?

To verify if skewness and kurtosis functions in python's scipy package is biased, we would like to make a set of standardized random normal values. As we know, a standard normal set, which $\mu = 0$ and $\sigma = 1$, the skewness and kurtosis are expected to be 0. In that case, we can easily verify the accuracy by conducting a two sides t-test to compare the mean of the skewness and kurtosis and zero respectively. Then, we can get a p-value from the test statistics. By comparing the p-value with $\alpha = 0.05$, we can reject the null hypothesis that $\mu = 0$ if $pvalue < \alpha$, that is, we are statistically confident that the functions that calculate skewness and kurtosis functions in python scipy package are biased. On the other hand, If $pvalue > \alpha$, we will fail to reject our null hypothesis that $\mu = 0$. We can conclude that the functions that calculate skewness and kurtosis functions in python scipy package are not biased.

First, we generate 100 standardized normal values, and then apply skewness and kurtosis function in scipy package: skew(data, kurtosis(data). To ensure avoiding test bias, we run this process 10000 times and record all skewness value and kurtosis value. Taking the mean of skewness and kurtosis respectively. Then, find t-statistics by mean of skewness and kurtosis. Finally, use function stats.ttest_1samp() to generate p-values.

Result

Skewness function t-test

TtestResult($statistics = 0.52$, $pvalue = 0.60$, $df = 9999$)

Kurtosis function t-test

TtestResult($statistics = -12.18$, $pvalue = 7.33e - 34$, $df = 9999$)

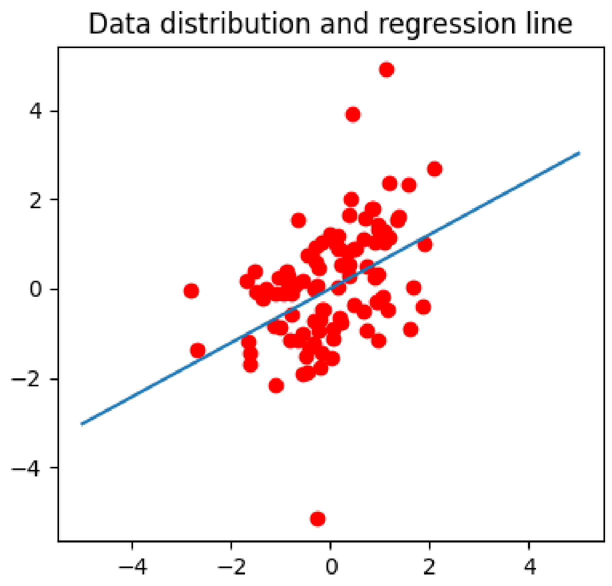
Conclusion

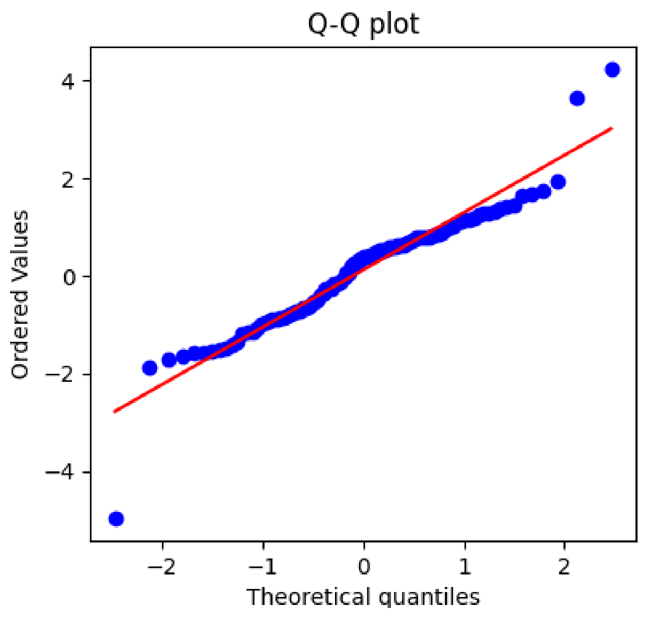
By our large sample size, we can see pvalue of Skewness function is way larger than 0.05, however, pvalue for kurtosis is approximately to zero. Thus, we fail to reject our null hypothesis in the skewness function, that is, we can say skewness function in python is unbiased. However, we reject the null hypothesis for kurtosis that $\mu = 0$. So the kurtosis function in python is biased.

Problem 2

Data fitting with OLS model

Using pandas package in python to read the given .csv file. Then we using statsmodel.api package to fit the data with OLS model by statsmodel.api.OLS(Y,X), where Y, X are data given in the file. Error vector $= Y - X * \beta$. By drawing a Q-Q plot of the error vector. We could see this is a highly skewed distribution





Data fitting using MLE given the assumption of normality and t-distribution

First, write the function that return negative of log likelihood of normal distrtibution. Then, use the minimize function to find θ that can maximize the value of -LL(log likelihood). $\theta = 0.605$, After get the θ, we can find AIC of the model by $2k - 2log(L)$, where $k = p + d$, and p is slope number + 1, and d is 1. Then, $AIC = 6 - 2 * LL = 325.98$

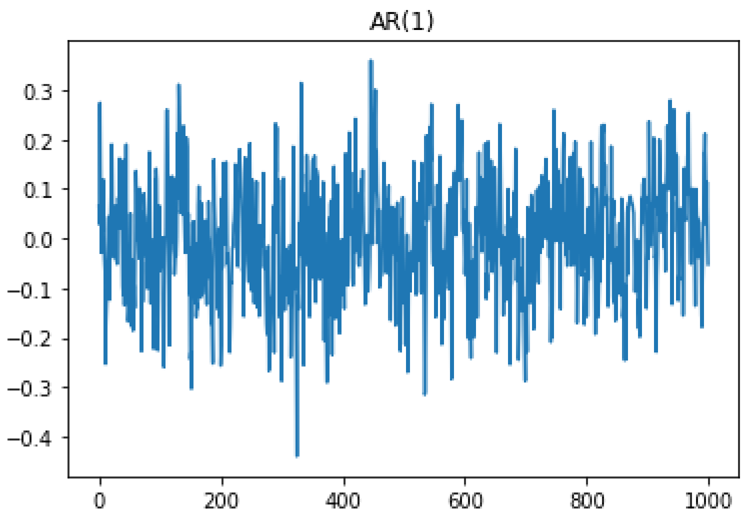
Then, by the same way, write the function that calculate the log likelihood of t distrtibution. $\theta = 0.966$, $AIC = 284.44$. By comparing the AIC of two assumption, $284.44 < 325.98$, we can conclude that MLE with assumption of T distribution is better at fitting the data.

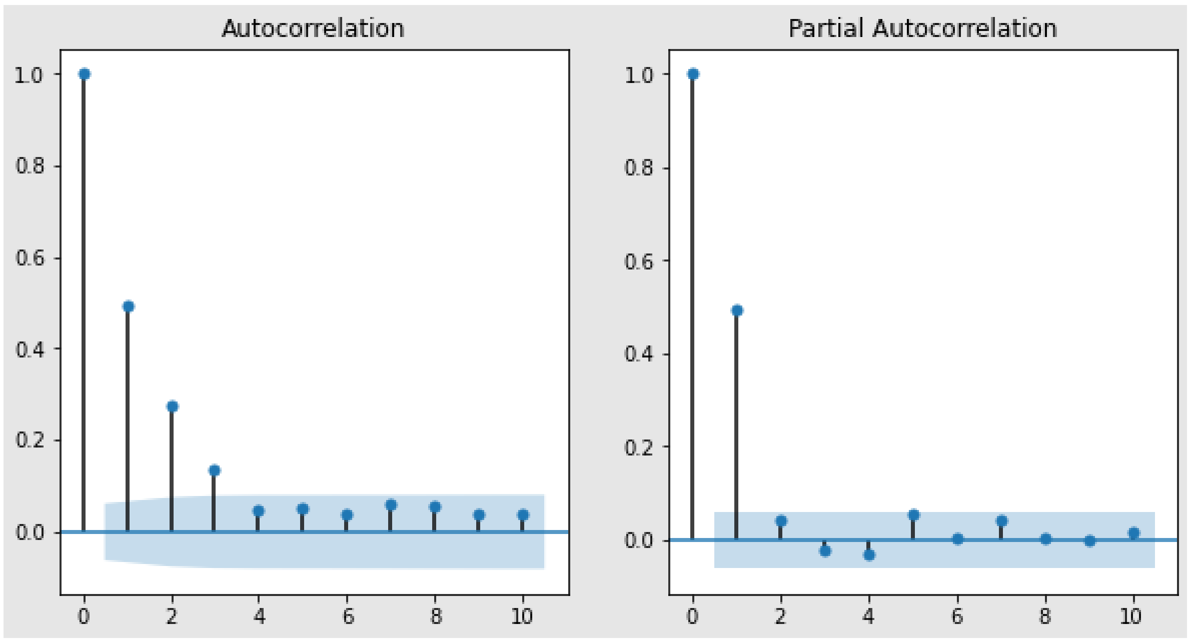
We acutally a relatively large difference between MLE with assumption of normality and MLE with t-distribution. And AIC shows the latter one perform better. However, the parameters of OLS model and MLE with assumption of normality are identical. It shows that if we breaking the normality assumption, we will get a relatively unreliable model fitting.

Problem 3

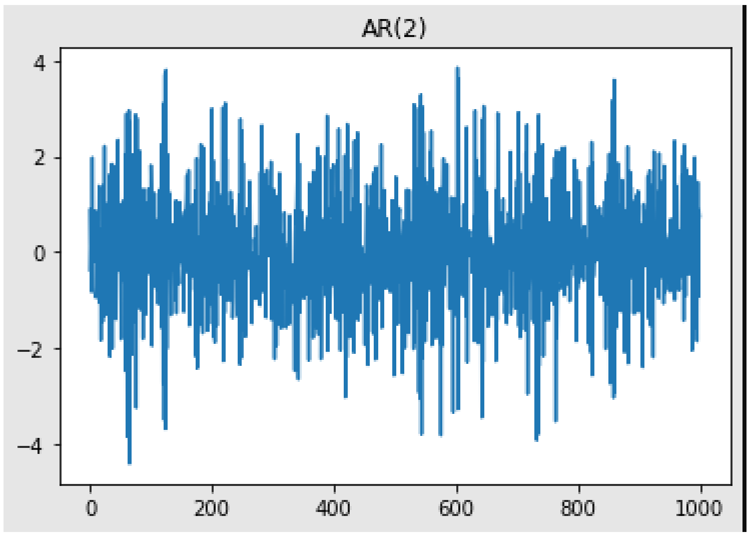
To simulate AR(1), we use the arima_process module from the statsmodels library. We use y = arma_generate_sample(ar_coefs, [1], scale=0.1, nsample=1000, burnin=50), where ar_coefs are the coefs (beta) we set for the simulation, for AR(1), we set it $[1, 0.5]$, where 1 is the coefficient for the constant term. scale is the standard deviation of the noise, and set a burnin=50 because we need to run error term to avoid the small bias from the beginning of generating data.

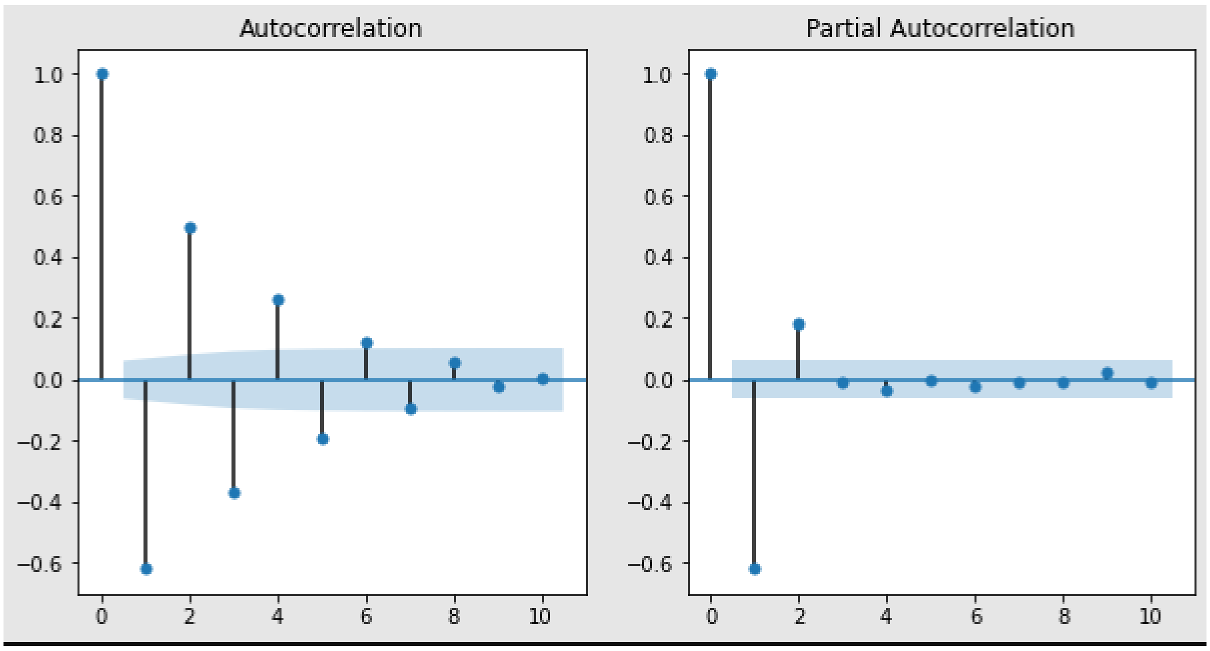
Then plot ACF and PACF by function plot_acf and plot_pacf in statsmodels.graphics.tsaplots module.



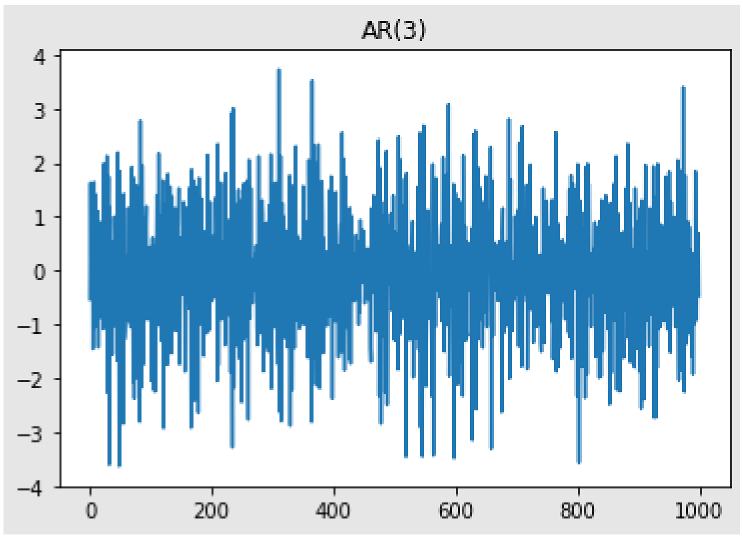


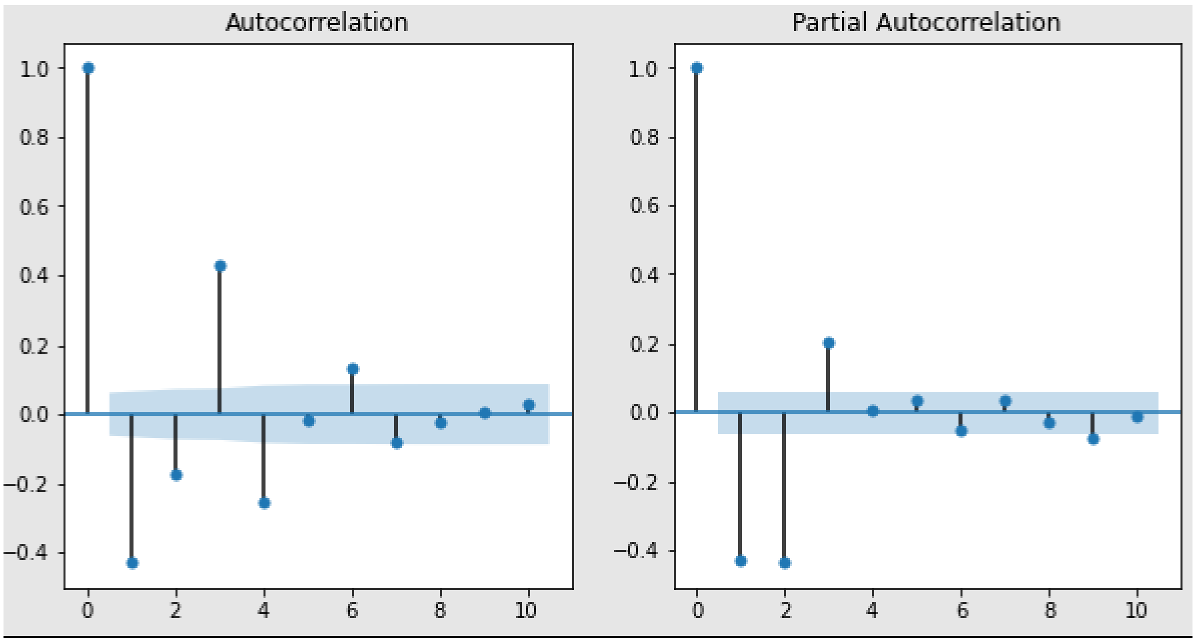
For AR(2), we can use the same simulation function, but we need to add another parameter in ar_coefs, to make it $[1, 0.5, -0.2]$, so similarly we can plot the graphs as following:





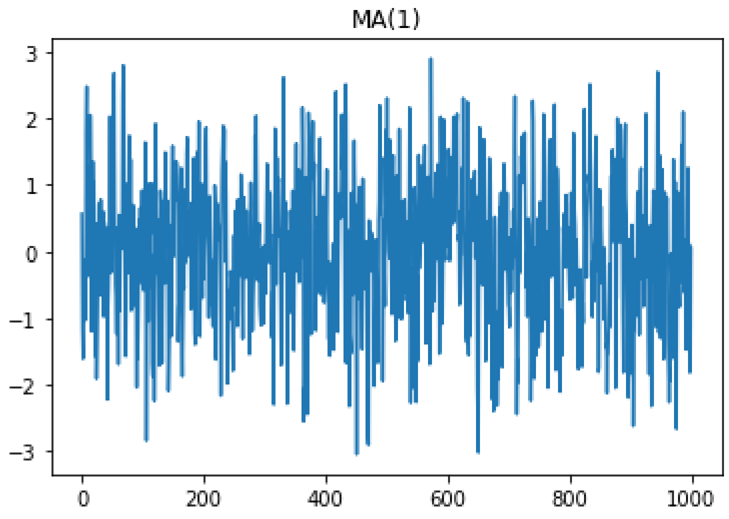
Similarly for AR(3), we add another β based on AR(2), ar_coefs = $[1, 0.5, 0.3, -0.2]$, we can plot the graphs as following:

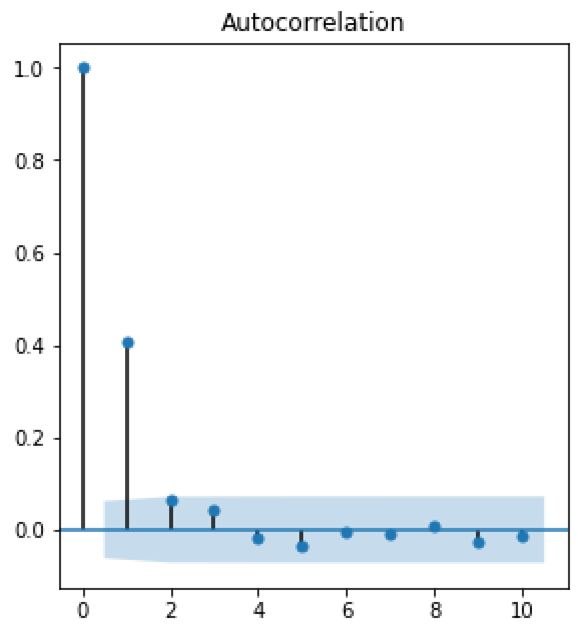


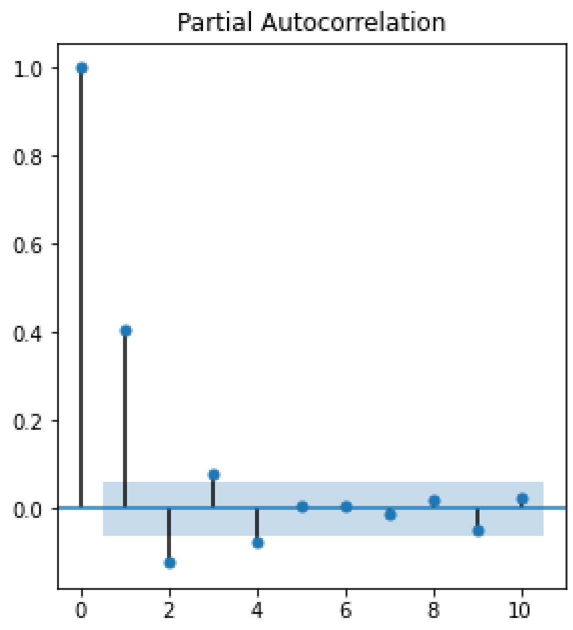


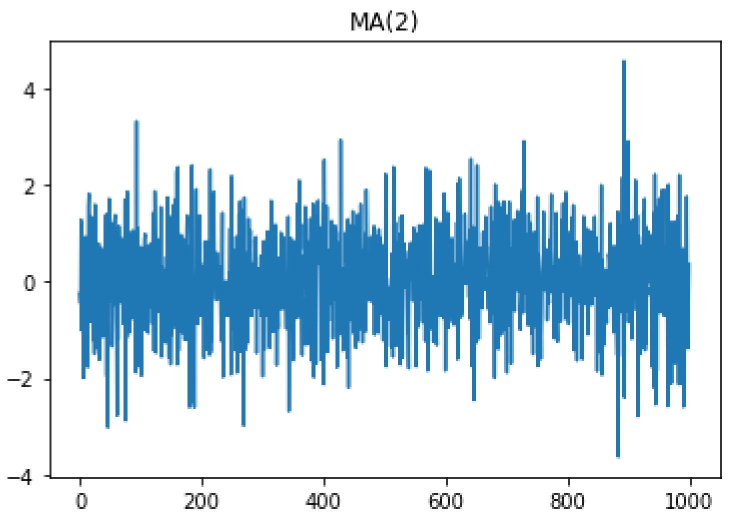
To simulate MA(1), we use the arima_process module from the statsmodels library. We use y = arma_generate_sample([1], ma_coefs, scale=0.1, nsample=1000, burnin=50), where ma_coefs are the coefs (theta) we set for the simulation, for AR(1), we set it $[1, 0.5]$, where 1 is the coefficient for the constant term. scale is the standard deviation of the noise, and set a burnin=50 because we need to run error term to avoid the small bias from the beginning of generating data.

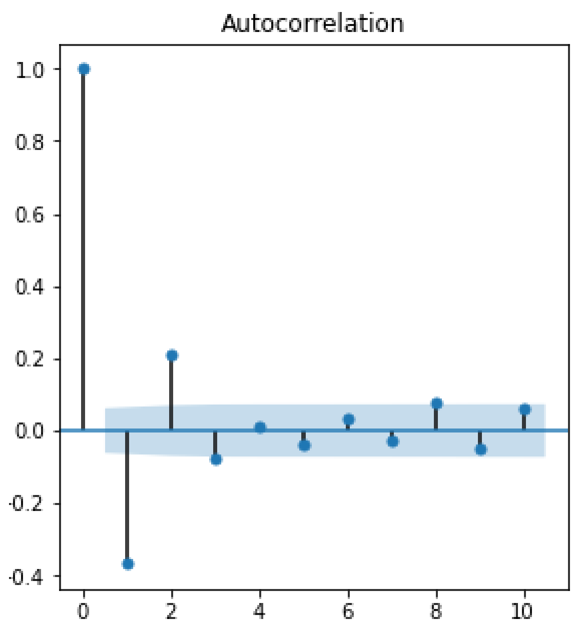
Similar to we done in AR(1) to AR(3), we can generating simulation data and ploting.

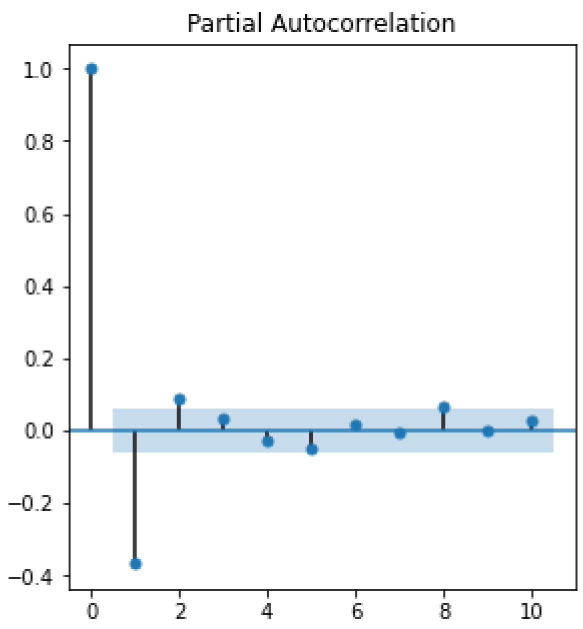


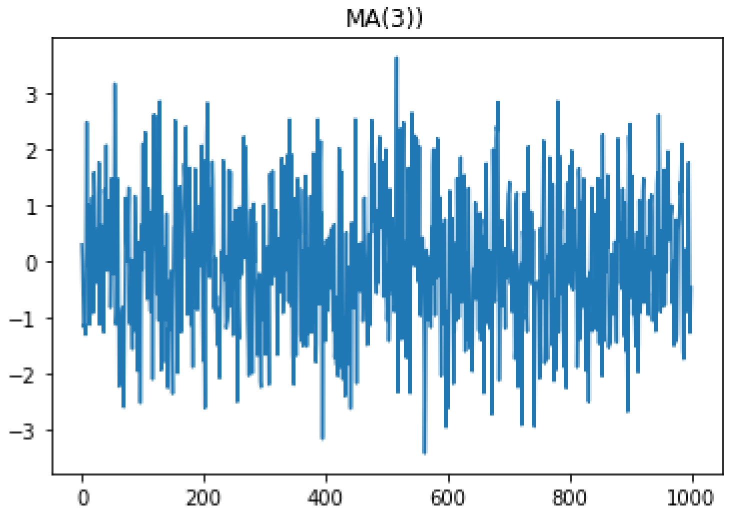

Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10

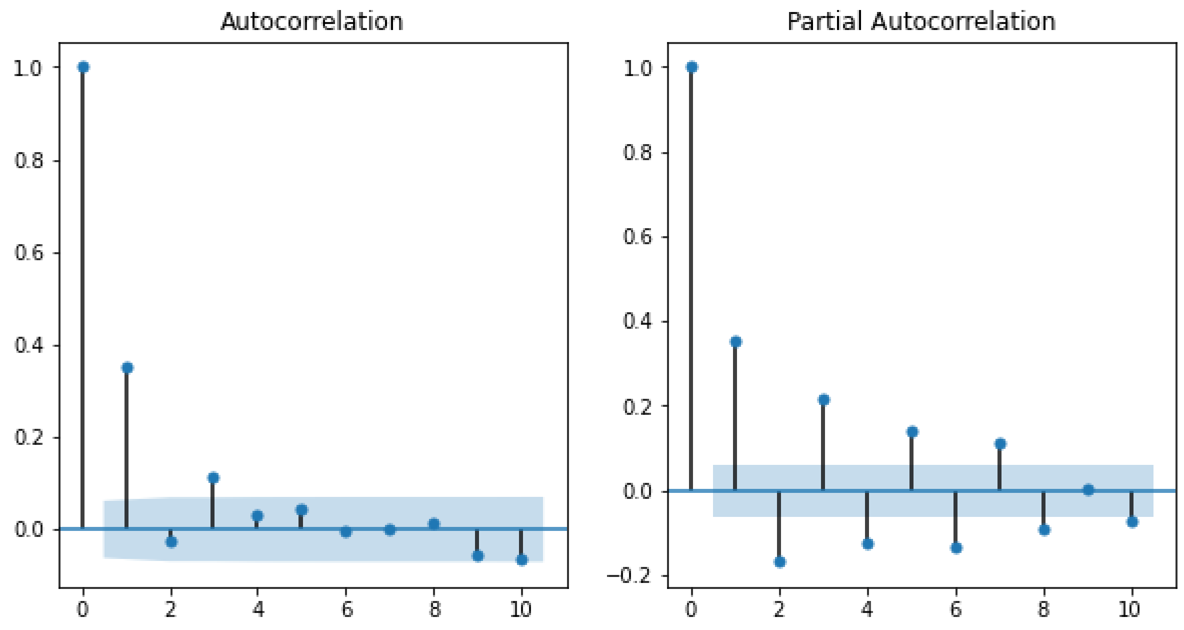

Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10


MA(2)
4
2
0
−2
−4
0
200
400
600
800
1000


Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-0.4
0
2
4
6
8
10


Partial Autocorrelation
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
2
4
6
8
10





Conclusion

For both AR(p) and MA(p) processes, the PACF will have non-zero coefficients for the first p lags and will taper off for the lags beyond that. The ACF will also have non-zero coefficients for the first p lags, but it will drop faster than that in PACF

We can identify the order of each process by looking the points above and below the blue range, usually the number of points beyond will represent the order of process.