Problem 1

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.

The function for skewness is unbiased, while the function for kurtosis is **not** unbiased.

To get these conclusions, I use hypothesis testing respectively. First, I set null hypotheses to be: H_0: skewness function is unbiased; H_0: kurtosis function is unbiased, and threshold, or alpha, to be 5%. Then, I start following steps listed in the class notes for testing whether skewness and kurtosis functions are biased or not:

1. Sample 100,000 standardized random normal values
2. Calculate the skewness and kurtosis
3. Sample the skewness and kurtosis by repeating steps 1 and 2 100 times
4. Calculate mean and standard deviation of skewness and kurtosis respectively
5. Calculate T statistics with null hypotheses being both functions are unbiased (which means both population means are 0)
6. Use the CDF function to calculate the p-value corresponding to a two-sided test for both skewness and kurtosis

I conduct the first three steps with a for loop, and calculate skewness and kurtosis for every of 100 samples, and store them into two separate data frames. Then, for steps four and five, I employ a describe method in a package called statsmodels to get means and standard deviations in a more convenient way, and those statistics are used for calculations of two T-statistics. Eventually, I take advantage of another package called scipy to perform the two-sided test and generate two p-values respectively for skewness and kurtosis.

When N = 100,

Trial 1:
```
skewness p-value: [0.42594084]
kurtosis p-value: [8.21543123e-227]
```

Trial 2:
```
skewness p-value: [0.50279043]
kurtosis p-value: [2.43208087e-231]
```

When N = 500,

Trial 1:
```
skewness p-value: [0.57578215]
kurtosis p-value: [0.]
```

Trial 2:
```
skewness p-value: [0.89719628]
kurtosis p-value: [0.]
```

When N = 1000,

Trial 1:
```
skewness p-value: [0.77113476]
kurtosis p-value: [0.]
```

Trial 2:
```
skewness p-value: [0.42628066]
kurtosis p-value: [0.]
```

As seen from resulting p-values above, at different sample sizes, p-values for skewness are greater than the pre-set threshold 5%, while those for kurtosis are effectively zero, and, therefore, smaller than 5%. Consequently, I fail to reject the null hypothesis for skewness, and reject the null hypothesis for kurtosis, concluding that the function of skewness is unbiased, while the function of kurtosis is not unbiased.

Problem 2

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?

Part 1

```
residuals

0    -0.838485
1     0.835296
2     1.027428
3     1.319711
4    -0.152317
        ...
95   -1.590264
96   -1.694848
97    0.434878
98    0.402261
99   -0.922319
Length: 100, dtype: float64
```

A brief summary of the error vector, shown on the left. For the complete vector, please run codes attached in the folder.

Residuals, or errors, of the OLS model run over problem2.csv are not distributed perfectly normally.

I first read data from problem2.csv and store them into a pandas DataFrame. Then, I extract data for X and Y respectively from the DataFrame, and employ the OLS model, included in the statsmodel package, to fit those data into the model. The summary of results is screenshot below.

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.195
Model:	OLS	**Adj. R-squared:**	0.186
Method:	Least Squares	**F-statistic:**	23.68
Date:	Fri, 27 Jan 2023	**Prob (F-statistic):**	4.34e-06
Time:	20:40:13	**Log-Likelihood:**	-159.99
No. Observations:	100	**AIC:**	324.0
Df Residuals:	98	**BIC:**	329.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.1198	0.121	0.990	0.325	-0.120	0.360
x	0.6052	0.124	4.867	0.000	0.358	0.852

Omnibus:	14.146	**Durbin-Watson:**	1.885
Prob(Omnibus):	0.001	**Jarque-Bera (JB):**	43.673
Skew:	-0.267	**Prob(JB):**	3.28e-10
Kurtosis:	6.193	**Cond. No.**	1.03

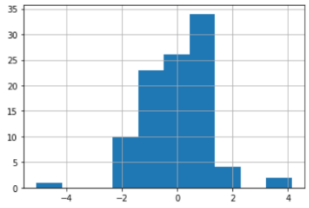
Then, I store residuals of the model from the results I derived from using the OLS model into a pandas DataFrame, and run two tests over those residuals - Lilliefors test and Shapiro Wilk test – both working to check normality of a group of data. For Lilliefors test, the result is shown below on the left hand side.

```
L_p_value

(0.09544408169056062, 0.025756222802914546)
```

```
SW_p_value

ShapiroResult(statistic=0.9383853673934937, pvalue=0.00015388781321235)
```

For Shapiro Wilk test, the result is shown below on the right hand side above. Both of these results could be interpreted in the way that residuals of the model are not distributed normally. If plotting the residuals as a histogram, as shown below, we could also conclude that residuals are not distributed normally.

```
<AxesSubplot:>
```



Part 2

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?

The latter one, with the assumption of a T distribution, is the best fit.

Due to involvement of MLE, I take these two questions as optimization problems. For each case, I first construct a function with parameters set and formula built for log likelihood. Then, after setting constraints of the optimization problem, I input initial guesses to optimize, in this case minimize, the negative log likelihood returned from the function constructed previously.

For the one using MLE given the assumption of normality, I get the following table as result:

```
lik_normal

      fun: 159.99209668916242
 hess_inv: array([[ 0.015659  ,  0.00068704, -0.00019829],
       [ 0.00068704,  0.01527205, -0.00025659],
       [-0.00019829, -0.00025659,  0.00742578]])
      jac: array([-1.90734863e-06, -1.90734863e-06, -3.81469727e-06])
  message: 'Optimization terminated successfully.'
     nfev: 72
      nit: 13
     njev: 18
   status: 0
  success: True
        x: array([0.60520483, 0.11983623, 1.19839411])
```

As seen from the last line, the first component of the array is the beta estimator derived from the MLE, and the second component of the array is the intercept estimator derived from the MLE. These two are nearly the same as results got from conducting OLS in part 1, which confirms the statement included in the class notes that "if assuming normality of errors, MLE would generate same/similar estimates as OLS". Also, the log likelihood, listed as negative log likelihood in the first line of the table, is the same as that result of OLS, -159.99209668916242.

On the contrary, for the other one using MLE assuming a T-distribution of errors, I get the following table as result:

```
lik_t

      fun: 155.51528727578858
 hess_inv: array([[ 1.43031893e-02,  9.65428929e-04,  1.50326057e-02,
         9.65428929e-04],
       [ 9.65428929e-04,  5.03341190e-01, -1.00979510e-02,
        -4.96658810e-01],
       [ 1.50326057e-02, -1.00979510e-02,  7.98892336e+00,
        -1.00979510e-02],
       [ 9.65428929e-04, -4.96658810e-01, -1.00979510e-02,
         5.03341190e-01]])
      jac: array([1.90734863e-06, 1.90734863e-06, 0.00000000e+00, 1.90734863e-06])
  message: 'Optimization terminated successfully.'
     nfev: 80
      nit: 14
     njev: 16
   status: 0
  success: True
        x: array([0.55893056, 0.06998071, 6.757348  , 0.06998071])
```

With the first element of the array in the last line being the estimated beta, coefficient of the independent variable, the result is different from the former one. Also, for this one, the log likelihood is -155.5152827578858, which is larger than that of the previous one. Since, the larger the log likelihood is, the more it fits, the one using MLE assuming a T-distribution is the best fit.

Note: for this part, credits to https://www.earthinversion.com/statistics/maximum-likelihood-estimation-with-examples-in-python/ for example of using MLE optimization

Part 3

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?

The fitted parameter of the former one, assuming normally distributed errors, is about 0.6052, while the one for the latter one is about 0.55. As concluded from last part, the one assuming a T-distribution of errors is the one fitting best. We may conclude that if normality of errors are violated, a T-distribution followed by errors would be the best distribution for estimation of parameters. Also, breaking of the normality assumption would make the estimation of parameters not reliable, contributing to a greater or a smaller confidence interval.

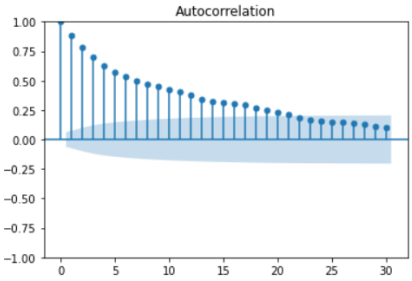
Problem 3

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?

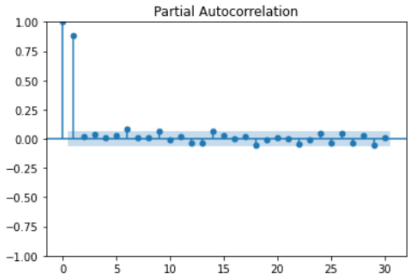
The codes for process simulations are in the python file within the folder. For each process, ACF and PACF graphs are screenshot below:

AR(1)

ACF

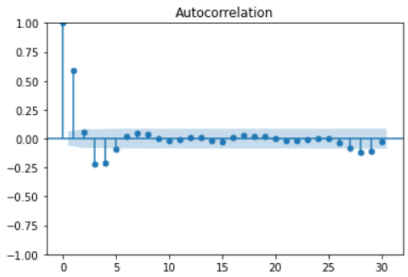


PACF

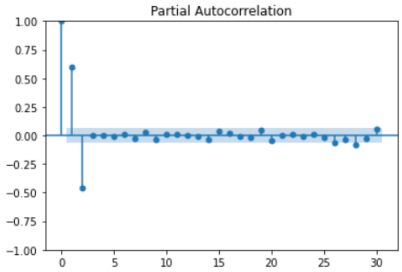


AR(2)

ACF

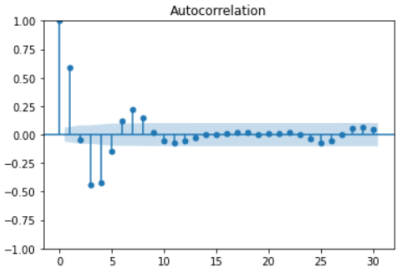


PACF

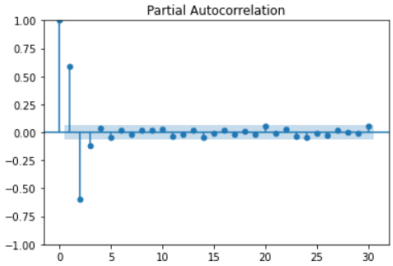


AR(3)

ACF

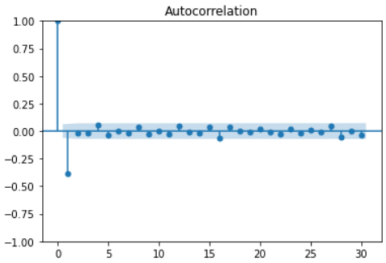


PACF

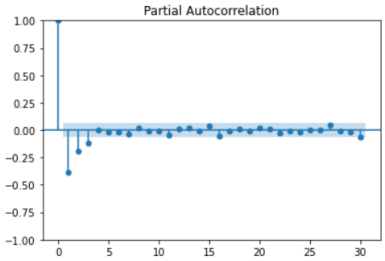


MA(1)

ACF

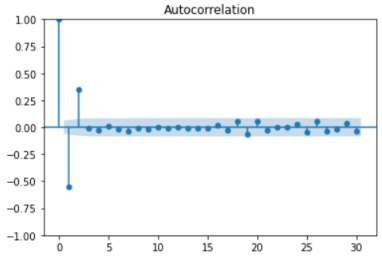


PACF

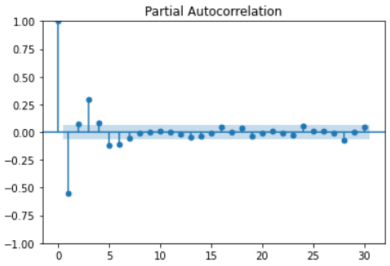


MA(2)

ACF

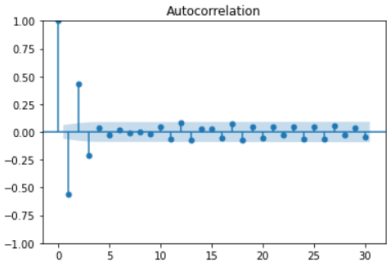


PACF

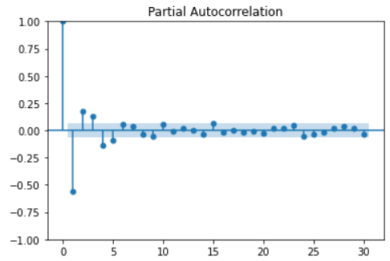


MA(3)

ACF



PACF



Upon how to discern different types and orders of processes through ACF and PACF graphs, we could see trends as below:

1. For AR processes, ACF graphs are mostly converging/decaying geometrically, while PACF graphs have significantly large observations, or great cut-offs, at the same lag as their levels. For example, for AR(1) process, there is a significantly large observation at lag 1 in its corresponding PACF graph.
2. For MA processes, vice versa. MA processes have PACF graphs with trends of geometric decaying/converging, while their ACF graphs usually have significantly large observations, or great cut-offs, at the same lag as their levels. For example, for MA(1) process, there is a great cut-off at lag 1 in its PACF graph.

For convenience, these two trends are made into the following table:

Graphs \ Processes	AR(x)	MA(y)
ACF	Converging/decaying geometrically	Large observation/cut-off at lag x
PACF	Large observation/cut-off at lag x	Converging/decaying geometrically