Multivariate Modeling
DATS 6450-15
Lab # 6
Taisha Ferguson
March 11, 2020, 2020 (TF)

Abstract

The primary purpose of this lab to simulate the processes for Auto-regressive and Moving Average models manually in Python and then with the Python library "scipy." In this lab it was shown that I was able to produce the same results with my manual processes as with the "scipy" library. In addition to simulating those process, the coefficients for the Auto-regressive models were estimated using the Least Square Estimation (LSE) method. It was my observation that increasing the sample size of the observations did not improve the estimation of the parameters of the model.

Introduction

In this lab the Auto-regressive and Moving Average models, both of order 2, were estimated manually in Python and also with the Scipy library. Auto-regressive and Moving average models are both time series methods use for predicting future outcomes based on past values. Auto-regressive models are estimated using the prior values of the dependent variable and Moving Averages are based on error terms, also known as white noise. The order of the model describes how many prior values or errors are used in the calculation of the future values.

The coefficients for both models were given as:

$$\text{AR(2)} : y(t) - 0.5y(t-1) - 0.2y(t-2) = e(t)$$
$$\text{MA(2)}: y(t) = e(t) + 0.5e(t-1) + 0.2e(t-2)$$

Using these coefficients, the future y values were simulated in Python and then using the dlsim function from the scipy library.

Methods, Theory, and Procedures

<u>Simulating the AR(2) Process</u>

In order the estimate the Auto-regressive processes shown in the Introduction, I first created error terms with mean 0 and standard deviation 1 using the Numpy library's random function. These parameters were given in the exercise. After, creating the error terms, also known as white noise, the y values were generated using a for loop that applied the given coefficients to previous y values with the first value being equal to the first error term. After the y values were computed manually, the results were also simulated using the dlsim function from the scipy library. It can be seen below that the values for both processes were the same.

```
The first few values of y(t) for Manual AR(2) process: [2.222106   0.98720185 2.92910054 2.3251778  3.66483989]
```

```
y(t) values using scipy for AR(2) process: [[2.222106  ]
 [0.98720185]
 [2.92910054]
 [2.3251778 ]
 [3.66483989]]
```

Estimating the Coefficients Using LSE

After the sample y values were generated using the steps explained above, the coefficients of the model were estimated using the Least Square Estimation (LSE) Method. The equation for LSE is below:

$$\hat{\boldsymbol{a}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$$

This process was done for 1000, 5000, and 10000 samples. The estimation of each round is shown below.

```
Estimated parameters using LSE with 1000 samples: [-0.63674912 -0.32902086]
```

```
Estimated parameters using LSE with 5000 samples: [-0.64192004 -0.32416699]
```

```
Estimated parameters using LSE with 10000 samples: [-0.63825544 -0.3276216 ]
```

Based on this it appears that increasing the samples size not improve the coefficient estimations.

Simulating the MA(2) Process

Similar to the AR(2) process, I first created error terms with mean 0 and standard deviation 1 using the Numpy libraries random function. After, creating the error terms, the future y values were computed using the previous error or white noise values using a for loop in Python. After the manual computation is was also simulated using the dlsim function from the scipy library. It can be seen below that values were the same for both processes.

```
First 5 values of y(t) for manual MA(2) process: [ 0.70156941  0.65776842  0.99030601  1.00495413 -1.04944264]
```

```
First 5 values for y(t) for MA(2) process using scipy:  [[ 0.70156941]
 [ 0.65776842]
 [ 0.99030601]
 [ 1.00495413]
 [-1.04944264]]
```

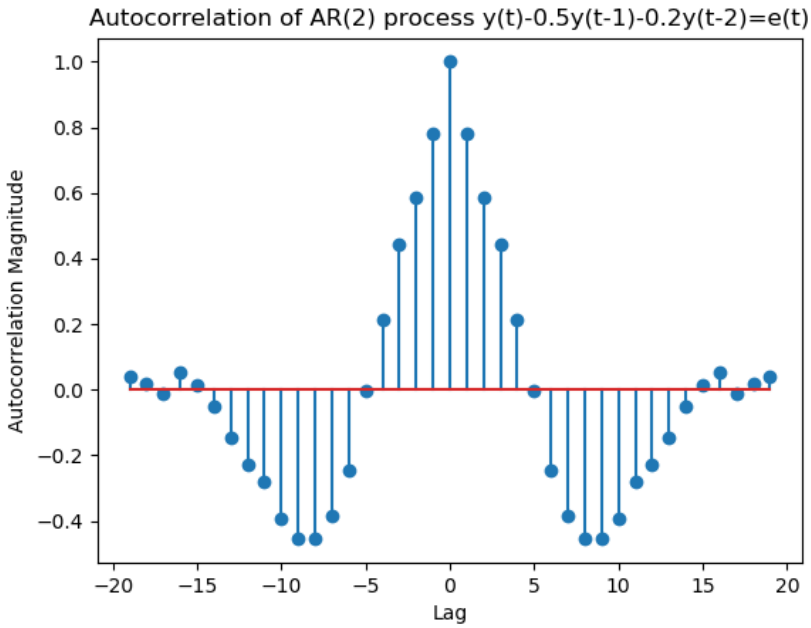
Answers to Asked Questions

1- Let consider an AR(2) process as

$$y(t) - 0.5y(t-1) - 0.2y(t-2) = e(t)$$

a. Using the python code, developed in previous labs, calculate autocorrelations for 20 lags and plot them versus number of lags. Write down your observation about the ACF of above process.

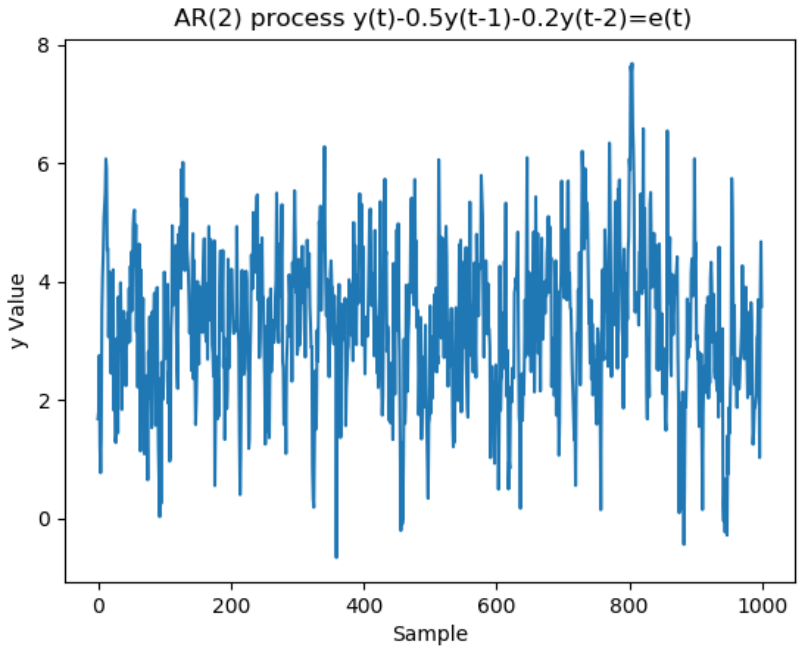
There appears to be an inverse pattern every 5 lags because the values begin to change direction every 5 lags.



b. Apply the ADF-test and check if this is a stationary process. Explain your answer.

Yes, the dataset is stationary because the results of the ADF test shown below has p-value below the 5% threshold. Also, the graph below shows that the mean and variance are the same over time.

```
ADF Statistic: -11.239040
p-value: 0.000000
Critical Values:
ADF Statistic: -11.239040
p-value: 0.000000
    1%: -3.437
    5%: -2.864
    10%: -2.568
```



2- Write the AR(2) process in question 1, as multiple regression model and using the least square estimate (LSE), estimate the true parameters a_1 and a_2 (-0.5, -0.2). Display the estimated parameters values at the console. What is the affect of additional samples on the accuracy of the estimate? Justify the answer by running your code for 5000 and 10000 data samples.

This was explained in the methods section. Increasing the sample size did not have any significant impact on the estimation of the coefficients.

3- Generalized your code in the previous question, such when the code runs it asks a user the following questions:

a. ***Enter number of samples :***
b. ***Enter the order # of the AR process:***
c. ***Enter the corresponding parameters of AR process :***

Your code should simulate the AR process based on the entered information (a, b, c) and estimate the AR parameters accordingly. The estimated parameters must be close to the entered numbers in part c. Display the estimated parameters and the true values at the console.

I was unable to generalize my code to accomplish all of the goals listed above. My function was able to take in the input values but I had a hard time creating incorporating the changes in indices for the different orders. The code that I was working on is in the appendix section but is commented out.

4- Let consider an MA(2) process as

$$\boldsymbol{y(t) = e(t) + 0.5e(t-1) + 0.2e(t-2)}$$

a. **Using the python code, developed in previous labs, calculate autocorrelations for 20 lags and plot them versus number of lags. Write down your observation about the ACF of above process.**

b. **Increase the data samples to 10000 and then 100000. Observe the ACF. Write down your observation. There is a difference between the ACF of an AR process and MA process. What is the main difference?**
c. **Display the first 5 values of y(t) at the console.**
d. Using various techniques learned in previous labs, Is this stationary process? Explain your answer.

Conclusion

The primary purpose of this lab to simulate the processes for Auto-regressive and Moving Average models manually in Python and then with the Python library "scipy." It was shown that I was able to produce the same results using different methods. I was also able to estimate the coefficients of the Auto-regressive method using the LSE method. Increasing the sample size did not seem to have an impact the accuracy of the coffecients. I was unsuccessful in generalizing the code in order to simulate the different order AR processes and then estimate the coefficients

Appendix – Python Code

```
# Import Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import signal
from numpy.linalg import inv
from statsmodels.tsa.stattools import adfuller



# 1 - Let consider an AR(2) process as y(t)-0.5y(t-1)-0.2y(t-2)=e(t). Where e(t) is a
WN (0,1).
# a. Using a for loop simulate above process for 1000 samples. Assume all initial
conditions to be zero.

N = 1000
mean= 1
std =1
```

```
e = std * np.random.randn(N) + mean
y = np.zeros((len(e)))
y = np.zeros((len(e)))
for i in range(len(e)):
    if i == 0:
        y[i] = e[i]
    elif i == 1:
        y[i] = 0.5 *y[i-1] + e[i]
    else:
        y[i] = 0.5 *y[i-1] + 0.2*y[i-2]+e[i]

# b. Plot the y(t) with respect to number of samples.

plt.figure()
plt.plot(y)
plt.title(" AR(2) process y(t)-0.5y(t-1)-0.2y(t-2)=e(t)")
plt.ylabel("y Value")
plt.xlabel('Sample')
plt.show()

# c. Using the python code, developed in previous labs, calculate autocorrelations
# for 20 lags and plot them versus number of lags. Write down your observation about
the ACF of above process.
def autocorrelation(y):
    y_bar = np.sum(y) / len(y)
    acf_list = []
    numerator = 0
    denomintator = 0
    n = len(y)
    for i in range(len(y)):
        denomintator += np.square(y[i] - y_bar)
    p = 0
    while p < n:
        r = n - p
        for j, z in zip(range(p, n), range(0, r)):
            numerator += ((y[j] - y_bar) * (y[z] - y_bar))
        quotient = numerator / denomintator
        acf_list.append(quotient)
        numerator = 0
        p = p + 1
    reverselist = acf_list[::-1]
    fulllist = reverselist+acf_list[1:]
    rangey = np.arange((-len(y)+1),(len(y)))
    df=pd.DataFrame({'ACFValue':fulllist}, index=rangey)
    print(df)
    return df

ACF_Forecast_Errors = autocorrelation(np.array(y[0:20]))
plt.stem(ACF_Forecast_Errors.index, ACF_Forecast_Errors.ACFValue)
plt.title("Autocorrelation of AR(2) process y(t)-0.5y(t-1)-0.2y(t-2)=e(t)")
plt.ylabel("Autocorrelation Magnitude")
plt.xlabel("Lag")
plt.show()


# d. Display the first 5 values of y(t) at the console.
print("The first few values of y(t) for Manual AR(2) process:",y[0:5])

# e. Apply the ADF-test and check if this is a stationary process. Explain your
answer.
def ADF_Cal(x):
```

```
    result = adfuller(x)
    print("ADF Statistic: %f" %result[0])
    print("p-value: %f" % result[1])
    print("Critical Values:")
    a= print("ADF Statistic: %f" %result[0])
    b= print("p-value: %f" % result[1])
    for key, value in result[4].items():
        print('\t%s: %.3f' %(key, value))
    return result

result=ADF_Cal(y)


# 2- Using the "scipy" python package and "dlsim" command, simulate the AR(2) process
in question 1

num = [1,0,0]
den = [1, -0.5, -0.2]
system = [num,den,1]

y_new= signal.dlsim(system,e)


# a. Display the first 5 values of y(t) at the console.
print("y(t) values using scipy for AR(2) process:",y_new[1][0:5])


# 3- Write the AR(2) process in question 1, as multiple regression model and using the
least square estimate (LSE),
#  estimate the true parameters a_1 and  a_2 (-0.5, -0.2).
#  Display the estimated parameters values at the console.
#  What is the affect of additional samples on the accuracy of the estimate?
#  Justify the answer by running your code for 5000 and 10000 data samples.

X = np.zeros((N,2))
print(X.shape)
for i in range(N-1):
    X[i]=[-y[i+1],-y[i]]
X=X[:-2,:]
y= y[2:]
print(X)
print(X.shape)
print(y.shape)
X_transpose = X.transpose()
Y = y
B_hat = inv(X_transpose.dot(X))
B_hat = B_hat.dot(X.transpose())
B_hat = B_hat.dot(Y)
print("Estimated parameters using LSE with 1000 samples:", B_hat)

N = 5000
mean= 1
std =1
e = std * np.random.randn(N) + mean
y = np.zeros((len(e)))
y = np.zeros((len(e)))
for i in range(len(e)):
    if i == 0:
        y[i] = e[i]
    elif i == 1:
        y[i] = 0.5 *y[i-1] + e[i]
```

```
    else:
        y[i] = 0.5 *y[i-1] + 0.2*y[i-2]+e[i]
X = np.zeros((N,2))
print(X.shape)
for i in range(N-1):
    X[i]=[-y[i+1],-y[i]]
X=X[:-2,:]
y= y[2:]
print(X)
print(X.shape)
print(y.shape)
X_transpose = X.transpose()
Y = y
B_hat = inv(X_transpose.dot(X))
B_hat = B_hat.dot(X.transpose())
B_hat = B_hat.dot(Y)
print("Estimated parameters using LSE with 5000 samples:", B_hat)

N = 10000
mean= 1
std =1
e = std * np.random.randn(N) + mean
y = np.zeros((len(e)))
y = np.zeros((len(e)))
for i in range(len(e)):
    if i == 0:
        y[i] = e[i]
    elif i == 1:
        y[i] = 0.5 *y[i-1] + e[i]
    else:
        y[i] = 0.5 *y[i-1] + 0.2*y[i-2]+e[i]
X = np.zeros((N,2))
print(X.shape)
for i in range(N-1):
    X[i]=[-y[i+1],-y[i]]
X=X[:-2,:]
y= y[2:]
print(X)
print(X.shape)
print(y.shape)
X_transpose = X.transpose()
Y = y
B_hat = inv(X_transpose.dot(X))
B_hat = B_hat.dot(X.transpose())
B_hat = B_hat.dot(Y)
print("Estimated parameters using LSE with 10000 samples:", B_hat)

# 4- Generalized your code in the previous question, such when the code runs it asks a
user the following questions:
# a. Enter number of samples :
# b. Enter the order # of the AR process:
# c. Enter the corresponding parameters of AR process :
# Your code should simulate the AR process based on the entered information (a, b, c)
# and estimate the AR parameters accordingly.
# The estimated parameters must be close to the entered numbers in part c.
# Display the estimated parameters and the true values at the console

def generalized_AR_process():
    samples=int(input('Enter number of samples '))
    process = int(input('Enter # of the AR process  '))
    true = input('Enter the corresponding parameters of the AR process  ')
```

```
    userList = true.split()
    list1=[]
    for i in userList:
        ele=float(i)
        list1.append(ele)

    #simulate AR Process
    N = samples
    mean = 1
    std = 1

    # e = std * np.random.randn(N) + mean
    # y = np.zeros((len(e)))
    # y = np.zeros((len(e)))
    # for j in range(len(N)):
    #     for i in range (len(e)):
    #         y[j+N+1]=list1[j]*y[]
    #     order=process
    #     while j >= order:
    #         y[i] = userList[order-1] * y[i - 1] + 0.2 * y[i - 2] + e[i]
    #     for i in range(len(e)):
    #         if i == 0:
    #             y[i] = e[i]
    #         elif i == 1:
    #             y[i] = 0.5 * y[i - 1] + e[i]
    #         else:
    #             y[i] = 0.5 * y[i - 1] + 0.2 * y[i - 2] + e[i]



    print(samples)
    print(true)
    print(process)
    print(type(true))


generalized_AR_process()
# 5- Let consider an MA(2) process as
# y(t)=e(t)+0.5e(t-1)+0.2e(t-2)
# a.   Using a for loop simulate above process for 1000 samples. Assume all initial
conditions to be zero.

N = 1000
mean= 1
std =1

e = std * np.random.randn(N) + mean
y = np.zeros((len(e)))
y = np.zeros((len(e)))
for i in range(len(e)):
    if i == 0:
        y[i] = e[i]
    elif i == 1:
        y[i] = 0.5 *e[i-1] + e[i]
    else:
        y[i] = 0.5 *e[i-1] + 0.2*e[i-2]+e[i]



# b. Plot the y(t) with respect to number of samples.
plt.figure()
```

```
plt.plot(y)
plt.title(" MA(2) process y(t)=e(t)+0.5e(t-1)+0.2e(t-2)")
plt.ylabel("y Value")
plt.xlabel('Sample')
plt.show()

# c. Using the python code, developed in previous labs, calculate autocorrelations
# for 20 lags and plot them versus number of lags.
# Write down your observation about the ACF of above process.

ACF_Forecast_Errors = autocorrelation(np.array(y[0:20]))
plt.stem(ACF_Forecast_Errors.index, ACF_Forecast_Errors.ACFValue)
plt.title("Autocorrelation of MA(2) process y(t)=e(t)+0.5e(t-1)+0.2e(t-2)")
plt.ylabel("Autocorrelation Magnitude")
plt.xlabel("Lag")
plt.show()



# d. Increase the data samples to 10000 and then 100000.
# Observe the ACF. Write down your observation.
# There is a difference between the ACF of an AR process and MA process.
# What is the main difference?
N = 10000
mean= 1
std =1
e = std * np.random.randn(N) + mean
y = np.zeros((len(e)))
y = np.zeros((len(e)))
for i in range(len(e)):
    if i == 0:
        y[i] = e[i]
    elif i == 1:
        y[i] = 0.5 *e[i-1] + e[i]
    else:
        y[i] = 0.5 *e[i-1] + 0.2*e[i-2]+e[i]
ACF_Forecast_Errors = autocorrelation(np.array(y[0:20]))
plt.stem(ACF_Forecast_Errors.index, ACF_Forecast_Errors.ACFValue)
plt.title("Autocorrelation of MA(2) process y(t)=e(t)+0.5e(t-1)+0.2e(t-2) with
N=10,000")
plt.ylabel("Autocorrelation Magnitude")
plt.xlabel("Lag")
plt.show()

N = 100000
mean= 1
std =1
e = std * np.random.randn(N) + mean
y = np.zeros((len(e)))
y = np.zeros((len(e)))
for i in range(len(e)):
    if i == 0:
        y[i] = e[i]
    elif i == 1:
        y[i] = 0.5 *e[i-1] + e[i]
    else:
        y[i] = 0.5 *e[i-1] + 0.2*e[i-2]+e[i]
ACF_Forecast_Errors = autocorrelation(np.array(y[0:20]))
plt.stem(ACF_Forecast_Errors.index, ACF_Forecast_Errors.ACFValue)
plt.title("Autocorrelation of MA(2) process y(t)=e(t)+0.5e(t-1)+0.2e(t-2) with
N=100,000")
```

```
plt.ylabel("Autocorrelation Magnitude")
plt.xlabel("Lag")
plt.show()

# e. Display the first 5 values of y(t) at the console.
print("First 5 values of y(t) for manual MA(2) process:",y[0:5])

# f. Using various techniques learned in previous labs, Is this stationary process?
Explain your answer.

result=ADF_Cal(y)

# 6- Using the “scipy” python package and “dlsim” command, simulate the MA(2) process
in question 5.
# a. Display the first 5 values of y(t) at the console.

den = [1,0,0]
num = [1, 0.5, 0.2]
system = [num,den,1]

y_new= signal.dlsim(system,e)
print("First 5 values for y(t) for MA(2) process using scipy: ",y_new[1][0:5])

# b. Show that your answer to the previous part is identical to the answer in part e)
of the previous question.
```