The George Washington University

Laboratory Seven:

Differences in Autocorrelation Functions for Autoregressive Models

Fernando Zambrano

DATS 6450: Multivariate Modeling

Dr. Reza Jafari

25 March 2020

Abstract:

An autoregressive model predicts the future behavior of a series based upon its past behavior. There are three models that are discussed the autoregressive, AR(na), moving average, MA(nb), and Autoregressive Moving Average, ARMA(na,nb). Given a certain equation, the theoretical mean and variance of the three methods are calculated by hand. The theoretical results are compared with the results from implementing the equation and models in Python with varying sample sizes. Furthermore, the autocorrelation function (ACF) plots for each method are compared with varying sample sizes.

Introduction:

Forecasting models can be simple and complex. One popular method is the use of regression, which is trying to predict the value of one variable based upon the changes and values of another variable. This can often lead down a deep rabbit hole of choosing and testing several variables in search of the perfect model. However, what about using the variable in question to predict itself? Can it offer any important insight or significant reliability to predict its future values? These questions propagate the autoregressive model which is a regression model that uses the change in past values of a variable to forecast its future values. For short-term forecasts the AR(na) and MA(nb) models are the best. For long-term forecasts the ARMA(na,nb) is better suited.

Methods & Theory:

The autoregressive models are similar to multivariable linear regression, where the predictors are lagged versions of the series with their own coefficients. This model can have up to *na* predictors or coefficients which depends on the number of lags taken into the model. This also determines the order of the model. An AR(2) model will take into account the series lagged at t -1 and t -2 steps. This concept often used to refer to AR models as "AR (na) models." One of the principle requirements of AR models is that they require stationarity – no trend or seasonality. There needs to be a constant level of variance and autocorrelation throughout the entire series.

A powerful characteristic of AR models is that they have "long memory." Each observation at k lags before have an effect on the current observation, and on each succeeding observation, at either large or small quantiles. Hence the information from the very beginning is stored through the entire data. The effects of the first values have little effect on the current observations IF the coefficient of the first observation is less than 1.

The formula for AR(na) is below.

$$y(t) + a_1 y(t-1) + a_2 y(t-2) + \ldots + a_{n_a} y(t-n_a) = \epsilon(t)$$

where $y(t)$ is the variable of interest and $\epsilon(t)$ is white noise $(WN \sim (0, \sigma_\epsilon^2)$.

Since AR models are implementing regression, the predictors or coefficients for each lagged series can be estimated using Least Square Estimate (LSE). The optimal coefficients for a certain AR(na) model can be solved through matrix algebra through the following formula:

Moving Average models or MA models are similar to AR models, but rather than making forecasts with pervious based on past values it uses the pervious forecaster errors. The order of the MA model depends on the lags between forecast errors, or q. Hence moving average models are referred to as MA(nb) models. The equation for MA(nb) models is below, followed by its backshift operator formula.

$$y(t) = \epsilon(t) + b_1\epsilon(t-1) + b_2\epsilon(t-2) + \ldots + b_{n_b}\epsilon(t-n_b)$$

where $\epsilon(t)$ is white noise $WN \sim (0, \sigma_\epsilon^2)$. This is called a moving average model of order n_b , $\boldsymbol{MA(n_b)}$.

The Autoregressive Moving Average (ARMA) is a combination of AR(na) and MA(nb) models. The ARMA model is one the most popular methods used for time stationary time series analysis since it offers the minimum number of parameters to forecast the unknown. The equation for the ARMA(na,nb) model is below.

$$y(t) + a_1y(t-1) + a_2y(t-2) + \ldots + a_{n_a}y(t-n_a) = \epsilon(t) + b_1\epsilon(t-1) + b_2\epsilon(t-2) + \ldots + b_{n_b}\epsilon(t-n_b)$$

Implementation and Results:

The theoretical mean and variance will be calculated for the following three autoregressive processes'.

AR(2) process as : $y(t) - 0.5y(t-1) - 0.2y(t-2) = e(t)$

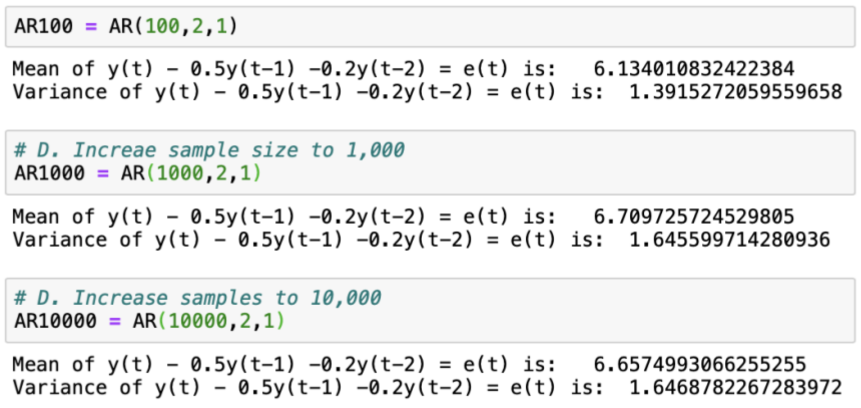
MA(2) process: $y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2)$

ARMA(2,2) process as $y(t) - 0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2)$

Where e(t) is the white noise WN (2,1).

The theoretical calculations for the mean and variance for this process are found in the appendix before the Python code.

Implementing the same processes in Python, starting with AR(2) and plotting the corresponding ACF plot at 20,40, and 80 lags.



```
AR100 = AR(100,2,1)
```

```
Mean of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) is:   6.134010832422384
Variance of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) is:  1.3915272059559658
```

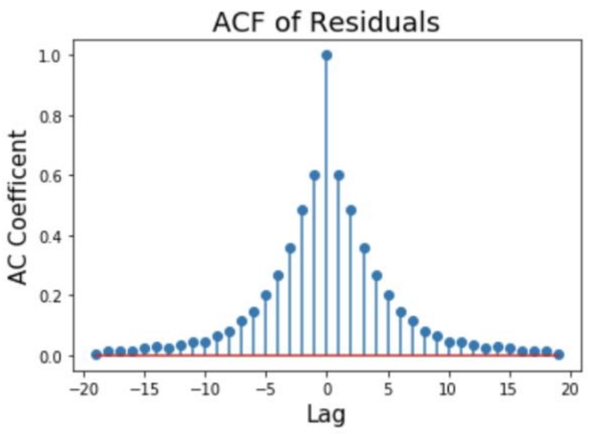
```
# D. Increae sample size to 1,000
AR1000 = AR(1000,2,1)
```

```
Mean of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) is:   6.709725724529805
Variance of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) is:  1.645599714280936
```

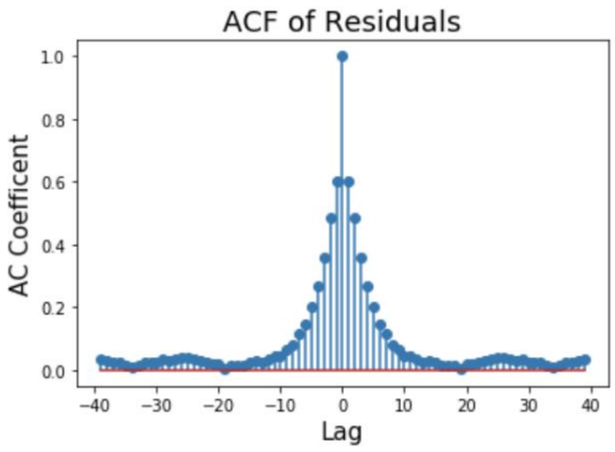
```
# D. Increase samples to 10,000
AR10000 = AR(10000,2,1)
```

```
Mean of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) is:   6.6574993066255255
Variance of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) is:  1.6468782267283972
```

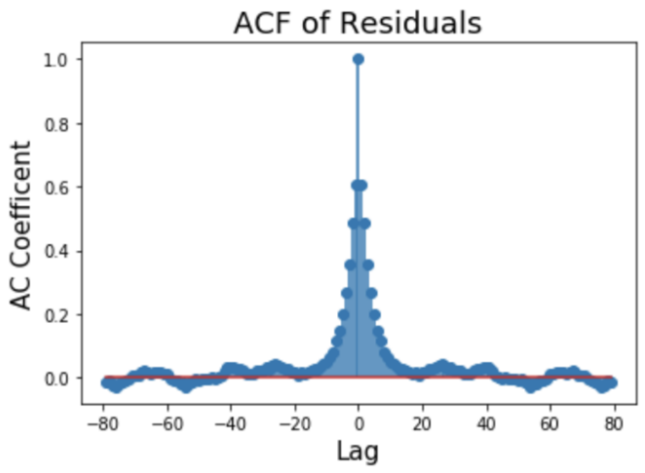
AR(2) with 20 Lags



AR(2) with 40 Lags

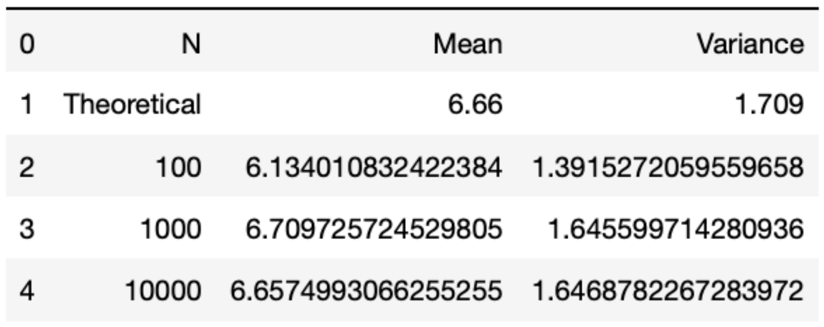


AR(2) with 80 Lags



The ACF of AR(2) shows that autocorrelation has slow decay with each lag. The ACF eventually oscillates from positive to negative around zero after 20 lags.

The Table below compiles the mean and variance for the AR(2) model to compare to the theoretical results.



0	N	Mean	Variance
1	Theoretical	6.66	1.709
2	100	6.134010832422384	1.3915272059559658
3	1000	6.709725724529805	1.645599714280936
4	10000	6.6574993066255255	1.6468782267283972

Comparing the theoretical and empirical results its clear there will be some error between the two. However, as the number of samples increases from 100 to 10,000 the accuracy tends to increase.

Implementing the same processes in Python, starting with MA(2) and plotting the corresponding ACF plot at 20,40, and 80 lags.

```
MA100 = MA(100,2,1)
```

```
Mean of y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2) is :   2.827382705749165
Variance of y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2)is:  0.9264701682001014
```

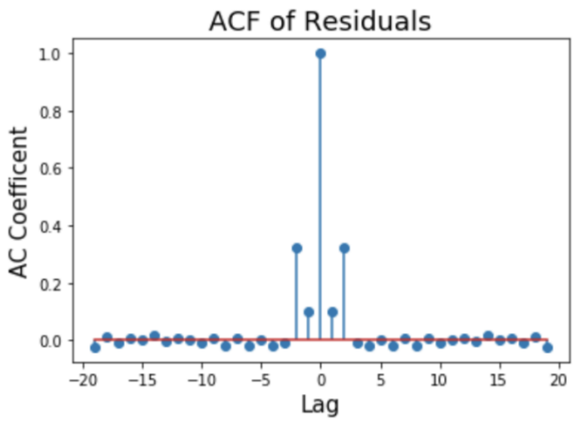
```
MA1000 = MA(1000,2,1)
```

```
Mean of y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2) is :   3.0271402639387235
Variance of y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2)is:  1.11936484907547
```

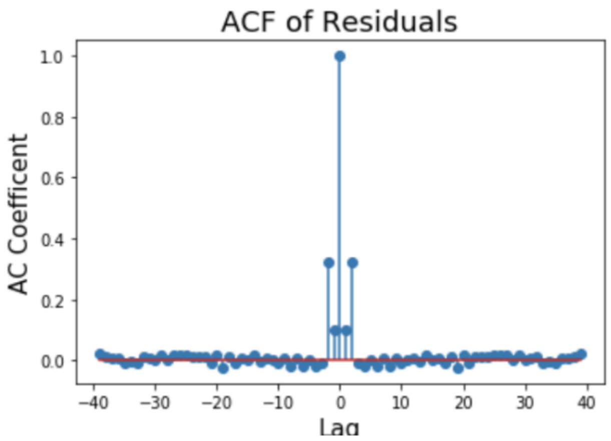
```
MA10000 = MA(10000,2,1)
```

```
Mean of y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2) is :   2.996563974901763
Variance of y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2)is:  1.1608904796459285
```

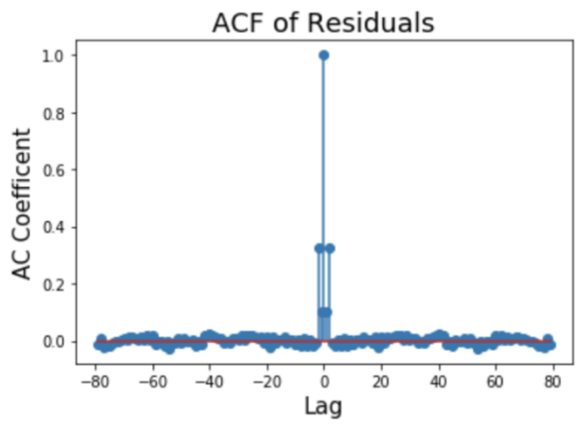
MA(2) with 20 Lags



MA(2) with 40 Lags



MA(2) with 80 Lags



The ACF of MA(2) shows that autocorrelation has a rapidly decaying autocorrelation with each lag up until the lag equal to the order of the process. After the lags equal to the order the ACF becomes zero.

The Table below compiles the mean and variance for the MA(2) model to compare to the theoretical results.

0	N	Mean	Variance
1	Theoretical	3	1.17
2	100	2.827382705749165	0.9264701682001014
3	1000	3.0271402639387235	1.11936484907547
4	10000	2.996563974901763	1.1608904796459285

Implementing the same processes in Python, starting with ARMA(2,2) and plotting the corresponding ACF plot at 20,40, and 80 lags.

```
ARMA100 = ARMA(100,2,1)
```

```
Mean of y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2) is
:   9.144764625198754
Variance of y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2)
is :  2.8058290517083733
```

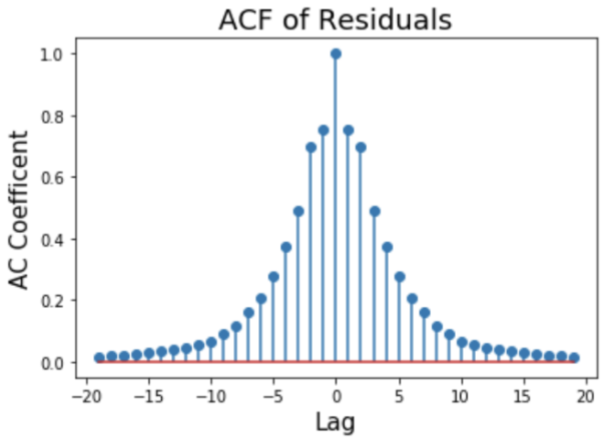
```
ARMA1000 = ARMA(1000,2,1)
```

```
Mean of y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2) is
:   10.058327428924692
Variance of y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2)
is :  2.939506967709488
```

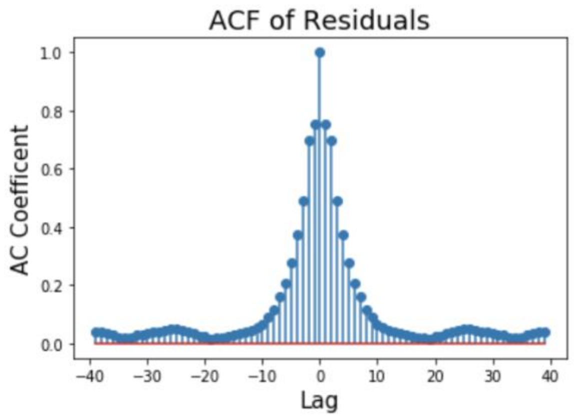
```
ARMA10000 = ARMA(10000,2,1)
```

```
Mean of y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2) is
:   9.985646228306974
Variance of y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2)
is :  2.8529705831441023
```

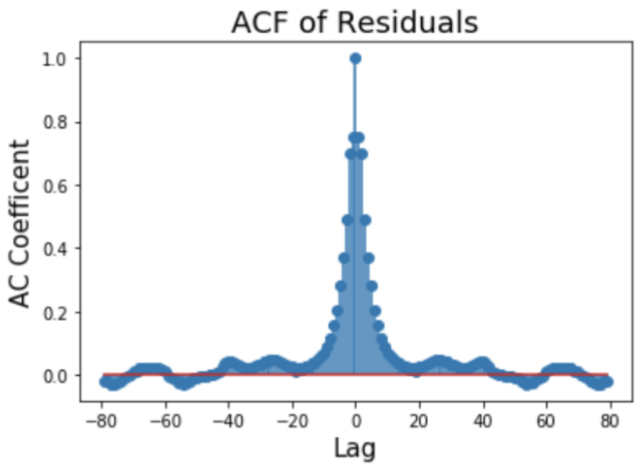
ARMA(2,2) with 20 Lags



ARMA(2,2) with 40 Lags

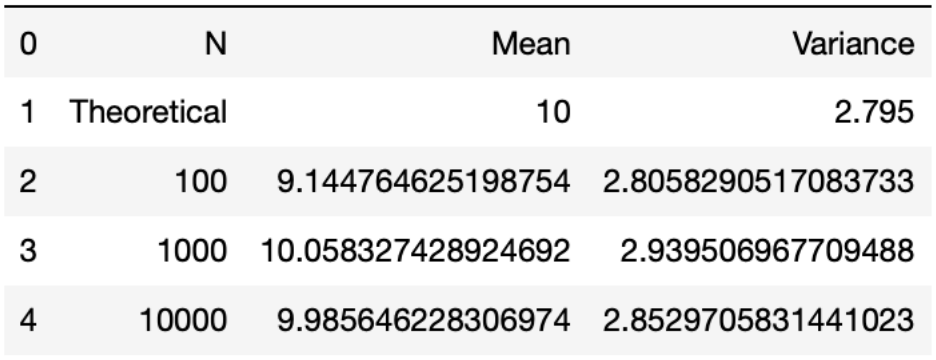


ARMA(2,2) with 80 Lags



The ACF of ARMA(2,2) shows that autocorrelation has a slow decaying autocorrelation similar to that of the AR(2) process, but slower. After 40 lags the ACF oscillates around zero.

The Table below compiles the mean and variance for the ARMA(2,2) model to compare to the theoretical results.

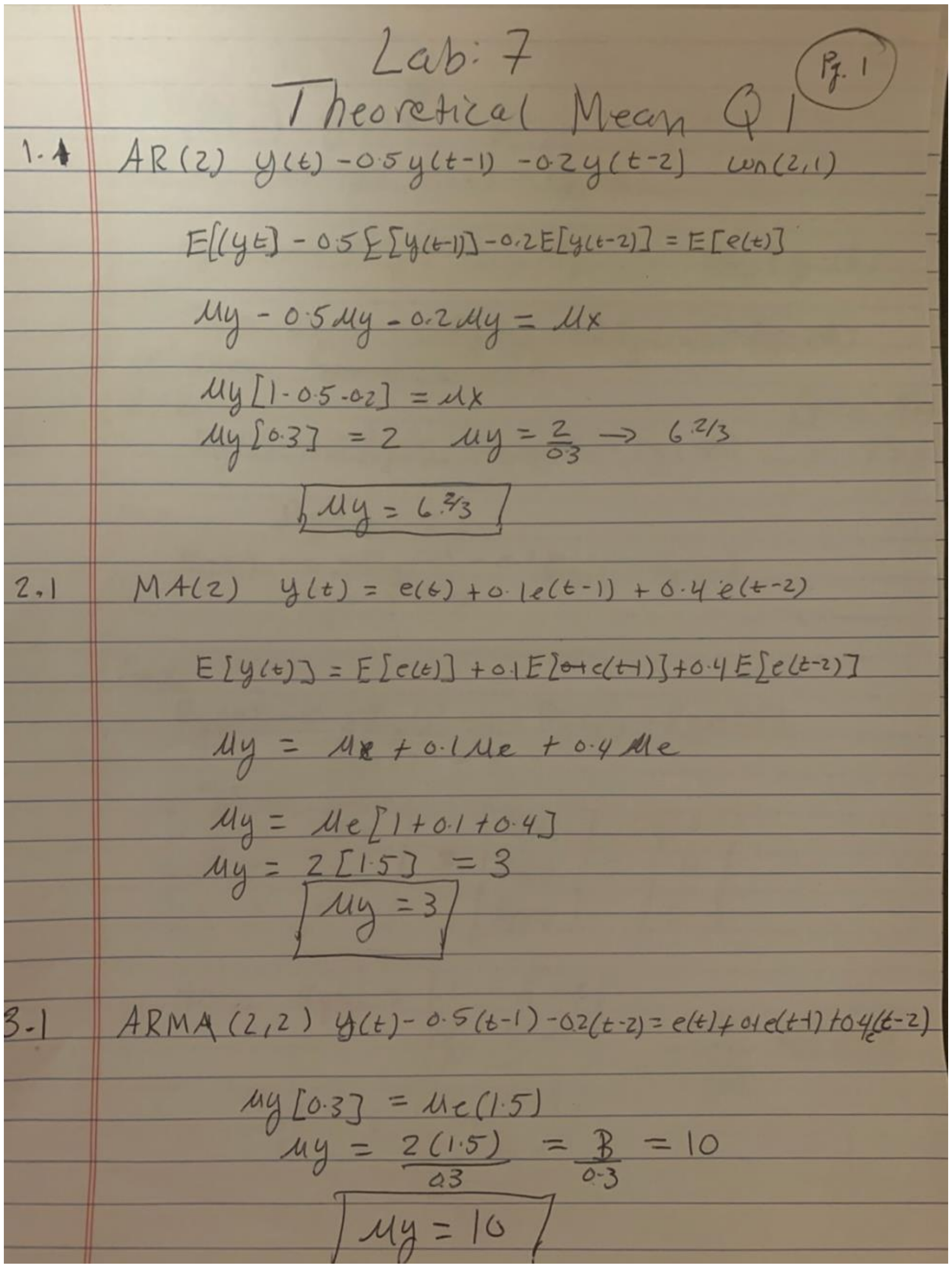


0	N	Mean	Variance
1	Theoretical	10	2.795
2	100	9.144764625198754	2.8058290517083733
3	1000	10.058327428924692	2.939506967709488
4	10000	9.985646228306974	2.8529705831441023

Conclusion:

Overall, there is clear difference between the MA(2) ACF and the ACF plots of AR(2) and ARMA(2,2). The clearest characteristic that sets MA(2) from the other processes is that after the lag equal to the order of the process, the ACF flattens out to zero. It is much more difficult to see visual differences between the AR(2) and the ARMA(2,2). Beyond the differences in ACF plots, all three process show an increase in their mean and variance accuracy as the number of samples increases with the theoretical values as the standard.

Appendix:



Lab: 7

Theoretical Mean Q1

Pg. 1

1.1 AR(2) $y(t) - 0.5y(t-1) - 0.2y(t-2)$ $wn(2,1)$

$E[(yt] - 0.5E[y(t-1)] - 0.2E[y(t-2)] = E[e(t)]$

$\mu y - 0.5\mu y - 0.2\mu y = \mu x$

$\mu y[1 - 0.5 - 0.2] = \mu x$

$\mu y[0.3] = 2 \quad \mu y = \frac{2}{0.3} \rightarrow 6^{2/3}$

$\boxed{\mu y = 6^{2/3}}$

2.1 MA(2) $y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2)$

$E[y(t)] = E[e(t)] + 0.1E[\cancel{0.1}e(t-1)] + 0.4E[e(t-2)]$

$\mu y = \mu \cancel{x} + 0.1\mu e + 0.4\mu e$

$\mu y = \mu e[1 + 0.1 + 0.4]$

$\mu y = 2[1.5] = 3$

$\boxed{\mu y = 3}$

3.1 ARMA(2,2) $y(t) - 0.5(t-1) - 0.2(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2)$

$\mu y[0.3] = \mu e(1.5)$

$\mu y = \frac{2(1.5)}{0.3} = \frac{3}{0.3} = 10$

$\boxed{\mu y = 10}$

Pg 2

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

Variance $y(t) = R_y(0)$

$R_y(\tau) - 0.5R_y(\tau-1) - 0.2R_y(\tau-2) = R_{ye}(\tau)$

$\tau = 0$ $R_y(0) - 0.5R_y(-1) - 0.2R_y(-2) = R_{ye}(0)$

* Impose symetrical ACF

$\tau = 0$

$R_y(0) - 0.5R_y(1) - 0.2R_y(2) = R_{ye}(0)$ $\begin{cases} g(0)\sigma_e^2 ; \tau \le 0 \\ 0 ; \tau > 0 \end{cases}$

$\tau = 1$ $R_{ye}(0) = 1$

$R_y(1) - 0.5R_y(0) - 0.2R_y(1) = R_{ye}(1)$

$R_{ye}(1) = 0$

$\tau = 2$

$R_y(2) - 0.5R_y(1) - 0.2R_y(0) = R_{ye}(2)$

$R_{ye}(2) = 0$

$$\begin{array}{ccc} R_y(0) & R_y(1) & R_y(2) \\ \end{array}$$
$$\begin{bmatrix} 1 & -0.5 & -0.2 \\ -.5 & .8 & 0 \\ -.2 & -.5 & 1 \end{bmatrix} * \begin{bmatrix} R_y(0) \\ R_y(1) \\ R_y(2) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

~~det~~ $R_y(0) = \dfrac{\begin{vmatrix} 1 & -.5 & -.2 \\ 0 & .8 & 0 \\ 0 & -.5 & 1 \end{vmatrix}}{\begin{vmatrix} 1 & -.5 & -.2 \\ -.5 & .8 & 0 \\ -.2 & -.5 & 1 \end{vmatrix}} \quad \dfrac{0.8}{= 0.468} = \boxed{1.709}$

$$R_y(1) = \frac{\begin{vmatrix} 1 & 1 & -.2 \\ -.5 & 0 & 0 \\ -.2 & 0 & 1 \end{vmatrix}}{0.468} = \frac{0.5}{0.468} = \boxed{1.068}$$

$$R_y(2) = \frac{\begin{vmatrix} 1 & -.5 & 1 \\ -.5 & .8 & 0 \\ -.2 & -.5 & 0 \end{vmatrix}}{0.468} = \frac{0.41}{0.468} = \boxed{0.876}$$

Variance $y(t) = R(0) = 1.709$

MA(2) $y(t) = e(t) + 0.1 e(t-1) + 0.4 e(t-2)$

Variance: $y(t) = R_y(0)$

F.E $R_y(\tau) = R_{ye}(\tau) + 0.1 R_{ye}(\tau-1) + 0.4 R_{ye}(\tau-2)$

$$R_{ye}(\tau) = \begin{cases} g(-\tau)\,\sigma_e^2 & : \tau \le 0 \\ 0 & : \tau > 0 \end{cases}$$

$g(t) = \delta(t) + 0.1\,\delta(t-1) + 0.4\,\delta(t-2)$

$t = 0$ $\quad g(0) = \delta(0) + 0.1\delta(-1) + 0.4\delta(-2)$

$= 1 + 0 + 0$

$\boxed{g(0) = 0}$

$t = 1$ $\quad g(1) = \delta(1) + 0.1\delta(0) + 0.4(-1)$

$g(1) = 0 + 0.1 + 0 = \boxed{0.1}$

$t = 2$ $\quad g(2) = \delta(2) + 0.1\delta(1) + 0.4\delta(0)$

$= 0 + 0 + .4$

$\boxed{g(2) = 0.4}$

$R_{ye}(0) = \cancel{1 \times 1 = 1}\ g(0)\sigma_e^2 = 1 \times 1 = 1$

$R_{ye}(-1) = \cancel{0.1 \times 1 = 0.1}\ g(1)\sigma_e^2 = 0.1 \times 1 = 0.1$

$R_{ye}(-2) = g(2)\sigma_e^2 = 0.4 \times 1 = 0.4$

$R_y(0) = R_{ye}(0) + 0.1\, R_{ye}(-1) + 0.4\, R_{ye}(-2)$

$R_y(0) = 1 + 0.1(0.1) + 0.4(0.4) = 1.17$

$\boxed{\text{Variance} = 1.17}$

ARMA (2,2) $y(t) - 0.5y(t-1) - 0.2y(t-2) = e(t) - 0.1e(t-1) - .4e(t-2)$

F.E. $R_y(\tau) - 0.5R_y(\tau-1) - 0.2R_y(\tau-2) = R_{ye}(\tau) + 0.1R_{ye}(\tau-1) + 0.4R_{ye}(\tau-2)$

Implement ACF symetry

$\tau = 0$ $R_y(0) - 0.5R_y(1) - 0.2R_y(2) = R_{ye}(0) + 0.1R_{ye}(-1) + 0.4R_{ye}(-2)$

$$\left\{\begin{array}{l} R_{ye}(0) = g(0)\sigma_e^2 \\ R_{ye}(-1) = g(1)\sigma_e^2 \\ R_{ye}(-2) = g(2)\sigma_e^2 \end{array}\right. \quad \begin{array}{l} e(t) \rightarrow \delta(t) \\ y(t) \rightarrow g(t) \end{array}$$

$t=0$ $g(0) - 0.5\overset{0}{g}(-1) - 0.2\overset{0}{g}(-2) = \overset{1}{\delta}(0) + 0.1\overset{0}{\delta}(-1) + 0.4\overset{0}{\delta}(-2)$

$g(0) = 1$

$t=1$ $g(1) - 0.5\overset{1}{g}(0) - 0.2\overset{0}{g}(-1) = \overset{0}{\delta}(1) + 0.1\overset{1}{\delta}(0) + 0.4\overset{0}{\delta}(-1)$

$g(1) - 0.5 = 0 + 0.1 + 0$

$g(1) = 0.1 + 0.5 = 0.6$

$t=2$ $g(2) - 0.5\overset{0}{g}(1) - 0.2\overset{1}{g}(0) = \overset{0}{\delta}(2) + 0.1\overset{0}{\delta}(1) + 0.4\delta(0)$

$g(2) - 0.2 = 0.4$

$g(2) = 0.6$

$Z=0$ $\quad R_y(0) - 0.5R_y(1) - 0.2R_y(2) = R_{ye}(0) + 0.1R_{ye}(-1) + 0.4R_{ye}(-2)$

$R_y(0) - 0.5R_y(1) - 0.2R_y(2) = (1*1) + (0.1*0.6) + (0.4*0.6)$

① $\quad R_y(0) - 0.5R_y(1) - 0.2R_y(2) = 1.3$

$Z=1$ $\quad R_y(1) - 0.5R_y(0) - 0.2R_y(1) = \underset{0}{R_{ye}(1)} + 0.1R_{ye}(0) + 0.4R_{ye}(-1)$

$\qquad\qquad\qquad 0 + 0.1*1 + 0.4*.6$

② $\quad R_y(1) - 0.5R_y(0) - 0.2R_y(1) = ~~1.24~~ \; 0.34$

$Z=2$ $\quad R_y(2) - 0.5R_y(1) - 0.2R_y(0) = \underset{0}{R_{ye}(2)} + 0.1\underset{0}{R_{ye}(1)} + 0.4R_{ye}(0)$

$\qquad\qquad\qquad 0.4*1$

③ $\quad R_y(2) - 0.5R_y(1) - 0.2R_y(0) = 0.4$

$$R_y(0) = \frac{\begin{vmatrix} 1.3 & -.5 & -.2 \\ 0.34 & .8 & 0 \\ 0.4 & -.5 & 1 \end{vmatrix}}{\begin{vmatrix} 1 & -.5 & -.2 \\ -.5 & .8 & 0 \\ -.2 & -.5 & 1 \end{vmatrix}} = \frac{1.308}{0.468} = \boxed{2.795}$$

* $\boxed{\text{Variance}}$ *

$$R_y(1) = \frac{\begin{vmatrix} 1 & 1.3 & -.2 \\ -.5 & 0.34 & 0 \\ -.2 & 0.4 & 1 \end{vmatrix}}{0.468} = \frac{1.0164}{0.468} = \boxed{2.1718}$$

$$R_y(2) = \frac{\begin{vmatrix} 1 & -.5 & 1.3 \\ -.5 & .8 & 0.34 \\ -.2 & -.5 & 0.4 \end{vmatrix}}{0.468} = \frac{0.957}{0.468} = \boxed{2.04487}$$

```
In[157]:



import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import signal
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
import math


# In[158]:



def AR(n,mean,std):
    np.random.seed(42)
    e = std * np.random.randn(n) + mean
    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]
    print("Mean of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) is:  ", y.mean())
    print("Variance of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) is: ", y.var())
    return y,n,y.mean(),y.var()


# In[159]:



def MA(n,mean,std):
    np.random.seed(42)
    e = std * np.random.randn(n) + mean
    y = np.zeros(len(e))
    for i in range(len(e)):
        if i == 0:
            y[i] = e[i]
        elif i == 1:
            y[i] = e[i] + 0.1 * e[i-1]
        else:
            y[i] = e[i] + 0.1 * e[i-1] + 0.4 * e[i-2]
    print("Mean of y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2) is :  ", y.mean())
    print("Variance of y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2)is: ", y.var())
    return y,n,y.mean(),y.var()


# In[160]:


def ARMA(n,mean,std):
    np.random.seed(42)
    e = std * np.random.randn(n) + mean
    y = np.zeros(len(e))
    for i in range(len(e)):
        if i == 0:
            y[i] = e[i]
        elif i == 1:
            y[i] = e[i] + 0.1 * e[i-1] + 0.5 * y[i-1]
        else:
            y[i] = e[i] + 0.1 * e[i-1] + 0.4 * e[i-2] + 0.5 * y[i-1] + 0.2 * y[i-2]
    print("Mean of y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2) is :  ", y.mean())
    print("Variance of y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2) is : ", y.var())
    return y,n,y.mean(),y.var()


# In[161]:


def ACF(data,lags):
    # convert input data into a numpy array
    data = np.array(data)
    # acf will store the autocorreltion coefficent at each lag interval
    # the first datapoint is always 1.0 since anything correlated with itsself is = 1
    acf = [1.0]
    # calculate the mean for the entire dataset
    y_bar = data.mean()
    print("The mean of this dataset is: ",y_bar)
    # subtract the mean from each observation
    yy_bar = data - y_bar
    # clacualte the total variance for the data set
    total_variance = sum(np.square(yy_bar))
    #print("The total variance for this dataset is: ", total_variance)
    # perform a forloop over the dataset with the desired number of lags
    # range is 1,lags b/c the first iteration calcualtes T1
    for i in range(1,lags):
        # first nparray is removing the last element each iteration
        yy_bar_bottom = yy_bar[:-i]
        # second nparray removes the first element each interation
        yy_bar_top = yy_bar[i:]
        # take the sum of of the product of each nparray each iteration
        yy = sum(yy_bar_top * yy_bar_bottom)
        # divide the sum by total variance and append to resulting acf list
        acf.append(yy/total_variance)
    return acf


# In[162]:


def acf_plot(y):
    #y = y.tolist()
    y_rev = y[::-1]
    y_rev.extend(y[1:])
    print(len(y_rev))
    lb = -(math.floor(len(y_rev)/2))
    hb = -(lb-1)
    x = np.array(list(range(lb,hb)))
    figure = plt.stem(x,y_rev,use_line_collection=True)
    plt.xlabel('Lag', fontsize=15)
    plt.ylabel('AC Coefficent', fontsize=15)
    plt.title('ACF of Residuals',fontsize=18)
    plt.show()

    #return y_rev


# In[163]:


# Example of AR(2)
# y(t) - 0.5y(t-1) -0.2y(t-2) = e(t)
# white noise WN (2,1).


# In[164]:


# 1
# A. calculate the theoretical mean and variance of the data


# In[165]:


# B. Create 100 samples of the AR(2) process above
# C. Calculate the experimental mean and variance


# In[166]:


AR100 = AR(100,2,1)
```

```
# In[167]:


# D. Increae sample size to 1,000
AR1000 = AR(1000,2,1)


# In[168]:


# D. Increase samples to 10,000
AR10000 = AR(10000,2,1)


# In[169]:


# E. Create table to store resutls
# F. Add resutls from 1k and 10k samples to table

header = np.array(["N","Mean","Variance"])
actual = np.array(["Theoretical","6.66","1.709"])
table = np.array([header,actual,AR100[1:],AR1000[1:],AR10000[1:]])
df_AR_results = pd.DataFrame(data=table)
df_AR_results


# In[170]:


# G plot the autocorrelation of y(t)
# 20, 40, and 80 lags


# In[171]:
```

```
acf_20 = ACF(AR10000[0],20)
acf_plot(acf_20)


# In[172]:


acf_40 = ACF(AR10000[0],40)
acf_plot(acf_40)


# In[173]:


acf_80 = ACF(AR10000[0],80)
acf_plot(acf_80)


# In[174]:


# 2 100 samples for MA(2) y(t) = e(t) + 0.1e(t-1) + 0.4e(t-2)


# In[175]:


MA100 = MA(100,2,1)


# In[176]:


MA1000 = MA(1000,2,1)


# In[177]:
```

```
MA10000 = MA(10000,2,1)


# In[178]:


# E. Create table to store resutls
# F. Add resutls from 1k and 10k samples to table

header1 = np.array(["N","Mean","Variance"])
actual1 = np.array(["Theoretical","3","1.17"])
table1 = np.array([header1,actual1,MA100[1:],MA1000[1:],MA10000[1:]])
df_MA_results = pd.DataFrame(data=table1)
df_MA_results


# In[179]:


# G plot the autocorrelation of y(t)
# 20, 40, and 80 lags
acf_201 = ACF(MA10000[0],20)
acf_plot(acf_201)


# In[180]:


acf_401 = ACF(MA10000[0],40)
acf_plot(acf_401)


# In[181]:
```

```
acf_801 = ACF(MA10000[0],80)
acf_plot(acf_801)


# In[182]:


# 3 100 samples for ARMA(2,2) y(t) -0.5y(t-1) - 0.2y(t-2) = e(t) + 0.1e(t-1) + 0.4e(t-2)


# In[183]:


ARMA100 = ARMA(100,2,1)


# In[184]:


ARMA1000 = ARMA(1000,2,1)


# In[185]:


ARMA10000 = ARMA(10000,2,1)


# In[192]:


# E. Create table to store resutls
# F. Add resutls from 1k and 10k samples to table

header2 = np.array(["N","Mean","Variance"])
actual2 = np.array(["Theoretical","10","2.795"])
table2 = np.array([header2,actual2,ARMA100[1:],ARMA1000[1:],ARMA10000[1:]])
```

```
df_ARMA_results = pd.DataFrame(data=table2)
df_ARMA_results


# In[187]:


# G plot the autocorrelation of y(t)
# 20, 40, and 80 lags
acf_202 = ACF(ARMA10000[0],20)
acf_plot(acf_202)


# In[188]:


acf_402 = ACF(ARMA10000[0],40)
acf_plot(acf_402)


# In[189]:


acf_802 = ACF(ARMA10000[0],80)
acf_plot(acf_802)


# In[ ]:
```