The George Washington University

Laboratory Six:

Autoregressive Models for Time Series Analysis

Fernando Zambrano

DATS 6450: Multivariate Modeling

Dr. Reza Jafari

11 March 2020

Abstract:

An autoregressive model predicts the future behavior of a series based upon its past behavior. There are two primary forms, the autoregressive, AR(na), and moving average, MA(nb).These two method are implemented using Python and the SciPy package to show the differences in their autocorrelation functions (ACF), and how the ACF changes 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, or AR model, which is a regression model that uses the change in past values of a variable to forecast its future values. This can be useful for short-term forecasts.

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:

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

where:

$$X = \begin{pmatrix} -y(n_a-1) & -y(n_a-2) & \cdots & -y(0) \\ -y(n_a) & -y(n_a-1) & \cdots & -y(1) \\ \vdots & \vdots & \vdots & \vdots \\ -y(n_a+T-1) & -y(n_a+T-2) & \cdots & -y(T) \end{pmatrix}$$

$$\boldsymbol{Y} = \begin{pmatrix} y(n_a) \\ y(n_a+1) \\ \vdots \\ y(n_a+T) \end{pmatrix} \hat{\boldsymbol{a}} = \begin{pmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_{n_a} \end{pmatrix}$$

Backshift operator notation allows for a simpler notation for a certain AR(na) models which reveal the transfer function of the system. Understanding the transfer function helps code the model using "dlsim" with he SciPy package in Python. Below is the backshift operator notation for AR(na) model.

$$y(t-1) = q^{-1}y(t)$$
$$\vdots$$
$$y(t-n_a) = q^{-n_a}y(t)$$

Using the backshift operator, $AR(n_a)$ model can be written as :

$$\boxed{\frac{y(t)}{\epsilon(t)} = \frac{1}{1 + a_1q^{-1} + a_2q^{-2} + \ldots + a_{n_a}q^{-n_a}} = \frac{1}{A(q)}}$$

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 , $\boldsymbol{MA(n_b)}$.

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

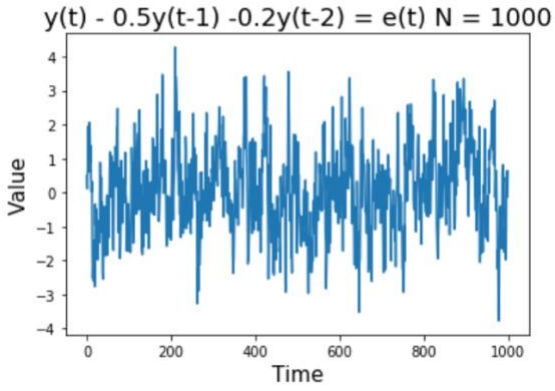
Implementation and Results:

1. Consider an AR(2) process with the following equation:

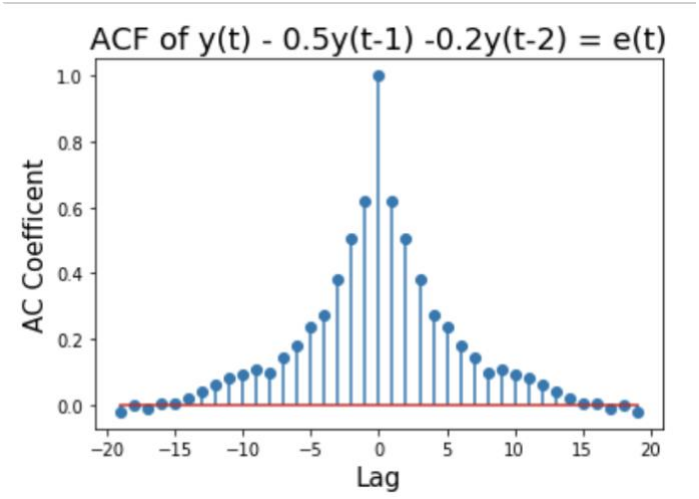
$$y(t) - 0.5y(t-1) - 0.2y(t-2) = e(t)$$

Where $e(t)$ is a WN (0,1).

a. Implement the model using a for-loop in Python with 1000 samples and initial conditions as zero. Plot the data.



b. Plot the ACF of y(t) with 20 lags.



The ACF for the AR(2) model demonstrates an slowing decay in autocorrelation as lags increase. The ACF demonstrates that around the +-10th lag autocorrelation switches from positive to negative. Lags between -5 and +5 show the quickest rate of change in autocorrelation.

c. Show the first 5 values of y(t)

```
First valeus of y(t): [0.49671415 0.11009278 0.80207776 1.94608729 0.89930582]
```

d. Apply and ADF test to check for stationarity. The series is stationary since the p-value for the ADF test is 0, which is less than the critical value threshold of 0.05.

```
ADF Statistic: -11.218917
p-value: 0.000000
Critical Values:
        1%: -3.436919
        5%: -2.864440
        10%: -2.568314
```

2. Using SciPy and dslim command simulate the AR(2) process using the equation in question 1.
    a. The first 5 values in y(t) using the first method

```
First valeus of y(t): [0.49671415 0.11009278 0.80207776 1.94608729 0.89930582]
```

    b. The first 5 values in y(t) using the SciPy method

```
First values of y(t) using dslim: [[0.49671415]
 [0.11009278]
 [0.80207776]
 [1.94608729]
 [0.89930582]]
```

    c. The values are the same regardless of method.
3. Use LSE to estimate the true parameters of the equation, a1 and a2 (-.5,-.2).
    a. Estimated parameters

```
test1 = AR_LSE(1000,2,None)
```

```
a1 and a2:
 [[-0.49506429]
 [-0.20184547]]
```

    b. Effect of increase sample size
        i. 5000 samples

```
test2 = AR_LSE(5000,2,None)
```

```
a1 and a2:
 [[-0.48876427]
 [-0.20459609]]
```

        ii. 100000 samples

```
test3 = AR_LSE(10000,2,None)
```

```
a1 and a2:
 [[-0.48691626]
 [-0.19331823]]
```

c. There does not seem to be any clear and noticeable effect by increasing the sample size, other than a slight decrease in accuracy for parameter estimation.

4. Generalize the code to allow input for sample size, order, and parameters.

a.
```
# Question 4
def AR_gen():
    a = int(input("\nEnter the numbers of samples :"))
    b = int(input("\nEnter the order # of the AR process :"))
    c = np.array([(input("\nEnter corresponding parameters of AR process :"))])

    results = AR_LSE(a,b,c)

    return results
```

b. 5,000 samples

```
test_5000 = AR_gen()
```

```
Enter the numbers of samples :5000

Enter the order # of the AR process :2

Enter corresponding parameters of AR process :0.5,0.2

a1 and a2:
 [[-0.48876427]
 [-0.20459609]]
True Parameters:  ['0.5,0.2']
```

i.

c. 10,000 samples

```
test_10000 = AR_gen()
```

```
Enter the numbers of samples :10000

Enter the order # of the AR process :2

Enter corresponding parameters of AR process :0.5,0.2

a1 and a2:
 [[-0.48691626]
 [-0.19331823]]
True Parameters:  ['0.5,0.2']
```

i.

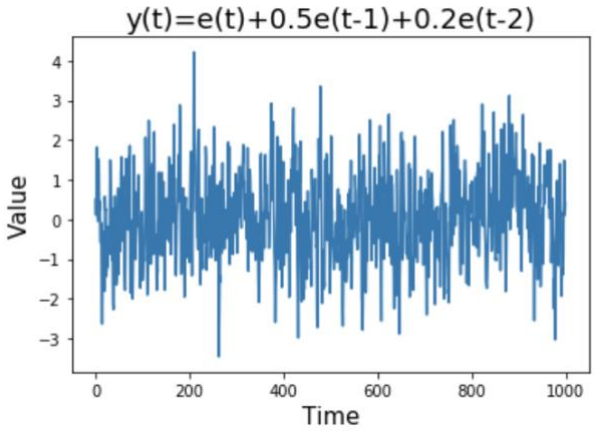
5. Consider the moving average - MA(2) process as :

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

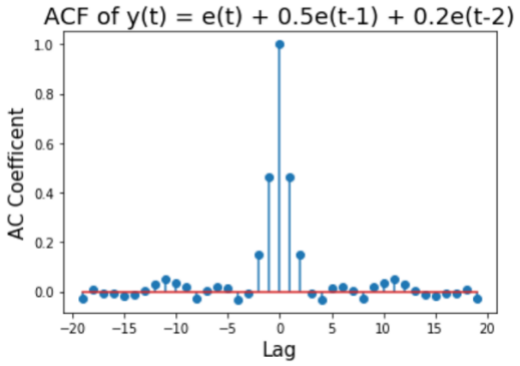
Where $e(t)$ is a WN (0,1).

a. Use a for-loop to simulate the process with 1000 samples with initial conditions set to zero.

b. Plot y(t).

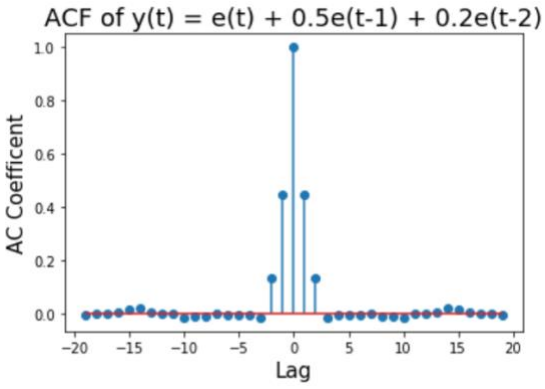


c. Calculate ACF at 20 lags with 1,000 samples and plot.



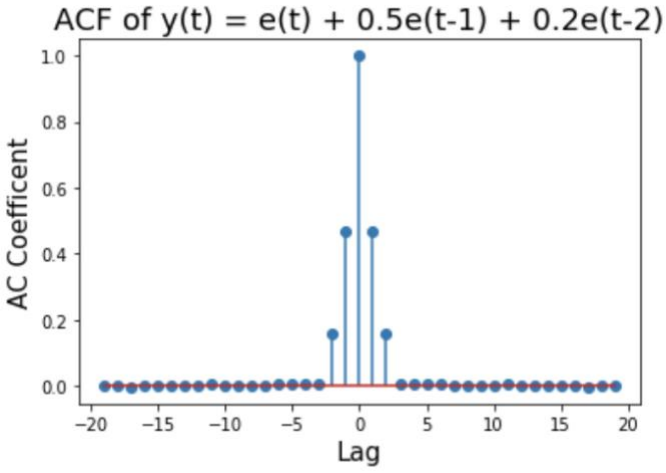
Lags greater than the order(2), the ACF sharply declines to 0, but still oscillates over and under 0 for the subsequent lags.

d. ACF with 20 lags and 10,000 samples.



With the 10,000 samples, the ACF becomes smother after the lags greater than the order (2) compared to the ACF with 1,000 samples.

ACF with 20 lags and 100,000 samples



Again, with the sample increase the ACF is much smoother to the point that after lags are greater than the order (2), the ACF is almost exactly zero. The difference between the ACF's of AR and MA is that the ACF of AR smoothly decays with each lag and may oscillate between positive and negative. On the other hand, MA ACF's decay only up to the lag that is equal to the MA order. Once lags are great than the order the ACF becomes zero.

e. Show the top 5 samples in y(t)

```
First values of y(t): [-1.17817969 -0.15623944  1.09154244  1.39713164  1.68422065]
```

f. Demonstrate the that this dataset is stationary.

Since the p-value from the ADF test is lower than the critical value threshold of 0.5, the dataset is considered as stationary. Furthermore, from a visual analysis there is no obvious trend nor clear variation in the variance of the dataset.

```
ADF Statistic: -127.209612
p-value: 0.000000
Critical Values:
        1%: -3.430415
        5%: -2.861569
        10%: -2.566785
```

6. Using SciPy perform the same process with dlsim on MA(2).
   a. Display the first 5 values of y(t):

```
y_new[1][0:5]
```

```
array([[-1.17817969],
       [-0.15623944],
       [ 1.09154244],
       [ 1.39713164],
       [ 1.68422065]])
```

   b. These are the same values of y(t) as demonstrated with the results in question 5.e.

Conclusion:

There was little effect in parameter estimation by increasing the sample sizes. If anything, the accuracy slightly decreased with increased sample sizes. Unlike parameter estimation, an increase in sample size had a clear effect in the ACF plots. As sample size increases the ACF plots become smoother and more accurate. This effect was clearest in the ACFs of the MA(2) process. There is an obvious difference when visually comparing the ACF of an AR model with that of an MA model. AR models have a smooth decay that can oscillate between positive and negative as lags increase, which contrasts with the sharp drop in MA models once lags surpass the order of the model. There is no difference in calculation and parameter estimations for autoregression between using for-loops and the dlsim method in the SciPy package in Python. However, the dlsim is much easier and quicker to create than coding multiple lines for a for-loop.

Appendix:

```
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


# In[3]:


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[4]:


# np.random.seed(42) fix


# In[5]:


def acf_plot(y):
    #y = y.tolist()
    y_rev = y[::-1]
    y_rev.extend(y[1:])
    print(len(y_rev))
    return y_rev


# In[6]:


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


# In[7]:


def a_co(x,y):
```

```
    #print("XT shape:",x.T.shape)
    #print("X shape:",x.shape)
    x_xt = np.mat(x.T) * np.mat(x)
    #print("\nXT * X :\n",x_xt)
    #print("\nXT * X shape:",x_xt.shape)
    det_x_xt = np.linalg.det(x_xt)
    #print("\nDeterminent:\n",det_x_xt)
    x_xtt = np.linalg.inv(x_xt)
    #print("\nInverse XT * X:\n",x_xtt)
    #print("\nShape Inverse XT: \n", x_xtt.shape)
    xt_x_xt = np.mat(x_xtt) * np.mat(x.T)
    #print("\nInverse XT * XT:\n", xt_x_xt)
    a = np.mat(xt_x_xt) * np.mat(y)
    print("\na1 and a2: \n",a)
    #B0 = float(B[0])
    #B1 = float(B[1])
    #print('The intercept for this linear regression model is: ', B0)
    #print('The slope for this linear regression model is: ', B1)
    #return B0,B1
    return a


# In[8]:


# 1
# example of AR(1)
# y(t) + .5y(t-1) = e(t)

# Example of AR(2)
# y(t) - 0.5y(t-1) -0.2y(t-2) = e(t)


# In[9]:


# 1.a simulate 1000 samples of AR(2)
```

```
n = 1000
mean = 0
std = 1
np.random.seed(42)
e = std * np.random.randn(n) + mean


# In[10]:


# 1a.
# 1.b plot y(t)
# np.zeros? is that because we assume all initial conditions = 0?
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]
plt.figure()
plt.plot(y)
plt.xlabel('Time', fontsize=15)
plt.ylabel('Value', fontsize=15)
plt.title('y(t) - 0.5y(t-1) -0.2y(t-2) = e(t) N = 1000',fontsize=18)

plt.show()


# In[11]:


# 1.c Plot the ACF of y(t) for the first 20 lags
y_acf = ACF(y,20)
y_acf_plt = acf_plot(y_acf)
```

```
# In[12]:


x = np.array(list(range(-19,20)))
figure = plt.stem(x,y_acf_plt,use_line_collection=True)
#plt.figure(figsize=(20,20))
plt.xlabel('Lag', fontsize=15)
plt.ylabel('AC Coefficent', fontsize=15)
plt.title('ACF of y(t) - 0.5y(t-1) -0.2y(t-2) = e(t)',fontsize=18)
plt.show()


# In[16]:


# 1.d Display top 5 values of y(t)
print("First valeus of y(t):",y[0:5])


# In[18]:


# 1.e Apply ADF test to y
adf_y = ADF_Cal(y)


# In[19]:


# Given the ADF test, y is defenitly stationary becaue of the low p-value


# In[20]:


# 2 Repeat
```

```
# scipy process
num = [1,0,0]
den = [1,-0.5,-0.2]
system = (num,den,1)
y_new = signal.dlsim(system,e)


# In[28]:


print("First values of y(t) using dslim:",y_new[1][0:5])


# In[29]:


plt.plot(y_new[1])


# In[30]:


# 3
# Rewrite question 1 as a multiple regression model using LSE


# In[31]:


y[0:5]


# In[32]:


def AR_LSE(N,na,c):
    mean = 0
```

```
    std = 1
    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]

    x0 = np.zeros(len(e) - na)
    x1 = np.zeros(len(e) - na)
    y1 = np.zeros(len(e) - na)
    for i in range(len(e) - na):
        x0[i] = -y[na + i - 1]
        x1[i] = -y[na + i - 2]
        y1[i] = y[i + na]

    X = np.vstack([x0,x1])
    X = X.T
    y1 = np.vstack(y1)

    a = a_co(X,y1)
    if c == None:
        pass
    else:
        print("True Parameters: ",c)
    return a


# In[35]:


test1 = AR_LSE(1000,2,None)
```

 # In[36]:

```
test2 = AR_LSE(5000,2,None)
```

 # In[37]:

```
test3 = AR_LSE(10000,2,None)
```

 # In[40]:

```
# Question 4
def AR_gen():
    a = int(input("\nEnter the numbers of samples e.g 5000 :"))
    b = int(input("\nEnter the order # of the AR process e.g 2 :"))
    c = np.array([(input("\nEnter corresponding parameters of AR process eg 0.5,0.2:"))])

    results = AR_LSE(a,b,c)

    return results
```

 # In[42]:

```
test_5000 = AR_gen()
```

 # In[43]:

```
test_10000 = AR_gen()
```

 # In[44]:

 # Question 5

 # In[45]:

 # Let consider an MA(2) process as

 # y(t)=e(t)+0.5e(t-1)+0.2e(t-2)

 # In[46]:

```
n_ma = 1000
mean = 0
std = 1
np.random.seed(42)
e_ma = std * np.random.randn(n_ma) + mean
y_ma = np.zeros(len(e_ma))
for i in range(len(e_ma)):
    if i == 0:
        y_ma[i] = e_ma[i]
    elif i == 1:
        y_ma[i] = e_ma[i] + 0.5 * e_ma[i-1]
    else:
        y_ma[i] = e_ma[i] + 0.5 * e_ma[i - 1] + 0.2 * e_ma[i - 2]
plt.figure()
plt.xlabel('Time', fontsize=15)
plt.ylabel('Value', fontsize=15)
plt.title('y(t)=e(t)+0.5e(t-1)+0.2e(t-2)',fontsize=18)
plt.plot(y_ma)
```

```
plt.show()
```

```
# In[47]:
```

```
# 5.c Plot the ACF of y(t) for the first 20 lags
yma_acf = ACF(y_ma,20)
yma_acf_plt = acf_plot(yma_acf)
```

```
# In[48]:
```

```
x = np.array(list(range(-19,20)))
figure = plt.stem(x,yma_acf_plt,use_line_collection=True)
#plt.figure(figsize=(20,20))
plt.xlabel('Lag', fontsize=15)
plt.ylabel('AC Coefficent', fontsize=15)
plt.title('ACF of y(t) = e(t) + 0.5e(t-1) + 0.2e(t-2)',fontsize=18)
plt.show()
```

```
# In[49]:
```

```
y_ma[:5]
```

```
# In[50]:
```

```
n_ma = 10000
mean = 0
std = 1
e_ma = std * np.random.randn(n_ma) + mean
y_ma = np.zeros(len(e_ma))
```

```
for i in range(len(e_ma)):
    if i == 0:
        y_ma[i] = e_ma[i]
    elif i == 1:
        y_ma[i] = e_ma[i] + 0.5 * e_ma[i-1]
    else:
        y_ma[i] = e_ma[i] + 0.5 * e_ma[i - 1] + 0.2 * e_ma[i - 2]
plt.figure()
plt.plot(y_ma)

plt.show()
```

```
# In[51]:
```

```
# 5.c Plot the ACF of y(t) for the first 20 lags
yma_acf = ACF(y_ma,20)
yma_acf_plt = acf_plot(yma_acf)
```

```
# In[52]:
```

```
x = np.array(list(range(-19,20)))
figure = plt.stem(x,yma_acf_plt,use_line_collection=True)
#plt.figure(figsize=(20,20))
plt.xlabel('Lag', fontsize=15)
plt.ylabel('AC Coefficent', fontsize=15)
plt.title('ACF of y(t) = e(t) + 0.5e(t-1) + 0.2e(t-2)',fontsize=18)
plt.show()
```

```
# In[53]:
```

```
y_ma[:5]
```

```
# In[54]:


n_ma = 100000
mean = 0
std = 1
e_ma = std * np.random.randn(n_ma) + mean
y_ma = np.zeros(len(e_ma))
for i in range(len(e_ma)):
    if i == 0:
        y_ma[i] = e_ma[i]
    elif i == 1:
        y_ma[i] = e_ma[i] + 0.5 * e_ma[i-1]
    else:
        y_ma[i] = e_ma[i] + 0.5 * e_ma[i - 1] + 0.2 * e_ma[i - 2]
plt.figure()
plt.plot(y_ma)

plt.show()


# In[55]:


# 5.c Plot the ACF of y(t) for the first 20 lags
yma_acf = ACF(y_ma,20)
yma_acf_plt = acf_plot(yma_acf)


# In[56]:


x = np.array(list(range(-19,20)))
figure = plt.stem(x,yma_acf_plt,use_line_collection=True)
#plt.figure(figsize=(20,20))
```

```
plt.xlabel('Lag', fontsize=15)
plt.ylabel('AC Coefficent', fontsize=15)
plt.title('ACF of y(t) = e(t) + 0.5e(t-1) + 0.2e(t-2)',fontsize=18)
plt.show()
```

 # In[60]:

```
print("First values of y(t):",y_ma[:5])
```

 # In[61]:

```
# Perform and ADF test on y(t) with 100,000 samples
ADF_ma = ADF_Cal(y_ma)
```

 # In[62]:

 # 6-

 # Using the “scipy” python package and “dlsim” command, simulate the MA(2) process in question 5.

 # In[63]:

```
# 2 Repeat
# scipy process
num = [1,.5,.2]
den = [1,0,0]
system = (num,den,1)
y_new = signal.dlsim(system,e_ma)
```

 # In[64]:

```
y_new[1][0:5]
```

 # In[ ]:

 # In[ ]:

References: