Lab Course: Distributed Data Analytics
Exercise Sheet 5
Krithika Murugesan - 277537

Exercise 1A : Complex Data : Time Series Forecasting

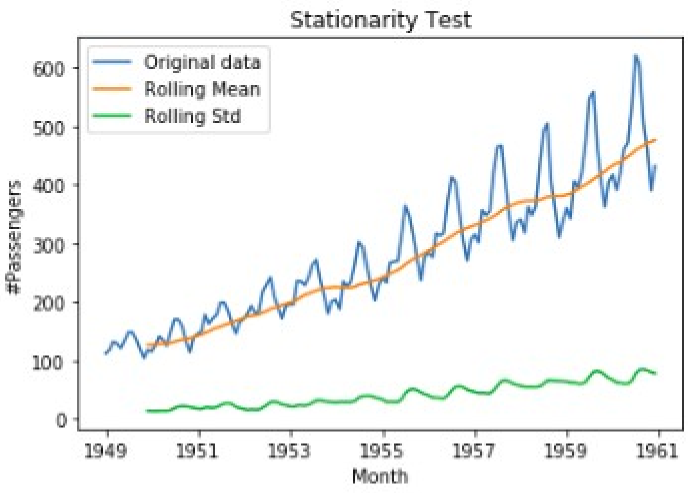
The given datasets are time series data, the values are learned with respect to the time. Such data is best represented in a pandas series. The two datasets are loaded into the pandas series, with their respective time as the index, that is they have only one column of data now with respect to time.

```
# Load Air passengers data
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
airDf = pd.read_csv('AirPassengers.csv',parse_dates=True,header =0)
airDf['Month'] = pd.to_datetime(airDf['Month'])
airDf.set_index('Month', inplace=True)
print("Air passengers : \n\n",airDf.head())

# Load daily minimum temperature data
tempDf = pd.read_csv('daily-minimum-temperatures-in-me.csv',header = 0,parse_dates=True,)
tempDf['Date'] = pd.to_datetime(tempDf['Date'])
tempDf.set_index('Date', inplace=True)
print("\n\nDaily minimum temperatures: \n\n",tempDf.head())
```

```
Air passengers :

             #Passengers
Month
1949-01-01          112
1949-02-01          118
1949-03-01          132
1949-04-01          129
1949-05-01          121


Daily minimum temperatures:

             Temperatures
Date
1981-01-01          20.7
1981-01-02          17.9
1981-01-03          18.8
1981-01-04          14.6
1981-01-05          15.8
```

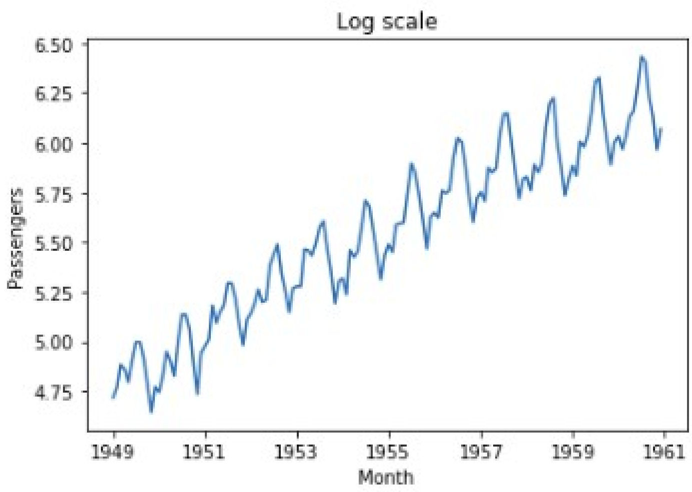
To use ARIMA models on data, it has to be stationary, the statistical properties of the series has to be constant independent of the time. This is to ensure similar behavior of data while prediction which otherwise will cause anomalies. Here we check for a constant mean and standard deviation, plotting the rolling mean and std for the two datasets, the graphs are as follows,

```
def stationaryCheck(dfRecv,x,y):
    #Original series
    df = dfRecv.copy(deep = True)
    plt.plot(df,label = 'Original data')
    plt.xlabel(x)
    plt.ylabel(y)
    plt.title("Stationarity Test")
    df['rollingMean'] = df[y].rolling(window = 12).mean()
    df['rollingStd'] = df[y].rolling(window = 12).std()
    plt.plot(df['rollingMean'],label = 'Rolling Mean')
    plt.plot(df['rollingStd'],label = 'Rolling Std')
    plt.legend()
    plt.show()
```



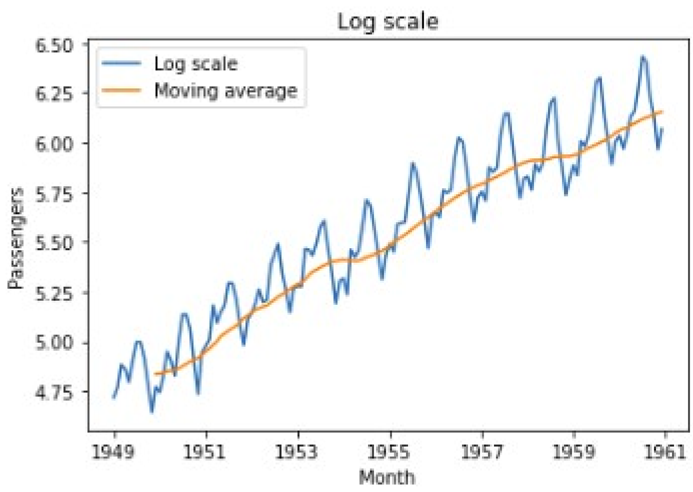
For Air passengers dataset, it can be seen that mean is not constant over time or the data has a trend therefore it is not stationary, in order to make it stationary we have to penalize large values due to the positive trend. To achieve this we use a log transformation

```
#Making the series stationary
airDf = np.log(airDf)
plt.plot(airDf)
plt.xlabel("Month")
plt.ylabel("Passengers")
plt.title("Log scale")
plt.show()
```



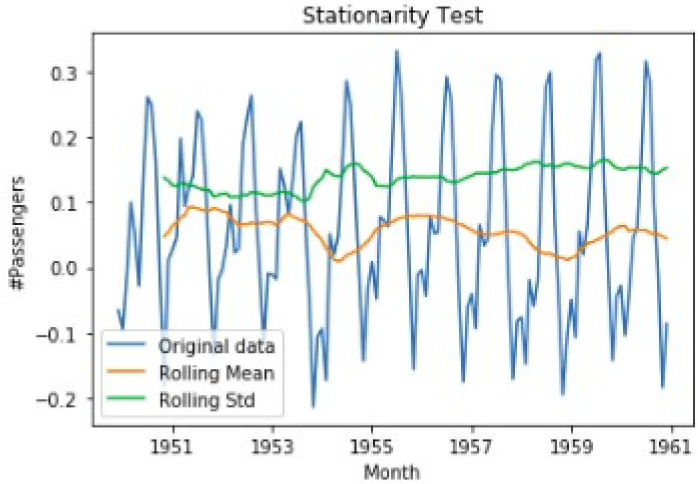
The graph includes a lot of noise, to cancel out this rolling mean is applied to the log transformation for smoothing the curves, this is subtracted from the transformed data to get a stationary series .

```
#Reducing the noise
ma = pd.rolling_mean(airDf,12)
plt.plot(airDf, label = "Log scale")
plt.plot(ma,label = "Moving average")
plt.xlabel("Month")
plt.ylabel("Passengers")
plt.title("Log scale")
plt.legend()
plt.show()
```



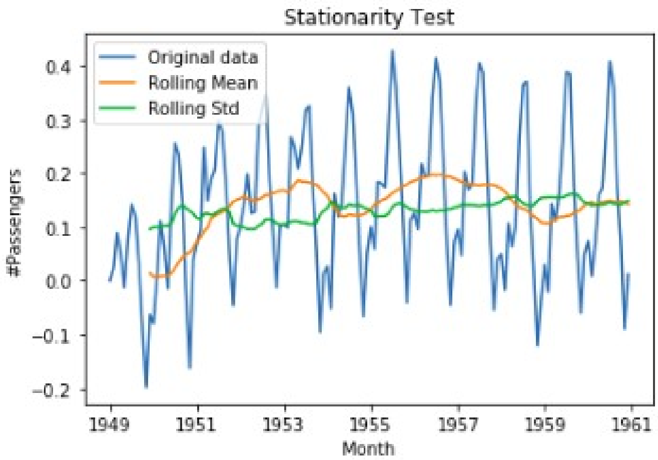
The subtraction leads a few Nan values in the beginning as they don't have a predecessor to subtract from, these values are removed. The resulting series is

```
#Stationarity check
stationaryCheck(airDfNew,"Month","#Passengers")
```

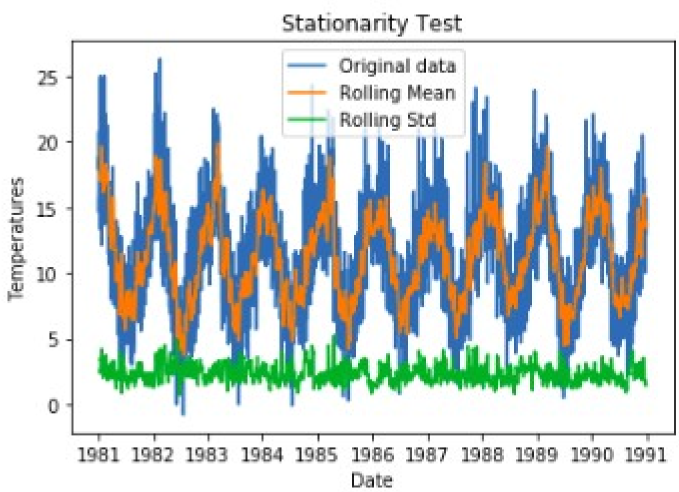


Here we know the time period as 12 months, but for other cases exponentially weighted moving average can be used.

```
#Still not happy!
expA = pd.ewma(airDf,halflife = 12)
#Making it stationary
airDfNew = (airDf - expA)
airDfNew = airDfNew.dropna()
#Stationarity check
stationaryCheck(airDfNew,"Month","#Passengers")
```

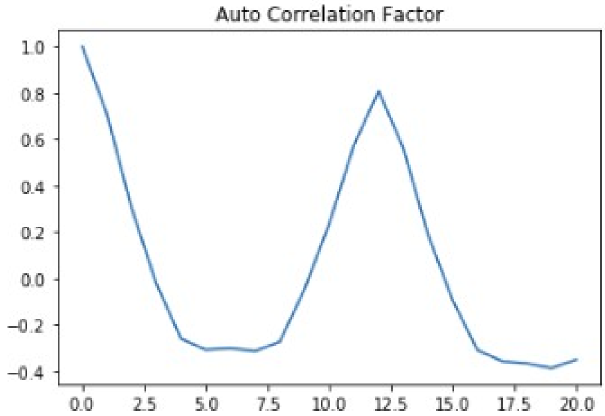


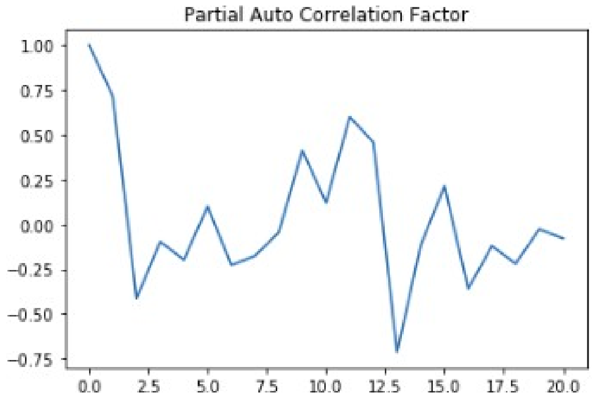
This method also makes the time series stationary. Now we are geared up to forecast! Performing stationary test on the Minimum daily temperatures data, it can be seen that it is almost stationary and no extra effort is needed.



Exercise 1B

ARIMA is Auto Regressive Integrated Moving Average, it is similar to linear regression where predictors depend on parameters (p,q,d). p is the number of Auto-Regressive terms, q is number of moving average terms and d is number of non-seasonal differences. These can be determined from the ACF (Autocorrelation Function) and PACF (Partial Autocorrelation Function) plots that measures correlation between time lagged elements and measure of degree of association respectively.



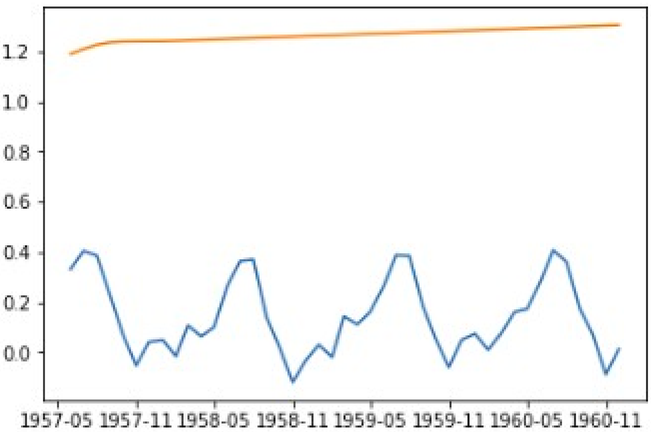


Predicting the values for the dataset, using ARIMA

```
trainIndex = int(np.floor(0.7*airDfNew.shape[0]))
testIndex = int(np.ceil(0.7*airDfNew.shape[0]))
model = ARIMA(airDfNew[0:trainIndex], order = (2,1,1))
model.fit = model.fit(disp = 0)
```

```
forecast = model.fit.forecast(steps = 43)
forecastTrue = np.exp(forecast[0])
```

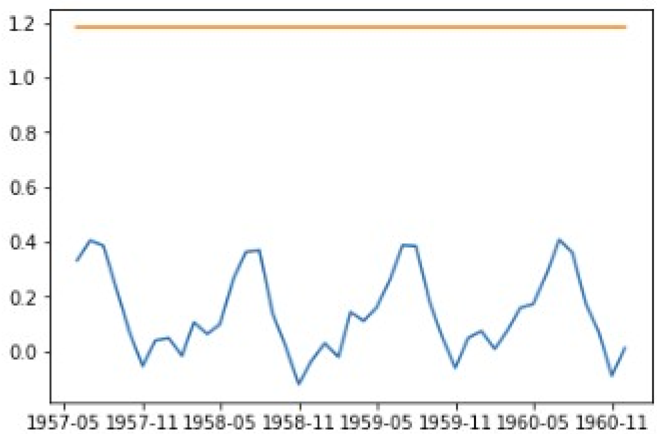
The performance on validation data is as follows, after converting back to original scale from log scale by applying forecast to exponential function



As the data has seasons, including as part of the series by using SARIMA, gives the following results.

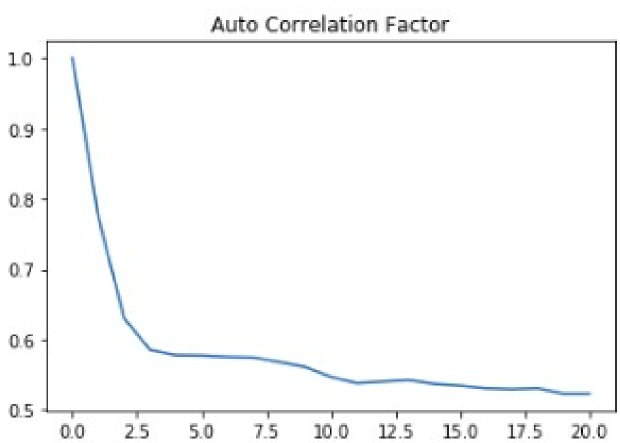
```python
import statsmodels.api as sm

mod = sm.tsa.statespace.SARIMAX(airDfNew[0:trainIndex], trend='n', order=(0,1,0), seasonal_order=(1,1,1,12))
results = mod.fit()
print (results.summary())
```

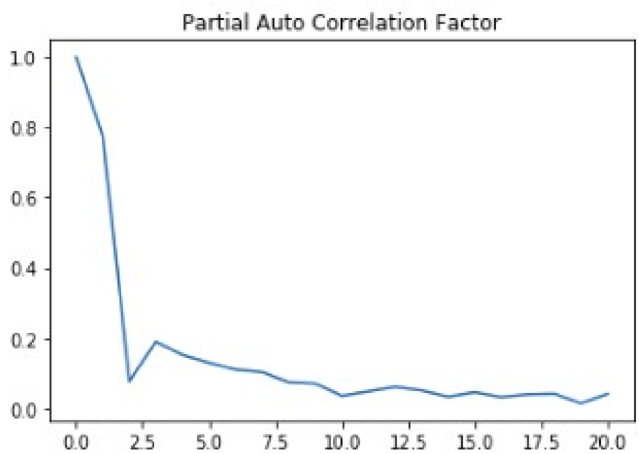
```
                                 Statespace Model Results
==========================================================================================
Dep. Variable:                         #Passengers   No. Observations:                  100
Model:             SARIMAX(0, 1, 0)x(1, 1, 1, 12)   Log Likelihood                 154.809
Date:                             Fri, 01 Jun 2018   AIC                           -303.619
Time:                                     12:53:25   BIC                           -295.803
Sample:                                 01-01-1949   HQIC                          -300.456
                                      - 04-01-1957
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.S.L12      -0.0273      0.197     -0.139      0.890      -0.413       0.359
ma.S.L12      -0.6266      0.246     -2.548      0.011      -1.108      -0.145
sigma2         0.0015      0.000      6.933      0.000       0.001       0.002
===================================================================================
Ljung-Box (Q):                       51.05   Jarque-Bera (JB):                 0.24
Prob(Q):                              0.11   Prob(JB):                         0.89
Heteroskedasticity (H):               0.31   Skew:                            -0.01
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.26
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```



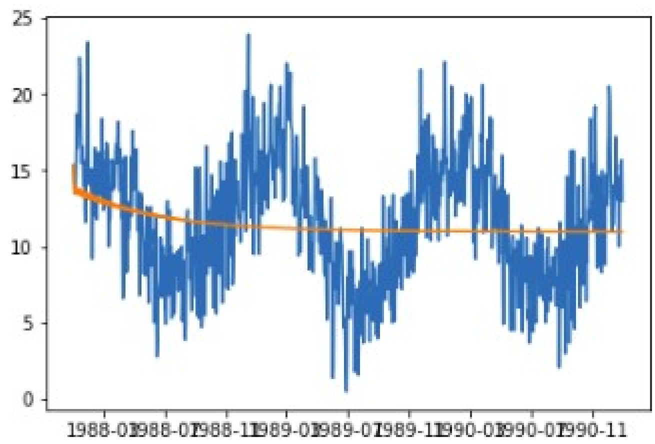
The validation RMSE for ARIMA is `2337.02115873,` while for SARIMA is 2000.35834733. Here SARIMA performs better due to seasonality in the data.

Performing the same ARIMA models for minimum daily temperature the results are as follows,





Using ARIMA the forecast is as follows on validation dataset



The RMSE is 5600.63747469 for this dataset.

Exercise 2 : Logistic Regression on Olivetti faces dataset

Olivetti faces dataset has the images of different people's face and a unique identifier for each person. The task at hand is to classify them using logistic regression in tensorflow. The data is loaded into pandas data frame

```
from sklearn import datasets
data = datasets.fetch_olivetti_faces(data_home=None, shuffle=True, random_state=0)
```

```
print(data.keys())
```

```
dict_keys(['data', 'images', 'target', 'DESCR'])
```

```
import matplotlib.pyplot as plt
for i in range(5):
    face = data.images[i]
    plt.subplot(1, 10, i + 1)
    plt.imshow(face.reshape((64, 64)), cmap='gray')
    plt.axis('off')
plt.show()
```

Then, they are split into test and train data:

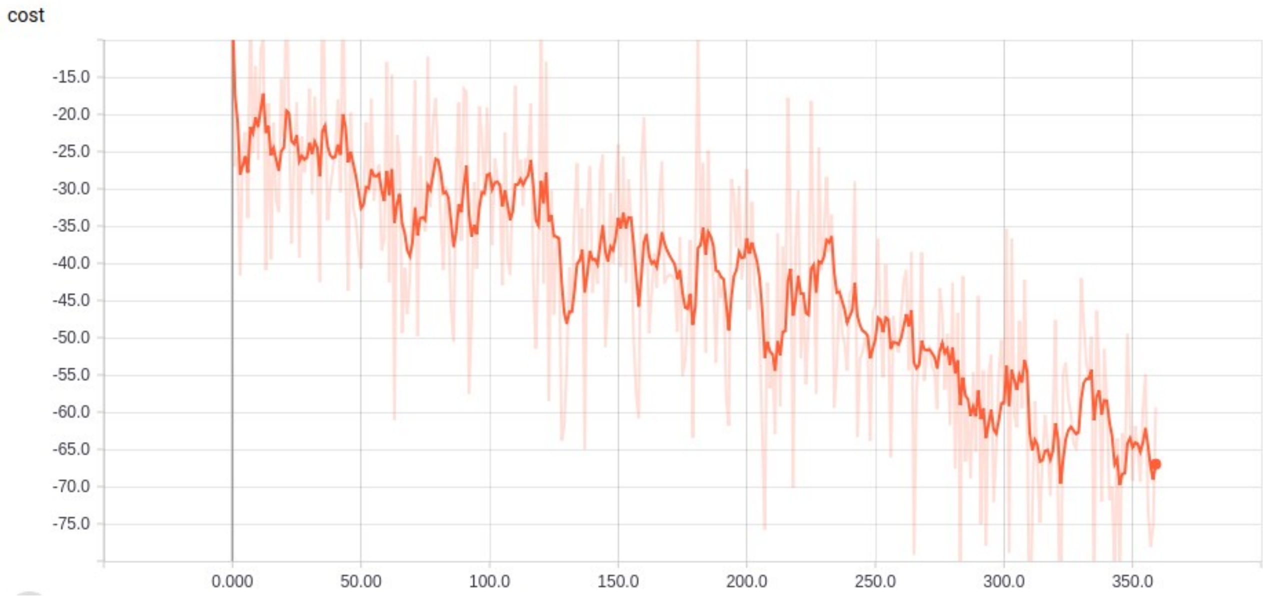
```
import numpy as np
X = data.data
Y = data.target.reshape([data.target.shape[0],1])
print(X.shape)
trainX = X[:360,]
print(trainX.shape)
testX  = X[360:,]
trainY = Y[:360,]
testY  = Y[360:,]
batchSize = 1
batches = int (360/batchSize)
batchX = np.array_split(trainX,(360/batchSize),axis = 0)
batchY = np.array_split(trainY,(360/batchSize),axis = 0)
```

```
(400, 4096)
(360, 4096)
```

Then, the train data is online stochastic gradient is performed by minimizing cross-entropy. Tensor board is used to display the results. The accuracy in the test set is 1, that is the model classifies all the unseen faces correctly. This is unusual but considering the small dataset, and frequent repetition of the similar data(same faces) several times might have caused this.

```
import tensorflow as tf

# reset everything to rerun in jupyter
tf.reset_default_graph()

# config
learning_rate = 0.0001
training_epochs = 1
logs_path = "/tmp/olivette/5"

# input images
with tf.name_scope('input'):
    # Predictors
    x = tf.placeholder(tf.float32, shape=[None, 4096], name="x-input")
    # target 10 output classes
    y_ = tf.placeholder(tf.float32, shape=[None, 1], name="y-input")

# Weights
with tf.name_scope("weights"):
    W = tf.Variable(tf.random_normal([4096, 1]))

# bias
with tf.name_scope("biases"):
    b = tf.Variable(tf.ones([1]))

# model
with tf.name_scope("softmax"):
    # y is our prediction
    h = tf.matmul(x, W) + b
    y = tf.nn.softmax(h)

# cost function
with tf.name_scope('cross_entropy'):
    cross_entropy = -tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(labels=y, logits=h))

# specify optimizer
with tf.name_scope('train'):
    # GradientDescent
    train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cross_entropy)

with tf.name_scope('Accuracy'):
    # Accuracy
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# create a summary for cost and accuracy
tf.summary.scalar("cost", cross_entropy)
tf.summary.scalar("accuracy", accuracy)

# merge all summaries
summary_op = tf.summary.merge_all()

with tf.Session() as sess:
    # initilaizing variables
    sess.run(tf.global_variables_initializer())

    # log writer object
    writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())

    # training
    for epoch in range(training_epochs):

        for i in range(batches):
            batch_x, batch_y = batchX[i],batchY[i]

            # perform the operations
            _,c, summary = sess.run([train_op,cross_entropy, summary_op], feed_dict={x: batch_x, y_: batch_y})
            print("Iteration ", i, "  cost ", c)
            # write log
            writer.add_summary(summary, i)
            writer.flush()
        if epoch % 5 == 0:
            print ("Epoch: ", epoch)
    print ("Accuracy: ", accuracy.eval(feed_dict={x: testX, y_: testY}))
```

The cost function in each iteration varies as follows



The RMSE for ARIMA is `131977209.699`