Brief Tech Approach: DARPA FFT Challenge

Hanfeng Zhai & Timonthy Sands

Sibley School of Mechanical and Aerospace Engineering,
Cornell University, Ithaca, NY

November 7, 2021

Abstract

The DARPA FFT Challenge aims to predict the motion (location) of drifters on oceans. Competitors were given a set of locations and related parameters of the drifters on the ocean in 20 days and required to predict their possible location for the next 5 days. Here, we proposed a series of methods of time-series forecasting, based on autoregressive modeling (AR), autoregressive moving-average (ARMA), including posterior residuals (*PS*) and exponential forgetting (*EF*) to infer the locations of drifters (as visualized in figure 1). Note that some state-of-art methods including random forest and Kriging regression might also be employed to compare with our methods. We will consider the effects of turbulence and sub-mesoscale ocean features by inputting the related data as control signals to the regression model. To prove the feasibility of our approaches, herein included two simple examples of inferring unknown locations of drifters with AR and ARMA in `python`. In our approach, $\frac{4}{5}$ of location data (latitude & longitude) per drifters' ID were fed into the model for training, and $\frac{1}{5}$ of rest data were used to test the accuracy of the model. Random tests on a few drifters' data indicate ARMA displays generally higher accuracy yet costs higher CPU computation time. A brief mathematical model and corresponding `python` implementations are attached in this file. With the proposed examples, the users can apply our code to predict whatever drifters among the 94 drifters, by specifying the value `jj` in the code (as in the sample data) and comparing both the methods.

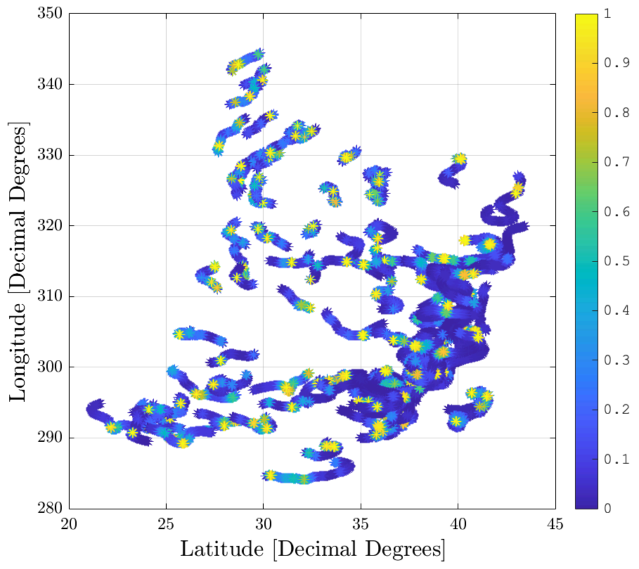


Figure 1: The location plot of the ocean drifters provided by DARPA.

1 Autoregressive Modeling

An autoregressive model has dynamics given by

$$y_t = \delta + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \epsilon_t.$$

The complete specification is

$$y_t = \delta_0 + \delta_1 t + \phi_1 y_{t-1} + \ldots + \phi_p y_{t-p} + \sum_{i=1}^{s-1} \gamma_i d_i + \sum_{j=1}^{m} \kappa_j x_{t,j} + \epsilon_t.$$

where: d_i is a seasonal dummy that is 1 if $mod(t, period) = i$. Period 0 is excluded if the model contains a constant (c is in trend). t is a time trend $(1, 2, \ldots)$ that starts with 1 in the first observation. $x_{t,j}$ are exogenous regressors. Note these are time-aligned to the left-hand-side variable when defining a model. ϵ_t is assumed to be a white noise process. [Reference]

[1]:
```
import timeit
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import pandas_datareader as pdr
import seaborn as sns
from statsmodels.tsa.api import acf, graphics, pacf
from statsmodels.tsa.ar_model import AutoReg, ar_select_order
import statsmodels.api as sm
import scipy.io
from matplotlib import pyplot
import numpy as np
import sys
sys.path.insert(0, '/DARPA/')
from statsmodels.graphics.tsaplots import plot_predict
from statsmodels.tsa.arima_process import arma_generate_sample
from statsmodels.tsa.arima.model import ARIMA
np.random.seed(12345)
```

[2]:
```
plt.rcParams["font.family"] = "Times New Roman"
plt.rcParams.update(plt.rcParamsDefault)
sns.set_style("darkgrid")
pd.plotting.register_matplotlib_converters()
sns.mpl.rc("figure", figsize=(16, 6))
sns.mpl.rc("font", size=16)
jj = 76 # IMPORTANT: specify which spotter you want to predict
```

[3]:
```
# READ DATA
with open('lat_trans.txt', 'r') as f1:
        data1 = f1.read().split(); floats1 = []
        for elem1 in data1:
            try:
                floats1.append(float(elem1))
            except ValueError:
                pass

lat = np.array(data1, dtype = np.float64);lat = np.array_split(lat, 93)
train1 = lat; trainlat = train1

with open('long_trans.txt', 'r') as f2:
        data2 = f2.read().split(); floats2 = []
        for elem2 in data2:
            try:
                floats2.append(float(elem2))
            except ValueError:
                pass

long = np.array(data2, dtype = np.float64);long = np.array_split(long, 93)
train2 = long; trainlong = train2
```

[4]:
```
# for i in range(1,93):
sellat = ar_select_order(trainlat[jj][0:440], 15, "bic", old_names=False)
sellong = ar_select_order(trainlong[jj][0:440], 15, "bic", old_names=False)
```

[5]:
```
reslat = sellat.model.fit()
reslong = sellong.model.fit()
print(reslat.summary())
print(reslong.summary())
time_AR = timeit.default_timer()
```

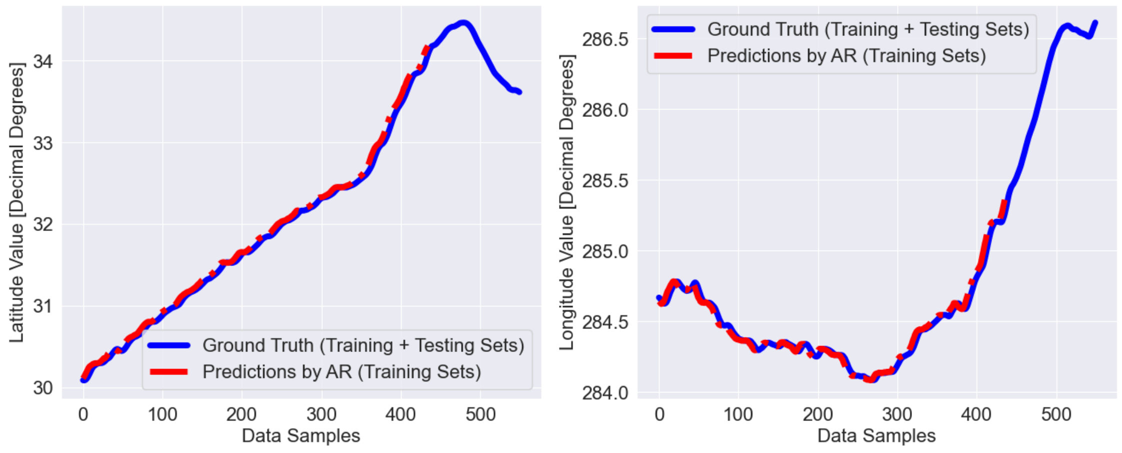
```
                            AutoReg Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  440
Model:                     AutoReg(5)   Log Likelihood                2284.054
Method:               Conditional MLE   S.D. of innovations              0.001
Date:                Sat, 06 Nov 2021   AIC                            -13.307
Time:                        14:02:57   BIC                            -13.242
Sample:                             5   HQIC                           -13.281
                                  440
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0061      0.002     -2.931      0.003      -0.010      -0.002
y.L1           2.5354      0.048     52.878      0.000       2.441       2.629
y.L2          -2.1383      0.131    -16.305      0.000      -2.395      -1.881
y.L3           0.7450      0.163      4.567      0.000       0.425       1.065
y.L4          -0.3006      0.131     -2.293      0.022      -0.558      -0.044
y.L5           0.1586      0.048      3.309      0.001       0.065       0.253
                                    Roots
```

```
==============================================================================
                  Real          Imaginary           Modulus         Frequency
------------------------------------------------------------------------------
AR.1            0.9974           -0.0000j            0.9974           -0.0000
AR.2            1.0675           -0.2592j            1.0986           -0.0379
AR.3            1.0675           +0.2592j            1.0986            0.0379
AR.4           -0.6188           -2.2032j            2.2884           -0.2936
AR.5           -0.6188           +2.2032j            2.2884            0.2936
------------------------------------------------------------------------------
                            AutoReg Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  440
Model:                     AutoReg(4)   Log Likelihood                2210.027
Method:               Conditional MLE   S.D. of innovations              0.002
Date:                Sat, 06 Nov 2021   AIC                            -12.948
Time:                        14:02:57   BIC                            -12.892
Sample:                             4   HQIC                           -12.926
                                  440
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.1950      0.082     -2.371      0.018      -0.356      -0.034
y.L1           2.5090      0.047     52.867      0.000       2.416       2.602
y.L2          -1.9139      0.129    -14.825      0.000      -2.167      -1.661
y.L3           0.2408      0.130      1.858      0.063      -0.013       0.495
y.L4           0.1647      0.048      3.426      0.001       0.070       0.259
                                    Roots
==============================================================================
                  Real          Imaginary           Modulus         Frequency
------------------------------------------------------------------------------
AR.1            0.9894           -0.0000j            0.9894           -0.0000
AR.2            1.1194           -0.2345j            1.1437           -0.0329
AR.3            1.1194           +0.2345j            1.1437            0.0329
AR.4           -4.6902           -0.0000j            4.6902           -0.5000
------------------------------------------------------------------------------
```

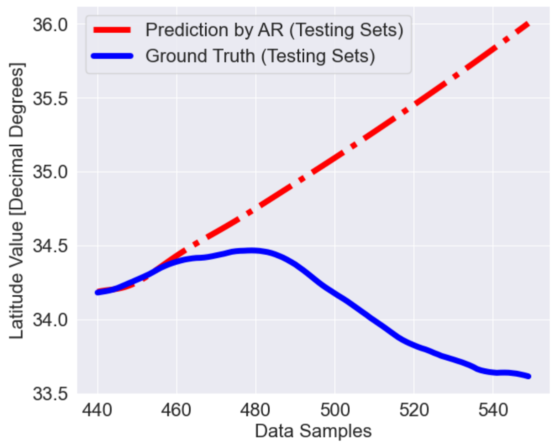
[6]:
```
predlat = reslat.predict()
predlong = reslong.predict()
```

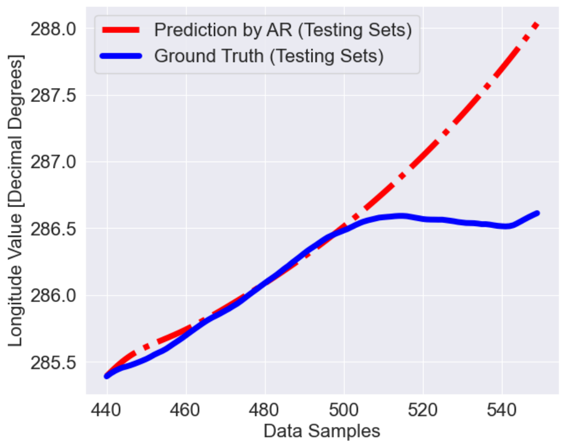
[7]:
```
fig, axs = plt.subplots(1,2)
# plt.rcParams["figure.figsize"] = (5,5)
axs[0].plot(trainlat[jj], linewidth=5, color = 'blue', label = "Ground Truth␣
 ↪(Training + Testing Sets)")
axs[0].plot(predlat, linewidth=5, linestyle="-.", color = 'red', label =␣
 ↪"Predictions by AR (Training Sets)")
axs[0].set(xlabel="Data Samples", ylabel="Latitude Value [Decimal Degrees]")
```

```
axs[1].plot(trainlong[jj], linewidth=5, color = 'blue', label = "Ground Truth␣
 ↪(Training + Testing Sets)")
axs[1].plot(predlong, linewidth=5, linestyle="-.", color = 'red', label =␣
 ↪"Predictions by AR (Training Sets)")
axs[1].set(xlabel="Data Samples", ylabel="Longitude Value [Decimal Degrees]")
for ax in axs.flat:
    ax.legend()
plt.legend()
plt.show()
```



```
[8]: # print(res.forecast(steps=151))
fig, axs = plt.subplots(1,2)
x = np.arange(0, 550, 1)
# plt.rcParams["figure.figsize"] = (5,5)
axs[0].plot(x[440:550], reslat.forecast(steps=110), linewidth=5, linestyle="-.",␣
 ↪color = 'red', label = "Prediction by AR (Testing Sets)")
axs[0].plot(x[440:550], trainlat[jj][440:550], linewidth=5, color = 'blue',␣
 ↪label = "Ground Truth (Testing Sets)")
axs[0].set(xlabel="Data Samples", ylabel="Latitude Value [Decimal Degrees]")
axs[1].plot(x[440:550], reslong.forecast(steps=110), linewidth=5, linestyle="-.
 ↪", color = 'red', label = "Prediction by AR (Testing Sets)")
axs[1].plot(x[440:550], trainlong[jj][440:550], linewidth=5, color = 'blue',␣
 ↪label = "Ground Truth (Testing Sets)")
axs[1].set(xlabel="Data Samples", ylabel="Longitude Value [Decimal Degrees]")
for ax in axs.flat:
    ax.legend()
plt.show()
predlat = reslat.forecast(steps=110)
predlong = reslong.forecast(steps=110)
```





---

2 Autoregressive Modeling-Average (ARMA)

Autoregressive Moving-Average (ARMA) model that correlates the input data with past data history of the target predicted data.

- **Autoregressive Model**

Based on the previous intro, The notation $\mathcal{AR}(p)$ refers to the **autoregressive model** of order p. The $\mathcal{AR}(p)$ model is written

$$y_t = c + \sum_{i=1}^{p} \varphi_i y_{t-1} + \varepsilon_t$$

where $\varphi_1, \ldots, \varphi_p$ are parameters, c is a constant, and the random variable ε_t is white noise.

Some constraints are necessary on the values of the parameters so that the model remains stationary. For example, processes in the $\mathcal{AR}(1)$ model with $|\varphi_1| \geq 1$ are not stationary.

- **Moving-Average Model**

The notation $\mathcal{MA}(q)$ refers to the moving average model of order q:

$$y_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}\, y_t$$

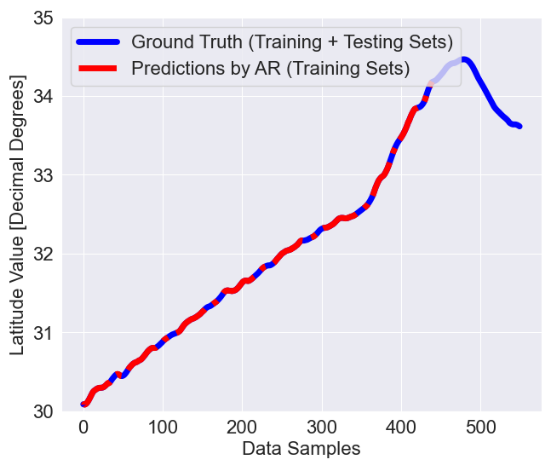
where the $\theta_1, ..., \theta_q$ are the parameters of the model, μ is the expectation of y_t (often assumed to equal 0), and the ε_t, ε_{t-1},... are again, white noise error terms.

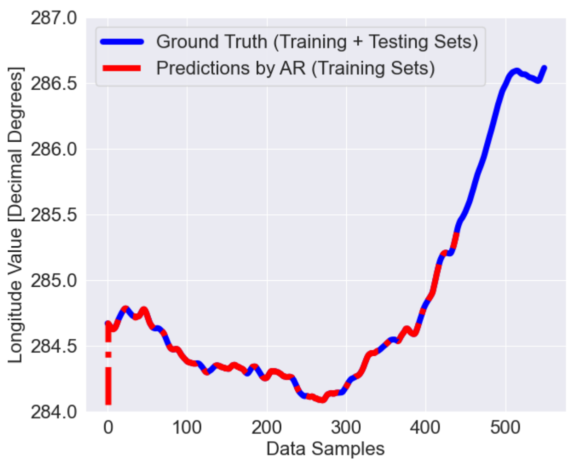
The notation $\mathcal{ARMA}(p, q)$ refers to the model with p autoregressive terms and q moving-average terms. This model contains the $\mathcal{AR}(p)$ and $\mathcal{MA}(q)$ models,

$$y_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i y_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i}.$$

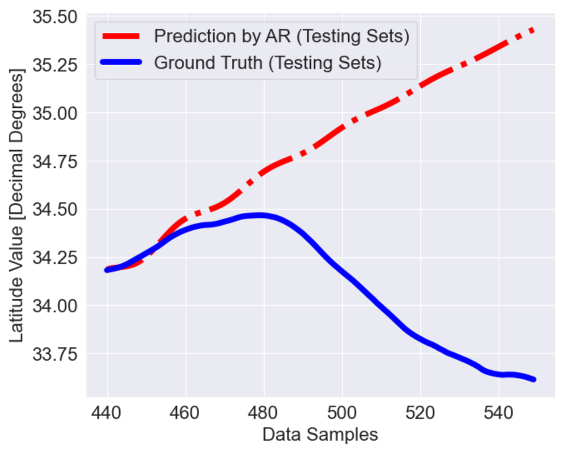
[9]:
```
y = trainlat[jj][0:440]
X = trainlong[jj][0:440]
yy = pd.Series(y, X)
XX = pd.Series(X, y)
arma_modlat = ARIMA(yy, order=(15,1,0), trend="n")
arma_reslat = arma_modlat.fit()
arma_modlong = ARIMA(XX, order=(15,1,0), trend="n")
arma_reslong = arma_modlong.fit()
time_ARMA = timeit.default_timer()
predlat_arma = arma_reslat.predict()
predlong_arma = arma_reslong.predict()
```

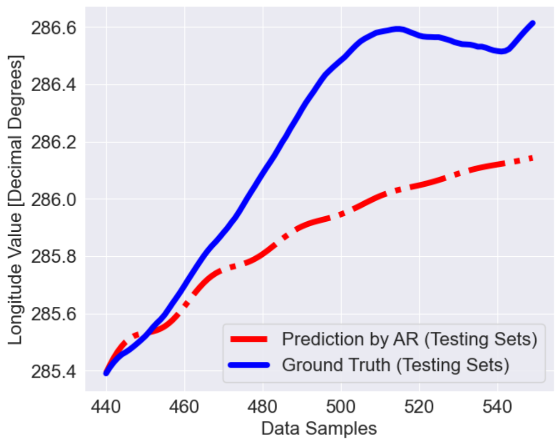
[10]:
```
sns.set_style("darkgrid")
pd.plotting.register_matplotlib_converters()
sns.mpl.rc("figure", figsize=(16, 6))
sns.mpl.rc("font", size=16)
fig, axs = plt.subplots(1,2)
axs[0].plot(trainlat[jj], linewidth=5, color = 'blue', label = "Ground Truth␣
 ↪(Training + Testing Sets)")
axs[0].plot(np.array(predlat_arma), linewidth=5, linestyle="-.", color = 'red',␣
 ↪label = "Predictions by AR (Training Sets)")
axs[0].set(xlabel="Data Samples", ylabel="Latitude Value [Decimal Degrees]")
axs[0].set_ylim(30,35)
# plt.rcParams["figure.figsize"] = (5,5)
axs[1].plot(trainlong[jj], linewidth=5, color = 'blue', label = "Ground Truth␣
 ↪(Training + Testing Sets)")
axs[1].plot(np.array(predlong_arma), linewidth=5, linestyle="-.", color = 'red',␣
 ↪label = "Predictions by AR (Training Sets)")
axs[1].set(xlabel="Data Samples", ylabel="Longitude Value [Decimal Degrees]")
axs[1].set_ylim(284, 287)
for ax in axs.flat:
    ax.legend()
plt.legend()
plt.show()
```



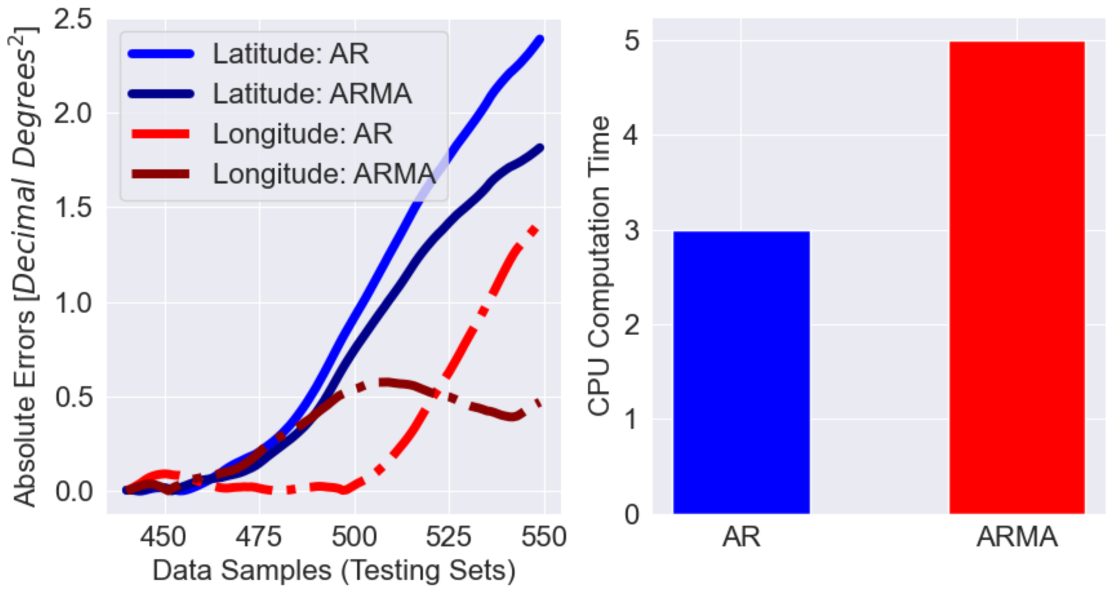


[11]:
```
predlatarma = arma_reslat.forecast(steps=110)
predlongarma = arma_reslong.forecast(steps=110)
fig, axs = plt.subplots(1,2)
x = np.arange(0, 550, 1)
axs[0].plot(x[440:550], arma_reslat.forecast(steps=110), linewidth=5,␣
 ↪linestyle="-.", color = 'red', label = "Prediction by AR (Testing Sets)")
axs[0].plot(x[440:550], trainlat[jj][440:550], linewidth=5, color = 'blue',␣
 ↪label = "Ground Truth (Testing Sets)")
axs[0].set(xlabel="Data Samples", ylabel="Latitude Value [Decimal Degrees]")
axs[1].plot(x[440:550], arma_reslong.forecast(steps=110), linewidth=5,␣
 ↪linestyle="-.", color = 'red', label = "Prediction by AR (Testing Sets)")
axs[1].plot(x[440:550], trainlong[jj][440:550], linewidth=5, color = 'blue',␣
 ↪label = "Ground Truth (Testing Sets)")
axs[1].set(xlabel="Data Samples", ylabel="Longitude Value [Decimal Degrees]")
for ax in axs.flat:
    ax.legend()
plt.show()
```





[12]:
```
plt.rcParams["figure.figsize"] = (10,5)
fig, axs = plt.subplots(1,2)
x = np.arange(440, 550, 1)
dat1 = [3]; dat2 = [5]#[time_AR,time_ARMA]
lab1 = ['AR']; lab2 = ['ARMA']
armaaselat = np.abs(predlatarma - trainlat[jj][440:550])
armaaselong = np.abs(predlongarma - trainlong[jj][440:550])
axs[0].plot(x, aselat, linewidth=5, color = 'blue', label = "Latitude: AR")
axs[0].plot(x, armaaselat, linewidth=5, color = 'darkblue', label = "Latitude:␣
 ↪ARMA")
axs[0].plot(x, aselong, linewidth=5, linestyle="-.", color = 'red', label =␣
 ↪"Longitude: AR")
axs[0].plot(x, armaaselong, linewidth=5, linestyle="-.", color = 'darkred',␣
 ↪label = "Longitude: ARMA")
axs[0].set_xlabel("Data Samples (Testing Sets)")
axs[0].set_ylabel("Absolute Errors [$Decimal\ Degrees^2$]")
axs[0].legend()
axs[1].bar(lab1,dat1,color = 'b',width = 0.5)
axs[1].bar(lab2,dat2,color = 'r',width = 0.5)
axs[1].set_ylabel("CPU Computation Time")
plt.show()
```


Absolute Errors [$Decimal\ Degrees^2$]
Latitude: AR
Latitude: ARMA
Longitude: AR
Longitude: ARMA
Data Samples (Testing Sets)
CPU Computation Time
AR
ARMA