Advanced Topics in Signal Processing Project 2

Name: Siatiras Evangelos

AM: EN2190001

The project has been implemented using python 3.6.

In a python environment run the below commands before the execution of the script.

pip install spectrum

pip install scipy

pip install matplotlib

pip install numpy

Note that .mat files should be in the same directory with the python script.

Ex_A

The linear prediction approach used to predict future values of a time-series is as follows.

Consider the following model e.g.

$$y(n) \approx a_1 y(n-1) + a_2 y(n-2) + a_3 y(n-3)$$

Then having the first 200 given samples as our training set our goal is to find the optimal $\hat{\theta}$ estimator which can be solved using least squares. Finding

$$\theta = (\theta_0, \theta_1, \theta_2)$$

Can be viewed as one of solving an overdetermined system of equations. For example, if $y(n)$ is available for $0 \leq n \leq N-1$ and we seek a third order linear predictor, then the overdetermined set of equations are given by

$$\begin{bmatrix} y(3) \\ y(4) \\ \vdots \\ y(N-1) \end{bmatrix} \approx \begin{bmatrix} y(2) & y(1) & y(0) \\ y(3) & y(2) & y(1) \\ \vdots & \vdots & \vdots \\ y(N-2) & y(N-3) & y(N-4) \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

Which we can write as $\bar{y} = \theta^T X$ where X is a matrix of size $(N-3) \times 3$.

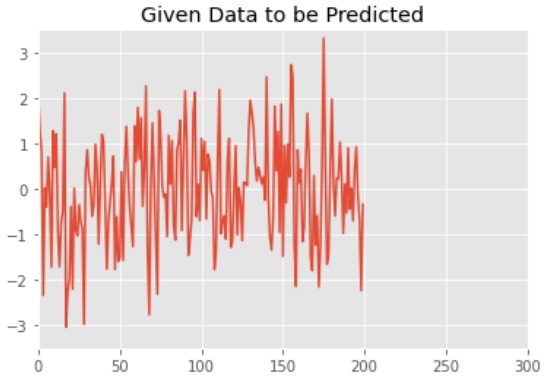
In order to visualize efficiently this rational I create a dictionary in order to represent the first two matrixes above, and then I create a vector from the dictionary adding one more column needed in order to be able to

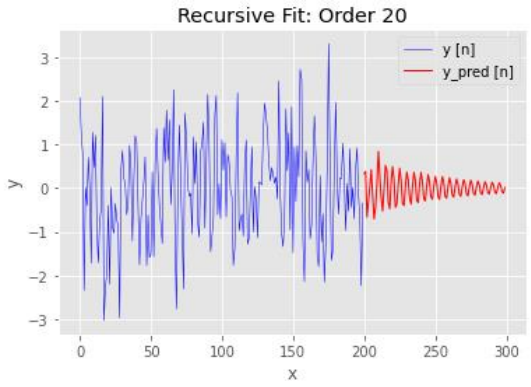
```
d = {}
for j in np.arange (order, len(self.y)):
  if (j-1-order >= 0):
    d['y' + str(j)] = self.y[j-order:j:1][::-1]
  else:
    d['y' + str(j)] = self.y[0:order:1][::-1]

# for key, value in d.items() :
#   print (key, value)

X = np.column_stack(d.values()).T
```

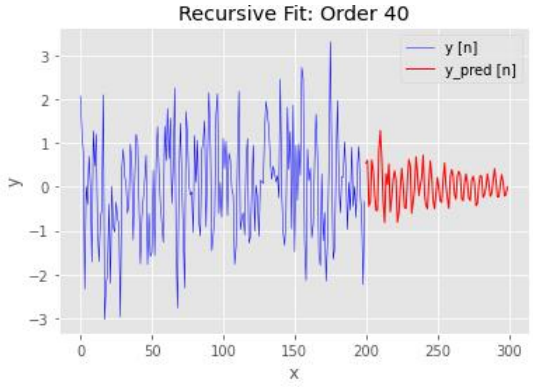
calculate the vector θ.

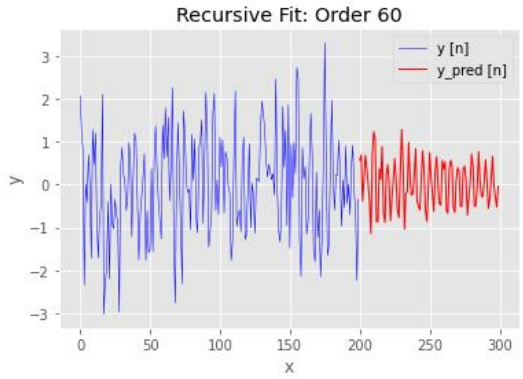
Then using the least squares, we calculate the parameter vector θ using the following equation $\theta = (X^T X)^{-1} X^T \bar{y}$

```
theta = np.matmul(np.matmul(linalg.pinv(np.matmul(np.transpose(X),X)), np.transpose(X)), self.y[order:])
```









Once the coefficients θ_i are found then $y(n)$ for $n > N$ can be estimated using the above-mentioned recursive difference equation. In the above plots 200 samples of data are available i.e. $y(n)$ for $0 < n < 199$ .For these samples a p-order =20,40,60 linear predictor is obtained by least squares and the subsequent 100 samples are predicted.

The implementation as below:

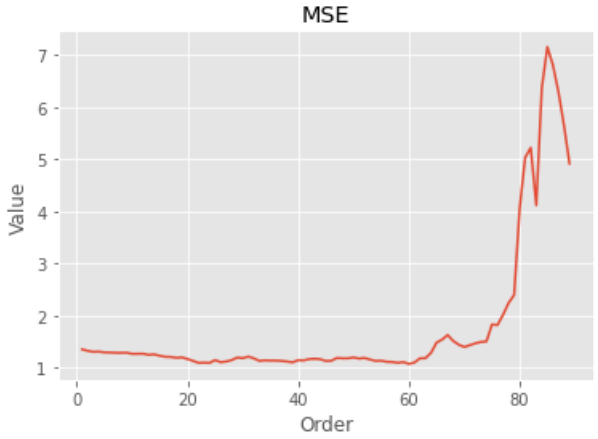
```
for j in np.arange (start, len(y_pts)):
  for i in np.arange(0, len(self.theta)):
    if (j-i-1 < start):
      self.line [j] += self.theta[i] * y_pts[j -i -1]
    else:
      self.line [j] += self.theta[i] * self.line[j -i -1]
```

We define start variable as the starting point of our test set. At the beginning the prediction of the new samples, the recursive model should have available all the samples needed (where the number of the samples is directly related with the order) therefore we get the required samples from the training set which is already introduced to our model. Then we predict every sample in the same recursive way by using the previously predicted samples.

Now our next goal is to find the optimal order of our model in the terms of least Squares. We define our MSE as follows:

```
mse_test_set = np.sum((test_set[start:] - PR.line[start:]) ** 2)*0.01
```

Intuitively with this equation we calculate the sum of distances (errors) between the actual and the predicted sample multiplied by 0.01.



In the above figure is cleared that as we increase our model order it is starting to be more and more sensitive in predicting new samples. Our model is learning adopting very much the training set and it is unable to predict efficiently samples that have not introduced to the model in the training and produces errors. This is known as overfitting of the data. For example, for a model of order 80 the MSE increases dramatically as when we train our model and calculating the parameter vector we use the 80 subsequent samples (for each predicted sample) of 200 available from our training set. So any slight difference in the test set that it is not introduced in the training produce a much bigger error. From the above line in the plot we can easily extract the order of our model that produces the min MSE is implemented as follows.

```
line = []
min_mse = 100
order = 0
for i in np.arange (1,90,1):
  line = np.append(line, PerformRegression('A.mat',200,i,plot=False))
  if (min_mse > float(np.min(line))):
    min_mse = np.min(line)
    order = i

# print (line)
# print (order)
plt.plot(np.arange (1,90,1), line)
plt.title('MSE')
plt.xlabel('Order')
plt.ylabel('Value')
plt.show()
PerformRegression('A.mat',200,order,plot=True ,mse=True)
```

The min MSE is produced for a model of order 60 with value ≈ 1.074 *and plot as shown in the above section.*

```
MSE of Model Order: 60 is: 1.0747783407431846
```

Ex_2

In this active noise cancelation task, we will use and FIR Wiener filter as we have learned.

An FIR Wiener filter of $p-1$ order it is designed based on Wiener-Hopf equations $\sum_{l=0}^{p-1} w(l)\, r_x(k-l) = r_{yx}(k)$, $k = 0,1, \dots, p-1$ or $R_x w = r_{yx}$ in which $r_x(k)$ and $r_{yx}(k)$ are the auto- and cross-correlations from the stationary $x(n) = u_2(n)$ and y(n).So for given lag or filter order,

At first we calculate the autocorrelation of u_2 :

```
  def autocorr(self):
    c = np.correlate(self.u2, self.u2, 'full')
    mid = len(c)//2
    # print (c[mid:mid+lag])
    acov = c[mid:mid+self.lag]
    acor = acov/acov[0]
    return(acor)
```

Then the cross correlation of y and u_2 :

```
 def crosscorr(self):
   c = np.correlate(self.y, self.u2, 'full')
   mid = len(c)//2
   # print (c[mid:mid+lag])
   ccov = c[mid:mid+self.lag]
   crosscor = ccov/ccov[0]
   return(crosscor)
```

Next, we compute the wiener filter coefficients:

```
  def compute_coeffs(self):
    ac = self.autocorr()
    R = linalg.toeplitz(ac[:self.lag])
    self.r = self.crosscorr()[:self.lag+1]
    self.w_opt = linalg.inv(R).dot(self.r)
    return self.w_opt
```

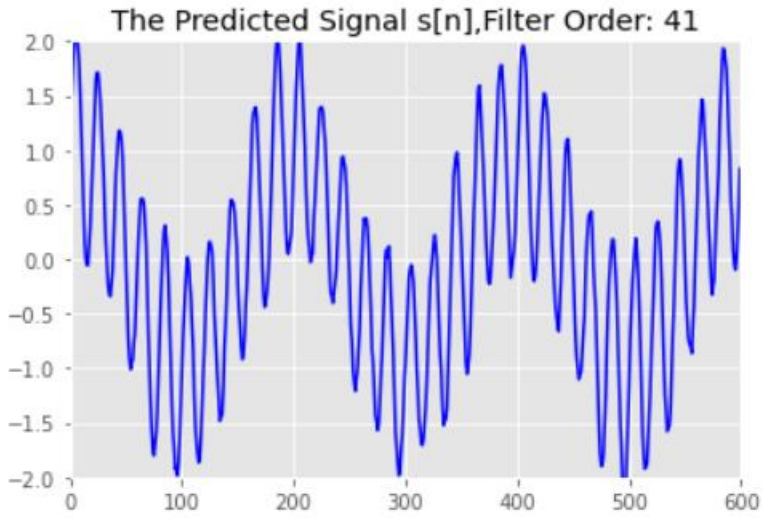
In order to predict the unknown added noise quantity u_1 :

```
  def predict_noise(self):
    self.u1 = sc.lfilter(w.w_opt, [1], self.u2)
    return self.u1
```

Then by a couple of experiments for some filter orders we end up that a range of orders with their predicted results been very closed to the required.

```
for i in np.arange(40,50,1):
  w = WienerFIRFilter(u2 ,y ,i)
  w.compute_coeffs()
  w.predict_noise()
  w.predict_signal()
  w.plot_predicted_signal()
```

Specifically, for a filter of order 41 we get the following result which is almost the same with the desired signal.


The Predicted Signal s[n],Filter Order: 41
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
100
200
300
400
500
600