Carthage University

Tunisia Polytechnic School

La Marsa, Tunisia

Theory of Signals and Systems

MODELISATION OF SIGNALS USING AUTOREGRESSION MODEL

Khaireddine Medhioub

khaireddine.medhioub@ept.ucar.tn

Hamouda Baghdadi

hamouda.bagdadi@ept.ucar.tn

Supervisor(s): Dr. Ines Bousnina

Tunisia Polytechnic School, La Marsa, Tunisia

11/02/2022

Contents

List of Figures

List of Tables

1. Introduction

There exist a full load of models used for forecasting a time series variable. A well known model is multiple regression model in which we forecast the variable of interest using a linear combination of predictors. A particular model of that type is the Auto-regression model.

Auto-regression is a time series model that uses observations from previous time steps as input to a regression equation to predict the value at the next step.
An auto-regressive model of order **p** can be written as:

$$Y(n) + a_1 * Y(n-1) + ... + a_p * Y(n-p) = W(n) \tag{1}$$

where W(n) is white noise. This is like a multiple regression but with lagged values of Y(n) as predictors.

In our project, we will explore two approaches forecasting new values of an audio sound with white noise, using the Auto-regression model. We will start by implementing simply the basic model then we will see how the model can forecast the value of the signal while incrementally feeding our dataset with new observation (real value) at each instant. In the end, we will change the parameter p (window) and observe its impact on the result.

2. Model Approches

2.1 Setting the environment

First, we import the libraries needed for the implementation of the model:

- torchaudio: to import the audio.
- Matplotlib: to visualize the signals.
- `Statsmodels.tsa.ar_model: to use the model.`
- sklearn.metrics: to calculate the MSE.
- Pandas: to read the csv file containing the audio values.

```
import torch
import torchaudio
import pandas as  pd
from pandas import read_csv
from matplotlib import pyplot
from pandas.plotting import lag_plot
from pandas import DataFrame
from pandas import concat
from statsmodels.tsa.ar_model import AutoReg
from sklearn.metrics import mean_squared_error
from math import sqrt
```

Then we import the audio placed in the drive, transform it to a pandas dataframe, then to a CSV file storing it also in the drive, then we create a series by reading the CSV file.

```
x, sr = torchaudio.load('/content/drive/MyDrive/xylophone.wav')
xn, srn = torchaudio.load('/content/drive/MyDrive/whitenoise.wav')

#Dataframe which contains the value of the audio signal in chronological order
wave1 = pd.DataFrame(x.numpy())
#Dataframe which contains the value of the white noise in chronological order
wn = pd.DataFrame(xn.numpy())

wave1.to_csv('wave1.csv', index=False)
!cp wave1.csv "drive/My Drive" #store the dataframe in the drive

wn.to_csv('wn.csv', index=False)
!cp wn.csv "drive/My Drive"

series = read_csv('wave1.csv', header=0, index_col=0)
series=series.T
seriesn = read_csv('wn.csv', header=0, index_col=0)
seriesn=seriesn.T
```

Both signals, the original audio and the noise, need to have the same length (same number of values= same number of instances). So we are going to restrict the used white noise to centered values around the pick (mean), knowing that we used a white noise with normal distribution.

```
n=(len(seriesn)-len(series))//2
seriesn=seriesn[n+1:len(seriesn)-n]
```

The statsmodels library provides an auto-regression model where you must specify an appropriate lag value and trains a linear regression model.

We can use this model by first creating the model AutoReg() and then calling fit() to train it on our dataset. This returns an AutoRegResults object.

Once fit, we can use the model to make a prediction by calling the predict() function for a number of observations in the future.

2.2 Simple Model

In this approach, we create and evaluate a static auto-regressive model. We split the dataset into a train set and a testset. The we feed the train set to the AutoReg model alongside the value of the order p (window) of the model. This will give us the parameters a_1, a_2, ..., a_p of the equation (1).

Afterwards, the predict function related to the AutoReg model compute the future values we want to forecast.

Important Notes

- For the first predicted value at time n, we use the p previous values from the train dataset. However, the second predicted value (at time n+1), we use p-1 values from the train dataset (Y(n-p+1), ..., Y(n-1)) **plus the predicted value at time n**. An so on for the next values.
- The train dataset contains both the original audio and the white noise whereas the test dataset contains only the original audio (without the noise). The intuition behind this choice is that our model has two roles; the first one is obvious, which is to **forecast** the signal for future instances. The second role is kind of hidden role, which is to **extract** the original audio from the one combined with noise.

The complete implementation is listed below:

```
# split dataset
X = series.values + seriesn.values
train = X[1:len(X)-100]
X1 = series.values
test= X1[len(X)-100:]

# train autoregression
model = AutoReg(train, lags=50)
```

```
model_fit = model.fit()
print('Coefficients: %s' % model_fit.params)

# make predictions
predictions = model_fit.predict(start=len(train),
                                end=len(train)+len(test)-1,
                                dynamic=False)
for i in range(len(predictions)):
  print('predicted=%f, expected=%f' % (predictions[i], test[i]))
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)

# plot results
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```

Using an order of **p = lags = 15** to predict 100 future observations, we obtained Test RMSE=0.011 and the graph shown below :

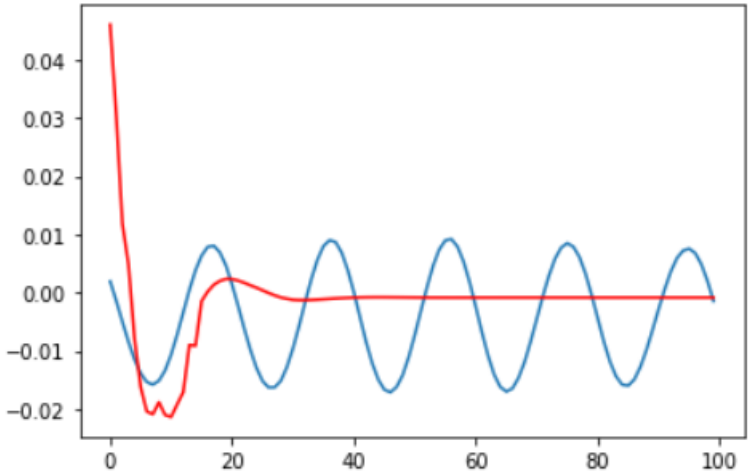

Figure 1: The prediction made by the Simple Model (in red) versus the real observation (in blue) for each instant

2.3 Advanced Model

The intuition behind this approach comes from observing the real values of the signal once the prediction has been made. In other terms, for the first predicted value at time n, we use the p previous values from the train dataset (the same as the previous approach). However, for the second predicted value (at time n+1), we use p-1 values from the train dataset (Y(n-p), ..., Y(n-1)) **plus the real value at time n** instead the predicted value and the key difference between the two approches.

We can look at this approach from other interesting perspective, well illustrated in making predictions in the stock market. So, now we at instant n-1 and we know the prices of a certain stock X for the previous p **real** values (observed values). Our goal is to predict its value at instant n, so we use the p previous observations and we obtain the **predicted** value of X at instant n. Now, n is the present and we can make

an observation and know its real value. Once we do that, we use this observation to make prediction for instant n+1.

Our motivation is to use every **deterministic information** (real observations) given at certain instant to make predictions in the future.

So, we implemented a for-loop for this intuition of the prediction process:

```
# split dataset
X = series.values + seriesn.values
train = X[1:len(X)-100]
X1 = series.values
test= X1[len(X)-100:]

# train autoregression
window = 50
model = AutoReg(train, lags=50)
model_fit = model.fit()
coef = model_fit.params

# walk forward over time steps in test
history = train[len(train)-window:]
history = [history[i] for i in range(len(history))]
predictions = list()
for t in range(len(test)):
  length = len(history)
  lag = [history[i] for i in range(length-window,length)]
  yhat = coef[0]
  for d in range(window):
    yhat += coef[d+1] * lag[window-d-1]
  obs = test[t]
  predictions.append(yhat)
  history.append(obs)
  print('predicted=%f, expected=%f' % (yhat, obs))
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)

# plot
pyplot.plot(test)
pyplot.plot(predictions, color='red')
pyplot.show()
```

Using an order of $\mathbf{p = lags = 15}$ to predict 100 future observations, we obtained Test RMSE=0.009 and the graph shown below :

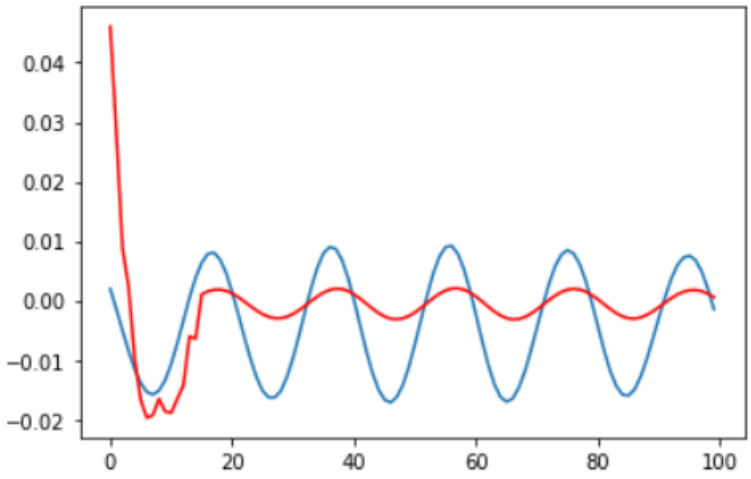

Figure 2: The prediction made by the Advanced Model (in red) versus the real observation (in blue) for each instant

2.4 Impact of parameters (window)

In this section, we are going to analyze the effect of the chosen window on the model accuracy. So, we are going to run the model using different values of the window (p) and see if we can find any pattern explaining the accuracy variation.

2.4.1 Case of Simple Model

To study the impact of the used window, we run the model using different values of window and each time, we calculate the accuracy metric (RMSE). The following table shows the results.

window	5	10	15	20	25	30	35
RMSE	0.00981	0.009223	0.010852	0.011218	0.013677	0.013862	0.014613
window	40	45	50	55	60	65	70
RMSE	0.015021	0.016912	0.017007	0.017245	0.017914	0.018386	0.017952

Table 1: RMSE of the Simple Model for different values of window

The variation of RMSE in function of the window is more precisely shown in the following graph.

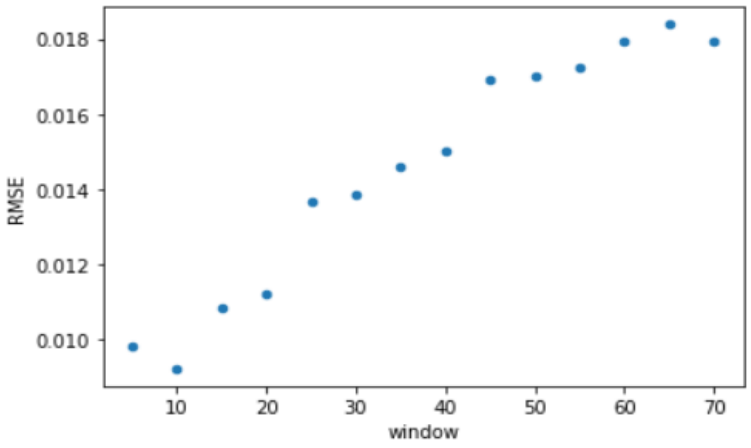

Figure 3: $\text{RMSE}_{SM} = f(window)$

Based on the founded results, we can safely conclude that we can obtain more precious and accurate model (smaller RMSE) if we use a small window. In other terms, to forecast a future value of the signal, we only need the few precious values.

2.4.2 Case of Advanced Model

As explained previously, to study the impact of the used window, we run the model using different values of window and each time, we calculate the accuracy metric (RMSE). The following table shows the results.

window	5	10	15	20	25	30	35
RMSE	0.008979	0.008139	0.009431	0.009669	0.011308	0.01142	0.01147
window	40	45	50	55	60	65	70
RMSE	0.011874	0.013278	0.01344	0.013627	0.013833	0.014323	0.01395

Table 2: RMSE of the Advanced Model for different values of window
The variation of RMSE in function of the window is more precisely shown in the following graph.

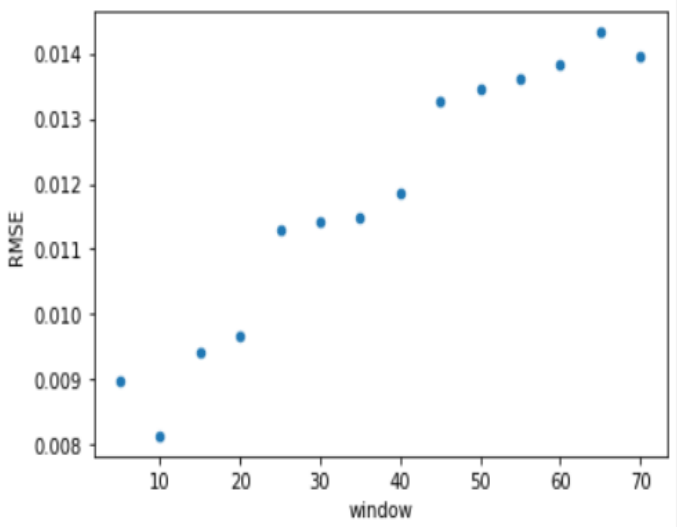

Figure 4: $\text{RMSE}_{AM} = f(window)$

Based on the founded results, we can make the same conclusion as the previous model. So, using a small window increases the accuracy of the Advanced Model (small RMSE) and enables us to make better predictions.

3. Conclusion

In this report, we chose a signal (generated by xylophone) and we added to it a white noise with normal distribution. Then, we did the modeling of the result signal using, first, a simple model (which uses the previous predicted values to forecast future values). As an attempt to reduce the error of our model and so to get more accurate predictions, we advance the model by adding, each time we make a prediction, the real observation of the predicted value in our train dataset. As the results have shown, the advanced model has given more accurate predictions (smaller RMSE than the simple model). As a next step to build a better model, we studied the influence of the main parameter of the model on its accuracy (measured by RMSE), we found that using a small window reduces the model's error.

As a final conclusion, the best solution proposed by this report to modelize the given signal and to forecast its future values is to use the Advanced Model with a small window.