Time series Analysis & Modeling

DATS 6450

LAB # 9- GPAC Table Implementation

The main purpose of this LAB is to implement the GPAC array covered in lecture using Python program and test the accuracy of your code using an ARMA(n_a,n_b) model. It is permitted to simulate the ARMA process using the statsmodels. Everyone needs to write their own GPAC code that generates GPAC table for various numbers of rows and columns.

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
```

1- Develop a python code that generates ARMA (n_a,n_b) process. The program should be written in a way that askes a user to enter the following information. Hint: Use statsemodels library.
   a. *Enter the number of data samples:* _____
   b. *Enter the mean of white noise:* _____
   c. *Enter the variance of the white noise:* _____
   d. *Enter AR order:* _____
   e. *Enter MA order:* _____
   f. *Enter the coefficients of AR (you need to include a hint how this should be entered):*____
   g. *Enter the coefficients of MA (you need to include a hint how this should be entered):*____

2- Edit the python code in step 1 that implement the GPAC table using the following equation. The output should be the GPAC table.

$$\phi_{kk}^{j} = \frac{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j+k) \end{vmatrix}}{\begin{vmatrix} \hat{R}_y(j) & \hat{R}_y(j-1) & \dots & \hat{R}_y(j-k+1) \\ \hat{R}_y(j+1) & \hat{R}_y(j) & \dots & \hat{R}_y(j-k+2) \\ \vdots & \vdots & \vdots & \vdots \\ \hat{R}_y(j+k-1) & \hat{R}_y(j+k-2) & \dots & \hat{R}_y(j) \end{vmatrix}}$$

j \ k	1	2	⋯	n_a
0	ϕ_{11}^{0}	ϕ_{22}^{0}	⋯	
1	ϕ_{11}^{1}	ϕ_{22}^{1}	⋯	
⋮	⋮	⋮		
n_b				

$-a_{n_a}$	0	0	0
$-a_{n_a}$	$\frac{0}{0}$	$\frac{0}{0}$	$\frac{0}{0}$
$-a_{n_a}$	$\frac{0}{0}$	$\frac{0}{0}$	$\frac{0}{0}$

Where $\hat{R}_y(j)$ is the estimated autocorrelation of y(t) at lag j.

3- Using the developed code above, simulate ARMA(1,0) for 1000 samples as follows:

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

Use statsemodels library to simulate above ARMA (1,2) process. You can use the .generate_sample (# of samples, scales = std of WN noise) + mean (y) where mean(y) = $\frac{\mu e(1+\sum bi)}{1+\sum ai}$

4- Using python program, estimate ACF for y(t) using the ACF function developed in previous labs. Number of lags = 15.

5- Using the estimated ACF from previous question, display GPAC table for k=7 and j=7. Do you see a pattern of constant column 0.5 and a row of zeros? What is the estimated n_a and what is the estimated n_b?

6- Increase the number of samples to 5000 and 10000. Do the numbers in the pattern converge?

7- Repeat above steps for the following 7 examples with 10000 samples which should cover the followings:
   a. Display the GPAC table (k=7, j=7) in your report and highlight the pattern that read the process orders. Assign the correct label for each column (i.e. k=1, k=2, …) and each row (i.e. j=0, j=1, ...)
   b. Display the ACF for the y(t) for 15 lags.
   c. Based on the observed pattern in the GPAC table, what is the estimated n_a and estimated n_b. Compare the estimated order versus the true order. Write down your observations.
   d. Make sure to include the .py code that can be run and verify the results.

Example 2: ARMA (0,1): y(t) = e(t) + 0.5e(t-1)
Example 3: ARMA (1,1): y(t) + 0.5y(t-1) = e(t) + 0.5e(t-1)
Example 4: ARMA (2,0): y(t) + 0.5y(t-1) + 0.2y(t-2) = e(t)
Example 5: ARMA (2,1): y(t) + 0.5y(t-1) + 0.2y(t-2) = e(t) - 0.5e(t-1)
Example 6: ARMA (1,2): y(t) + 0.5y(t-1) = e(t) + 0.5e(t-1) - 0.4e(t-2)
Example 7: ARMA (0,2): y(t) = e(t) + 0.5e(t-1) - 0.4e(t-2)
Example 8: ARMA (2,2): y(t)+0.5y(t-1) +0.2y(t-2) = e(t)+0.5e(t-1) - 0.4e(t-2)

Upload the **solution <u>report (as a single pdf</u>**) plus **<u>the .py file(s)</u>** through BB by the due date.