Econometrics Assignment

Debajyoti Maity
B2230023
Department of Computer Science
Ramakrishna Mission Vivekananda Educational and Research Institute
Belur Math, Howrah
Pin - 711 202, West Bengal

Contents

1 Question1

1.1 1.a

```
import numpy as np

# Given data
time = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
yt = np.array([3, 7, -4, -6, 1, 9, -3, -7, 1, 9, -3, -7])
forecast_1 = np.array([2, 4, -3, -4, 0, 12, -6, -4, 4, 15, -1, -11])
forecast_2 = np.array([7, 9, -8, -10, 7, 9, -8, -10, 7, 9, -8, -10])
---------------------------------------
# Mean Absolute Error (MAE)
def calculate_mae(actual, forecast):
    return np.mean(np.abs(actual - forecast))

mae_1 = calculate_mae(yt, forecast_1)
mae_2 = calculate_mae(yt, forecast_2)

# Mean Squared Error (MSE)
def calculate_mse(actual, forecast):
    return np.mean((actual - forecast)**2)

mse_1 = calculate_mse(yt, forecast_1)
mse_2 = calculate_mse(yt, forecast_2)

# Root Mean Squared Error (RMSE)
def calculate_rmse(actual, forecast):
    return np.sqrt(np.mean((actual - forecast)**2))

rmse_1 = calculate_rmse(yt, forecast_1)
```

```
rmse_2 = calculate_rmse(yt, forecast_2)
# Print the results
print("Method 1:")
print(f"MAE: {mae_1}")
print(f"MSE: {mse_1}")
print(f"RMSE: {rmse_1}")
print("\nMethod 2:")
print(f"MAE: {mae_2}")
print(f"MSE: {mse_2}")
print(f"RMSE: {rmse_2}")
```

	MAE	**MSE**	**RMSE**
Method 1	2.67	9.0	3.0
Method 2	3.5	16.0	4.0

Table 1: Comparison of Error Metrics for Method 1 and Method 2

From Table 1, it is evident that Method 1 outperforms Method 2, as the error metrics (MAE, MSE, and RMSE) for Method 1 are consistently lower.

From these results, it is evident that Method 1 outperforms Method 2. The lower values of Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) in Method 1 suggest a better forecast compared to Method 2.

1.2 1.b

```
import numpy as np

# Given data
time = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
yt = np.array([3, 7, -4, -6, 1, 9, -3, -7, 1, 9, -3, -7])
forecast_1 = np.array([2, 4, -3, -4, 0, 12, -6, -4, 4, 15, -1, -11])
forecast_2 = np.array([7, 9, -8, -10, 7, 9, -8, -10, 7, 9, -8, -10])

#Compute forecast errors for both methods
error_1 = yt - forecast_1
error_2 = yt - forecast_2

# Calculate MAE for both methods
mae_1 = np.mean(np.abs(error_1))
mae_2 = np.mean(np.abs(error_2))

# Calculate Mean Squared Error (MSE) for both methods
mse_1 = np.mean(error_1**2)
mse_2 = np.mean(error_2**2)

# Calculate RMSE for both methods
```

```
rmse_1 = np.sqrt(np.mean(error_1**2))
rmse_2 = np.sqrt(np.mean(error_2**2))

print(f"MAE for Method 1: {mae_1}")
print(f"MAE for Method 2: {mae_2}")
print("---"*20)
print(f"MSE for Method 1: {mse_1}")
print(f"MSE for Method 2: {mse_2}")
print("---"*20)

print(f"RMSE for Method 1: {rmse_1}")
print(f"RMSE for Method 2: {rmse_2}")
print("---"*20)

# print(f"MAPE for Method 1: {mape_1}%")
# print(f"MAPE for Method 2: {mape_2}%")
# print("---"*20)
```

	MAE	**MSE**	**RMSE**
Method 1	2.67	9.0	3.0
Method 2	3.5	16.0	4.0

Table 2: Comparison of Error Metrics for Method 1 and Method 2

From Table 2, it is evident that Method 1 outperforms Method 2, as the error metrics (MAE, MSE, and RMSE) for Method 1 are consistently lower.

1.3 1.c

```
import numpy as np

# Given data
time = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
yt = np.array([3, 7, -4, -6, 1, 9, -3, -7, 1, 9, -3, -7])
forecast_1 = np.array([2, 4, -3, -4, 0, 12, -6, -4, 4, 15, -1, -11])
forecast_2 = np.array([7, 9, -8, -10, 7, 9, -8, -10, 7, 9, -8, -10])

# Select only the last four observations
yt_last_four = yt[-4:]
forecast_1_last_four = forecast_1[-4:]
forecast_2_last_four = forecast_2[-4:]

#Compute forecast errors for both methods
error_1 = yt_last_four - forecast_1_last_four
error_2 = yt_last_four - forecast_2_last_four
```

```
# Calculate MAE for both methods
mae_1 = np.mean(np.abs(error_1))
mae_2 = np.mean(np.abs(error_2))

# Calculate Mean Squared Error (MSE) for both methods
mse_1 = np.mean(error_1**2)
mse_2 = np.mean(error_2**2)

# Calculate RMSE for both methods
rmse_1 = np.sqrt(np.mean(error_1**2))
rmse_2 = np.sqrt(np.mean(error_2**2))
print(f"MAE for Method 1: {mae_1}")
print(f"MAE for Method 2: {mae_2}")
print("---"*20)
print(f"MSE for Method 1: {mse_1}")
print(f"MSE for Method 2: {mse_2}")
print("---"*20)

print(f"RMSE for Method 1: {rmse_1}")
print(f"RMSE for Method 2: {rmse_2}")
print("---"*20)
```

Metric	**Method 1**	**Method 2**
MAE	3.75	3.5
MSE	16.25	17.5
RMSE	4.03	4.18

Table 3: Comparison of Error Metrics for Method 1 and Method 2

From Table 3, we observe that Method 2 performs slightly better than Method 1 in terms of Mean Absolute Error (MAE), while Method 1 has a lower Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). The choice of the preferred method may depend on the specific requirements and priorities.

1.4 1.d

```
import numpy as np
from scipy.stats import linregress

# Given data
time = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
yt = np.array([3, 7, -4, -6, 1, 9, -3, -7, 1, 9, -3, -7])

# Fit a linear trend model
slope, intercept, _, _, _ = linregress(time, yt)
```

```
# Forecast values for t = 1, 2, ..., 12
forecast_linear_trend = intercept + slope * np.arange(1, 13)

# Compare with the previous methods
forecast_1 = np.array([2, 4, -3, -4, 0, 12, -6, -4, 4, 15, -1, -11])
forecast_2 = np.array([7, 9, -8, -10, 7, 9, -8, -10, 7, 9, -8, -10])

print("Actual Values:", yt)
print("Forecast Method 1:", forecast_1)
print("Forecast Method 2:", forecast_2)
print("Forecast Linear Trend Model:", forecast_linear_trend)
---------answer--------
Actual Values: [ 3  7 -4 -6  1  9 -3 -7  1  9 -3 -7]
Forecast Method 1: [  2   4  -3  -4   0  12  -6  -4   4  15  -1 -11]
Forecast Method 2: [  7   9  -8 -10   7   9  -8 -10   7   9  -8 -10]
Forecast Linear Trend Model: [ 2.11538462  1.73076923  1.34615385  0.96153846  0.57692308  0.1923076
 -0.19230769 -0.57692308 -0.96153846 -1.34615385 -1.73076923 -2.11538462]
 -----------------------
 import matplotlib.pyplot as plt

# Given data
forecast_linear_trend
forecast_1
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, forecast_1, label='forecast_1', marker='o')
plt.plot(time, forecast_2, label='forecast_2', marker='o')
plt.plot(time, yt, label='Actual Values', marker='o')

plt.plot(time, forecast_linear_trend, label='linear_trend', linestyle='--', marker='o')

plt.title('Time Series Plot')
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend()
plt.grid(True)
plt.savefig("linear_trend_twomethods.png")
plt.show()


import numpy as np

# Given data
time = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
yt = np.array([3, 7, -4, -6, 1, 9, -3, -7, 1, 9, -3, -7])
forecast_1 = np.array([2, 4, -3, -4, 0, 12, -6, -4, 4, 15, -1, -11])
```

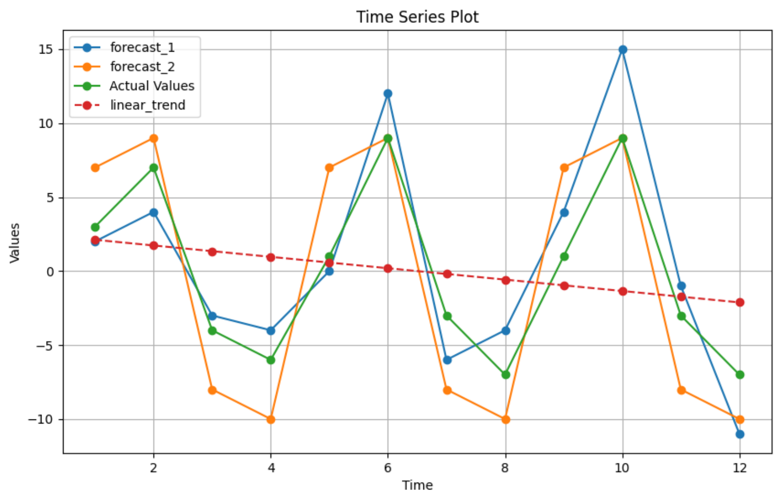


Figure 1: Comparison linear_trend with forecast1 and forecast2

```
forecast_2 = np.array([7, 9, -8, -10, 7, 9, -8, -10, 7, 9, -8, -10])

#Compute forecast errors for both methods
error_1 = yt - forecast_1
error_2 = yt - forecast_2
error_3 = yt - forecast_linear_trend


# Calculate MAE for both methods
mae_1 = np.mean(np.abs(error_1))
mae_2 = np.mean(np.abs(error_2))
mae_3 = np.mean(np.abs(error_3))

# Calculate Mean Squared Error (MSE) for both methods
mse_1 = np.mean(error_1**2)
mse_2 = np.mean(error_2**2)
mse_3 = np.mean(error_3**2)


# Calculate RMSE for both methods
rmse_1 = np.sqrt(np.mean(error_1**2))
rmse_2 = np.sqrt(np.mean(error_2**2))
rmse_3 = np.sqrt(np.mean(error_3**2))
print(f"MAE for Method 1: {mae_1}")
print(f"MAE for Method 2: {mae_2}")
print(f"MAE for linear trend: {mae_3}")

print("---"*20)
print(f"MSE for Method 1: {mse_1}")
print(f"MSE for Method 2: {mse_2}")
print(f"MSE for linear trend: {mse_3}")

print("---"*20)

print(f"RMSE for Method 1: {rmse_1}")
print(f"RMSE for Method 2: {rmse_2}")
print(f"RMSE for linear trend: {rmse_3}")

print("---"*20)
```

Metric	Method 1	Method 2	Linear Trend
MAE	2.67	3.5	4.62
MSE	9.0	16.0	30.74
RMSE	3.0	4.0	5.54

Table 4: Comparison of Error Metrics for Method 1, Method 2, and Linear Trend Model

From Table 4, it is apparent that the linear trend model exhibits higher errors (MAE, MSE, RMSE) compared to both Method 1 and Method 2. The higher values suggest that the linear trend model has a weaker predictive performance in capturing the underlying patterns of the data.

Therefore, based on the given error metrics, it can be concluded that the linear trend model is not as effective as either Method 1 or Method 2 in forecasting the given data. Depending on the specific requirements and priorities, Method 1 or Method 2 may be preferred over the linear trend model for more accurate predictions.

1.5 1.e

```
import numpy as np
import matplotlib.pyplot as plt

def simple_exponential_smoothing(y, alpha):
    """
    Perform simple exponential smoothing on a time series.

    Parameters:
    - y: Time series data as a numpy array.
    - alpha: Smoothing parameter (0 < alpha < 1).

    Returns:
    - Forecasted values.
    """
    n = len(y)
    forecast = np.zeros_like(y, dtype=float)
    forecast[0] = y[0]  # Initial forecast is the same as the first observation

    for t in range(1, n):
        forecast[t] = alpha * y[t-1] + (1 - alpha) * forecast[t-1]

    return forecast

# Choose alpha (smoothing parameter)
alpha = 0.1
# Perform simple exponential smoothing
forecast_ses_1 = simple_exponential_smoothing(yt, alpha)

#Choose alpha (smoothing parameter)
alpha = 0.3

# Perform simple exponential smoothing
forecast_ses_3 = simple_exponential_smoothing(yt, alpha)
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, yt, label='Actual Values', marker='o',color = 'green')
plt.plot(time, forecast_ses_1, label=f'SES Forecast ( = 0.1)', linestyle='--
   ',marker='o',color = 'blue')
plt.plot(time, forecast_ses_3, label=f'SES Forecast ( = 0.3)', linestyle='--',
     marker='o',color = 'purple')

plt.title('Simple Exponential Smoothing Forecasting')
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend()
plt.grid(True)
plt.savefig("exponential_smoothing.png")
plt.show()


# Mean Squared Error (MSE)
def calculate_mse(actual, forecast):
    return np.mean((actual - forecast)**2)

mse_alpha_0_1  = calculate_mse(yt, forecast_ses_1)
mse_alpha_0_3 = calculate_mse(yt, forecast_ses_3)
print(f"MSE for SES Forecast ( = 0.1) : {mse_alpha_0_1 }")
print(f"MSE for SES Forecast ( = 0.3) : {mse_alpha_0_3}")
print("---"*20)
optimal_alpha = 0.1 if mse_alpha_0_1 < mse_alpha_0_3 else 0.3
print("Optimal value of smoothing parameter alpha:", optimal_alpha)
```

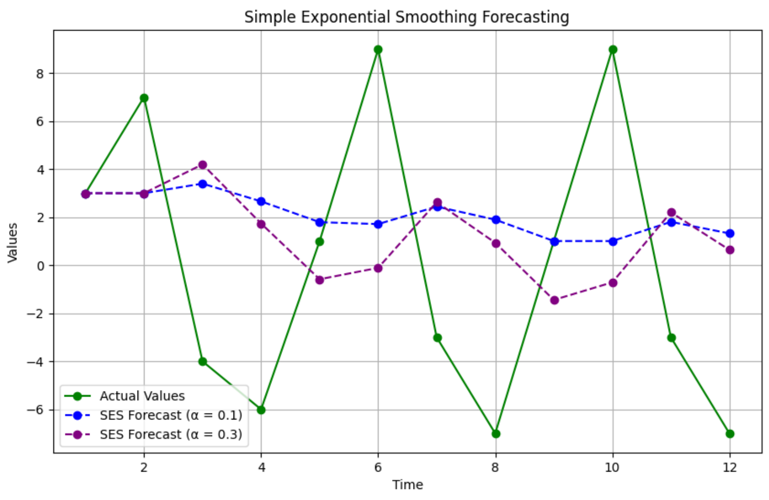


Figure 2: Exponential smoothing

Smoothing Parameter (α)	**MSE for SES Forecast**
0.1	38.72
0.3	42.41

Table 5: Mean Squared Error (MSE) for SES Forecast with Different Smoothing Parameters

The optimal value of the smoothing parameter, α, is determined to be 0.1 based on the analysis of MSE values from 5.

1.6 1.f

```
import numpy as np
from scipy.stats import linregress
import matplotlib.pyplot as plt

# Given data
time = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
yt = np.array([3, 7, -4, -6, 1, 9, -3, -7, 1, 9, -3, -7])

# Fit a linear trend model
slope, intercept, _, _, _ = linregress(time, yt)

# Forecast values for t = 1, 2, ..., 12
```

```
forecast_linear_trend = intercept + slope * np.arange(1, 13)

# Detrend the time series
detrended_series = yt - forecast_linear_trend

# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, detrended_series, label='detrended_series', marker='o',color = 'blue')

plt.title('detrended_series')
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend()
plt.grid(True)
plt.savefig("detrend_series.png")
plt.show()
```

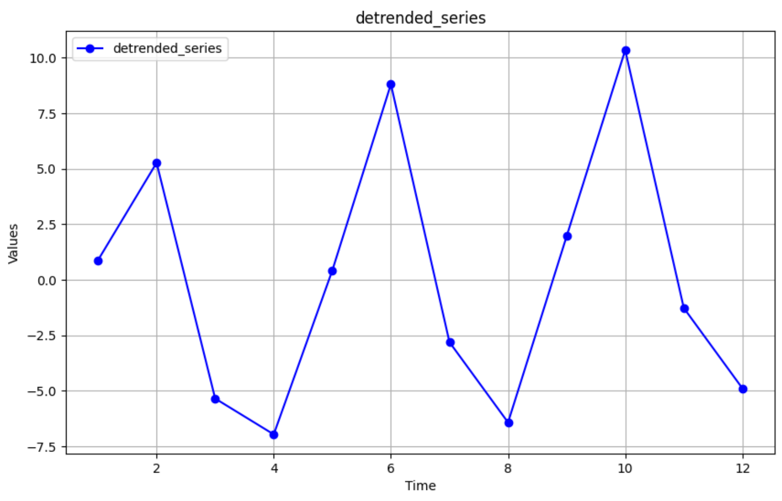


Figure 3: detrend series plot

```
import numpy as np

# Given data
time = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

# Function to estimate seasonal component for a given time t
def estimate_seasonal(t, d, detrended_data):
    quarter_t = (t - 1) % d + 1
    seasonal_period = d
    seasonal_components = [np.mean(detrended_data[i::seasonal_period]) for i in
                range(seasonal_period)]
    quarter_data = seasonal_components[(quarter_t - 1)]
    return quarter_data
estimate_seasonal(24,4,detrended_series)
```

The value of s_{24} is -6.08974358974359

1.7 1.g

```
import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_pacf

# Your time series data
yt = np.array([3, 7, -4, -6, 1, 9, -3, -7, 1, 9, -3, -7])

# Define the AR(4) model without constant term
p = 4

model = sm.tsa.AutoReg(yt, lags=p)
result = model.fit()

# Extract coefficients
ar_coefficients = result.params

# Make predictions
predicted_values = result.predict(start=0, end=len(yt)-1)

# Plot the actual and predicted time series
plt.figure(figsize=(10, 6))
plt.plot(yt, label='Actual', marker='o')
plt.plot(predicted_values, label='Predicted', linestyle='--', marker='o')
plt.title('Actual vs Predicted Time Series')
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend()
plt.savefig("ar4.png")
plt.show()
```

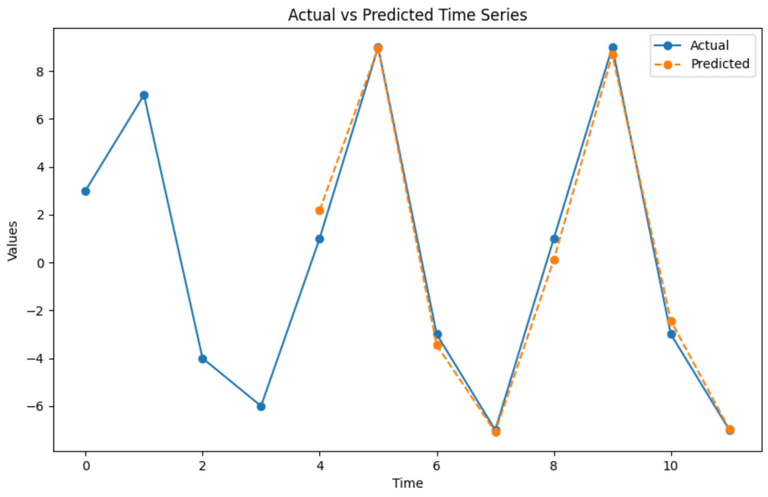


Figure 4: Autoregressive-4 process

```
# Plotting
plt.figure(figsize=(10, 6))
plt.plot(time, yt, label='Actual Values', marker='o')
plt.plot(time, forecast_1, label='Forecast 1', linestyle='--', marker='o')
plt.plot(time, forecast_2, label='Forecast 2', linestyle='--', marker='o')
plt.plot(time, forecast_ar4, label='AR(4) Forecast', linestyle='--', marker='o')

plt.title('Comparison of Forecasts (AR(4) vs Method 1 vs Method 2)')
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend()
plt.grid(True)
plt.savefig("ar4_method2.png")
plt.show()


# Mean Absolute Error (MAE)
def calculate_mae(actual, forecast):
    return np.mean(np.abs(actual - forecast))

mae_1 = calculate_mae(yt, forecast_1)
mae_2 = calculate_mae(yt, forecast_2)
#mae_3 = calculate_mae(yt, forecast_linear_trend)
mae_4 = calculate_mae(yt[4:], forecast_ar4[4:])

# Mean Squared Error (MSE)
```

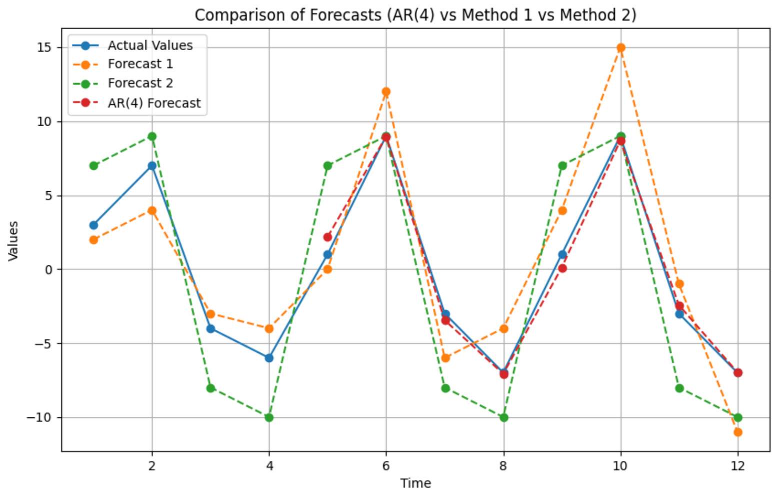


Figure 5: Comparison AR(4) with forecast1 and forecast2

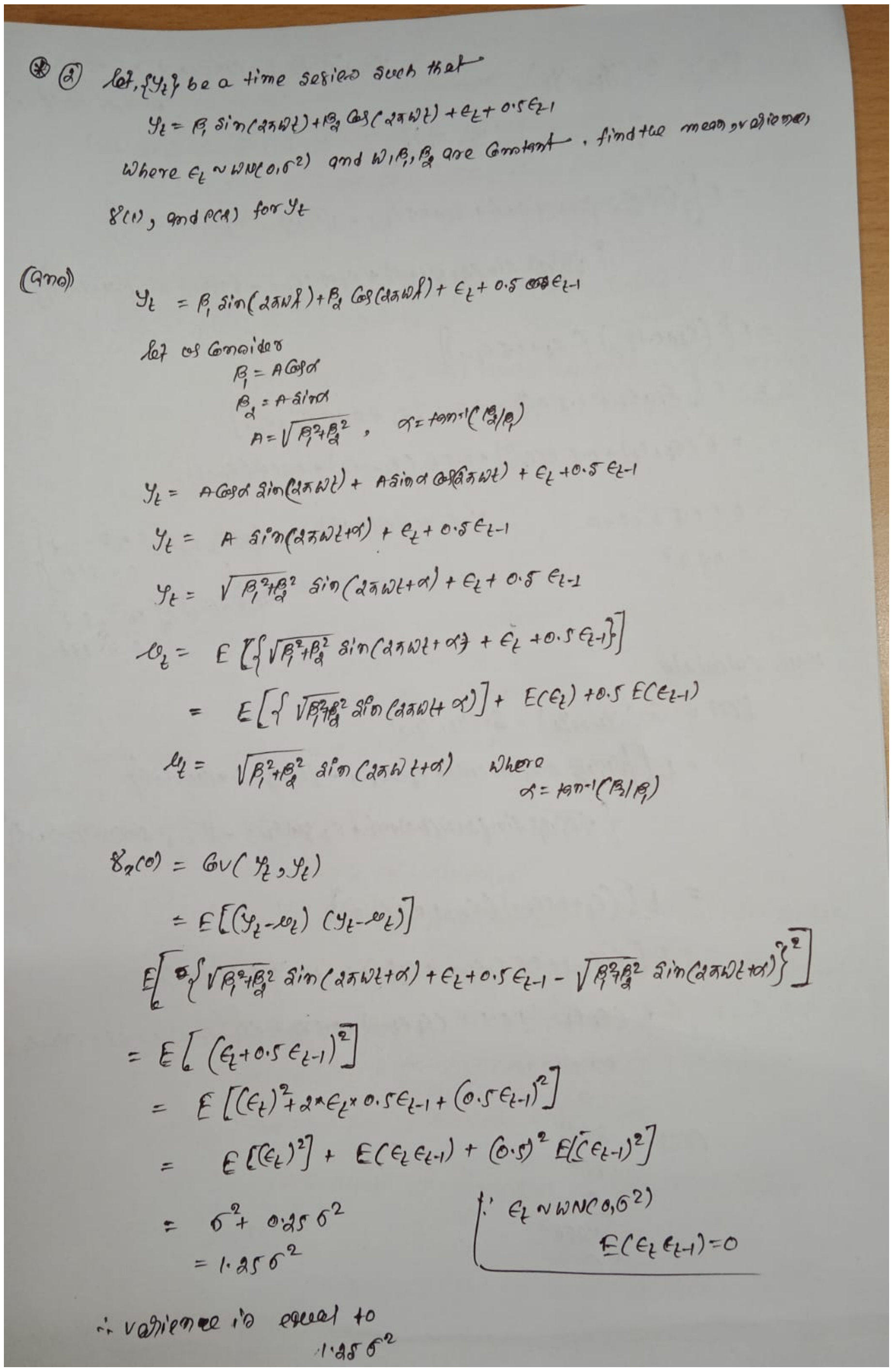
```
def calculate_mse(actual, forecast):
    return np.mean((actual - forecast)**2)

mse_1 = calculate_mse(yt, forecast_1)
mse_2 = calculate_mse(yt, forecast_2)
#mse_3 = calculate_mse(yt, forecast_linear_trend)
mse_4 = calculate_mse(yt[4:], forecast_ar4[4:])

# Root Mean Squared Error (RMSE)
def calculate_rmse(actual, forecast):
    return np.sqrt(np.mean((actual - forecast)**2))

rmse_1 = calculate_rmse(yt, forecast_1)
rmse_2 = calculate_rmse(yt, forecast_2)
#rmse_3 = calculate_rmse(yt, forecast_linear_trend)
rmse_4 = calculate_rmse(yt[4:], forecast_ar4[4:])

--------------------------------------------

print(f"MAE for Method 1: {mae_1}")
print(f"MAE for Method 2: {mae_2}")
print(f"MAE for AR(4): {mae_4}")

print("---"*20)
print(f"MSE for Method 1: {mse_1}")
print(f"MSE for Method 2: {mse_2}")
print(f"MSE for AR(4): {mse_4}")

print("---"*20)

print(f"RMSE for Method 1: {rmse_1}")
print(f"RMSE for Method 2: {rmse_2}")
print(f"RMSE for Ar(4): {rmse_4}")

print("---"*20)
```

Metric	**Method 1**	**Method 2**	**AR(4)**
MAE	2.67	3.5	0.44
MSE	9.0	16.0	0.35
RMSE	3.0	4.0	0.59

Table 6: Comparison of Error Metrics for Method 1, Method 2, and AR(4) model

From Table 6, it is observed that the AR(4) model outperforms both Method 1 and Method 2 in terms of MAE, MSE, and RMSE. The lower values of these error metrics for the AR(4) model suggest better accuracy in predicting the data compared to the other methods.

2 Question2



(2) let, $\{y_t\}$ be a time series such that

$$y_t = \beta_1 \sin(2\pi\omega t) + \beta_2 \cos(2\pi\omega t) + \epsilon_t + 0.5\epsilon_{t-1}$$

where $\epsilon_t \sim WN(0, \sigma^2)$ and ω, β_1, β_2 are constant. find the mean, variance, $\gamma(1)$, and $\rho(1)$ for y_t

(Ans)

$$y_t = \beta_1 \sin(2\pi\omega t) + \beta_2 \cos(2\pi\omega t) + \epsilon_t + 0.5\,\epsilon_{t-1}$$

let us consider

$$\beta_1 = A\cos\alpha$$
$$\beta_2 = A\sin\alpha$$
$$A = \sqrt{\beta_1^2+\beta_2^2}, \quad \alpha = \tan^{-1}(\beta_2/\beta_1)$$

$$y_t = A\cos\alpha \sin(2\pi\omega t) + A\sin\alpha \cos(2\pi\omega t) + \epsilon_t + 0.5\,\epsilon_{t-1}$$

$$y_t = A \sin(2\pi\omega t+\alpha) + \epsilon_t + 0.5\,\epsilon_{t-1}$$

$$y_t = \sqrt{\beta_1^2+\beta_2^2}\,\sin(2\pi\omega t+\alpha) + \epsilon_t + 0.5\,\epsilon_{t-1}$$

$$\mu_t = E\left[\left\{\sqrt{\beta_1^2+\beta_2^2}\,\sin(2\pi\omega t+\alpha) + \epsilon_t + 0.5\,\epsilon_{t-1}\right\}\right]$$

$$= E\left[\left\{\sqrt{\beta_1^2+\beta_2^2}\,\sin(2\pi\omega t+\alpha)\right\}\right] + E(\epsilon_t) + 0.5\,E(\epsilon_{t-1})$$

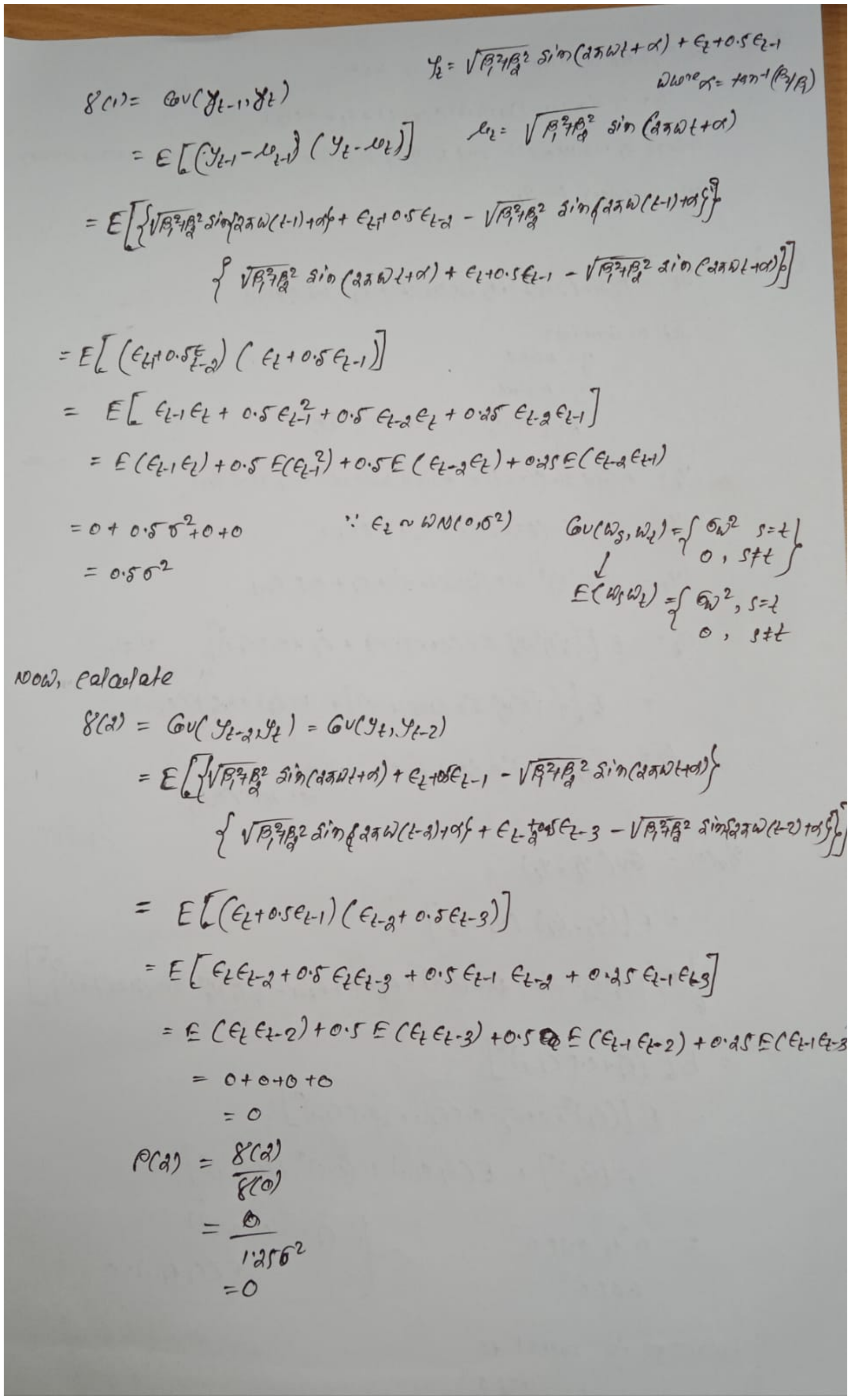
$$\mu_t = \sqrt{\beta_1^2+\beta_2^2}\,\sin(2\pi\omega t+\alpha) \quad \text{where } \alpha = \tan^{-1}(\beta_2/\beta_1)$$

$$\gamma_x(0) = Cov(y_t, y_t)$$
$$= E[(y_t-\mu_t)(y_t-\mu_t)]$$
$$E\left[\left\{\sqrt{\beta_1^2+\beta_2^2}\,\sin(2\pi\omega t+\alpha) + \epsilon_t + 0.5\,\epsilon_{t-1} - \sqrt{\beta_1^2+\beta_2^2}\,\sin(2\pi\omega t+\alpha)\right\}^2\right]$$
$$= E\left[(\epsilon_t + 0.5\,\epsilon_{t-1})^2\right]$$
$$= E\left[(\epsilon_t)^2 + 2*\epsilon_t \times 0.5\epsilon_{t-1} + (0.5\,\epsilon_{t-1})^2\right]$$
$$= E[(\epsilon_t)^2] + E(\epsilon_t\epsilon_{t-1}) + (0.5)^2 E[(\epsilon_{t-1})^2]$$
$$= \sigma^2 + 0.25\sigma^2$$
$$= 1.25\sigma^2$$

$\because \epsilon_t \sim WN(0,\sigma^2)$
$E(\epsilon_t\epsilon_{t-1}) = 0$

$\therefore$ varience is equal to $1.25\sigma^2$

Figure 6: question2



$$y_t = \sqrt{\beta_1^2+\beta_2^2}\sin(2\pi\omega t+\alpha) + \epsilon_t + 0.5\epsilon_{t-1}$$

where $\alpha = \tan^{-1}(\beta_1/\beta_2)$

$$\mu_t = \sqrt{\beta_1^2+\beta_2^2}\sin(2\pi\omega t+\alpha)$$

$$\gamma(1) = Cov(y_{t-1}, y_t)$$

$$= E[(y_{t-1}-\mu_{t-1})(y_t-\mu_t)]$$

$$= E\Big[\{\sqrt{\beta_1^2+\beta_2^2}\sin\{2\pi\omega(t-1)+\alpha\} + \epsilon_{t-1} + 0.5\epsilon_{t-2} - \sqrt{\beta_1^2+\beta_2^2}\sin\{2\pi\omega(t-1)+\alpha\}\}$$
$$\{\sqrt{\beta_1^2+\beta_2^2}\sin(2\pi\omega t+\alpha) + \epsilon_t + 0.5\epsilon_{t-1} - \sqrt{\beta_1^2+\beta_2^2}\sin(2\pi\omega t+\alpha)\}\Big]$$

$$= E[(\epsilon_{t-1}+0.5\epsilon_{t-2})(\epsilon_t+0.5\epsilon_{t-1})]$$

$$= E[\epsilon_{t-1}\epsilon_t + 0.5\epsilon_{t-1}^2 + 0.5\epsilon_{t-2}\epsilon_t + 0.25\epsilon_{t-2}\epsilon_{t-1}]$$

$$= E(\epsilon_{t-1}\epsilon_t) + 0.5E(\epsilon_{t-1}^2) + 0.5E(\epsilon_{t-2}\epsilon_t) + 0.25E(\epsilon_{t-2}\epsilon_{t-1})$$

$$= 0 + 0.5\sigma^2 + 0 + 0$$

$$= 0.5\sigma^2$$

$\because \epsilon_t \sim WN(0,\sigma^2)$

$$Cov(w_s, w_t) = \begin{cases}\sigma_w^2 & s=t\\ 0, & s\neq t\end{cases}$$
$$\downarrow$$
$$E(w_s w_t) = \begin{cases}\sigma_w^2, & s=t\\ 0, & s\neq t\end{cases}$$

Now, calculate

$$\gamma(2) = Cov(y_{t-2}, y_t) = Cov(y_t, y_{t-2})$$

$$= E\Big[\{\sqrt{\beta_1^2+\beta_2^2}\sin(2\pi\omega t+\alpha) + \epsilon_t + 0.5\epsilon_{t-1} - \sqrt{\beta_1^2+\beta_2^2}\sin(2\pi\omega t+\alpha)\}$$
$$\{\sqrt{\beta_1^2+\beta_2^2}\sin\{2\pi\omega(t-2)+\alpha\} + \epsilon_{t-2} + 0.5\epsilon_{t-3} - \sqrt{\beta_1^2+\beta_2^2}\sin\{2\pi\omega(t-2)+\alpha\}\}\Big]$$

$$= E[(\epsilon_t+0.5\epsilon_{t-1})(\epsilon_{t-2}+0.5\epsilon_{t-3})]$$

$$= E[\epsilon_t\epsilon_{t-2} + 0.5\epsilon_t\epsilon_{t-3} + 0.5\epsilon_{t-1}\epsilon_{t-2} + 0.25\epsilon_{t-1}\epsilon_{t-3}]$$

$$= E(\epsilon_t\epsilon_{t-2}) + 0.5E(\epsilon_t\epsilon_{t-3}) + 0.5E(\epsilon_{t-1}\epsilon_{t-2}) + 0.25E(\epsilon_{t-1}\epsilon_{t-3}$$

$$= 0+0+0+0$$

$$= 0$$

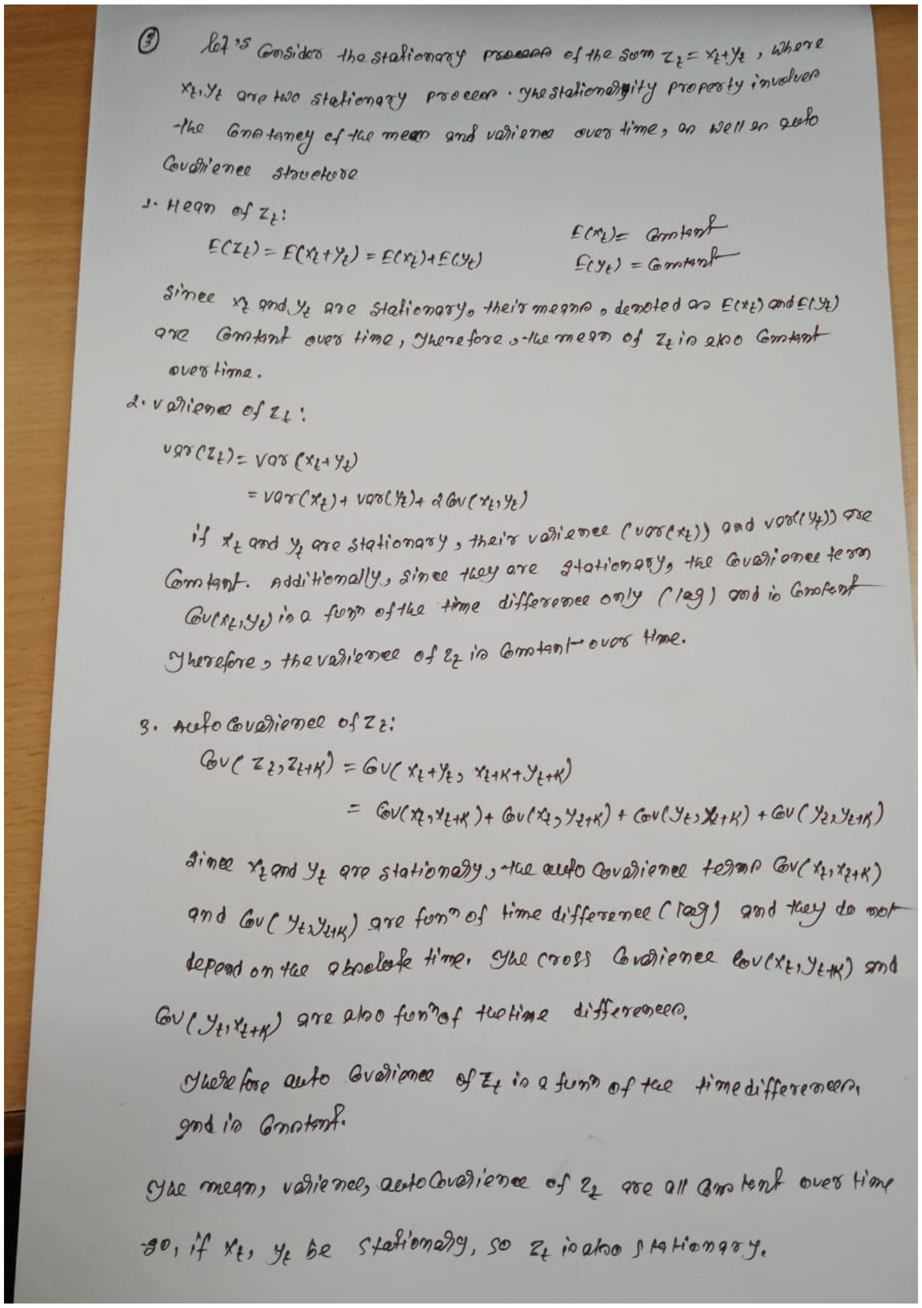
$$\rho(2) = \frac{\gamma(2)}{\gamma(0)}$$

$$= \frac{0}{1.25\sigma^2}$$

$$= 0$$

Figure 7: question2

3 Question3



③ Let's consider the stationary ~~process~~ process of the sum $Z_t = X_t + Y_t$, where X_t, Y_t are two stationary process. The stationarity property involves the constancy of the mean and varience over time, as well as auto covarience structure

1. Mean of Z_t:

$$E(Z_t) = E(X_t + Y_t) = E(X_t) + E(Y_t) \qquad \begin{array}{l} E(X_t) = \text{Constant} \\ E(Y_t) = \text{Constant} \end{array}$$

Since X_t and Y_t are stationary, their means, denoted as $E(X_t)$ and $E(Y_t)$ are constant over time, Therefore, the mean of Z_t is also constant over time.

2. Varience of Z_t:

$$\begin{aligned} var(Z_t) &= var(X_t + Y_t) \\ &= var(X_t) + var(Y_t) + 2Cov(X_t, Y_t) \end{aligned}$$

if X_t and Y_t are stationary, their varience ($var(X_t)$) and $var(Y_t)$) are constant. Additionally, since they are stationary, the covarience term $Cov(X_t, Y_t)$ is a funⁿ of the time difference only (lag) and is constant

Therefore, the varience of Z_t is constant over time.

3. Auto covarience of Z_t:

$$\begin{aligned} Cov(Z_t, Z_{t+K}) &= Cov(X_t + Y_t, X_{t+K} + Y_{t+K}) \\ &= Cov(X_t, X_{t+K}) + Cov(X_t, Y_{t+K}) + Cov(Y_t, X_{t+K}) + Cov(Y_t, Y_{t+K}) \end{aligned}$$

Since X_t and Y_t are stationary, the auto covarience terms $Cov(X_t, X_{t+K})$ and $Cov(Y_t, Y_{t+K})$ are funⁿ of time difference (lag) and they do not depend on the absolute time. The cross covarience $Cov(X_t, Y_{t+K})$ and $Cov(Y_t, X_{t+K})$ are also funⁿ of the time differences.

Therefore auto covarience of Z_t is a funⁿ of the time differences, and is constant.

The mean, varience, auto covarience of Z_t are all constant over time

So, if X_t, Y_t be stationary, so Z_t is also stationary.

Figure 8: question3