Practiacl 1  
Aim: Fitting and plotting of modified exponential curve.   
  
import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

# Generate sample data np.random.seed(0)

x\_data = np.linspace(0, 4, 50)

a, b, c = 2.5, 1.3, 0.5

y\_data = (a \* np.exp(b \* x\_data) + c + 0.5 \*

np.random.normal(size=x\_data.size))

# Define the modified exponential function

def modified\_exponential(x, a, b, c):

return a \* np.exp(b \* x) + c

# Fit the curve

popt, pcov = curve\_fit(modified\_exponential, x\_data, y\_data, p0=(1, 1, 1))

# Get the fitted parameters

a\_fit, b\_fit, c\_fit = popt

print(f"Fitted parameters: a={a\_fit}, b={b\_fit}, c={c\_fit}")

# Generate y values using the fitted parameters

y\_fit = modified\_exponential(x\_data, \*popt)

# Plot the data and the fitted curve

plt.scatter(x\_data, y\_data, label='Data')

plt.plot(x\_data, y\_fit, color='red', label='Fitted curve')

plt.xlabel('x')

plt.ylabel('y')

plt.legend()

plt.show()

Practical 2  
Aim: Fitting and plotting of Gompertz curve.

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

def gompertz(t, a, b, c):

return a \* np.exp(-b \* np.exp(-c \* t))

# Parameters for the true Gompertz function

a\_true = 100

b\_true = 2

c\_true = 0.1

# Generate synthetic data

t\_data = np.linspace(0, 50, 100)

y\_true = gompertz(t\_data, a\_true, b\_true, c\_true)

noise = np.random.normal(0, 5, size=t\_data.shape)

y\_data = y\_true + noise

# Initial guess for the parameters initial\_guess = [90, 1, 0.1]

# Fit the curve

popt, pcov = curve\_fit(gompertz, t\_data, y\_data, p0=initial\_guess)

# Extract the fitted parameters

a\_fit, b\_fit, c\_fit = popt

print(f"Fitted parameters: a = {a\_fit}, b = {b\_fit}, c ={c\_fit}")

# Generate the fitted curve

y\_fit = gompertz(t\_data, a\_fit, b\_fit, c\_fit)

# Plot the data and the fitted curve

plt.figure(figsize=(10, 6))

plt.scatter(t\_data, y\_data, label='Data', color='blue', s=10)

plt.plot(t\_data, y\_true, label='True Gompertz Curve', color='green', linestyle='dashed')

plt.plot(t\_data, y\_fit, label='Fitted Gompertz Curve', color='red')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend()

plt.title('Gompertz Curve Fitting')

plt.show()

# Practical 3

### Aim: Fitting and plotting of logistic curve.

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import curve\_fit

def logistic(t, L, k, t0):

return L / (1 + np.exp(-k \* (t - t0)))

# Parameters for the true logistic function

L\_true = 100

k\_true = 0.1

t0\_true = 10

# Generate synthetic data

t\_data = np.linspace(0, 50, 100)

y\_true = logistic(t\_data, L\_true, k\_true, t0\_true)

noise = np.random.normal(0, 5, size=t\_data.shape)

y\_data = y\_true + noise

# Initial guess for the parameters

initial\_guess = [90, 0.1, 10]

# Fit the curve

popt, pcov = curve\_fit(logistic, t\_data, y\_data, p0=initial\_guess)

# Extract the fitted parameters

L\_fit, k\_fit, t0\_fit = popt

print(f"Fitted parameters: L = {L\_fit}, k = {k\_fit}, t0= {t0\_fit}")

# Generate the fitted curve

y\_fit = logistic(t\_data, L\_fit, k\_fit, t0\_fit)

# Plot the data and the fitted curve

plt.figure(figsize=(10, 6))

plt.scatter(t\_data, y\_data, label='Data', color='blue', s=10)

plt.plot(t\_data, y\_true, label='True Logistic Curve', color='green', linestyle='dashed')

plt.plot(t\_data, y\_fit, label='Fitted Logistic Curve', color='red')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend()

plt.title('Logistic Curve Fitting')

plt.show()

Practical 4

Aim: Fitting of trend by Moving Average Method.

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

np.random.seed(0)

# For reproducibility

# Generate synthetic time series data

time = np.arange(1, 101)

trend = 0.5 \* time

# Linear trend

seasonal = 10 \* np.sin(0.2 \* time)

# Seasonal component

noise = np.random.normal(0, 5, size=time.shape)

# Random

noise

data = trend + seasonal + noise

# Create a pandas DataFrame for convenience

df = pd.DataFrame({'Time': time, 'Data': data})

# Define the window size

window\_size = 5

# Apply the moving average

df['Moving\_Average'] = df['Data'].rolling(window=window\_size).mean()

# Plot the data

plt.figure(figsize=(12, 6))

plt.plot(df['Time'], df['Data'], label='Original Data', color='blue', alpha=0.6)

plt.plot(df['Time'], df['Moving\_Average'], label=f'Moving Average (window={window\_size})', color='red')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend()

plt.title('Trend Fitting Using Moving Average Method')

plt.show()

Practical 5

Aim: Measurement of Seasonal indices Ratio-to-Trend method.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

np.random.seed(0)

# Generate synthetic time series data

periods = 4

# Number of seasons in a year

years = 5

time = np.arange(1, periods \* years + 1)

trend = np.linspace(10, 20, periods \* years)

# Linear trend

seasonal\_pattern = np.tile([1.2, 0.8, 1.5, 0.5], years)

# Seasonal component

noise = np.random.normal(0, 0.1, size=time.shape)

# Random noise

data = trend \* seasonal\_pattern + noise

# Create a pandas DataFrame

df = pd.DataFrame({'Time': time, 'Data': data})

# Define the window size for the moving average (should be the length of one period)

window\_size = periods

# Calculate the moving average as the trend

df['Trend'] = df['Data'].rolling(window=window\_size, center=True).mean()

# Drop NaN values resulting from the moving average calculation

df.dropna(inplace=True)

# Calculate the ratio of actual values to the trend values

df['Ratio'] = df['Data'] / df['Trend']

# Compute the average ratio for each season

seasonal\_indices = df.groupby(df['Time'] % periods)['Ratio'].mean()

# Normalize the seasonal indices to have an average of 1

seasonal\_indices /= seasonal\_indices.mean()

print("Seasonal Indices:")

print(seasonal\_indices)

# Plot the original data and the trend

plt.figure(figsize=(12, 6))

plt.plot(df['Time'], df['Data'], label='Original Data', color='blue', alpha=0.6)

plt.plot(df['Time'], df['Trend'], label='Trend (Moving Average)', color='red')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend()

plt.title('Original Data and Trend')

plt.show()

# Plot the seasonal indices

plt.figure(figsize=(12, 6))

plt.bar(seasonal\_indices.index, seasonal\_indices.values, color='green')

plt.xlabel('Season')

plt.ylabel('Seasonal Index')

plt.title('Seasonal Indices (Ratio-to-Trend Method)')

plt.show()

Practical 6

Aim: Measurement of Seasonal indices Ratio-to-Moving Average method.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

np.random.seed(0)

# Generate synthetic time series data

periods = 4

# Number of seasons in a year

years = 5

time = np.arange(1, periods \* years + 1)

trend = np.linspace(10, 20, periods \* years)

# Linear trend

seasonal\_pattern = np.tile([1.2, 0.8, 1.5, 0.5], years)

# Seasonal component

noise = np.random.normal(0, 0.1, size=time.shape)

# Random noise

data = trend \* seasonal\_pattern + noise

# Create a pandas DataFrame

df = pd.DataFrame({'Time': time, 'Data': data})

# Define the window size for the moving average (should be the length of one period)

window\_size = periods

# Calculate the moving average as the trend

df['Moving\_Average'] = df['Data'].rolling(window=window\_size, center=True).mean()

# Drop NaN values resulting from the moving average calculation

df.dropna(inplace=True)

# Calculate the ratio of actual values to the moving average values

df['Ratio'] = df['Data'] / df['Moving\_Average']

# Compute the average ratio for each season

df['Season'] = df['Time'] % periods

seasonal\_indices = df.groupby('Season')['Ratio'].mean()

# Normalize the seasonal indices to have an average of 1

seasonal\_indices /= seasonal\_indices.mean()

print("Seasonal Indices:")

print(seasonal\_indices)

# Plot the original data and the moving average

plt.figure(figsize=(12, 6))

plt.plot(df['Time'], df['Data'], label='Original Data', color='blue', alpha=0.6)

plt.plot(df['Time'], df['Moving\_Average'], label='Moving Average', color='red')

plt.xlabel('Time')

plt.ylabel('Value')

plt.legend()

plt.title('Original Data and Moving Average')

plt.show()

# Plot the seasonal indices

plt.figure(figsize=(12, 6))

plt.bar(seasonal\_indices.index, seasonal\_indices.values,

color='green')

plt.xlabel('Season')

plt.ylabel('Seasonal Index')

plt.title('Seasonal Indices (Ratio-to-Moving Average Method)')

plt.show()

Practical 7

Aim: Measurement of seasonal indices Link Relative method.

# Example sales data for 3 years (36 months)

np.random.seed(0)

data = {

'Year': np.repeat([1, 2, 3], 12),

'Month': list(range(1, 13)) \* 3,

'Sales': np.random.randint(80, 120, size=36)

}

df = pd.DataFrame(data)

# Calculate Link Relatives

df['Link\_Relative'] = df['Sales'] / df['Sales'].shift(1)

df['Link\_Relative'].iloc[::12] = np.nan

# Reset for the start of each year

# Calculate Average Link Relatives

average\_link\_relatives = df.groupby('Month')['Link\_Relative'].mean()

# Adjusting so the average of indices is 100

average\_link\_relatives = average\_link\_relatives.fillna(1)

# Replace NaN with 1 for calculation

seasonal\_indices = (average\_link\_relatives / average\_link\_relatives.mean()) \* 100

seasonal\_indices

Practical 8

Aim: Measurement of seasonal indices Link Relative method.

Step-by-Step Example

1. Collect Data: Let's consider a time series of monthly sales data for 3 years

(36 months).

2. Calculate Differences: Calculate the first differences of the data.

3. Calculate the Variance of Differences: Compute the variance of these

differences.

Step 1: Collect Data

import numpy as np

import pandas as pd

# Example sales data for 3 years (36 months)

np.random.seed(0)

data = {

'Year': np.repeat([1, 2, 3], 12),

'Month': list(range(1, 13)) \* 3,

'Sales': np.random.randint(80, 120, size=36)

}

df = pd.DataFrame(data)

df.head()

2. Calculate Differences:

df['Difference'] = df['Sales'].diff()

df = df.dropna()

# Remove NaN values resulting from the difference calculation

df.head()

variance\_difference = df['Difference'].var()

variance\_difference

Let's implement this in Python and calculate the variance of the random

component.

import numpy as np

import pandas as pd   
  
3. Step 3

# Example sales data for 3 years (36 months)

np.random.seed(0)

data = {

'Year': np.repeat([1, 2, 3], 12),

'Month': list(range(1, 13)) \* 3,

'Sales': np.random.randint(80, 120, size=36)

}

df = pd.DataFrame(data)

# Calculate Differences

df['Difference'] = df['Sales'].diff()

df = df.dropna()

# Remove NaN values resulting from the difference calculation

# Calculate the Variance of Differences

variance\_difference = df['Difference'].var()

variance\_difference

Practical 9

Aim: Forecasting by exponential smoothing.