

**MCA 3rd semester**

**Python programming**

**Semester examination**

**Topic: Scipy**

**Submitted by**

Siri Paladi

**Under the guidance**

Krishnan R

**Python program on Scipy module**

import numpy as np

import matplotlib.pyplot as plt

from scipy.integrate import quad, solve\_ivp

from scipy.optimize import minimize, root

from scipy.linalg import lu

from scipy.signal import butter, sosfilt

from scipy.stats import ttest\_ind

from scipy.sparse import csr\_matrix

from scipy.interpolate import interp1d

from scipy.cluster.hierarchy import linkage, dendrogram

from scipy.spatial.distance import pdist

from scipy.fft import fft

from scipy.special import gamma

from sklearn.decomposition import PCA

from scipy.ndimage import gaussian\_filter

# 1. Numerical Integration

def numerical\_integration(func, a, b):

    result, error = quad(func, a, b)

    return result

print("1. Numerical Integration:", numerical\_integration(lambda x: x\*\*2, 0, 1))

# 2. Optimization

def optimize\_function(func, x0):

    result = minimize(func, x0)

    return result

result = optimize\_function(lambda x: (x - 3)\*\*2, x0=0)

print("\n2. Optimization Result:", result)

# 3. Linear Algebra

def lu\_decomposition(matrix):

    P, L, U = lu(matrix)

    return P, L, U

P, L, U = lu\_decomposition(np.array([[4, 3], [6, 3]]))

print("\n3. LU Decomposition:\nP:\n", P, "\nL:\n", L, "\nU:\n", U)

# 4. Signal Processing

def butterworth\_filter(data, cutoff, fs, order=5):

    sos = butter(order, cutoff, fs=fs, output='sos')

    filtered = sosfilt(sos, data)

    return filtered

data = np.sin(np.linspace(0, 10, 100))

filtered\_data = butterworth\_filter(data, cutoff=0.1, fs=10)

plt.figure()

plt.plot(data, label="Original")

plt.plot(filtered\_data, label="Filtered")

plt.legend()

plt.title("4. Butterworth Filter")

plt.show()

# 5. Statistics

def perform\_ttest(data1, data2):

    stat, p\_value = ttest\_ind(data1, data2)

    return stat, p\_value

stat, p\_value = perform\_ttest([1, 2, 3], [4, 5, 6])

print("\n5. T-Test: stat=", stat, ", p-value=", p\_value)

# 6. Sparse Matrix Operations

def sparse\_matrix\_operations(data):

    sparse = csr\_matrix(data)

    return sparse

sparse = sparse\_matrix\_operations([[1, 0, 0], [0, 0, 1], [1, 0, 0]])

print("\n6. Sparse Matrix:\n", sparse.toarray())

# 7. Interpolation

def interpolate\_data(x, y, x\_new):

    # Changed kind to 'linear' or 'quadratic' as cubic needs 4 or more data points.

    interpolation = interp1d(x, y, kind='linear')

    return interpolation(x\_new)

x\_new = np.linspace(0, 2, 100)

y\_interp = interpolate\_data([0, 1, 2], [0, 1, 4], x\_new)

# 8. Root Finding

def find\_root(func, x0):

    result = root(func, x0)

    return result

root\_result = find\_root(lambda x: x\*\*2 - 4, x0=1)

print("\n8. Root Finding Result:", root\_result.x)

# 9. ODE Solver

def solve\_ode(func, t\_span, y0):

    result = solve\_ivp(func, t\_span, y0)

    return result

ode\_result = solve\_ode(lambda t, y: -y, [0, 5], [1])

plt.figure()

plt.plot(ode\_result.t, ode\_result.y[0], label="ODE Solution")

plt.title("9. ODE Solution")

plt.legend()

plt.show()

# 10. Multivariate Data Analysis (PCA)

def perform\_pca(data, n\_components):

    pca = PCA(n\_components=n\_components)

    pca.fit(data)

    return pca.components\_

data = np.array([[1, 2], [3, 4], [5, 6]])

components = perform\_pca(data, 1)

print("\n10. PCA Components:", components)

# 11. Image Processing

def apply\_gaussian\_filter(image, sigma):

    filtered\_image = gaussian\_filter(image, sigma=sigma)

    return filtered\_image

image = np.array([[1, 2], [3, 4]])

filtered\_image = apply\_gaussian\_filter(image, sigma=1)

print("\n11. Filtered Image:\n", filtered\_image)

# 12. Monte Carlo Simulation

def monte\_carlo\_simulation(trials):

    samples = np.random.rand(trials)

    return samples.mean()

mean = monte\_carlo\_simulation(1000)

print("\n12. Monte Carlo Simulation Mean:", mean)

# 13. Fast Fourier Transform

def perform\_fft(data):

    transformed = fft(data)

    return transformed

data = [1, 2, 3, 4]

fft\_result = perform\_fft(data)

print("\n13. FFT Result:", fft\_result)

# 14. Special Functions

def compute\_gamma(value):

    return gamma(value)

gamma\_value = compute\_gamma(5)

print("\n14. Gamma Function Result:", gamma\_value)

# 15. Clustering with Distance Metrics

def hierarchical\_clustering(data):

    distances = pdist(data, metric='euclidean')

    linked = linkage(distances, method='ward')

    dendrogram(linked)

    plt.title("15. Hierarchical Clustering Dendrogram")

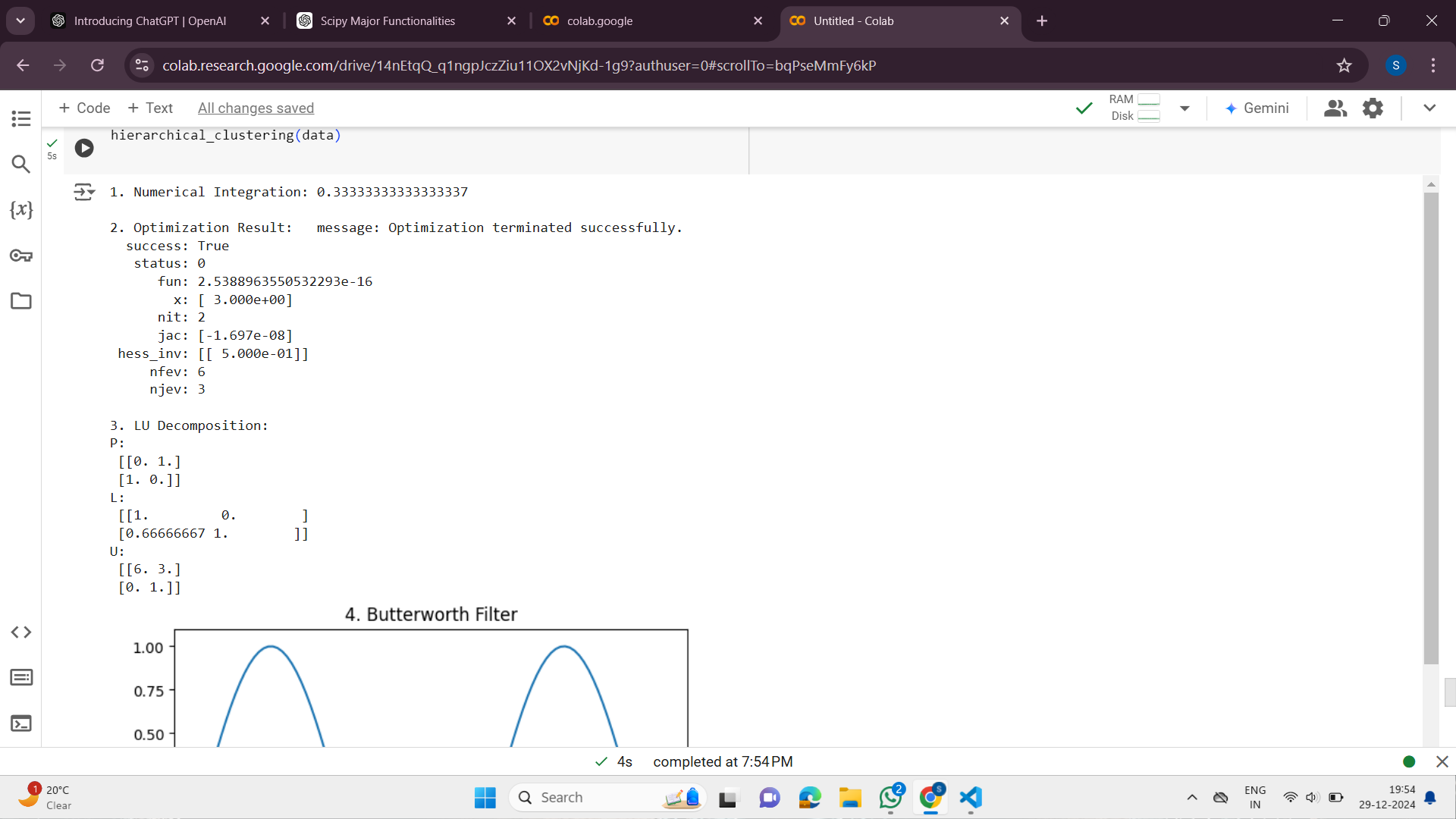
    plt.show()

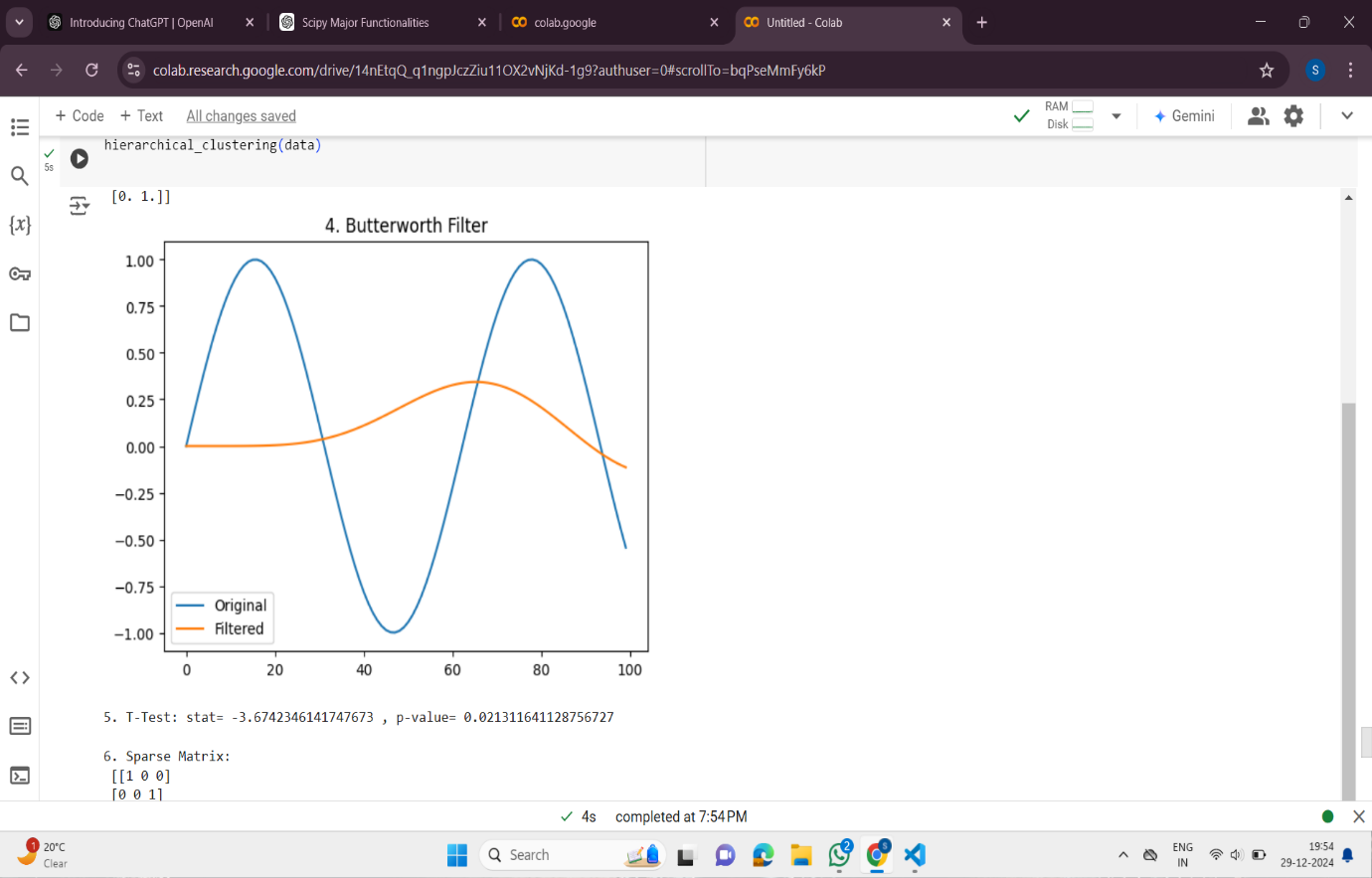
data = [[1, 2], [3, 4], [5, 6]]

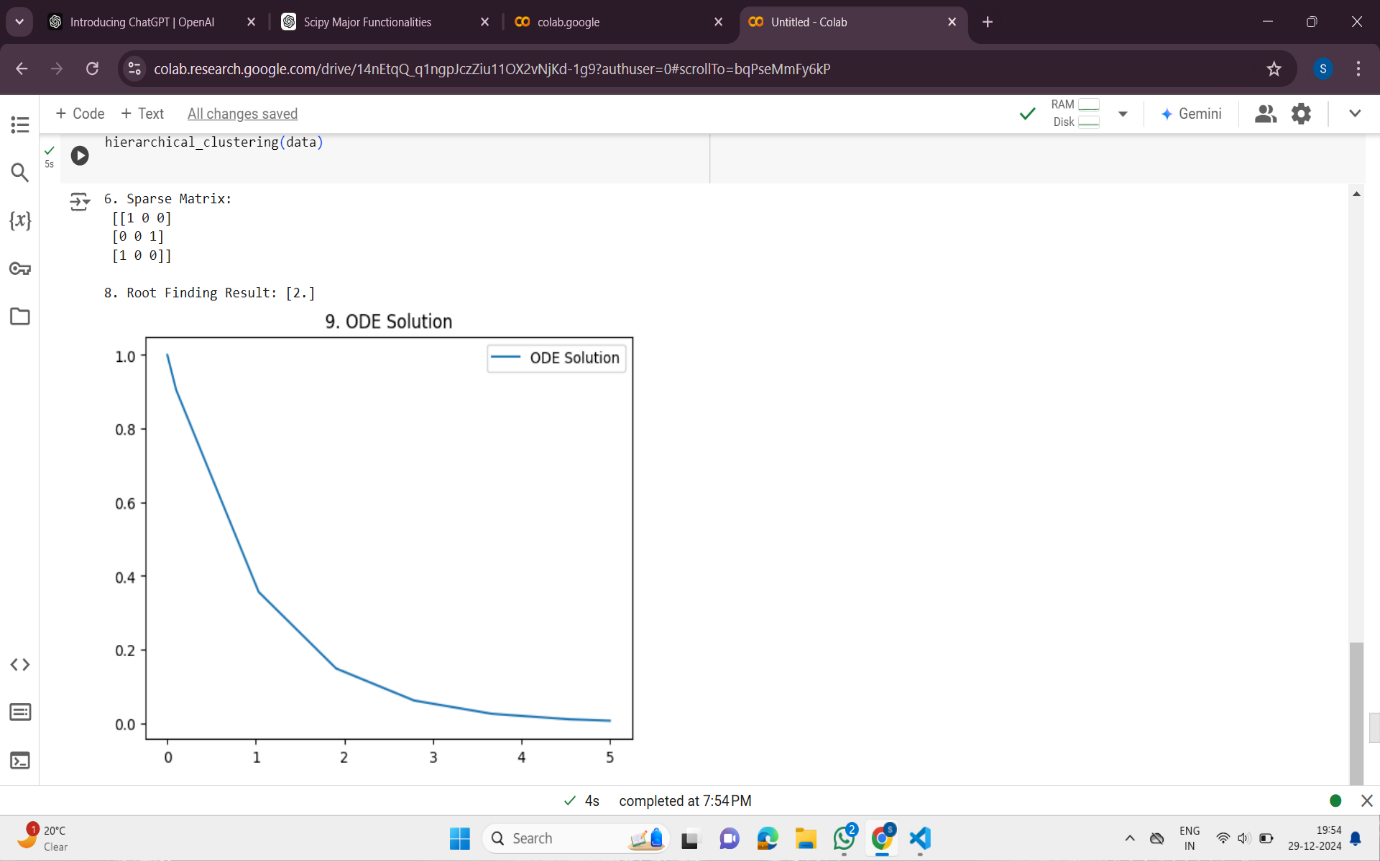
print("\n15. Hierarchical Clustering Dendrogram:")

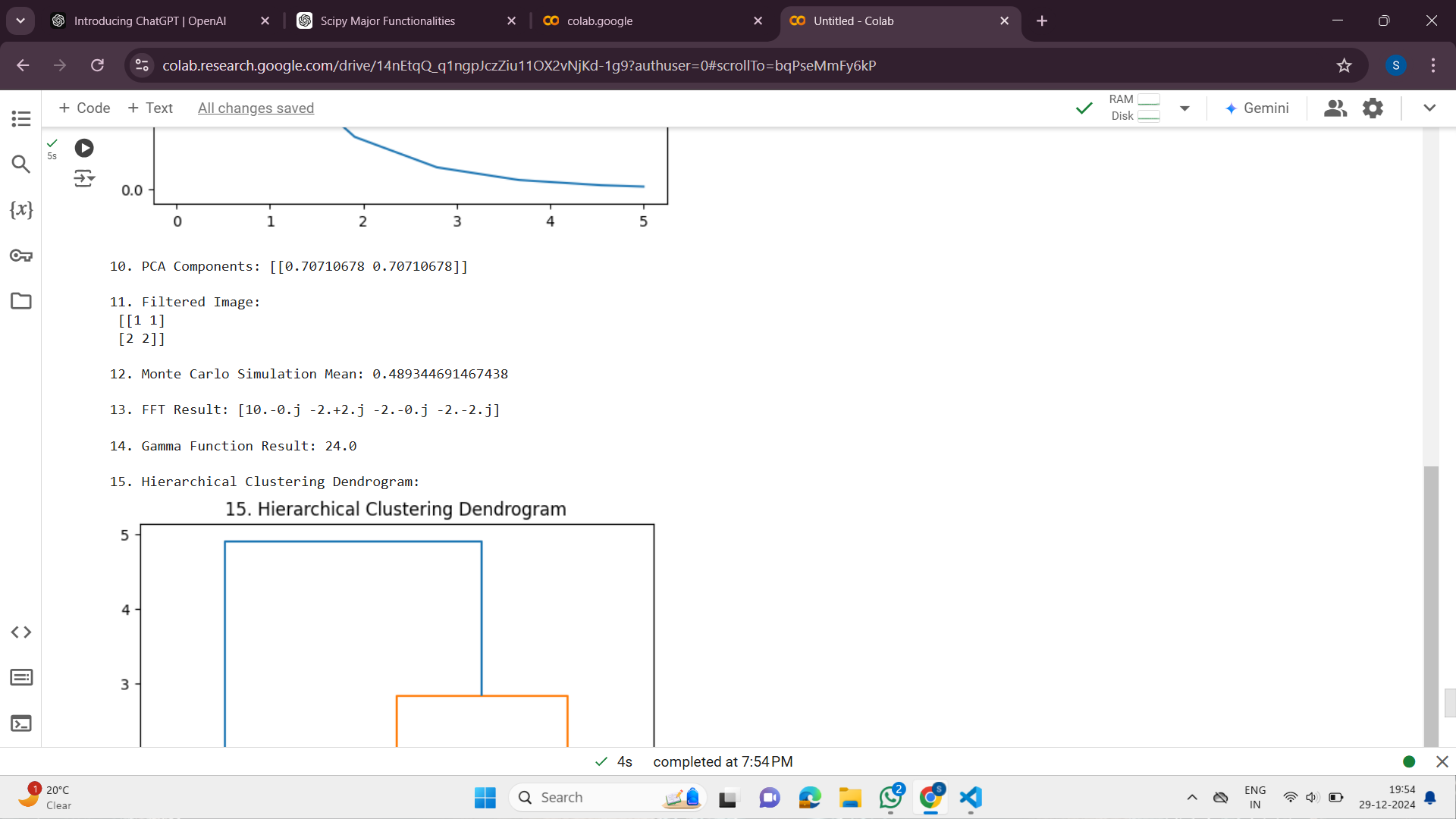
hierarchical\_clustering(data)

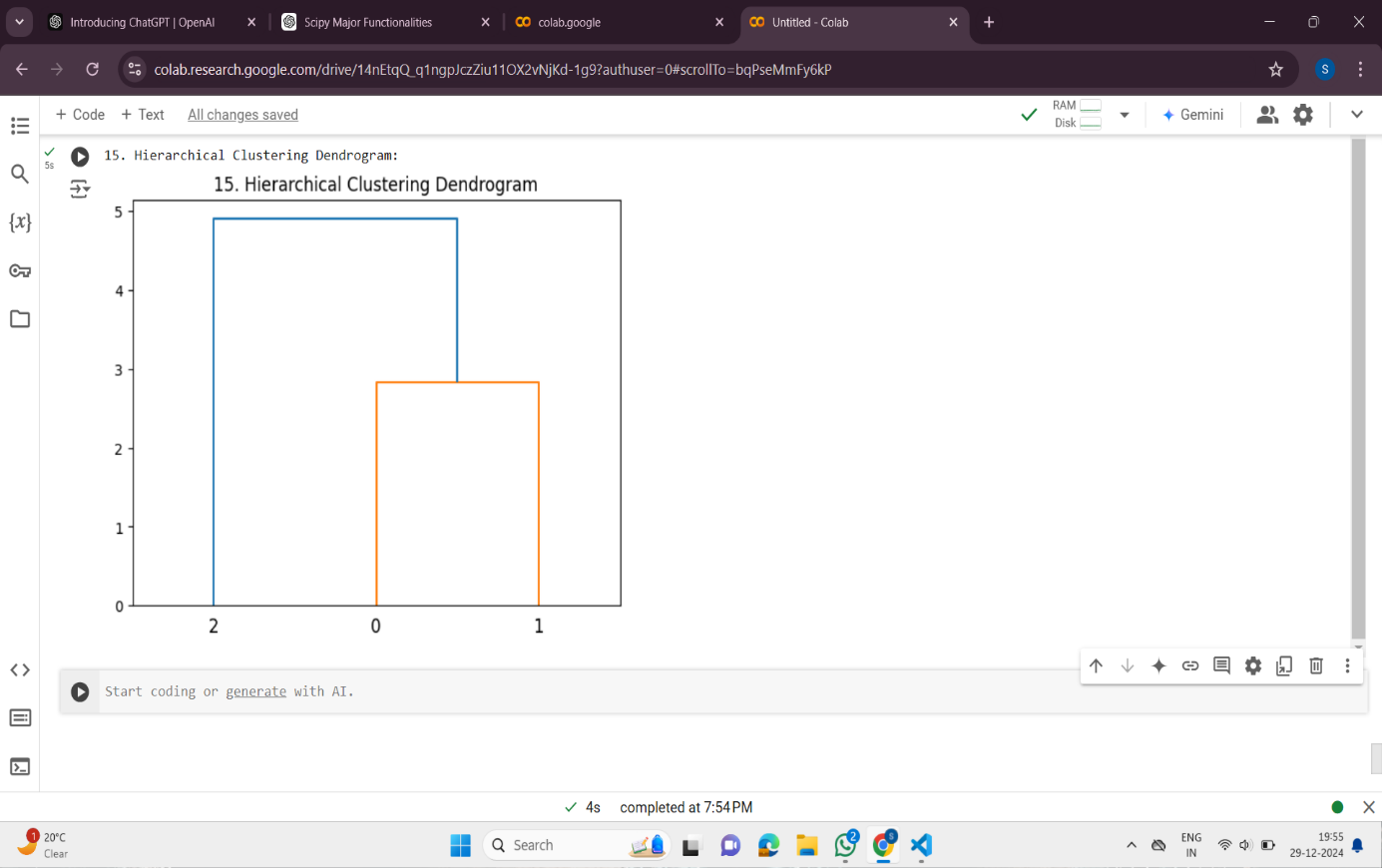
**output:**

****

****

****

****

****