INFORMATICS INSTITUTE OF TECHNOLOGY

In Collaboration with

ROBERT GORDON UNIVERSITY ABERDEEN

<u>Department of Computing</u>

Name: A.M Sandushke De Alwis

IIT number: 20200244

RGU number : 2117852

Module: CM2607 Advanced Mathematics for Data Science

Module Leader: Mrs. Ganesha Thondilege

Tutor: Ruwan Egodawatte

Handout Date: 3rd week of October

Submission Date: 27th of December 2022

Acknowledge

I would like to take this opportunity to thank all the IIT staff for giving us the opportunity to follow the degree artificial intelligence & data science and also l specially thank my lecturer and tutor for giving me the knowledge to do this report. Rather than the academic knowledge I got a lot of information while gathering information. So, at last, I would like to take the opportunity to thank for all the feedbacks that you provided for me and it really helped me to correct all our mistakes. Thank you

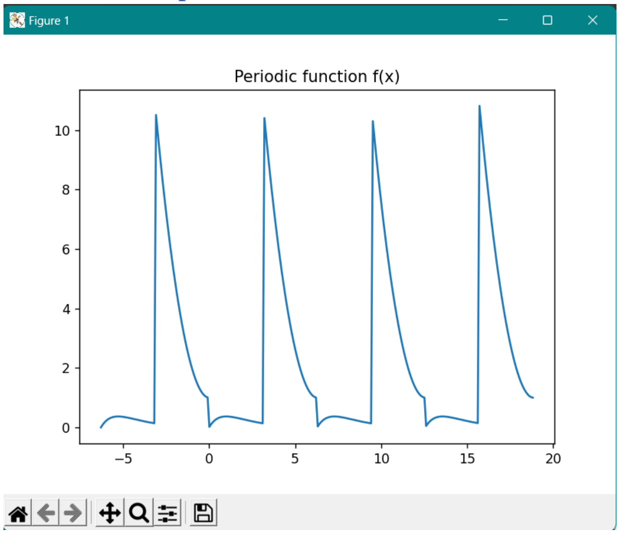
Question 1

a)

Code

```python
# a)
import matplotlib.pyplot as plt
import numpy as np

def f(x):
    if (0 > x >= -np.pi):
        return x ** 2 + 1

    if (np.pi >= x >= 0):
        return x * np.exp(-x)

    if (x < -np.pi):
        z = x+(2*np.pi)
        result = f(z)

    if (x > np.pi):
        z = x - (2 * np.pi)
        result = f(z)

    return result

# Create a sequence between -4.pi to +4.pi
xValues = np.linspace(-4*np.pi, 4*np.pi, 1000)
period = 2 * np.pi + xValues

yValues = [f(x) for x in period]

# Plot the function
plt.title("Periodic function f(x)")
plt.xlabel=("y")
plt.ylabel =("x")
plt.plot(period, yValues)
plt.show()
```

Screenshot of the output



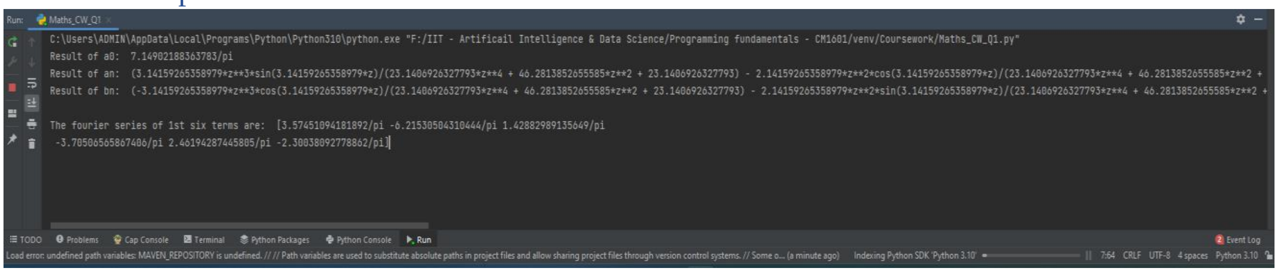
b)

Code

```
# b)
import numpy as np
import sympy as sym

x = sym.symbols("x")
z = sym.symbols("z")

seq = np.empty(6, dtype=object)

func1 = x**2 +1
func2 = x*sym.exp(-1*x)

#equation for a0
a0 = (1 / (2 * sym.pi)) * (func1.integrate((x, -np.pi, 0)) +
func2.integrate((x,0,np.pi)))
#equation for an
an = (1 / sym.pi) * (sym.integrate((func1 * sym.cos(z * x)), (x, -1 * np.pi,
0)) + sym.integrate((func2 * sym.cos(z * x)), (x, 0, np.pi)))
#equation for bn
bn = (1 / sym.pi) * (sym.integrate((func1 * sym.sin(z * x)), (x, -1 * np.pi,
0)) + sym.integrate((func2 * sym.sin(z * x)), (x, 0, np.pi)))

#Print the results
print("Result of a0: ", a0)
print("Result of an: ", an)
print("Result of bn: ", bn)

seq[0] = a0 / 2

#Print the an series
answer = 1
```

```python
for a in range(1, 3):
    an = (1 / sym.pi) * (sym.integrate((func1 * sym.cos(a * x)), (x, -1 *
np.pi, 0)) + sym.integrate((func2 * sym.cos(a * x)), (x, 0, np.pi)))
    seq[answer] = an
    answer+=1

#Print the bn series
answer = 3
for b in range(3, 6):
    bn = (1 / sym.pi) * (sym.integrate((func1 * sym.sin(b * x)), (x, -1 *
np.pi, 0)) + sym.integrate((func2 * sym.sin(b * x)), (x, 0, np.pi)))
    seq[answer] = bn
    answer +=1

print("")
print("The fourier series of 1st six terms are: ", seq, end="")
```

Screenshot of the output



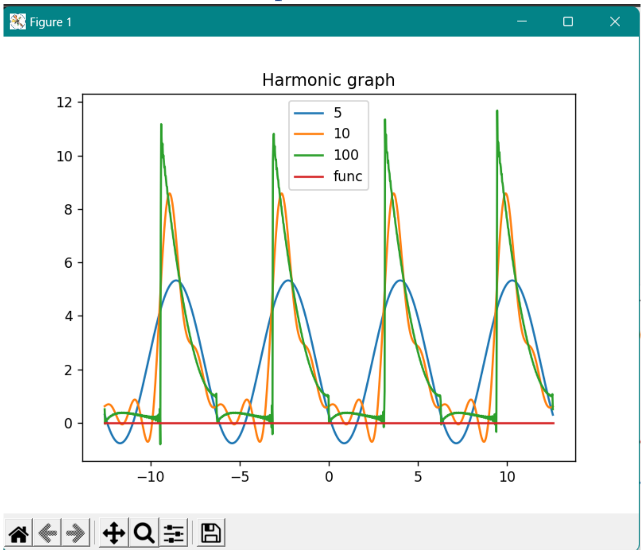
c)

Code

```python
# c)
import matplotlib.pyplot as plt
import numpy as np
import sympy as sym

x = sym.symbols("x")
z = sym.symbols("z")

seq = np.empty(150, dtype=object)
xRange = np.linspace(-4 * np.pi, 4 * np.pi, 1000)
y = np.zeros([151, 1000])

func1 = x ** 2 +1
func2 = x * sym.exp(-x)

#equation for a0
a0 = (1 / (2 * sym.pi)) * (func1.integrate((x, -np.pi, 0)) +
func2.integrate((x,0,np.pi)))
#equation for anu
an = (1 / sym.pi) * (sym.integrate((func1 * sym.cos(z * x)), (x, -1 * np.pi,
0)) + sym.integrate((func2 * sym.cos(z * x)), (x, 0, np.pi)))
```

```python
#equation for bn
bn = (1 / sym.pi) * (sym.integrate((func1 * sym.sin(z * x)), (x, -1 * np.pi,
0)) + sym.integrate((func2 * sym.sin(z * x)), (x, 0, np.pi)))

seq[0] = a0
f1 = sym.lambdify(x, seq[0], 'numpy')
y[0, :] = f1(xRange)

for i in range(1, 150):
    seq[i] = seq[i - 1] + an.subs(z, i) * sym.cos(i * x) + bn.subs(z, i) *
sym.sin(i * x)
    #print (n+1, ":", ms[z])
    f1 = sym.lambdify(x, seq[i], 'numpy')
    y[i, :] = f1(xRange)

for value in range(0, 1000):
    if 0 < xRange[value] < np.pi:
        y[150, value] = xRange[value] ** 2
    elif 2 * np.pi < xRange[value] <= 3 * np.pi:
        y[150, value] = (xRange[value] - 2 * np.pi) ** 2
    elif -2 * np.pi < xRange[value] <= np.pi:
        y[150, value] = (xRange[value] + 2 * np.pi) ** 2


plt.plot(xRange, y[0, :])
plt.plot(xRange, y[4, :])
plt.plot(xRange, y[149, :])
plt.plot(xRange, y[150, :])

#Plot the graph
plt.title("Harmonic graph")
plt.legend(["5", "10", "100", "func"])
plt.show()
```

Screenshot of the output

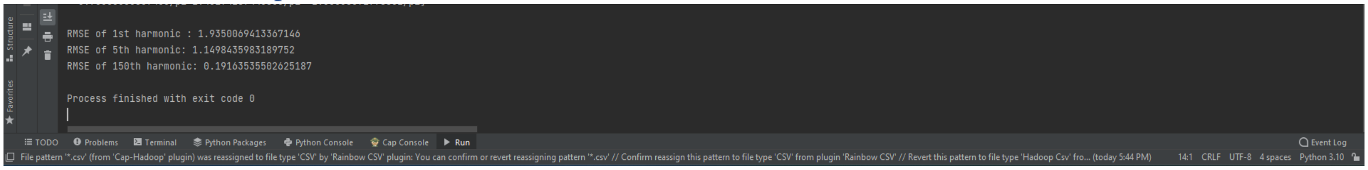


d)

Code

```
# d)
import math

#To get the RMSE between f(x) and 1st harmonic value
absolvalue = [y[1, :]]
pred = [yValues]
meanSquareError = np.square(np.subtract(absolvalue, pred)).mean()
#Root mean square error
rootSquareMeanError = math.sqrt(meanSquareError)
print("")
print("")
print("RMSE of 1st harmonic :", rootSquareMeanError)


#To get the RMSE between f(x) and 5th harmonic value
absolvalue = [y[4, :]]
meanSquareError = np.square(np.subtract(absolvalue, pred)).mean()
#Root mean square error
rootSquareMeanError = math.sqrt(meanSquareError)
print("RMSE of 5th harmonic:", rootSquareMeanError)


#To get the RMSE between f(x) and 150th harmonic value
absolvalue = [y[149, :]]
meanSquareError = np.square(np.subtract(absolvalue, pred)).mean()
#Root mean square error
rootSquareMeanError = math.sqrt(meanSquareError)
print("RMSE of 150th harmonic:", rootSquareMeanError)
```

Screenshot of the output



Question 2

```
"""" Aliasing occurs when the sampled signal is not sampled at a high enough
rate, which could result in overlapping
frequency components in the frequency domain(the frequency would be twice the
highest frequency present in the signal).
When DFT is implemented on the sampled signal, the higher frequency in the
signal gets 'aliased' to lower frequency
which would lead to incorrect results
"""
```

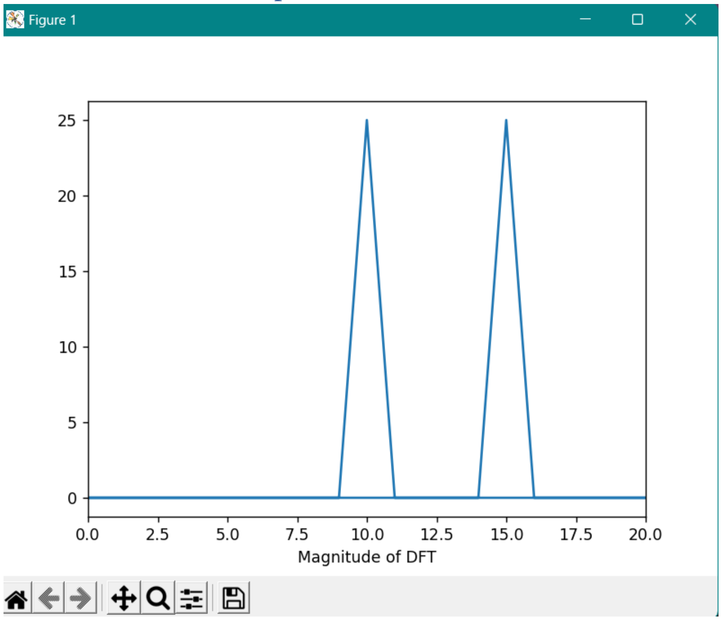
Code

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

# Get a signal with high frequency
sampleRate = 50
time = 1/sampleRate

range = np.arange(0, 1, time)

y = np.sin(2 * np.pi * 10 * range) + np.sin(2 * np.pi * 15 * range)

#DFT signal
x = np.fft.fft(y)

#frequency axis
freq = np.fft.fftfreq(len(y), time)

#Plot the DFT graph
plt.plot(freq, np.abs(x))
plt.xlim(0, 20)
plt.xlabel("Frequency in Hz")
plt.xlabel("Magnitude of DFT")
plt.show()
```

```
"""" In the above code, two frequency signals were generated with 10Hz and
50Hz.  When the numpy.ffy.fft is used,
it needs to plot a graph with two peaks DFT magnitude at frequencies of 10
and 50Hz. But because of aliasing, it will
show some extra peaks in other frequencies(change in frequency values) if the
sample rate is not high enough. Therefore,
it concludes that if the sampling rate is not accurate to represent the high
```

```
frequency of the signals, the result would aliased
"""
```

Screenshot of the output

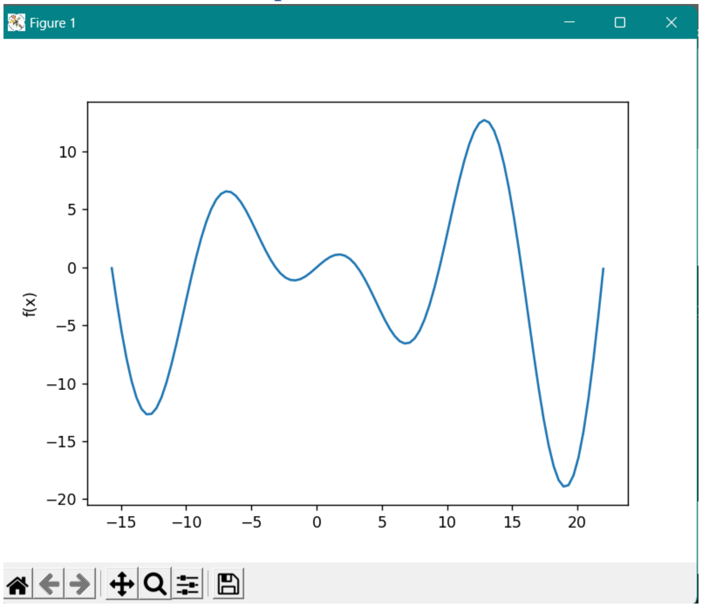


Question 3

Code

```
# a)
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(5 *-3.14, 7 * 3.14, 100)

def f(x):
    return x * np.cos(x/2)

plt.ylabel("f(x)")
plt.title("")
plt.plot(x, f(x))
plt.show()
```

Screenshot of the output

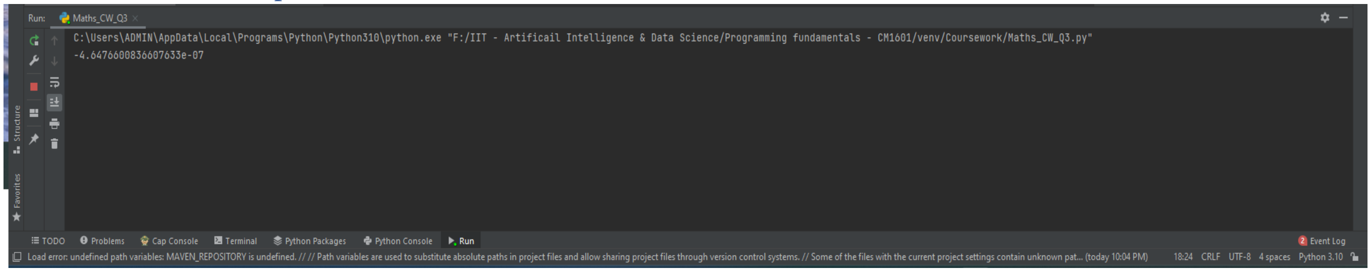


b)

Code

```
# b)
from math import pi, sin
import math

def taylorCosSum(x, values):
    # Initialize to 1
    result = 1

    for i in range(1, values + 1):
        numerator = (-1) ** i * x ** (2 * i)
        denominator = math.factorial(2 * i)
        result += numerator/denominator

    return result

x = math.pi/2
values = 5
print(taylorCosSum(x, values))
```

Screenshot of the output

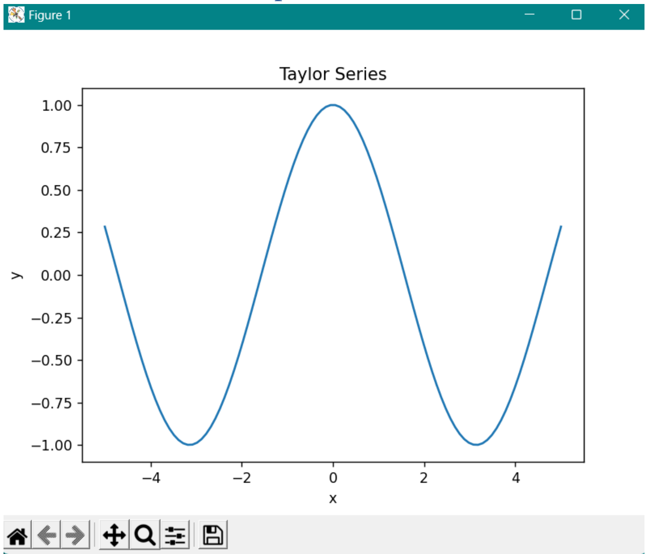


c)

Code

```
# c)
import sympy
import matplotlib.pyplot as plt

# Define the function
def f(x):
  return sympy.cos(x)

# series for the first 60 terms
x = sympy.Symbol('x')
taylorSeries = sympy.series(f(x), x0=sympy.pi/2, n=60).removeO()

range = np.linspace(-5, 5, 100)

# Get the taylor series for the given range
yValues = [taylorSeries.evalf(subs={x: xValues}) for xValues in range]

plt.title("Taylor Series")
plt.xlabel("x")
plt.ylabel("y")
plt.plot(range, yValues)
plt.show()
```

Screenshot of the output



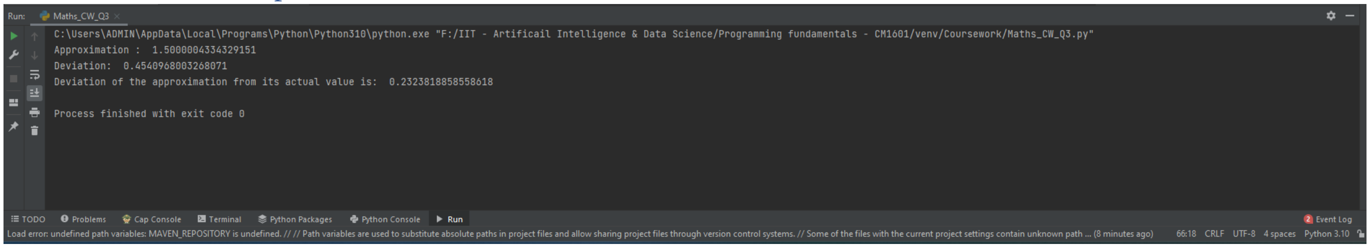
d)

Code

```
# d)
import numpy as np
import math

def cos(x):
    #taylor seies for x = pi/2
    n = 5  #Number of terms
    result = 1

    for i in range(n):
        result += (-1)**i * x**(2*i) / factorial(2*i)
    return result

def factorial(n):
    #Get the factorial
    result = 1
    for i in range(1, n+1):
        result *= i
    return result

#Approximate value at pi/3
x = math.pi/3
approx = cos(x)
print("Approximation : ", approx)

#deviation
actualValue = x * cos(math.pi/6)
approximate = cos(x)
absolError = abs(actualValue - approximate)
print("Deviation: ", absolError)
```

```
result = absolError/ actualValue
print("Deviation of the approximation from its actual value is: ", result)
```

Screenshot of the output



Question 4

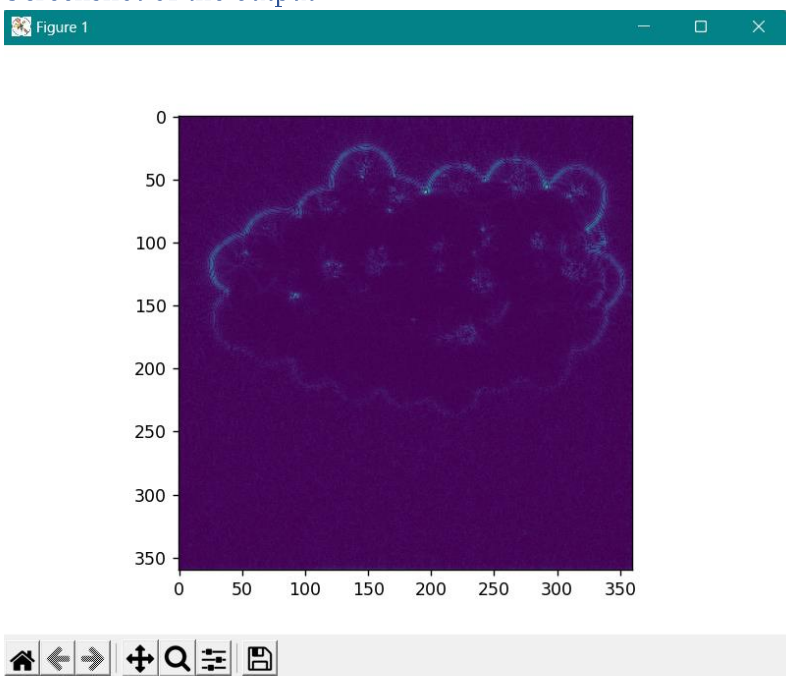
a)

Code

```
# a)

import numpy as np
from matplotlib import image as mpimg
import matplotlib.pyplot as plt
import scipy.fftpack as sfft

image = mpimg.imread("Fruit.jpg")

#fft
imagef = sfft.fft2(image)
plt.imshow(np.abs(imagef))
#image with fft shift
imagef = sfft.fftshift(imagef)
plt.imshow(np.abs(imagef))

#remove high frequencies
Image = np.zeros((360, 360), dtype=complex)
c = 180
r = 50
for m in range(0, 360):
    for n in range(0, 360):
        if (np.sqrt(((m-c)**2 + (n-c)**2))<r):
            Image[m, n] = imagef[m, n]

plt.imshow(np.abs(Image))
image1 = sfft.ifft2(Image)
plt.imshow(np.abs(image1))
# plt.show()

#remove low frequencies
Imagef1 = np.zeros((360, 360), dtype=complex)
c = 180
```

```
r = 90

for m in range(0, 360):
    for n in range(0, 360):
        if (np.sqrt(((m-c)**2 + (n-c)**2))>r):
            Imagef1[m, n] = imagef[m, n]

plt.imshow(np.abs(Imagef1))
image1 = sfft.ifft2(Imagef1)
plt.imshow(np.abs(image1))
plt.show()
```

Screenshot of the output



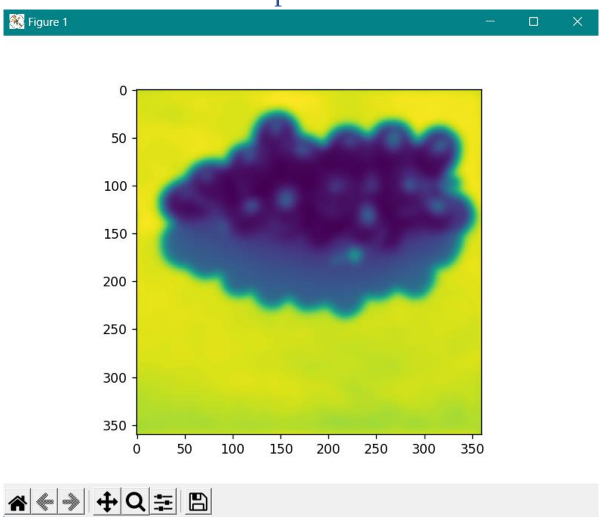
b)

Code

```
# b)
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import scipy.fftpack as sfft
import scipy.signal as signal

#load the image
img = mpimg.imread("Fruit.jpg")

#Gaussian filter
image = np.outer(signal.gaussian(360, 5), signal.gaussian(360, 5))
```

```
imagef = sfft.fft2(sfft.ifftshift(image))  #freq domain kernel
plt.imshow(np.abs(imagef))

imgf = sfft.fft2(img)
plt.imshow(np.abs(imagef))

imgB = imgf * imagef
plt.imshow(np.abs(imgB))

img1 = sfft.ifft2(imgB)
plt.imshow(np.abs(img1))
plt.show()
```

Screenshot of the output



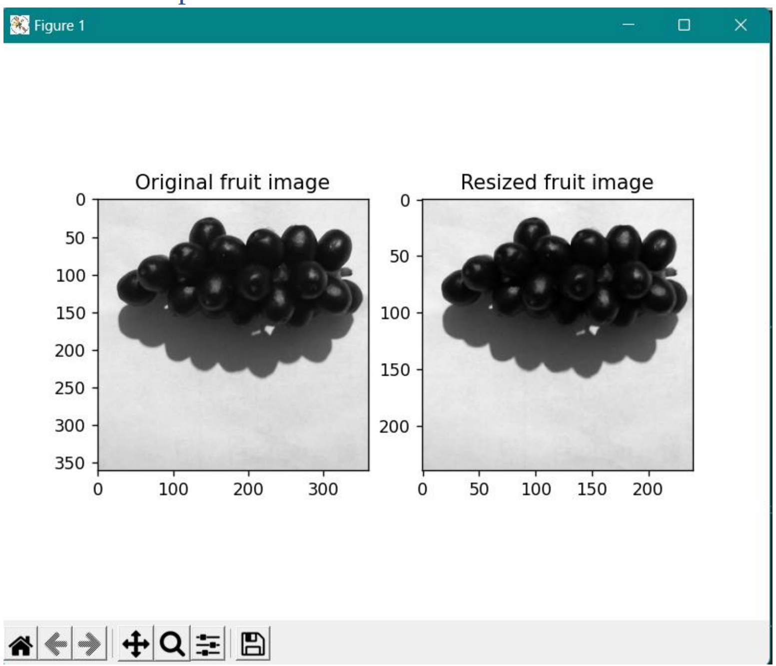
c)

Code

```
# c)
import numpy as np
from scipy.fftpack import dct
from PIL import Image

# Load the image
image = Image.open('Fruit.jpg')

# Convert the image to a numpy array
img = np.array(image)

#DCT to the fruit image
DCT = dct(img)

#Scale the image
resizedImage = image.resize((240, 240))

#Plot the original vs compressed fruit image
plt.subplot(1, 2, 1)
```

```
plt.imshow(image, cmap='gray')
plt.title('Original fruit image')
plt.subplot(1, 2, 2)
plt.imshow(resizedImage, cmap='gray')
plt.title('Resized fruit image')
plt.show()
```

Screenshot of the output



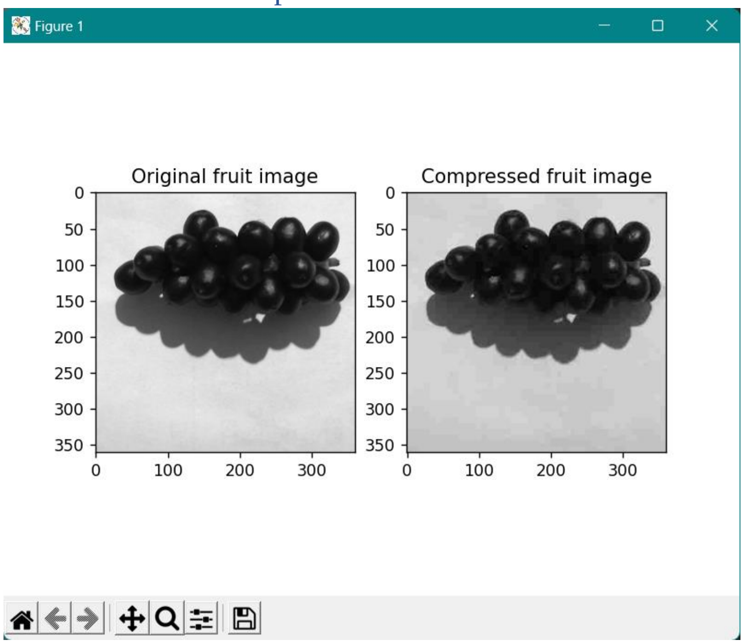
d)

Code

```
# d)
from PIL import Image

# Open the image
image = Image.open('Fruit.jpg')

# Set the image quality to 50 pixels
image.save('compressedImage.jpg', "JPEG", quality=20)

compressedImage = Image.open("compressedImage.jpg")

#Plot the original vs compressed fruit image
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original fruit image')
plt.subplot(1, 2, 2)
plt.imshow(compressedImage, cmap='gray')
```

```
plt.title('Compressed fruit image')
plt.show()
```

Screenshot of the output

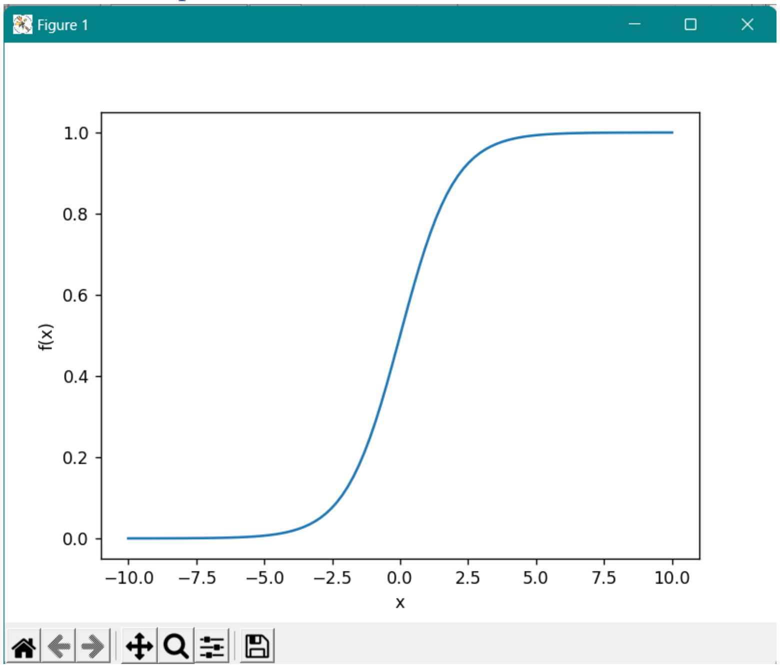


Question 5

Code

```
# a)
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-10, 10, 100)
y = 1/(1 + np.exp(-x))

plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.show()
```

Screenshot of the output

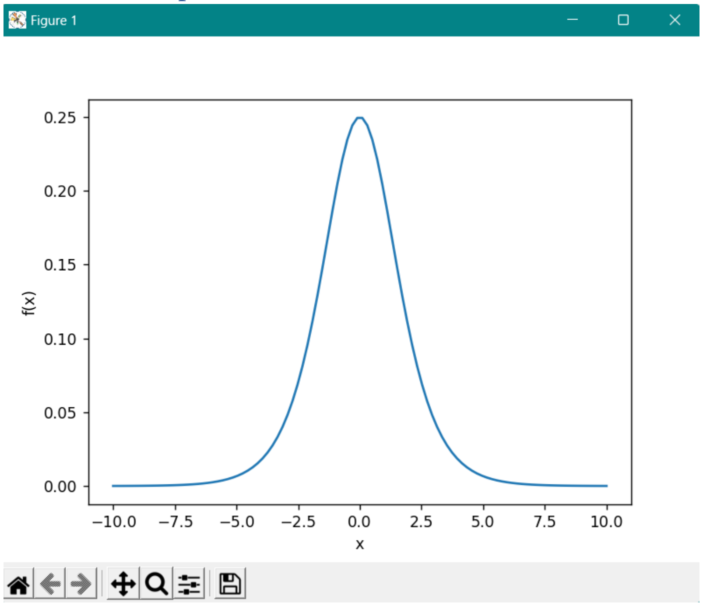


a)

Code

```
# b)
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-10, 10, 100)
y = 1/(1 + np.exp(-x))
df = y *(1-y)

plt.plot(x, df)
plt.xlabel("x")
plt.ylabel("f(x)")

plt.show()
```

Screenshot of the output



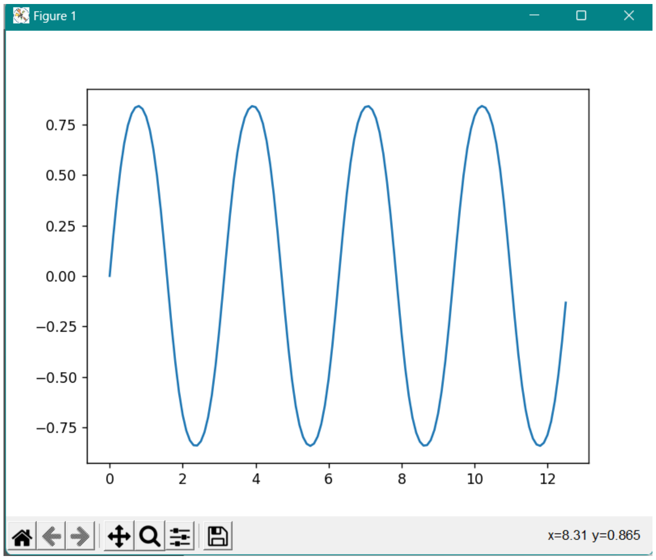
c)

i) sin sin(2x)

Code

```
# a)
import matplotlib.pyplot as plt
import numpy as np


x = np.arange(0, 4* np.pi, 0.1)
y = np.sin(np.sin(2*x))

plt.plot(x, y)
plt.show()
```

Screenshot of the output

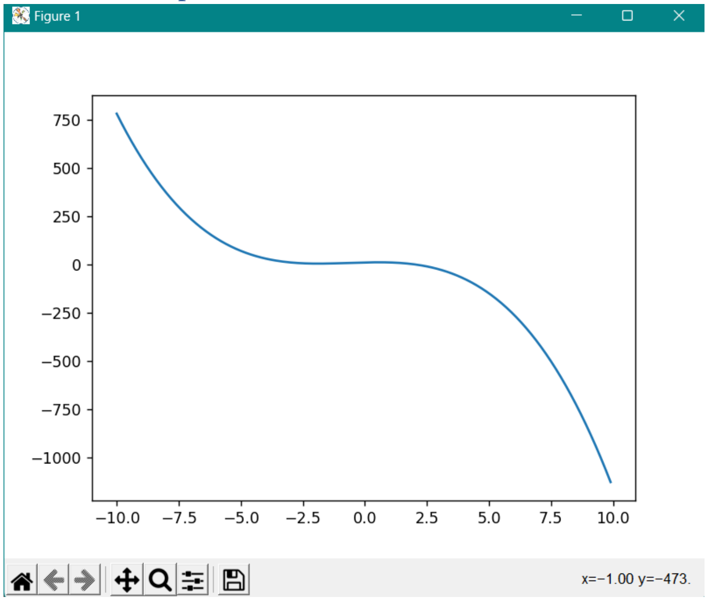


ii) -x**2 -2x**2 +3x+10

Code

```
# b)
import sympy as sym
import matplotlib.pyplot as plt
import numpy as np

x = sym.symbols('x')
fx = -x ** 3 - 2 * x ** 2 + 3 * x + 10
f4 = sym.lambdify(x, fx, 'numpy')
z = np.arange(-10, 10, 0.1)
result = f4(z)

plt.plot(z, result)
plt.show()
```

Screenshot of the output

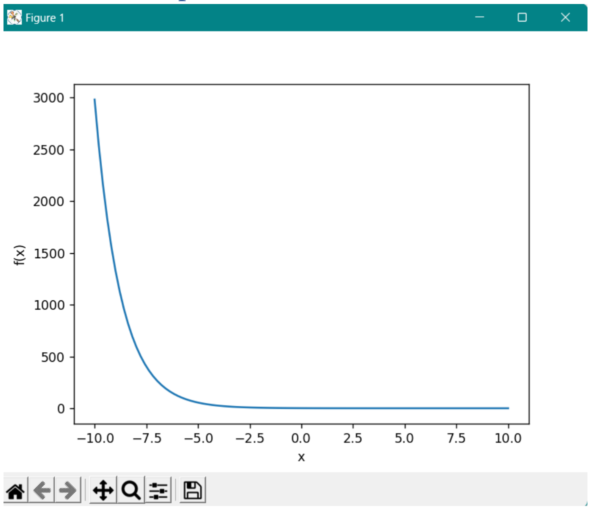


iii) e*-0.8x

Code

```
# c)
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(-0.8*x)

plt.plot(x, y)
plt.xlabel("x")
plt.ylabel("f(x)")

plt.show()
```

Screenshot of the output

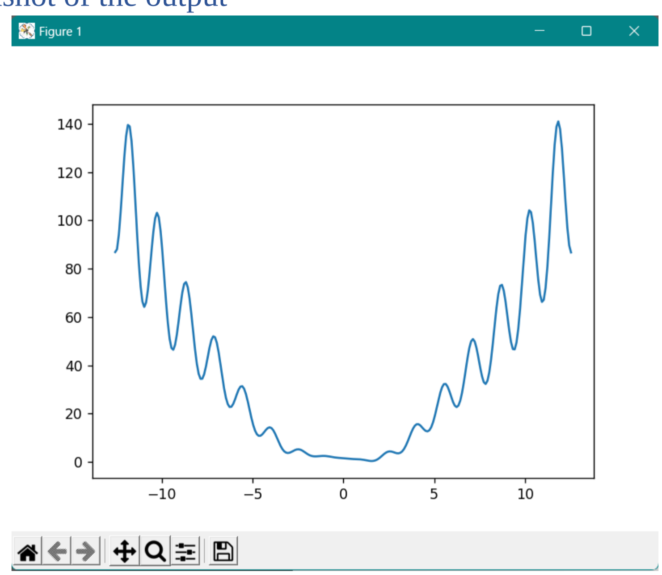


iv) x**2 cos cos(2x) -2 sin sin(x-pi/3)

Code

```
# d)
import matplotlib.pyplot as plt
import numpy as np
import math

x = np.arange(-4*math.pi, 4* math.pi, 0.1)
y = x**2 * np.cos(np.cos(2*x)) - 2 * np.sin(np.sin(x - 3.14/3))

plt.plot(x, y)
plt.show()
```

Screenshot of the output

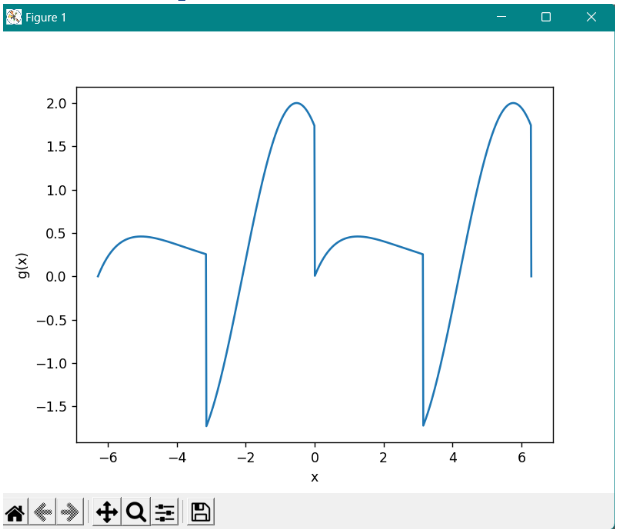


v) $g(x) = \{ 2\cos(x + \pi\ 6/) - \pi \leq x < 0$

$xe\ {-0.4x}\ 2\ 0 \leq x < \pi$

```python
# e)
import numpy as np
import matplotlib.pyplot as plt

def g(x):
    if (-np.pi <= x < 0):
        return 2 * np.cos(x + np.pi/6)

    if (0 <= x < np.pi):
        return x * np.exp(-0.4 * x*2)

    if (x <= -np.pi):
        z = x + (2 * np.pi)
        result = g(z)

    if (x > np.pi):
        z = x - (2 * np.pi)
        result = g(z)

    return result


# for the range between -2*pi and 2*pi
x = np.linspace(-2 * np.pi, 2 * np.pi, 1000)

# Calculate y values using the function g(x)
y = [g(result) for result in x]

# Plot the function
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('g(x)')
plt.show()
```

Screenshot of the output



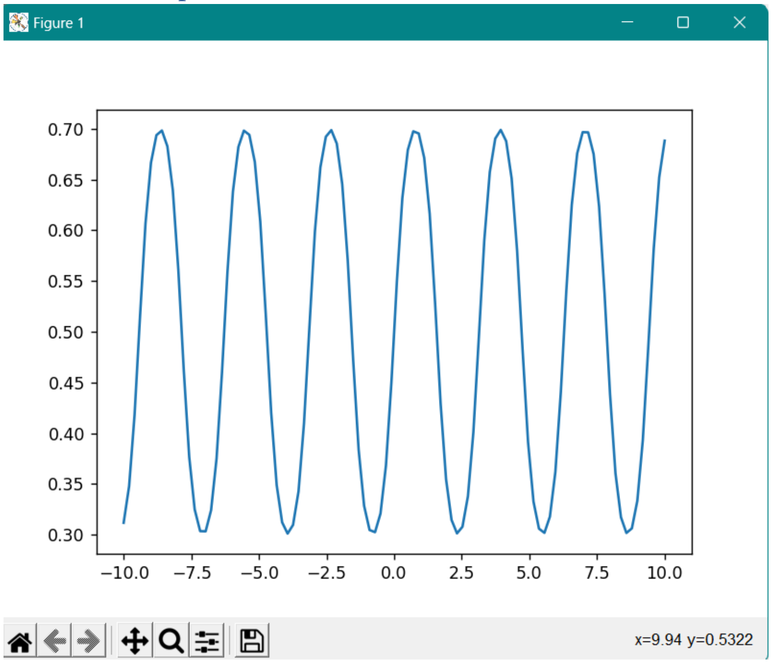
d)

i) sin sin(2x)

Code

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


def logistic(x):
    return (1 / (1 + np.exp(-x)))

def f(x):
    return np.sin(np.sin(2*x))

x = np.linspace(-10, 10, 100)
y=logistic(f(x))

plt.plot(x, y)
plt.show()
```

Screenshot of the output

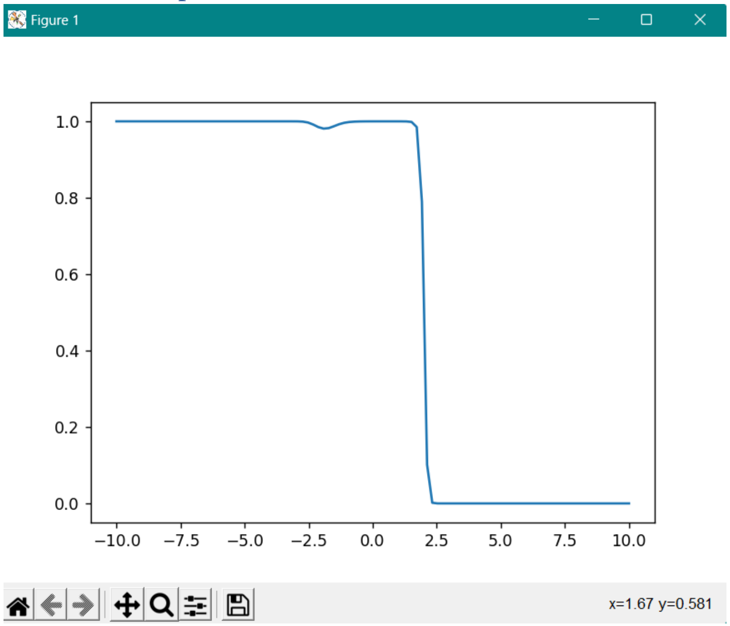


ii) -x**2 -2x**2 +3x+10

Code

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


def logistic(x):
    return (1 / (1 + np.exp(-x)))

def f(x):
    return -x**3 - 2 * x**2 + 3*x + 10

x = np.linspace(-10, 10, 100)
y= logistic(f(x))

plt.plot(x, y)
plt.show()
```

Screenshot of the output

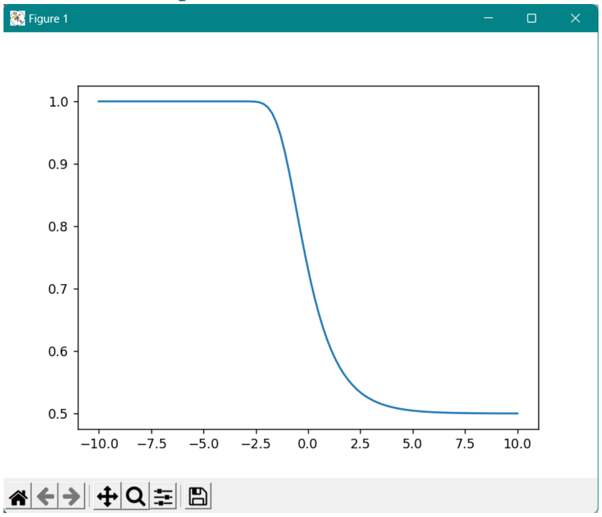


iii) e*-0.8x

Code

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


def logistic(x):
    return (1 / (1 + np.exp(-x)))

def f(x):
    return np.exp(-0.8*x)

x = np.linspace(-10, 10, 100)
y= logistic(f(x))

plt.plot(x, y)
plt.show()
```

Screenshot of the output

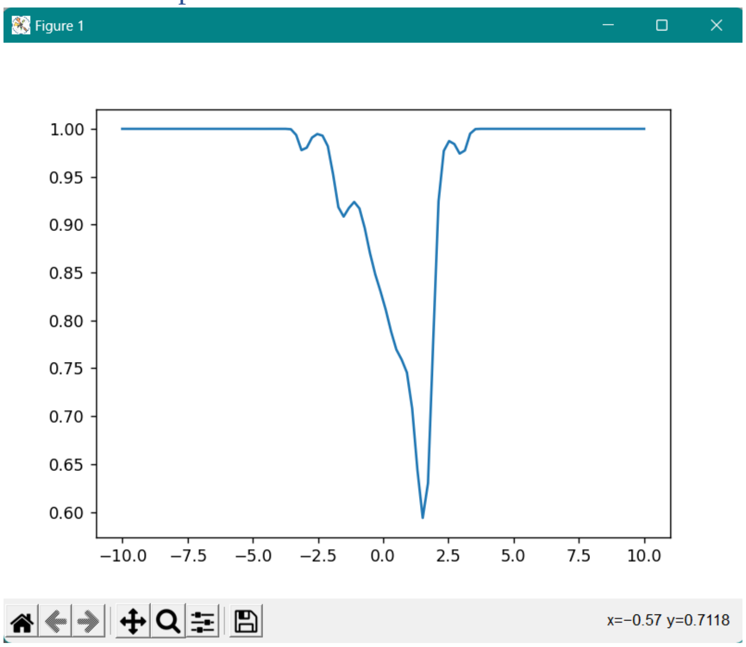


iv) x**2 cos cos(2x) -2 sin sin(x-pi/3)

Code

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


def logistic(x):
    return (1 / (1 + np.exp(-x)))

def f(x):
    return x**2 * np.cos(np.cos(2*x)) - 2 * np.sin(np.sin(x - 3.14/3))

x = np.linspace(-10, 10, 100)
y= logistic(f(x))

plt.plot(x, y)
plt.show()
```

Screenshot of the output



v) $\boldsymbol{g(x) = \{\ 2\cos(x + \pi\ 6/) - \pi \leq x < 0}$

$$\boldsymbol{xe\ {-0.4x}\ 2\ \ 0 \leq x < \pi}$$

Code

```
# v)
import numpy as np
import matplotlib.pyplot as plt

xVal=[]
for i in np.arange(-np.pi, np.pi):
    xVal.append(i * 0.1)

yValues=[]


def g(x):
    if -np.pi <= x < 0:
        return (1 / (1 + np.exp(-x))) * 2 * np.cos(x + np.pi/6)

    if 0 <= x < np.pi:
        return (1 / (1 + np.exp(-x))) * x * np.exp(-0.4 * x*2)

    if x <= -np.pi:
        z = x + (2 * np.pi)
        result = g(z)

    if x > np.pi:
        z = x - (2 * np.pi)
        result = g(z)
```

```
    return result

x = np.arange(-4 * np.pi, 4 * np.pi, 0.01)
y = [g(result) for result in x]

#Plot the graph
plt.plot(x, y, color="green")
plt.xlabel('x')
plt.ylabel('g(x)')
plt.show()
```

Screenshot of the output