Lab Report 3

By Seunghyun Park 1003105855 & Juann Jeon 1005210166
Question 1 done by Seunghyun Park
Question 2 done by Juann Jeon & Seunghyun Park
Question 3 done by Seunghyun Park
Lab Report done by Juann Jeon

Q1 a)
Pseudocode

import numpy as np
import matplotlib.pyplot as plt

#Define a function f(x)
def $f(x)$:
    return e^{-x^2}

# Analytic Value of the derivative
def $f_{true}(x)$:
    return $-\ 2xe^{-x^2}$

#Define a forward difference
def $f_{forward}(x, h)$:
    #c is derivative value, i.e. $c = \frac{df}{dx}$
    $c = \frac{f(x+h) - f(x)}{h}$
    return c

#Calculate the derivative and numerical error using for loop
for i in range(0, 17):
    $derivative = f_{forward}(x = 0.5, h = 10^{-16+i})$
    $error_{forward} = |f_{true}(0.5) - f_{forward}(0.5,\ 10^{-16+i})|$
    print("The derivative of function $f_{forward}$ when h =", 10^{-16+i}, "is", $derivative$)
    print("The absolute value of error in $f_{forward}$ when h =", 10^{-16+i},"is", $error$)

print('The analytic value of the derivative is', true_value(0.5))

Q1 a)
Table

Value of h	Numerical derivative value	Error
10^{-16}	-1.11e+00	3.31e-01
10^{-15}	-7.77e-01	1.64e-03
10^{-14}	-7.77e-01	1.64e-03
10^{-13}	-7.79e-01	5.76e-04
10^{-12}	-7.79e-01	2.07e-05
10^{-11}	-7.79e-01	1.54e-06
10^{-10}	-7.79e-01	6.85e-07
10^{-9}	-7.79e-01	1.86e-08
10^{-8}	-7.79e-01	7.45e-09
10^{-7}	-7.79e-01	3.85e-08
10^{-6}	-7.79e-01	3.89e-07
10^{-5}	-7.79e-01	3.89e-06
10^{-4}	-7.79e-01	3.89e-05
10^{-3}	-7.79e-01	3.89e-04
10^{-2}	-7.83e-01	3.83e-03
10^{-1}	-8.11e-01	3.24e-02
1	-6.73e-01	1.05e-01

Table 1: Numerical derivative value using forward differences and their errors with different values of h

Q1 b)

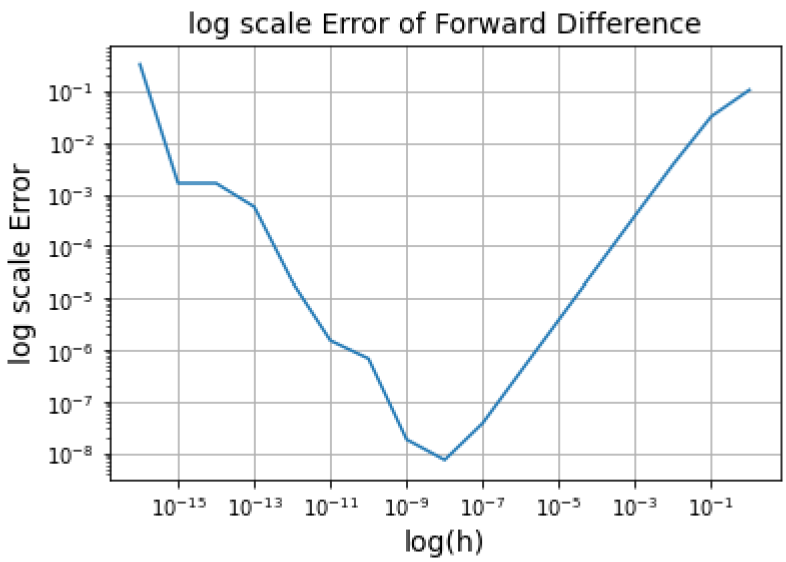

Figure 1: Error vs Value of h, in logarithmic scale

Figure 1 shows the Error vs Value of h in logarithmic scale using forward method. From Figure 1, we can notice that round off error dominates the left part of the plot (i.e. h less than 10^{-8}) while truncation error dominates the right part of the plot (i.e. h greater than 10^{-8}). If we look at the formula for the error, the round off error is given as $\epsilon_{round} = \frac{2C|f(x)|}{h}$ and truncation error is given as $\epsilon_{truncation} = \frac{1}{2}h|f^{''}(x)|$. Notice that in ϵ_{round}, the h is a denominator, while in $\epsilon_{truncation}$, the h is a factor. This means that when h is small, $\epsilon_{round} > \epsilon_{truncation}$ and when h is large, $\epsilon_{round} < \epsilon_{truncation}$. The final error ϵ is a sum of ϵ_{round} and $\epsilon_{truncation}$; i.e. $\epsilon = \frac{2C|f(x)|}{h} + \frac{1}{2}h|f^{''}(x)|$, which is given in the textbook (5.91). The reason why error is smallest around 10^{-8} is because as shown in the textbook (5.94), ϵ can be rewritten as $\epsilon = \sqrt{4C|f(x)f^{''}(x)|}$. With error constant $C = 10^{-16}$, we see that $\sqrt{C} = 10^{-8}$. So if we use the value of h that deviates further from 10^{-8}, it is expected that accuracy of derivation will fall due to the factor of $\sqrt{C} = 10^{-8}$.

Q1 c)
Pseudocode

#Define a function f(x)
def $f(x)$:
    return e^{-x^2}

#Analytic Value of the derivative
def $f_{true}(x)$:
    return $-\ 2xe^{-x^2}$

#Define a central difference
def $f_{central}(x, h)$:
    #c is derivative, i.e. $c\ =\ \frac{df}{dx}$
    $c\ =\ \frac{f(x+\frac{h}{2})-f(x-\frac{h}{2})}{h}$
    return c

#Calculate the derivative and numerical error using for loop
for i in range(0, 17):
    $derivative\ = f_{central}(0.5.\ 10^{-16+i})$
    $error_{central}$ = $|f_{true}(0.5)\ -\ f_{central}(0.5,\ 10^{-16+i})|$
    print("The derivative of function $f_{central}$ when h =", 10^{-16+i}, "is", $derivative$)
    print("The absolute value of error in $f_{central}$ when h =", 10^{-16+i},"is", $error$)

#Plot error_forward and error_central in log scale
plt.loglog(h,$error_{forward}$)
plt.loglog(h, $error_{central}$)

Q1 c)
Analysis

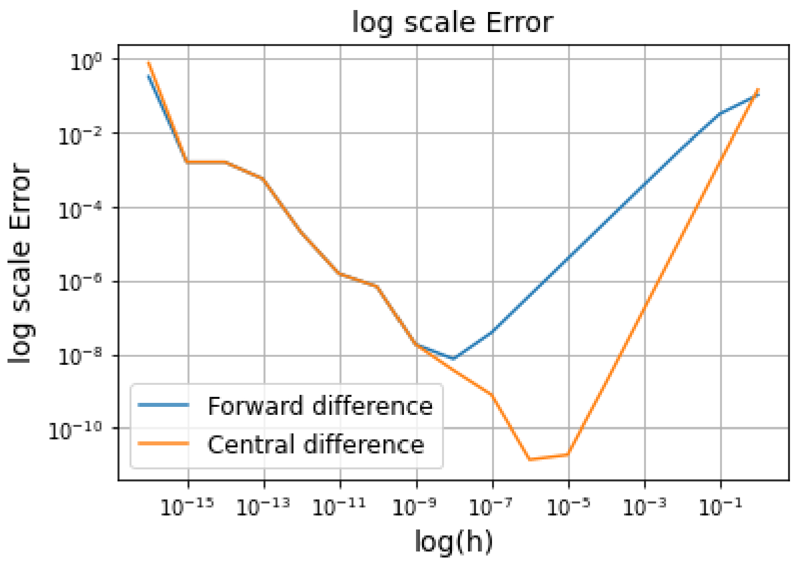

Figure 2: Error vs Value of h in logarithmic scale, for both forward and central differences

Figure 2 shows the error of forward and central differences in logarithmic scale. From textbook (5.101), final error for central differences is given as $\epsilon = (\frac{9}{8}C^2|f(x)|^2|f'''(x)|)^{\frac{1}{3}}$, so ideal value of h is around 10^{-5} with error around 10^{-10}. From Figure 2, we can see that derivating with central differences indeed gives better precision compared to forward differences at their ideal value of the h. However, we can see that when h deviates further from their respective ideal values, the derivation error using central differences gets bigger compared to forward differences, meaning central differences is not always better if h is far from being an ideal value.

Q2
Pseudocode

#(a)
from gaussxw import gaussxw
from scipy import constants
import matplotlib.pyplot as plt
import numpy as np

m = Mass of particle, in kg
k = Spring constant, in N/m
c = speed of light, in m/s
x_0 = initial position

# Define the integrand
def $g(x,\ x_0)$:

$f_x = x_0^2 - x^2$

$f_1 = k * f_x$

$f_2 = (2mc^2) + \frac{1}{2}k * f_x$

$f_3 = 2(mc^2 + \frac{1}{2}k * f_x)^2$

return $\frac{c\sqrt{f_1 * f_2}}{\sqrt{f_3}}$

a = Lower boundary of integral
b = x_0 Upper boundary of integral, initial position
N_1 = 8 # Number of Sample
N_2 = 16 # Number of Sample

x_1, w_1 = gaussxw(N_1)
xp_1 = 0.5 * (b - a) * x_1 + 0.5 * (b + a) # x points to take integral
wp_1 = 0.5 * (b - a) * w_1 # Weight on integral
s_1 = 0.0 # Initial value of integral with N = 8

# Calculate the integral when N = 8 using for loop
for i in range(N_1):
    s_1 += wp_1[i] * g(xp_1[i],x_0)
print("Period Using Gaussian quadrature with N = 8:", s_1)

x_2, w_2 = gaussxw(N_2)
xp_2 = 0.5 * (b - a) * x_2 + 0.5 * (b + a) # x points to take integral
wp_2 = 0.5 * (b - a) * w_2 # Weight on integral
s_2 = 0.0 # Initial value of integral with N = 16

# Calculate the integral when N = 16 using for loop
for i in range(N_2):

```
    s_2 += wp_2[i] * g(xp_2[i],x_0)
print("Period Using Gaussian quadrature with N = 16:", s_2)
```

\# Calculate the Classical period value

$classical\ = 2\pi\sqrt{m/k}$

\# Calculate the fractional error

fractional error = $\left|\frac{s_1 - classical}{classical}\right|$ # for N = 8

fractional error = $\left|\frac{s_2 - classical}{classical}\right|$ # for N = 16

#(b)

\# Calculate the integrands

$integrands\ = 4/g(x, x_0)$

\# Plot the result

plt.plot($integrands$ of n = 8 as a function of x)
plt.plot($integrands$ of n = 16 as a function of x)

\# Calculate the weighted value

$weighted\ = 4\omega_k/g_k$

\# Plot the result

plt.plot(weighted value of n = 8 as a function of x)
plt.plot(weighted value of n = 16 as a function of x)

#(c)

$x_c = \sqrt{c^2/k}$ # Calculate x_c

print(x_c)

#(d)

```
N_200 = 200 # Number of sample
x_200, w_200 = gaussxw(N_200)
xp_200 = 0.5 * (b - a) * x_200 + 0.5 * (b + a) # x points to take integral
wp_200 = 0.5 * (b - a) * w_200 # Weight on integral
s_200 = 0.0 # Initial value of integral with N = 200

# Calculate the integral when N = 200 using for loop
for i in range(N_200):
    s_200 += wp_200[i] * g(xp_200[i],x_0)

N_400 = 400 # Number of sample

x_400, w_400 = gaussxw(N_400)
```

xp_400 = 0.5 * (b - a) * x_400 + 0.5 * (b + a) # x points to take integral
wp_400 = 0.5 * (b - a) * w_400 # Weight on integral
s_400 = 0.0 # Initial value of integral with N = 400

# Calculate the integral when N = 400 using for loop
for i in range(N_400):
    s_400 += wp_400[i] * g(xp_400[i],x_0)

Error estimation
$\epsilon_N = I_{2N} - I_N$
print(Error estimation * 100 %)

#(e)
N = 200 # Number of Sample
a = 0 # Lower bound of the integral
$x_c = \sqrt{c^2/k}$
x_0 = np.linspace(1,10*x_c) # Assign x_0 value (1 < x_0< 10x_c)
T = [] # Assign empty list of period

# Calculate T as a function of x_0 using for loop
for i in range(0,len(x_0)):
        b = x_0[i] # Assign value of b
        x_41, w_41 = gaussxw(N_4)
        xp_41 = 0.5 * (b - a) * x_41 + 0.5 * (b + a)
        wp_41 = 0.5 * (b - a) * w_41
        s_41 = 0 # Assign initial value of integral
        for i in range(N):
                s_41 += (wp_41[i] * g(xp_41[i],b))
        T.append(s_41) # Add T value in the list

# Plot the result
plt.plot(T as a function of x_0 in the range 1m < x_0 < 10x_c)
# Classical limit
classical limit = $2\pi\sqrt{m/k}$

Large-amplitude relativistic limit
relativistic limit = 4*x_0/c

# Plot both limit
plt.scatter(classical limit at x_0=1)
plt.scatter(relativistic limit at x_0= $10x_c$)

Q2 a)

Given $x_0 = 0.01m,\ k = 12N/m,\ m = 1kg$ and c is the speed of light in m/s, notice it is indeed true that $\frac{k(x_0^2 - x^2)}{2} << mc^2$ for all $x \in [0, x_0]$ (we know that spring cannot move beyond the initial position we placed as if it does it violates the Energy Conservation Law).

Using `gaussxw.py` (details are given in Appendix E in textbook) with $N = 8$ and $N = 16$, we get the following result:

N	Output	Fractional error (in decimal)
Classical value	1.81e+00	0 (base)
$N = 8$	1.73e+00	4.61e-02
$N = 16$	1.77e+00	2.38e-02

Table 2: Output and their fractional error in decimal form using gaussian quadrature compared to classical value

The Table 2 shows the trend that for larger N, estimated fractional error is getting smaller.

Q2 b)

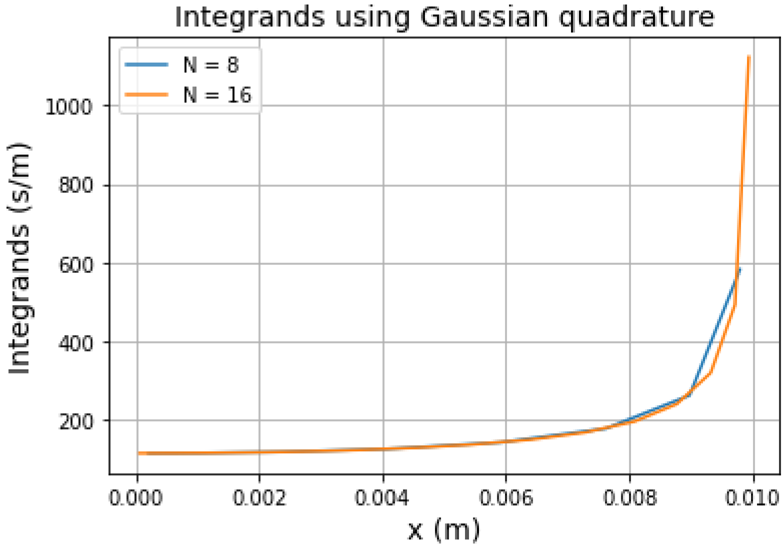

Figure 3: Integrands $4/g_k$ using Gaussian quadrature

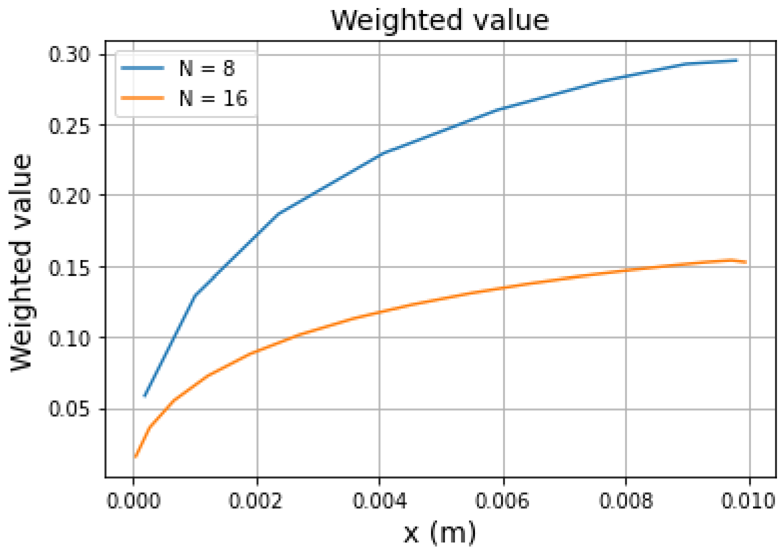

Figure 4: Weighted value $4w_k/g_k$ using Gaussian quadrature

Figure 3 shows the plot of integrands of $4/g_k$ and Figure 4 shows the plot of weighted value of $4w_k/g_k$. Notice that Figure 3 and Figure 4 shows very opposite behavior as the x_0 limit of integration is approached; Figure 3 seems to diverge to positive infinity while Figure 4 converges to some number. Figure 3 and Figure 4 suggests the importance of the weighted value w, and shows that larger the N is, it can take broader w which may increase the accuracy of calculation.

Q2 c)

On the classical view, we know that $v \approx \sqrt{k(x_0^{\ 2} - x^2)}$. Given $v = c$, $k = 12N/m$ and $x = 0m$, we get $x_0 = x_c = \frac{c}{\sqrt{12}} \approx 8.65 * 10^7 m$.

Q2 d)

Using the Gaussian quadrature with $N = 200$ and the same physical values from 2 a), we get the output 1.81025365200371. The error can be estimated using equation (3) in lab instruction: $\epsilon_N = I_{2N} - I_N$. To estimate the error of integral with $N = 200$, we also need to calculate the integral with $N = 400$. The equation (3) becomes $\epsilon_{200} = I_{400} - I_{200}$ and gives a value of 0.0017706424112220454 in decimal form. The estimate of the percentage error for the small amplitude case with $N = 200$ is 0.17706424112220454%

Q2 e)

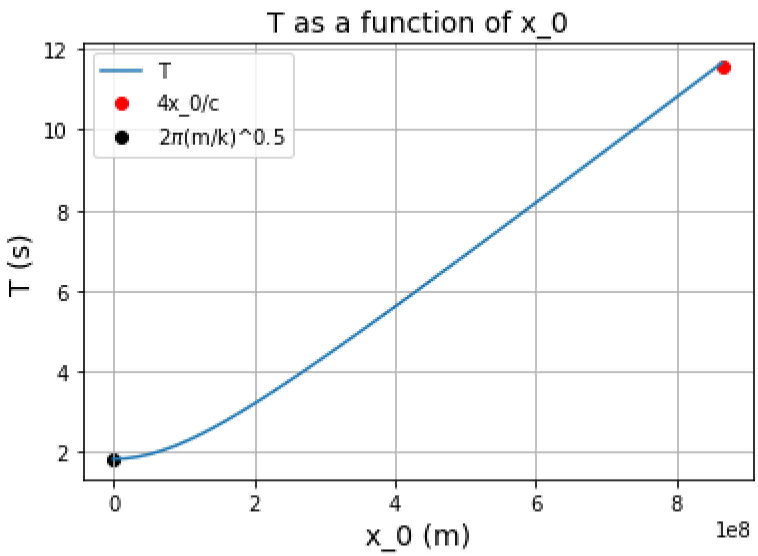

Figure 5: Plot for transition of classical to general

Figure 5 shows the plot of $T(x_0)$ for $x_0 \in [0,\ 10x_c]$ (the value of x_c is given in Q2 c)). Notice that for very low values of x_0, $T(x_0)$ shows non-linear motion, and after that it shows a linear trend. If we look at the left end point of the plot (i.e. $T(1)$), we see that T converges to the classical limit, $2\pi\sqrt{\frac{m}{k}}$. For right end point of the plot (i.e. $T(10x_c)$), we see that T converges to the large-amplitude relativistic limit, $\frac{4x_0}{c}$, as suggested at the beginning of the problem.

Q3 a)

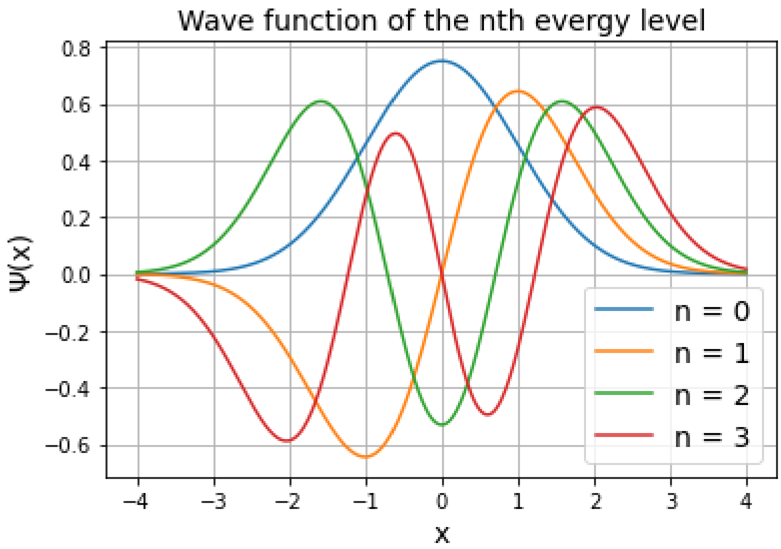

Figure 6: Harmonic oscillator wave function in the range $x \in [-\ 4, 4]$ for $n\ =\ 0, 1, 2, 3$

Figure 6 represents the quantum state of nth energy level of the one-dimensional quantum harmonic oscillator. Quantum state is defined as the mathematical probability distribution of the possible measurement in the quantum system. Areas under and over the each wave function at nth energy along the x axis is a probability density of that corresponding energy level in their quadratic potential well.

Q3 b)

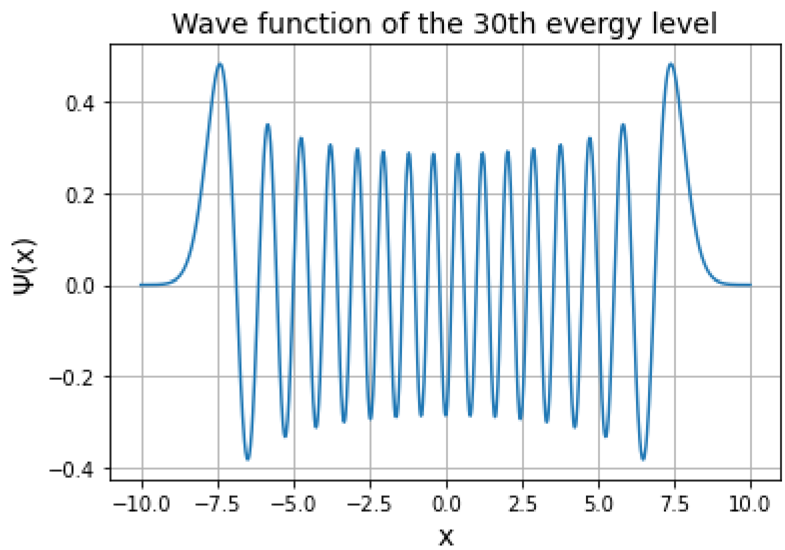

Figure 7: Harmonic oscillator wave function in the range $x \in [-\ 10, 10]$ for $n\ =\ 30$

Q3 c)

Pseudocode

import numpy as np
import matplotlib.pyplot as plt
import math
from gaussxw import gaussxw

#Define wavefunction
def function psi(n,x):

return $\frac{1}{\sqrt{2^n n!\sqrt{\pi}}}exp(-x^2/2)H_n(x)$

#Define derivative of wavefunction, $\frac{d\psi_n(x)}{dx}$
def function der_psi(n,x):

return $\frac{1}{\sqrt{2^n n!\sqrt{\pi}}}exp(-x^2/2)[-xH_n(x)+2nH_{n-1}(x)]$

#Define $x^2|\psi_n(x)|^2$, change variable x = tan(z)
The integrand becomes $tan^2(z)|\psi_n(tan(z))|^2\frac{1}{cos^2(z)}$
Define a function x_u(n,z) :

return $tan^2(z)|\psi_n(tan(z))|^2\frac{1}{cos^2(z)}$

Define $\left|\frac{d\psi_n(x)}{dx}\right|^2$, change variable x = tan(z)
The integrand becomes $\frac{|der\_psi(n,tan(x))|^2}{cos^2(z)}$
Define a function p_u(n,z):

return $\frac{|der\_psi(n,tan(x))|^2}{cos^2(z)}$

N = 100 # Number of points
a = -np.pi/2 # Lower bound of integral
b = np.pi/2 # Upper bound of integral
x,w = gaussxw(N)
xp = 0.5 * (b - a) * x + 0.5 * (b + a)
wp = 0.5 * (b - a) * w
s_x = 0.0 # Assign Initial value of integral = $<x^2>$
s_p = 0.0 # Assign Initial value of integral = $<p^2>$
ns = np.arange(0,16,1)   # n = (1,2,3, ..., 15)

```
po = [] #Assign a empty list for < x^2 >
mo =[] #Assign a empty list for < p^2 >


#Calculate <x^2> for a given value of n using for loop
for n in ns:
        for i in range(N):
                s_x += w[i] * x_u(n,xp[i])
        po.append(s_x)
        print('<x^2> for n =',n,' is',s_x)


#Calculate <p^2> for a given value of n using for loop
for n in ns:
        for i in range(N):
                s_p += w[i] * p_u(n,xp[i])
        mo.append(s_p)
        print('<p^2> for n =',n,' is',s_p)

E = [] #Assign a empty list for total energy

#Calculate the Energy using for loop
# E = 1/2 (< x^2 >  +  < p^2 >)
for i in range(0,len(po)):
        E.append(0.5*(po[i]+mo[i]))

#Print the total energy of the oscillator using for loop
for i in range(0,len(ns)):
        print('The total energy of the oscillator when n=',ns[i],'is',E[i])

u_x = np.sqrt(po) # uncertainty in position
u_m = np.sqrt(mo) # uncertainty in momentum

# Print uncertainty of position and momentum using for loop
for i in range(0,len(ns)):
        print('The uncertainty in position when n=',ns[i],'is',u_x[i])

for i in range(0,len(ns)):
        print('The uncertainty in momentum when n=',ns[i],'is',u_m[i])
```

Printed Output

```
<x^2> for n = 0  is 5.e-01
<x^2> for n = 1  is 1.50e+00
<x^2> for n = 2  is 2.50e+00
<x^2> for n = 3  is 3.5e+00
<x^2> for n = 4  is 4.5e+00
<x^2> for n = 5  is 5.5e+00
<x^2> for n = 6  is 6.50e+00
<x^2> for n = 7  is 7.50e+00
<x^2> for n = 8  is 8.5e+00
<x^2> for n = 9  is 9.5e+00
<x^2> for n = 10  is 1.05e+01
<x^2> for n = 11  is 1.15e+01
<x^2> for n = 12  is 1.25e+01
<x^2> for n = 13  is 1.35e+01
<x^2> for n = 14  is 1.45e+01
<x^2> for n = 15  is 1.55e+01
```

Figure 8: Value of $< x^2 >$ for given n

```
<p^2> for n = 0  is 5.e-01
<p^2> for n = 1  is 1.50e+00
<p^2> for n = 2  is 2.50e+00
<p^2> for n = 3  is 3.5e+00
<p^2> for n = 4  is 4.5e+00
<p^2> for n = 5  is 5.50e+00
<p^2> for n = 6  is 6.50e+00
<p^2> for n = 7  is 7.50e+00
<p^2> for n = 8  is 8.5e+00
<p^2> for n = 9  is 9.5e+00
<p^2> for n = 10  is 1.05e+01
<p^2> for n = 11  is 1.15e+01
<p^2> for n = 12  is 1.25e+01
<p^2> for n = 13  is 1.35e+01
<p^2> for n = 14  is 1.45e+01
<p^2> for n = 15  is 1.55e+01
```

Figure 9: Value of $< p^2 >$ for given n

```
The total energy of the oscillator when n= 0 is 5.e-01
The total energy of the oscillator when n= 1 is 1.50e+00
The total energy of the oscillator when n= 2 is 2.50e+00
The total energy of the oscillator when n= 3 is 3.5e+00
The total energy of the oscillator when n= 4 is 4.5e+00
The total energy of the oscillator when n= 5 is 5.5e+00
The total energy of the oscillator when n= 6 is 6.50e+00
The total energy of the oscillator when n= 7 is 7.50e+00
The total energy of the oscillator when n= 8 is 8.5e+00
The total energy of the oscillator when n= 9 is 9.5e+00
The total energy of the oscillator when n= 10 is 1.05e+01
The total energy of the oscillator when n= 11 is 1.15e+01
The total energy of the oscillator when n= 12 is 1.25e+01
The total energy of the oscillator when n= 13 is 1.35e+01
The total energy of the oscillator when n= 14 is 1.45e+01
The total energy of the oscillator when n= 15 is 1.55e+01
```

Figure 10: The total energy of the oscillator for given n

```
The uncertainty in position when n= 0 is 7.07e-01
The uncertainty in position when n= 1 is 1.22e+00
The uncertainty in position when n= 2 is 1.58e+00
The uncertainty in position when n= 3 is 1.87e+00
The uncertainty in position when n= 4 is 2.12e+00
The uncertainty in position when n= 5 is 2.35e+00
The uncertainty in position when n= 6 is 2.55e+00
The uncertainty in position when n= 7 is 2.74e+00
The uncertainty in position when n= 8 is 2.92e+00
The uncertainty in position when n= 9 is 3.08e+00
The uncertainty in position when n= 10 is 3.24e+00
The uncertainty in position when n= 11 is 3.39e+00
The uncertainty in position when n= 12 is 3.54e+00
The uncertainty in position when n= 13 is 3.67e+00
The uncertainty in position when n= 14 is 3.81e+00
The uncertainty in position when n= 15 is 3.94e+00
```

Figure 11: The uncertainty in position for given n

```
The uncertainty in momentum when n= 0 is 7.07e-01
The uncertainty in momentum when n= 1 is 1.22e+00
The uncertainty in momentum when n= 2 is 1.58e+00
The uncertainty in momentum when n= 3 is 1.87e+00
The uncertainty in momentum when n= 4 is 2.12e+00
The uncertainty in momentum when n= 5 is 2.35e+00
The uncertainty in momentum when n= 6 is 2.55e+00
The uncertainty in momentum when n= 7 is 2.74e+00
The uncertainty in momentum when n= 8 is 2.92e+00
The uncertainty in momentum when n= 9 is 3.08e+00
The uncertainty in momentum when n= 10 is 3.24e+00
The uncertainty in momentum when n= 11 is 3.39e+00
The uncertainty in momentum when n= 12 is 3.54e+00
The uncertainty in momentum when n= 13 is 3.67e+00
The uncertainty in momentum when n= 14 is 3.81e+00
The uncertainty in momentum when n= 15 is 3.94e+00
```

Figure 12: The uncertainty in momentum for given n

From Figure 11 and 12, we can see that the uncertainty in position and the uncertainty in momentum has the same value. This means $< x^2 >$ and $< p^2 >$ also have the same value as well. The total energy of the oscillator can be calculated by using the equation (14) from the lab instruction, $E = \frac{1}{2}(< x^2 > + < p^2 >)$. By putting in $< x^2 >$ and $< p^2 >$ values into the equation (14), we notice that the total energy of the oscillator has same value with $< x^2 >$ and $< p^2 >$ as well as it increases by 1 for each energy level. (i.e. $E(n + 1) \approx E(n) + 1$)