PHY 407 Lab 3

Landon Wang, Yinshi Liu

Contributions: Landon Wang wrote the entirety of Q1 and Q3, Yinshi Liu wrote the entirety of Q2. The report is compiled by Yinshi Liu.

Q1 A

The function we used is

$$f(x) = e^{-x^2} \tag{1.1}$$

Which analytically, have a derivative

$$f'(x) = -2xe^{-x^2} = -2x \cdot f(x) \tag{1.2}$$

Use forward differentiation method, we calculate

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} = \frac{e^{-(x+h)^2} - e^{-x^2}}{h} \tag{1.3}$$

For this part, we want to test the error based on a 64bit computer, therefore we have an array of h from 10^{-16} to 1. To achieve that, we calculate following in a for loop from -16 to 1

$$h[i+16] = 10^i \tag{1.4}$$

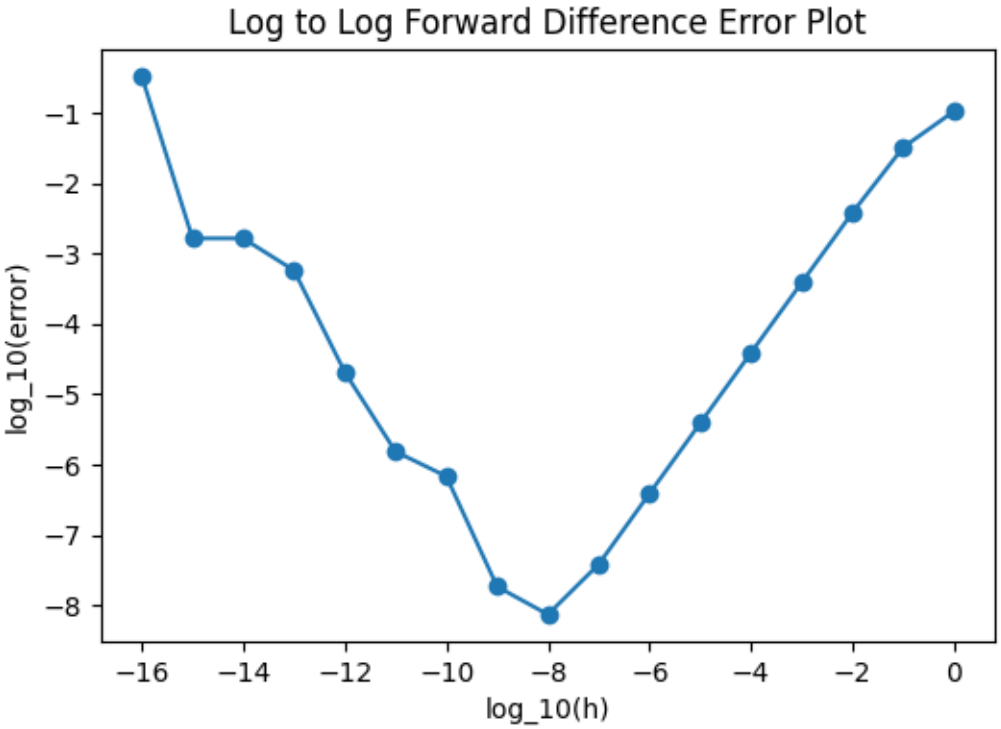
Pseudocode:

1. Define function based on equation 1.1
2. Define the differentiation method based on equation 1.3
3. Define an array of h within a loop from -16 to 0 follow equation 1.4
4. Set $x = 0.5$ for the test
5. Calculate analytic solution according to equation 1.2
6. Run a loop store and calculate numerical differentiation based on function defined in step 2 and absolute error based on analytic solution calculated in step 5

Here is the result:

h	f'(x)	error
10^-16	-1.11022	0.331422
10^-15	-0.777156	0.00164467
10^-14	-0.777156	0.00164467
10^-13	-0.779377	0.00057578
10^-12	-0.778821	2.06687e-05
10^-11	-0.778799	1.53576e-06
10^-10	-0.778801	6.84689e-07
10^-9	-0.778801	1.85549e-08
10^-8	-0.778801	7.45265e-09
10^-7	-0.778801	3.85389e-08
10^-6	-0.778801	3.89369e-07
10^-5	-0.778805	3.89393e-06
10^-4	-0.77884	3.89335e-05
10^-3	-0.77919	0.000388751
10^-2	-0.78263	0.00382907
10^-1	-0.811245	0.0324438
10^0	-0.673402	0.105399

Q1B FIGURE 1.1



We have a v shaped plot with the minimum at $h = 10^{-8}$.

Equation 5.91 from the textbook is

$$\epsilon = \frac{2C|f(x)|}{h} + \frac{1}{2} \tag{1.5}$$

Which the author followed up with some derivation and concludes that

$$\epsilon = h|f''(x)| = \sqrt{4C|f(x) \cdot f''(x)|} \tag{1.6}$$

Notice that C is dependent on the computer's hardware level design.

We have $C \approx 10^{-16}$ for our 64 bits computer which means h is in the level of 10^{-8}. Therefore, for $h > 10^{-8}$, truncation error dominates, and for $h < 10^{-8}$, rounding error dominates.

Q1C

Now, use central difference scheme, we use the differentiation function:

$$f'(x) \approx \frac{f\left(x + \frac{h}{2}\right) - f\left(x - \frac{h}{2}\right)}{h} = \frac{e^{-\left(x+\frac{h}{2}\right)^2} - e^{-\left(x-\frac{h}{2}\right)^2}}{h} \tag{1.7}$$

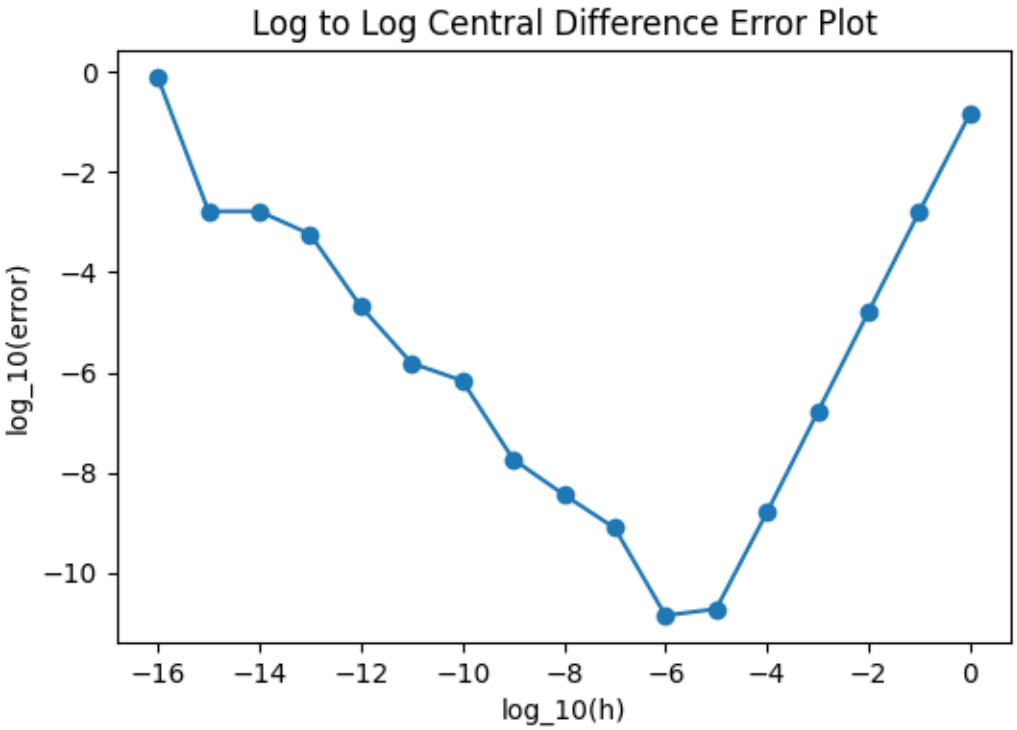
Everything else follows the methods of (a) and (b)

Pseudocode:

1. Define differentiation method according to equation 1.7
2. Use function f, array h, parameter x and analytic solution defined earlier in part (a)
3. Setup for loop according to step 6 of part (a) and store values in a different location

4. Plot the log to log error vs h for this differentiation method.

FIGURE 1.2



We noticed that the errors are smaller with the central differentiation scheme and the minimum error occurs at a larger $h = 10^{-6}$.

It is obviously better to use the central method compared to both forward and backward method because the average value of two sample points of central method is exactly the point we are looking for, whereas the forward and backward method has the average of two sample points slightly drafted from the center. According to the textbook derivation of the error, we have $h \approx C^{\frac{1}{3}}$, which matches the result we have, that is, we have not only the lower numbers to calculate if we trying to have a range of derivatives to calculate, we also have a better precision.

However, it is noticeable that for $h = 1$, we have a worse performance for central method which is always dependent to the equation we are calculating. When a function has its derivative about zero, the large interval estimation of the derivative could have very large error because the shape of the function is not able to be captured with the large h. In such case, it is possible that forward and backward method having a better performance than central, but this is simply a lucky guess that forward method happens to capture the shape more precisely.

Q2a

Pseudocode

1. Import the function gaussxw and gaussxwab from sample code gaussxw.py
2. Define constants and basic functions

   Define function g(x) as the result of:

   (Note: In the actual code, the equation is divided into different parts for ease of implementation)

$$g(x) = c\left(\frac{k(x_0^2 - x^2)\left(2mc^2 + \frac{k(x_0^2 - x^2)}{2}\right)}{2\left(mc^2 + \frac{k(x_0^2 - x^2)}{2}\right)^2}\right)^{\frac{1}{2}} \quad 2.1$$

3. Define function t(x) as:

$$t(x) = \frac{4}{g(x)} \quad 2.2$$

   Integrate the result of t(x) using the gaussxwab function with the following parameters: N1 = 8, N2 = 16, a = 0, b = 0.01.

   Sum up the results of T using a for loop:

   For each sample point, calculate the value of $w_i * t(x_i)$ and add it to T

   Iterate.

4. Calculate errors of N1 and N2 using the following estimation formula:

$$\epsilon_{frac} = \frac{I_{classical} - I_N}{I_{classical}} \quad 2.3$$

   print the results.

5. Calculate the classical limit:

$$T = 2\pi\sqrt{\frac{m}{k}} \quad 2.4$$

   Print the result.

Code output

```
period with N=8: 1.7301762343365563
period with N=16: 1.770715490242243
estimated fractional error (N=8) = 0.037918894960913364
estimated fractional error (N=16) = 0.019536375813847647
classical value = 1.8137993642342176
```

According to the code outputs, the period T when x_0 is small is indeed close to that of the classical limit. As the number of sampling points increases, the output gradually converges with the theoretical value. Additionally, as the number of sampling points increases, the fractional error for N = 16 is much smaller than that of N = 8. This does indeed prove that Gaussian quadrature can improve the accuracy of the result with relatively little sample

points.

Q2b

Pseudocode

1. Create empty arrays for $\frac{4}{g_k}$ and weight values $\frac{4w_k}{g_k}$ for each individual sample points N_i
2. In the for loops of part a, extract the values $w_k, 4/g_k$ and append them to the empty arrays.
3. Plot $\frac{4}{g_k}$ and $\frac{4w_k}{g_k}$ against each sample number N_i. Plot the results.

Code Output:

Figure 2.1

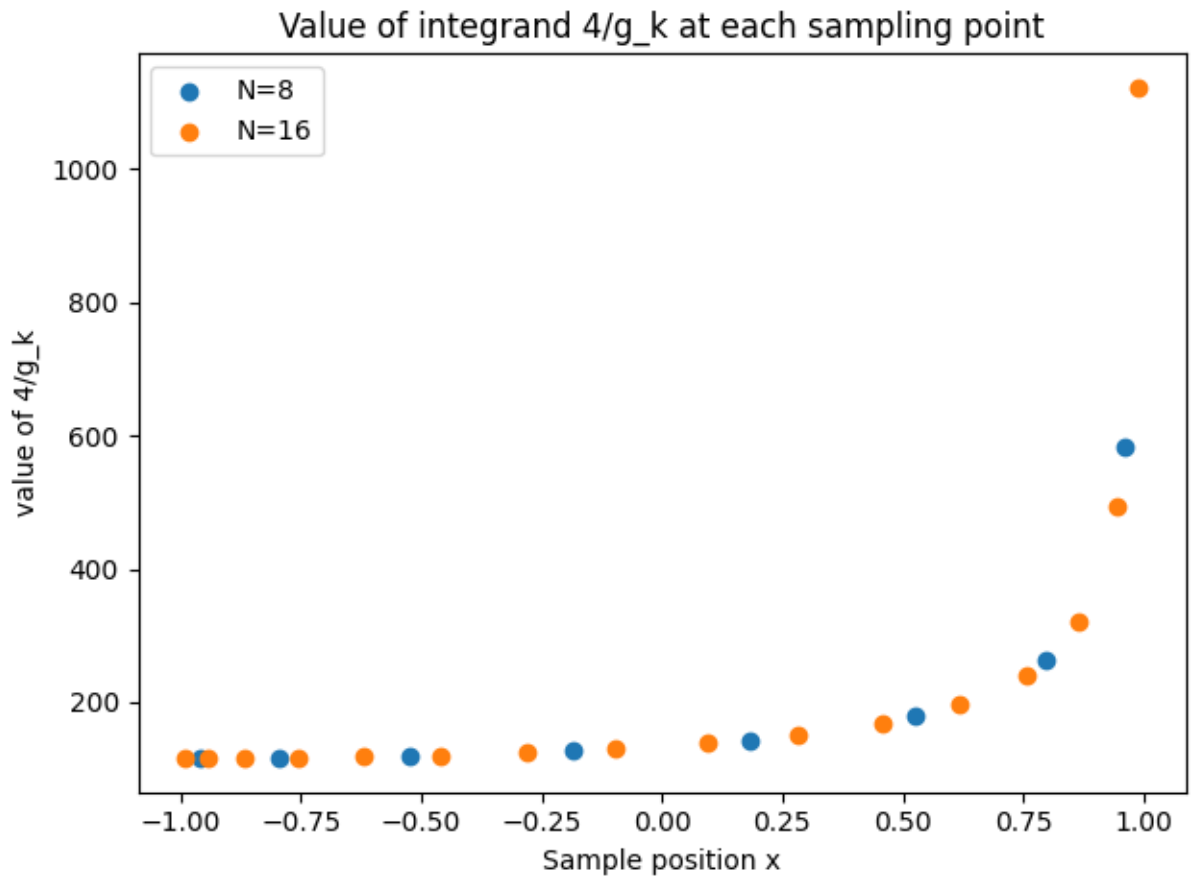


Figure 2.2

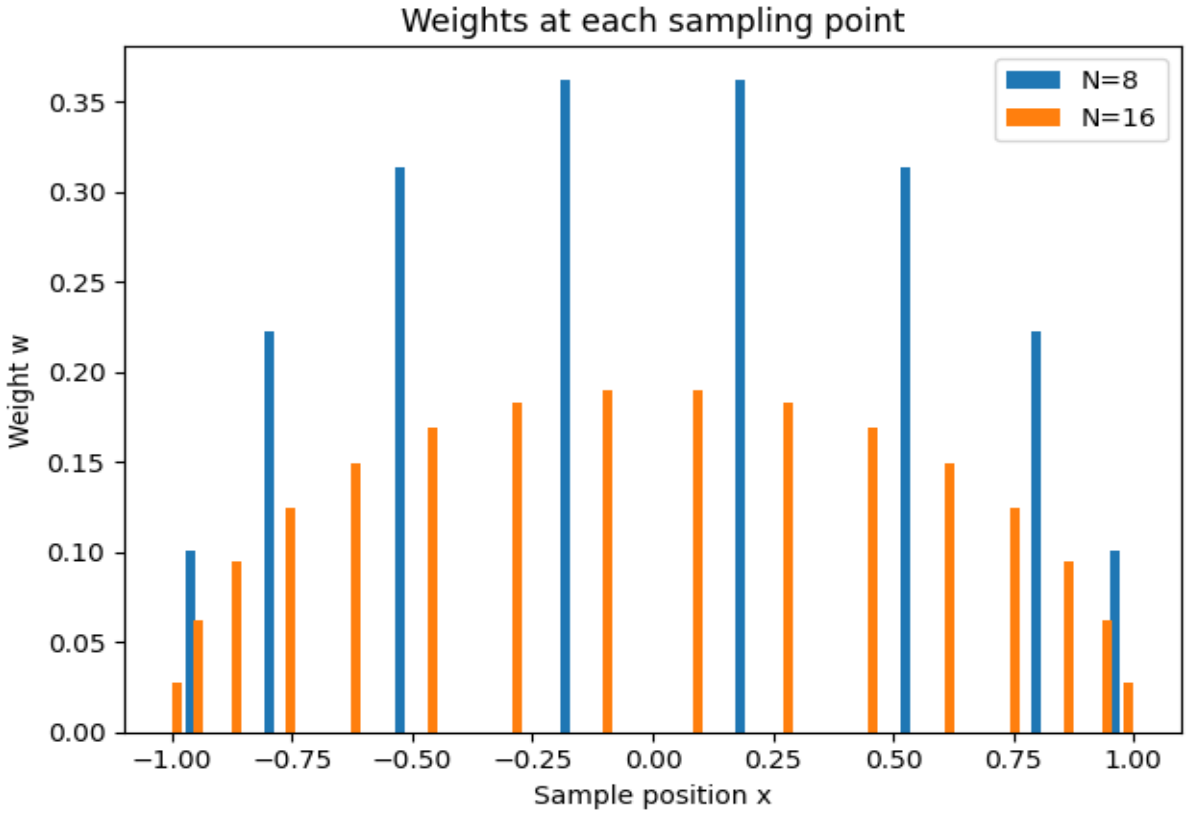
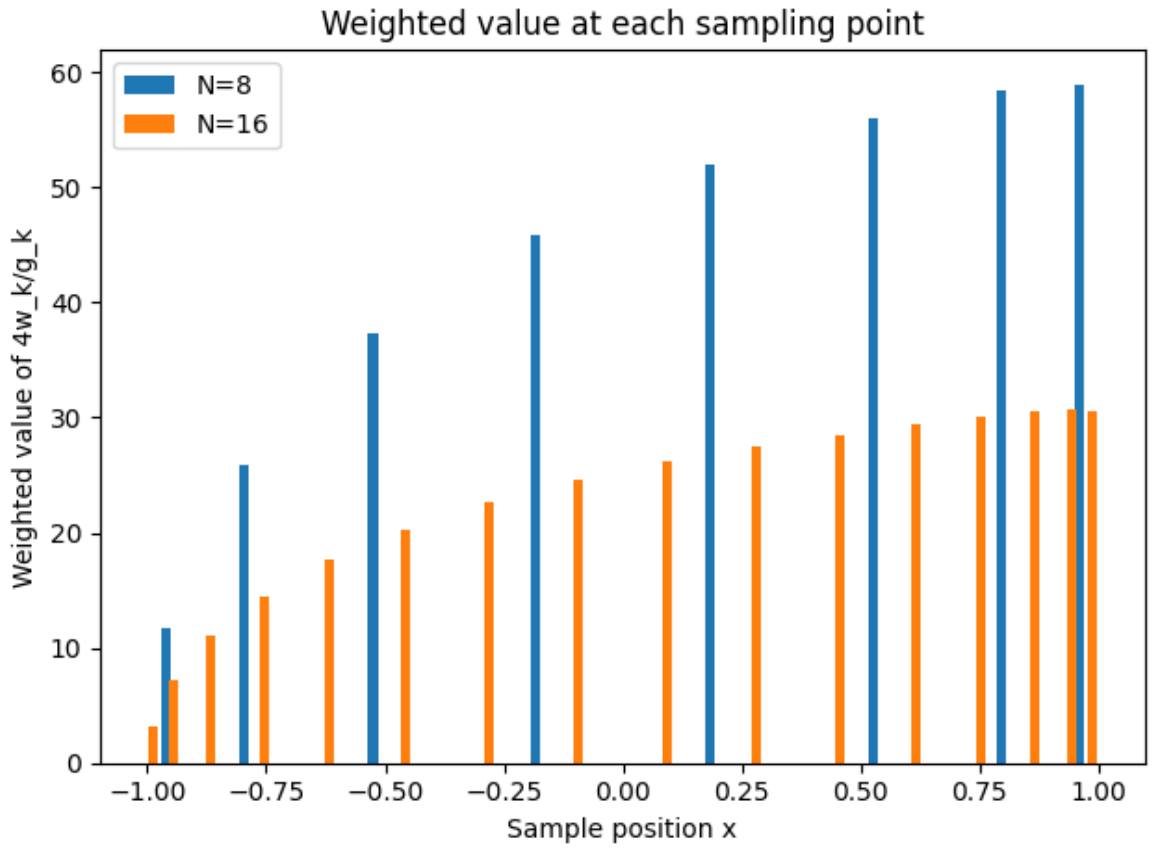

Weights at each sampling point
N=8
N=16
Weight w
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Sample position x

Figure 2.3



According to figure 2.1, the integral diverges towards infinity as x approach $x_0 = 0.01m$ (The upper limit). As the plot has demonstrated, the value of the final sample point for N = 16 is much higher than that of N=8, due to the fact that the final value is closer to the upper limit.

As figure 2.2 has shown, the weight of each point is reduced if more points are added. The weight of each point also decreases when they approach either the upper/lower limits of x. This effect is going to counteract the effect of having more (much larger) values closer to the upper limit, but it is still having an effect on the accuracy on the final result.

The effect is demonstrated in the weighted values graph (Figure 2.3), where the weighted values of sample points closer to the upper limit still carries relatively heavy weights in the result. This explains why Gaussian quadrature is not as accurate as it was supposed to be – more sapling points are needed to cover the diverging upper limit, thus a large amount of points are needed to produce a reasonably accurate result.

Q2c

Solve the following ODE:

$$m\frac{d^2x}{dt^2} = -kx \qquad 2.5$$

$$\frac{d^2x}{dt^2} + \frac{k}{m}x = 0 \tag{2.6}$$

which has the solution:

$$x(t) = x_0 \cos\left(\sqrt{\frac{k}{m}}t\right) \tag{2.7}$$

$$v(t) = \frac{dx}{dt} = -x_0\sqrt{\frac{k}{m}}\sin\left(\sqrt{\frac{k}{m}}t\right) \tag{2.8}$$

It is given that the particle travels at the speed of light when it passes the origin, which first happens at $t = \frac{1}{4}T$, and with $T = 2\pi\sqrt{\frac{m}{k}}$, the expression could be written as:

$$c = -x_0\sqrt{\frac{k}{m}}\sin\left(\frac{\pi}{2}\sqrt{\frac{m}{k}}\sqrt{\frac{k}{m}}\right) \tag{2.9}$$

additionally, since speed is the magnitude of velocity, the negative sign can be removed, which gives:

$$c = x_0\sqrt{\frac{k}{m}} \tag{2.10}$$

$$x_0 = c\sqrt{\frac{m}{k}} \tag{2.11}$$

plug in $k = 12\frac{N}{m}$, $m = 1\text{kg}$, $x_0 = \frac{1}{2\sqrt{3}}c = 8.66\times10^7\frac{m}{s}$.

Q2d

As the textbook has stated, the calculation of I_N and ϵ_N are not nested, therefore to extrapolate the error of I_{200} from ϵ_{16} and ϵ_8 (already known) is not possible. Additionally, due to the diverging nature of this integral, using the error reduction factors N^{-2N} (doubling N) and $\frac{c}{N^2}$ (adding an additional point) is unreliable as well, since adding more points will always lead to a larger value, as mentioned in part b. Therefore, the only way to estimate errors for N = 200 is to use the formula:

$$\epsilon_{frac} = \frac{I_{classical} - I_{200}}{I_{classical}} \tag{2.12}$$

which generated the following output from the code:

```
fractional error for N = 200: 0.0016078026473630399
```

Q2e

Pseudocode

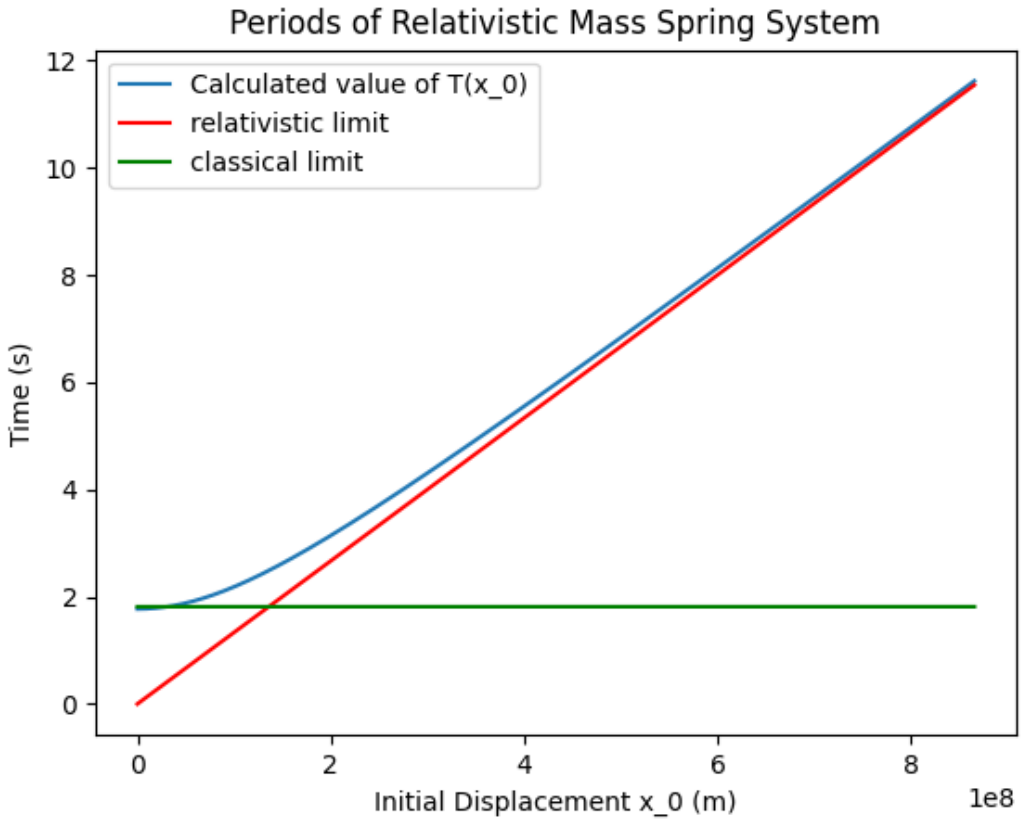
1. Define constants x_0, x_c, dx and create empty arrays for plotting
2. While x is smaller than x_c:
   Calculate x, w, using the gaussxw function with N = 16
   (Note: this is an arbitrary selection mainly aimed to reduce time usage while maintain a certain degree of accuracy)
   calculate xp, wp using:

$$xp = 0.5(b-a)x + 0.5(b+a) \tag{2.13}$$

$$wp = 0.5(b-a)w \tag{2.14}$$

   calculate the integral using for loop outlined in part a. Extract this value and add it to the empty arrays. Iterate.
3. Plot the result of T versus x_0, the classical limit, the relativistic limit.

Figure 2.4



When the value of x_0 is small, the period of the mass spring system is very similar to the classical value, which is expected for small amplitude oscillations. As the value of initial displacement reaches a certain threshold, the period of the system started to diverge from the classical limit and observes neither classical nor relativistic limits. As the initial displacement continues to grow, which makes the speed of the particle to approach the speed of light, the period of the particle started to converge towards the relativistic limit, but never to reach it since its speed is always lower than c.

Q3

In this question, we are asked to calculate the uncertainties of position and momentum to a wave equation. First, we define the wave equation been

$$\psi_n(x) = \frac{1}{\sqrt{2^n n!\sqrt{\pi}}} \exp\left(-\frac{x^2}{2}\right) H_n(x) \tag{3.1}$$

Where $H_n(x)$ is the Hermite polynomial, defined as

$$H_n = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \tag{3.2}$$

The first two terms are defined as:

$$H_0(x) = 1 \tag{3.3}$$

$$H_1(x) = 2x \tag{3.4}$$

To calculate the potential energy, we use

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} x^2 |\psi_n(x)|^2 dx \tag{3.5}$$

To numerically integrate over ∞, we change the integrating variable $x = \frac{z}{1-z^2}$, and $dx = \frac{1+z^2}{(1-z^2)^2} dz$. For such change of variable, the equation follows

$$\int_{-\infty}^{\infty} f(x)dx = \int_{-1}^{1} \frac{1+z^2}{(1-z^2)^2} f\left(\frac{z}{1-z^2}\right) dz \tag{3.6}$$

Therefore, the integrating function of potential energy $\langle x^2 \rangle$ becomes:

$$\langle x^2 \rangle = \int_{-1}^{1} \frac{1+z^2}{(1-z^2)^2} \left(\frac{z}{1-z^2}\right)^2 \left|\psi_n\left(\frac{z}{1-z^2}\right)\right|^2 dz \tag{3.7}$$

We also want to calculate kinetic energy:

$$\langle p^2 \rangle = \int_{-\infty}^{\infty} \left|\frac{d\psi_n(x)}{dx}\right|^2 dx \tag{3.8}$$

Notice the derivative inside equation 3.8, we noticed that

$$\frac{dH_n(x)}{dx} = 2nH_{n-1}(x) \tag{3.9}$$

Therefore

$$\frac{d\psi_n(x)}{dx} = \frac{1}{\sqrt{2^n n!\sqrt{\pi}}} \exp\left(-\frac{x^2}{2}\right) [-xH_n(x) + 2nH_{n-1}(x)] \tag{3.10}$$

The same infinite integral appears in equation 3.8, we apply the same change of variable:

$$\langle p^2 \rangle = \int_{-1}^{1} \frac{1+z^2}{(1-z^2)^2} \left|\psi_n{}'\left(\frac{z}{1-z^2}\right)\right|^2 dz \tag{3.11}$$

The total energy follows that

$$E = \frac{1}{2}(\langle x^2 \rangle + \langle p^2 \rangle) \quad 3.12$$

The uncertainty of position and momentum is the root mean square of position and momentum:

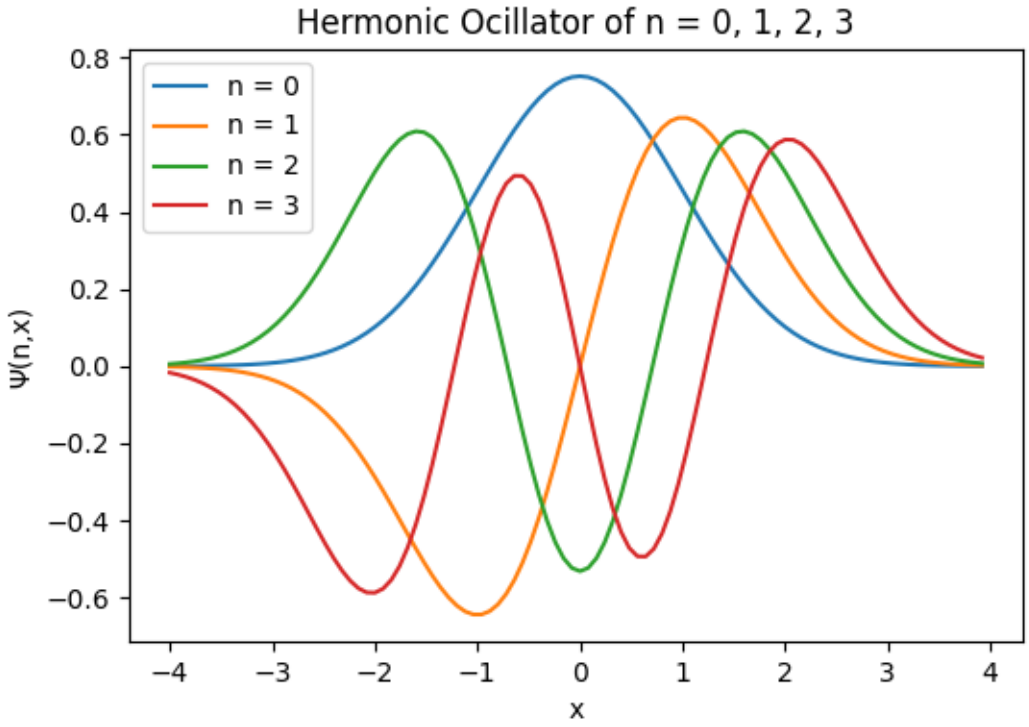
$$\Delta x = \sqrt{\langle x^2 \rangle} \quad 3.13$$

$$\Delta p = \sqrt{\langle p^2 \rangle} \quad 3.14$$

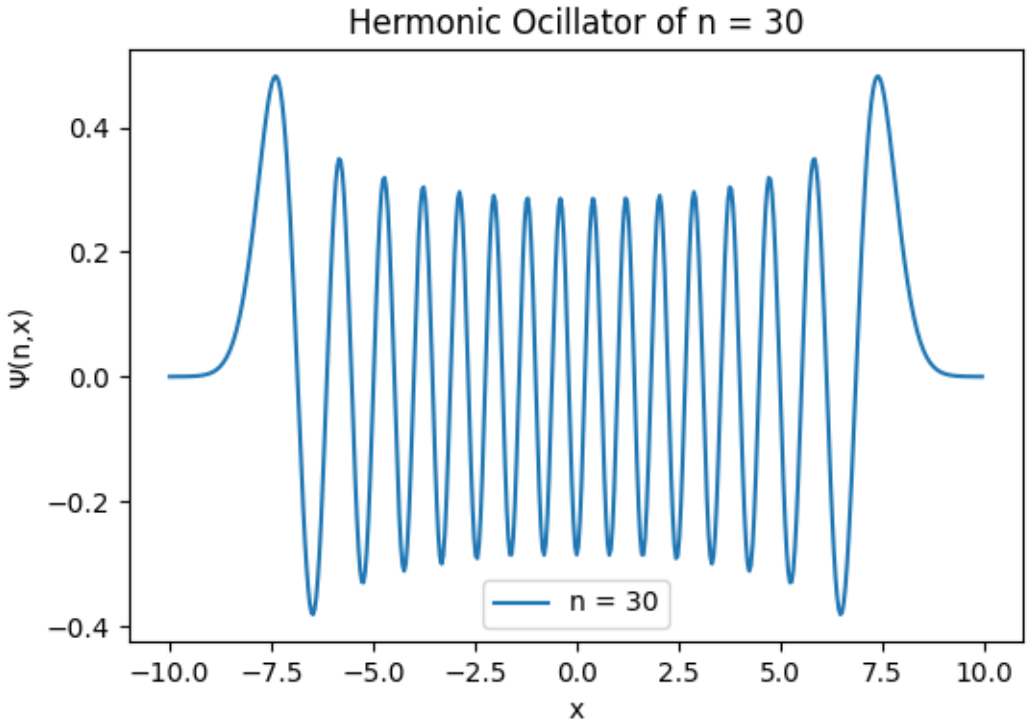
Pseudocodes:

1. Define Hermite polynomials according to equation 3.2 – 3.4
2. Define the wave equation according to equation 3,1
3. Define the derivative $\psi_n{}'(x)$ according to equation 3.10
4. Define the method gaussxw created by Mark Newman
   # FOR PART (a)
5. Define the range x using np function arange with lower limit, higher limit and step size (-4, 4, abs(4-(-4))/100)
6. Create a plot frame
7. Run a for loop over 0, 1, 2, 3 as the n and plot the wave function ψ_n on the same plot
   # FOR PART (b)
8. Redefine range x similar to step 5 with parameter (-10, 10, abs(4-(-4))/500)
9. Set n = 30 and plot wave function
   #FOR PART (c)
10. Define the integrating function average_x_sq according to the interior of equation 3.7
11. Define the integrating function average_p_sq according to the interior of equation 3.11
12. Define gaussian quadruple parameter N = 100 and run the gaussxw(N)
13. Setup a loop from n = 1 to 15 as required to calculate uncertainties
14. Create a data frame to store $n, \langle x^2 \rangle, \langle p^2 \rangle, E, \sqrt{\langle x^2 \rangle}, \sqrt{\langle p^2 \rangle}$
15. Run the integration loop and apply the integration for $\langle x^2 \rangle$ and $\langle p^2 \rangle$
16. Calculate $E, \sqrt{\langle x^2 \rangle}, \sqrt{\langle p^2 \rangle}$
17. Store data

Q3A FIGURE 3.1



Q3B FIGURE 3.2



Q3C CHART 3.1

n	$\langle x^2 \rangle$	$\langle p^2 \rangle$	E	$\sqrt{\langle x^2 \rangle}$	$\sqrt{\langle p^2 \rangle}$
0	0.5	0.5	0.5	0.707107	0.707107
1	1.5	1.5	1.5	1.22474	1.22474
2	2.5	2.5	2.5	1.58114	1.58114
3	3.5	3.5	3.5	1.87083	1.87083
4	4.5	4.5	4.5	2.12132	2.12132
5	5.5	5.5	5.5	2.34521	2.34521
6	6.5	6.5	6.5	2.54951	2.54951
7	7.5	7.5	7.5	2.73861	2.73861
8	8.49999	8.49998	8.49998	2.91547	2.91547
9	9.49985	9.49993	9.49989	3.08218	3.0822
10	10.4999	10.5004	10.5001	3.24035	3.24042
11	11.5022	11.5016	11.5019	3.39149	3.3914
12	12.5057	12.4976	12.5016	3.53634	3.53519
13	13.4864	13.4812	13.4838	3.67239	3.67168
14	14.4282	14.5028	14.4655	3.79845	3.80825
15	15.4941	15.6343	15.5642	3.93625	3.95402

Note: terminal print out result available when running the code

Since the system is not normalized, we are not having an exact energy print out in terms of $\hbar$. However, it is obvious that for E, it is a quantized follow that $E_n = 0.5 + n$, especially visible from n = 0 to 5 as the computational error are smaller than the rounding digit. The increased error on E is due to limited number of Gaussian quadrature used that is not sufficient for the highly fluctuating function.
Quantum harmonic oscillators has analytic total energy of

$$E_n = \frac{2n+1}{2}\hbar\omega \tag{3.15}$$

Which for our unnormalized system

$$E_0 = 0.5 = \frac{\hbar\omega}{2} \tag{3.16}$$

Where k is some normalization factor, we have

$$\hbar\omega = 1 \tag{3.17}$$

Heisenberg's uncertainty principle states that:

$$\Delta x \Delta p \geq \frac{\hbar}{2} \tag{3.18}$$

For our result, we have $\Delta x \Delta p \approx E$, (the approximation is due to rounding error, analytically it should be equal) the smallest energy is of $E_0 = \hbar\omega/2$

$$E_0 = \frac{\hbar\omega}{2} \geq \frac{\hbar}{2} \tag{3.19}$$

The above equation always holds because $\omega > 1$ for any particle considered in quantum limits. For any energy larger than E_0, the uncertainty is greater. Therefore, the Heisenberg's Uncertainty Principle holds in our system.

.