Lab 3 Report

Habiba Zaghloul, Augustine Tai

October 1, 2022

Q1 a

Pseudocode

- import numpy library
- use a for loop to create an array of h's ranging from 10^{-16} to 10^{0}, where h increases by a factor of 10 each time
- define the function we want to differentiate
- find the first derivative by using the forward difference method using a for loop that ranges to n=17 because we have 17 values of h
- calculate the true value of the derivative to compare with the values we got
- calculate the error by subtracting the true value from the calculated values and taking its absolute value
- use plt.step to plot a step function and plt.loglog to log scale e and h
- use a for loop again to calculate the derivative using the central difference method

h	f'	error
10^{-16}	-1.1102230246251565	3.894003915357024e-17
10^{-15}	-0.7771561172376095	3.8940039153570245e-16
10^{-14}	-0.7771561172376096	3.894003915357024e-15
10^{-13}	-0.7793765632868599	3.894003915357024e-14
10^{-12}	-0.7788214517745473	3.8940039153570244e-13
10^{-11}	-0.7787992473140548	3.894003915357024e-12
10^{-10}	-0.7788014677601041	3.894003915357024e-11
10^{-9}	-0.7788008016262893	3.894003915357025e-10
10^{-8}	**-0.778800790524059**	**3.8940039153570246e-09**
10^{-7}	-0.7788008216103037	3.894003915357024e-08
10^{-6}	-0.7788011724407795	3.894003915357024e-07
10^{-5}	-0.7788046770040856	3.894003915357025e-06
10^{-4}	-0.7788397166208494	3.894003915357025e-05
10^{-3}	-0.7791895344301247	0.0003894003915357024 4
10^{-2}	-0.782629857128958	0.003894003915357024 6
10^{-1}	-0.8112445700037385	0.03894003915357025
10^{0}	-0.6734015585095405	0.38940039153570244

Table 1: Forward difference method

The analytic value we get from differentiating the function

$$f(x) = \exp -x^2$$

at x=0.5 is

$$f'(0.5) = -0.7788007830714049$$

By looking at the table, the closest value to this is when $h = 10^{-8}$ coming at $f' = -0.778800790524059$

Q1 b

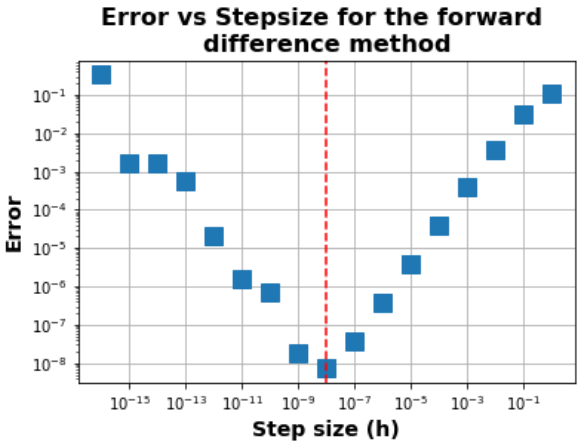


Figure 1: log-log plot - forward difference method

The rounding error dominates when we make h too small and the truncation error dominates when h is too big. We know this by looking at equation (5.91) in the text given as

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

The first term in this equation represents the rounding error and the second is the approximation/truncation error. Since we have h in the denominator in the first term, we see why the rounding error becomes really large when we make h too small and vice versa for the truncation error. Therefore, there is a point where this function reaches a minimum then increases again. By looking at the graph, the minimum is clearly at $h = 10^{-8}$

Q1 c

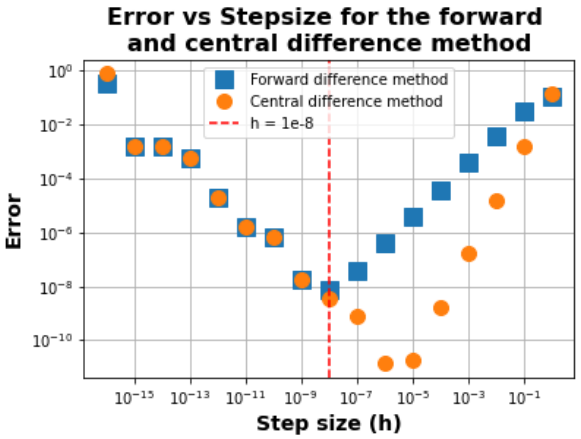


Figure 2: log-log plot forward + central difference

In this case, the central difference scheme was proven to be more accurate than the forward difference scheme because at its minimum, $h = 10^{-6}$ and the error was lower than $\epsilon = 10^{-10}$ while the error for the forward method at its minimum was $\epsilon = 10^{-8}$. This tells us that the central difference method is roughly 100 times more accurate.

The reason that the error for the central difference method is this much lower is because the truncation error has a higher order of h than the previous method.

$$\epsilon_{truncation} = \frac{1}{24}h^2|f'''(x)|$$

Q2

Pseudo-code for Q2 (all parts)

- import needed libraries (numpy, gaussxw, and scipy)
- define constants used (spring constant, mass, c, N, etc.)
  - note that i defined x_0 in terms of m (0.01m) instead of cm (1cm) to keep unit consistency
- define a function f(x), that returns the expression we are trying to integrate (equation 6-7)
- define a function, Gauss(), that uses Gaussian quadrature to integrate f(x)
  - calculate the sample points and weights using gaussxwab when given number of samples (N) and upper-lower limits
  - calculate the integral using the Gaussian method (follows Example 5.2 in the textbook)
  - calculate the period by multiplying the integral by 4 (equation 7)
  - calculate the period by multiplying the integral by 4 (equation 7) and outputs the calculated period, sample points, integrand, and weighted values when called
- calculate the classical value = $2\pi(\sqrt{m/k})$
- part a) calculate the fractional error of N=8 and N=16 using $\frac{|calculatedperiod-classicallimit|}{classicallimit}$
- part b) plot intergrand vs sample points and weighted values vs sample points plots for N=8 and 16 on the same plot (add legend for clarity)
- part c) calculate initial displacement xc needed for the particle to reach a speed of c at x=0
  - using equation v = $\sqrt{k(x_o^2 - x^2)}$, rearranging + substitute x = 0 and v = c results in $x_c = \sqrt{c^2/k)}$
- part d) calculate the integral for N=200 (using same code)
- calculate the percentage error for N=200 using equation: fractional error*100%
- part d) create an array using linspace for the range of x_0 values:

$$1 < x < 10x_c$$

- Modify the Gauss() function to calculate the period for the range of x_0 values (use np.linspace for range of values)
  - create empty space to store the 200 periods calculated for each x_0 value when N=200
  - add For loop into existing function to calculate the period at each x_0
  - the rest of the codes is the same as before and returns the 200 periods in a list to be plotted
- plot the periods against x_0, along with the classical and relativistic limit (found in lab handout)

Q2 a

Number of samples (N)	Period	Fractional error
8	1.7301762343365563	0.046103847838190645
16	1.770715490242243	0.023753384658487007

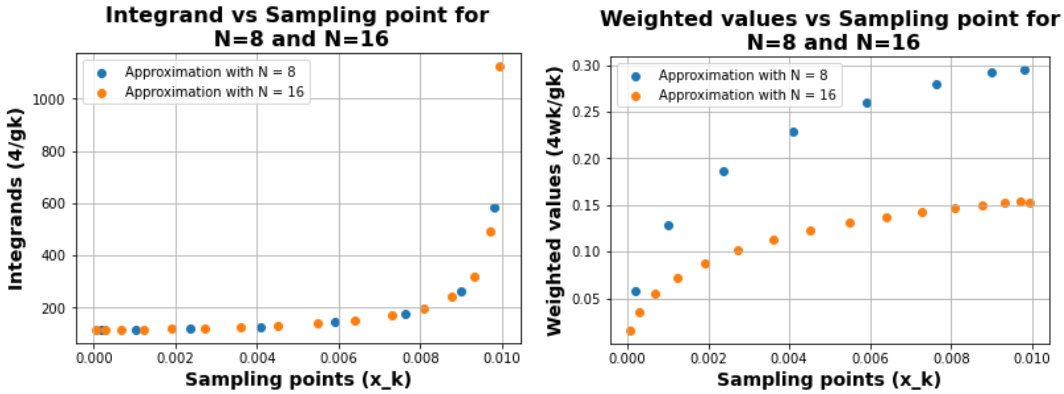
Table 2: Table of values containing the period and fractional error corresponding to the number of samples used

The true value of the period was determined to be:

$$period = 2\pi\sqrt{m/k} = 1.8137993642342176$$

By looking at Table 2, when N was increased from 8 to 16, the fractional error also decreased in value from 0.046 to 0.023. This relationship suggests that as N increases, the calculated period should also approach the true value.

Q2 b



(a) integrand vs sampling point

(b) weighted value vs sampling point

Figure 3: plots of integrand vs sampling points and weighted values for sampling points

By looking at Figure 3 a, as the position of the sampling points increases, the values of the integrand 4/gk increase as well. When comparing N=8 and N=16 in Figure 3 a, they are approximately the same, the only difference being that N=16 has more points. Based on this graph alone, there seems to be little to no difference between using N=8 and N=16 aside from the number of sample points. On the other hand, Figure 3 b reveals a significant difference between N=8 and N=16 when considering their weighted values. By observing 3 b, N=16 has more values but is weighted less, while N=8 has fewer values but is weighted more. In tandem with the error values calculated in part a, Figure 3 suggests that using more sample points will make the calculated period closer to the actual value. This makes sense as we know that by using more sample points in the Gaussian quadrature, we get smaller errors and because we use more values, each individual 'weight' would be smaller, explaining Figure 3 b. As a result, by using more sample points, we get a more accurate result.

Q2 c

By using equation: v = $\sqrt{k(x_o^2 - x^2)}$, substituting x=0 and v=c (light speed), and rearranging to find $x_0 = x_c$ we obtain: $x_c = \sqrt{c^2/k}$. Inserting values c from scipy (constants) and k=12N/m, the initial displacement of a particle initially at rest leading to a speed equal to c at x=0 is 86542628.16365978m

Q2 d

By changing the number of sample points to N=200, the percentage error of using N=200 when compared to the value given by the small amplitude case (classical) was calculated using:

$$(\frac{|calculatedperiod - classicallimit|}{classicallimit}) * 100$$

As a result, the percentage error when using N=200 was determined to be 0.19548536075292894%.

Q2 e

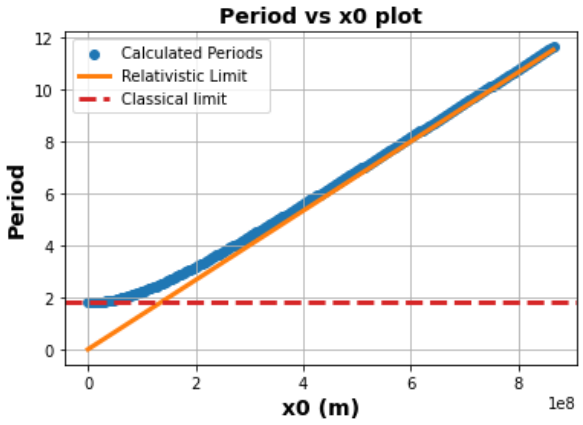


Figure 4: period vs initial position (x_0) plot along with the classical and relativistic limits

By observing Figure 4, for small values of x_0 (right-hand side), the period of oscillation follows the classical limit for a brief moment. However, as the value of x_0 increases, the period of oscillation follows the relativistic limit. As a result, the classical limit, when x_0 is small, and the relativistic limit, when x_0 is large, seems to hold true in Figure 4.

Q3 a

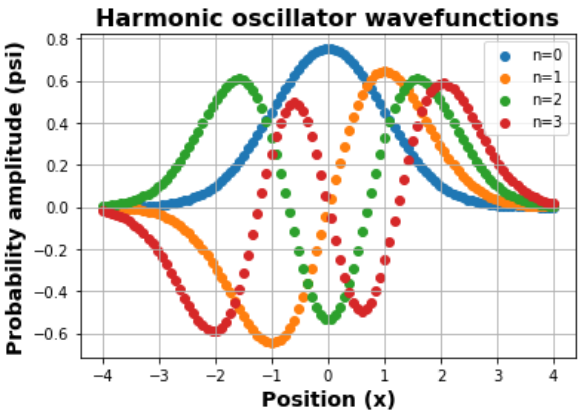


Figure 5: Harmonic oscillator wave functions for n=0, 1, 2, and 3

By using a function that calculates the Hermite function, the wave functions of n=0, 1, 2, and 3 were able to calculated using equation 8 (from the lab) and plotted in Figure 5.

The wave functions in figure 5 represent the first 4 position eigenstates of the particle in a harmonic oscillator.

Q3 b

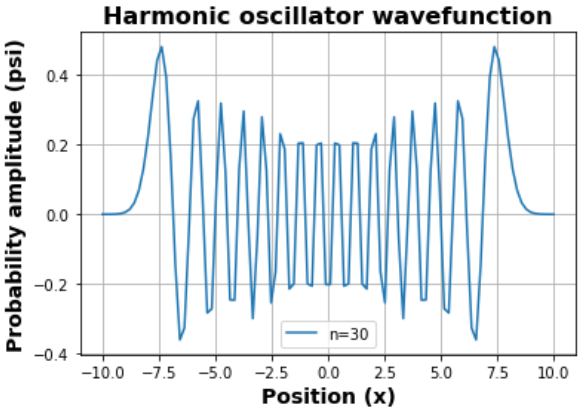


Figure 6: Harmonic oscillator wave function for n=30

Q3 c

Pseudocode for all of Q3 (see commented code for more details)

- import needed libraries: gaussxw file in same folder as code
- Part a) define a Hermite function when given x (position) and n (level): Hermite(x,n)
  - use If condition:when n = 0, return value of 0. Use Elif condition: when n = 1 return 2*x. Use Else condition: return equation 9 for all other cases (values greater than 1).
- define a function that calculates the wave function: Wave(x,n)
  - make storage space for the loop
  - use For loop to calculate the wave function values from -4 to 4 (x-values)
  - calculate wave function using equation 8 + return values when function is called
- define the range of x-values for the wavefunction using np.linspace
- Calculate the wave function for each energy level by calling the function Wave(x,n), where n=0, 1, 2, 3. Afterwards plot them on the same graph
- Part b) create a range of new x values that goes from -10 to 10, i used 100 points between these 2 values
- Call the function Wave(x,n) using the new range of x-values and n=30
- ***The code takes a while to this (less than 2-minutes), due to the way i created the Hermite function, I don't know how to fix this, but it still works despite taking around 1-2 minutes.
- Plot the resulting wavefunction for n=30 from -10 to 10
- Part c) set some constants to be used: sample points (N)=100, limits of integration (a,b) = -pi/2 and pi/2 respectively
- define a modified wavefunction for part c
  - we don't have a range of x-values to calculate + loop for the wavefunction. So "x[i]" will become just "x", since we are using the change of variables (z) later.
  - the rest of the function is exactly the same as before

- Define a function Gauss() that use Gaussian quadrature to integrate a function, the code is from Example 5.8, but with some variable name changes + additions
  - the Gauss() function takes in the function you want to integrate, number of sampling points, upper and lower limits, and the energy of the wavefunction (n)
- define a function Pos() that calculates pre-integrated x^2 based on equation 12 and the change of variables (z) indicated in example 5.8, please see commented code for more info
- use Gauss() function to integrate Pos() to obtain x^2
- define a function Derivative() which calculates the derivative of a wave function using equation 11
  - when n¿0 it follows the regular equation 11 for the derivative calculation
  - create a special case for when n=0. Because of the nature of the derivative defined in equation 10 where $H_{n-1}(x)$, at n=0 we have a probelem. To solve this a set $H_{n-1}(x)$=0 when n=0 to solve this problem. Although because of this the true value of the wavefunction derivative at n=0 may be wrong/altered which would effect p^2 as well.
- define a function Mom() that calculates pre-integrated p^2
  - uses equation 13 for the calculation along with the change of variables method indicated in 5.8
- use Gauss() function to integrate Mom() to obtain p^2
- Calculates + print x^2 and p^2 for values of n from 0 to 15. Use a For loop that changes the value of n when using the Gauss() function to integrate Pos() and Mom()
- Calculates + print energy (E). Loops (For loop) through values of n from 0 to 15 using equation 14
- Define a function Uncertainty() that takes the square root of x^2 and p^2 for n=0-15, using a For loop
- Optional plots the uncertainties of x^2 and p^2 to see relationship
- tried to be as brief as possible, please read commented code for more specific details

The uncertainty in position and uncertainty in momentum are directly proportional to one another. By plotting the uncertaintys against each other, we can see this relationship very clearly. We have a y=x relationship between these two parameters. Now since the uncertainty values are approximately

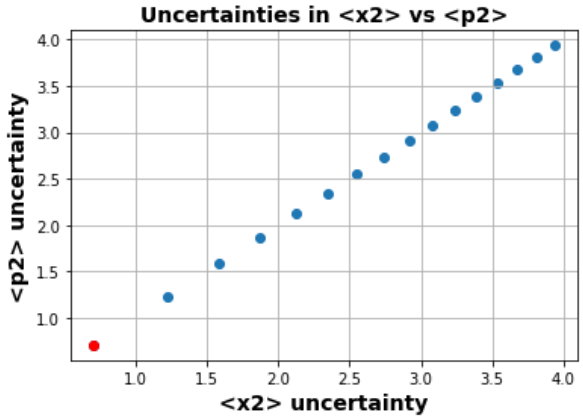


Figure 7: This plot shows that $\sqrt{< x^2 >} \propto \sqrt{< p^2 >}$

equal, we can deduce a simpler relationship for the energy. We know that the energy is

$$E = \frac{1}{2}(< x^2 > + < p^2 >)$$

but if we have

$$\sqrt{< x^2 >} \simeq \sqrt{< p^2 >}$$

then

$$< x^2 > \simeq < p^2 >$$

Substituting this in the energy equation we get

$$E = \frac{1}{2}(< x^2 > + < x^2 >)$$

$$= \frac{1}{2}(2 < x^2 >)$$

$$= < x^2 >$$

$$= < p^2 >$$