HW5 Exercise 5.13

Nina Martinez Diers*
Bryn Mawr College Department of Physics
(Dated: March 18, 2024)

In this assignment, an algorithm was written to calculate the wavefunction and uncertainty of a quantum harmonic oscillator

1. INTRODUCTION

The wavefunction of a particle in a one-dimensional harmonic oscillator is given by

$$\psi_n(x) = \frac{1}{\sqrt{2^n n! \sqrt{\pi}}} e^{-x^2/2} H_n(x) \tag{1}$$

at the n^{th} energy level.[1] $H_n(x)$ is the Hermite polynomial as a function of x for a given energy level n which is given by the relation: [1]

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x). \tag{2}$$

Using the substitution $n = n-1$, the Hermite polynomial function can be rewritten as

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x), \tag{3}$$

which is more practical for the iterative calculations of the algorithm.

2. EXPERIMENT

My initial attempt at coding an algorithm for the Hermite polynomials involved directly implementing Eq. 3 to iteratively calculate the n^{th} polynomial, as demonstrated by the pseudocode in Fig. 1. However, the runtime of this algorithm quickly became unreasonably high with larger n. To streamline the calculation, instead of having the algorithm work backwards first before calculating the Hermite polynomial, created a list of the Hermite polynomials, setting up a loop that used the last two elements of the list to calculate the next Hermite polynomial until the n^{th} polynomial had been calculated. This was done using a **while** loop that iterates until the length of the Hermite polynomial list has $n+1$ elements. The function then returns the n^{th} element in the list. This version of the Hermite polynomial algorithm was much more efficient and reduced the runtime from minutes to less than a second at $n = 30$. Using the Hermite polynomial algorithm, the wavefuction of the quantum harmonic oscillator at energy level n could be calculated using Eq. 1. To determine the quantum uncertainty in the position of a particle in the n^{th} level of a harmonic oscillator, we calculated the root-mean-square of the position $\sqrt{\langle x^2 \rangle}$ according to

$$\langle x^2 \rangle = \int_{-\infty}^{\infty} x^2 |\psi_n(x)|^2 \, dx. \tag{4}$$

Because the square of the wavefunction gives the probability distribution for the position of a quantum particle in the harmonic oscillator, Eq. 4 gives the expectation value for the square of the position of the particle.

Mark Newman's algorithm for gaussian quadrature is used to to perform the necessary integration of Eq. 4. [1] Because it is not possible to directly integrate over an infinite range, we implement a change of variables $x = tan(z)$:[1]

$$\langle x^2 \rangle = \int_{-\pi/2}^{\pi/2} \frac{tan^2(z)\,|\psi_n(tan(z))|^2}{cos^2(z)} \, dz. \tag{5}$$

```
Exercise 5.13: Quantum uncertainty in the harmonic oscillator
  wavefunction of n^th energy level:
  H(n+1,x) = 2x H(n,x) - 2n H(n-1,x) , or equivalently: H(n,x) = 2x H(n-1,x) - 2(n-1)H(n-2,x)
    H(n=0,x) = 1  &  H(n=1,x)=2x
pseudocode to calculate Hermite polynomials wrt n = integer & any given x
def H(n,x):
    if n=0:
      H=1
    elif n=1:
      H=2x
    else:
      H(n,x) = 2x H(n-1,x) - 2(n-1) H(n-2,x)
   return H
pseudocode to calculate harmonic oscillator wavefunctions for n=0,1,2,3  from -4≤x≤4
 for n in range(0,4):
   ψ=[] , x = np.linspace(-4,4,100, endpoints=True)
    for i in x:
      ψ.append((2**n * math.factorial(n)* math.pi**(1/2))**(-1/2) * exp(-x**2/2) * H(n,x))
    plt.plot(x, ψ, label = n)
plt.xlabel(r'$x$')
plt.ylabel(r'$\phi_n(x)$')
plt.legend()
plt.show()
outside of loop
                having this be inside the loop will make the ψn plots layer on top of each other without having to save the data

 n=30
 ψ=[]    x = np.linspace(-10,10, 20 , endpoints=True)
 for i in x:
   ψ.append((2**n * math.factorial(n)* math.pi**(1/2))**(-1/2) * exp(-x**2/2) * H(n,x))
```

FIG. 1: Pseudocode for generating the Hermite polynomials, graphing the wavefunction of a particle in a one-dimensional harmonic oscillator for different n, and graphing the wavefunction for $n = 30$.

*Electronic address: nmartinezd@brynmawr.edu

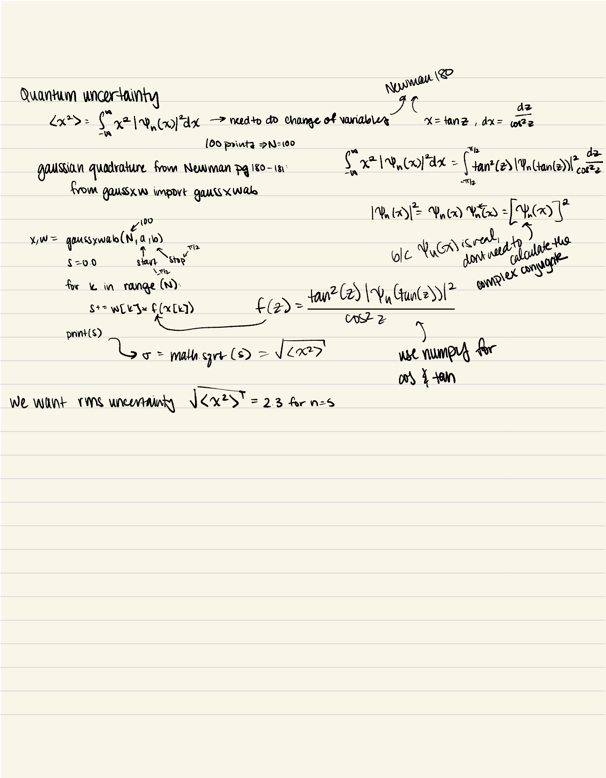

FIG. 2: Pseudocode for calculating the uncertainty in the position of the a particle in a one-dimensional harmonic oscillator at the n^{th}, and graphing the wavefunction for $n = 30$.

This allows us to change the limits of integration to a definite range, making it possible to integrate over the entire range of x-values for a given value of n.

3. RESULTS

Figure 3 shows the impact of increasing energy level n on the wavefuction of the the particle in the harmonic oscillator.

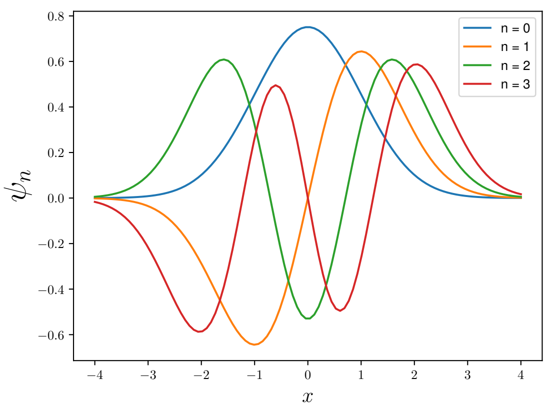

FIG. 3: The wavefunctions of a particle in a quantum harmonic oscillator with increasing energy levels.

When $n = 30$, the bounds where wavefunction stops oscillating and drops to zero is around $x = 9$, while for smaller n this is happens sooner: for example for $n = 1$ this occurs at $x = 3$. When n increases, the range of positions in the quantum well that the particle is likely to occupy increases. This behavior is shown in both Fig. 3 and Fig. 4.

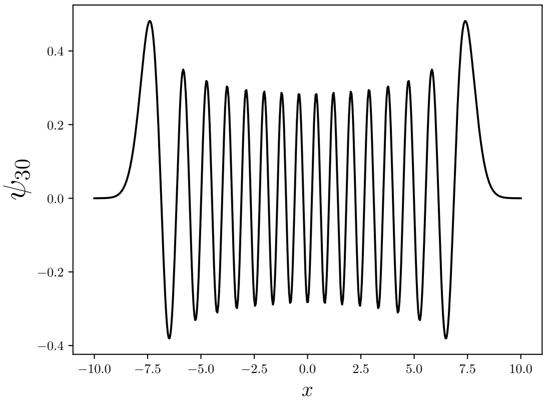

FIG. 4: The wavefunction of a particle in a quantum harmonic oscillator with energy level 30.

4. CONCLUSIONS

The algorithm to calculate quantum uncertainty returned the expected value of the quantum uncertainty for $n = 5$. The quantum uncertainty was found to be 2.3452078797796547 for $n = 5$.

[1] M. Newman, *Computational Physics* (2013), revised and expanded ed., ISBN 978-1-4801-4551-1.

[2] *How to add a variable to python plt.title?*, https://stackoverflow.com/questions/43757820/how-to-add-a-variable-to-python-plt-title.

Appendix A: Comprehension Questions

This assignment took about 5 hours. Writing the pseudocode took 1 hour, coding took 2 hours, and the write-up took 2 hours. I think it took me a reasonable amount of time to do this problem. Something that I enjoyed learning was using both strings and variables to create a label in matplotlib.[2] I used this to make the legend for the plot of the wavefuctions for multiple n.

The most important (and the most interesting) thing I learned was while I was writing an algorithm for the Hermite polynomials. The solution that was the most straight-forward to me was inefficient, but I realized that did not mean that the alternative algorithm had to be complicated. It was still helpful for me to write the inefficient algorithm before figuring out a way to optimize it using the **while** loop.