Monte Carlo Method and Parallelization for the Hydrogen Atom

Brianna Isola, Dominic Payne

Introduction

In this report, we will solve the probability density for a hydrogen atom and plot its orbitals via the Monte Carlo technique. We will be following the steps outlined in *A Smooth Path to plot Hydrogen Atom via Monte Carlo Method* by Lobo et al. (2019) [4] to solve for the probability. Following this, we perform a scalability analysis for runtime improvement using OpenMp to parallelize our code.

The workload was divided with Brianna primarily working on the main code, and Dominic focusing on its parallelization.

1.1 Implementation and Repository Overview

This code can be run completely through `Monte Carlo Hydrogen Atom Solution.ipynb`, where it will check for the correct directory and build them as needed. This code can also be run on the command line as,

```
~/build$ cmake ..
~//build$ make
~//build$ mpirun -n N hsolv
```

Where N is the number of threads to run on. If running on the command line, a `build` and `data` directory must be created in order to compile to code and save the output files to the correct location.
Our files are divided up as described below,

- `Monte Carlo Hydrogen Atom Solution.ipynb`: code compilation notebook and data plotter for Hydrogen orbitals
- `main_hydrogen.cxx`: main compilation code
- `hydrolib.h`: header file
- `functions.cxx`: functions for components of the wave function
- `polynomials.cxx`: functions for solving Lagurres' and Legendre polynomials
- `vecmath.cxx`: general math/ vector-handling functions
- `writetofile.cxx`: functions for writing data to file

The output produces data files labeled by the quantum numbers and the number of points. These can also be plotted in the ipynb.

2 Quantum Mechanics in 1D

The Schrodinger Equation describes the wavefunction of a quantum-mechanical system, and can describe evolution of particle motion over time. We can define the time-independent Schrodinger equation as follows [3],

$$i\hbar\frac{\partial\Psi}{\partial t} = -\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} + V\Psi$$

If the system consists of only one particle, we can use the integral

$$\int_a^b |\Psi(x,t)|^2 \, \mathrm{d}x$$

to define the probability density of finding the particle between points a and b with probability $|\psi(\vec{r},t)|^2 d^3r$. For a given wave function we have $\psi = \psi_{nlm}$ where nlm are the three quantum numbers [4]. Index n is the radial quantum number defined by a range $n = 1, 2, 3....$ The second term, l, is the azimuthal quantum number which is defined in the range of $l = 0, 1, 2...n-1$. Finally, we have m, the magnetic quantum number, which falls in range $m = -l, ...l$. For the sake of this project, we will keep m at a constant $m = 0$, which prevents our values from becoming complex numbers.

3 The Hydrogen Atom

The wavefunction we will be solving for is the hydrogen atom. The hydrogen atom is a system consisting of a single proton and electron where the motion of the electron can be completely described by the Coulomb force between the two particles and the kinetic energy of the electron [4]. The Hamiltonian form of the Schrodinger equation for the hydrogen atom can be written as,

$$-\left(\frac{\hbar}{2m}\nabla^2 + \frac{e^2}{4\pi\epsilon_0 r}\right)\psi(r,\theta,\phi) = E\psi(r,\theta,\phi)$$

where e is the charge of the electron, m is the mass, and ϵ_0 is the constant of electrical permittivity in a vacuum. From this equation we find the analytical solution for ψ,

$$\psi(r,\theta,\phi) = R(r)Y(\theta,\phi)$$

with a radial component $R(r)$ and an angular component $Y(\theta,\phi)$. The radial component is defined as,

$$R_{nl}(r) = -\sqrt{\left(\frac{2Z}{na_0}\right)^3 \frac{(n-l-1)!}{2n[(n+l)!]^3}} e^{-\rho/2}\rho^l L_{n+l}^{2l+1}(\rho)$$

with parameter ρ

$$\rho(r) := \frac{2Zr}{na_0}$$

and the Laguerre polynomial $L_{n+l}^{2l+1}(\rho)$

$$L_{n+l}^{2l+1}(\rho) = \sum_{k=0}^{n-l-1}(-1)^{k+1}\frac{[(n+l)!]^2}{(n-l-1-k)!(2l+1+k)!k!}\rho^k$$

The radial component is quite straightforward and is easily implemented into code; however, we have a bit more work to do with the other component. The angular component defined as,

$$Y_l^m(\theta,\phi) = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}} P_l^{|m|}(\cos(\theta)) e^{im\phi}$$

where we find use of the Legendre polynomial and the associated Legendre polynomial,

$$P_l^{|m|}(x) = (-1)^{|m|} \left(1-x^2\right)^{|m|/2} \frac{d^{|m|}}{dx^{|m|}} P_l(x)$$

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} \left(x^2-1\right)^l$$

To solve for these polynomials in our code, we will need to implement a few strategies and take advantage of a few relationships between these polynomials. This will be described in the next section; Regardless, we can see that by solving for $\psi_{nlm}(r,\theta,\phi)$, we can readily find our probability density

$$|\psi_{nlm}(r,\theta,\phi)|^2 = |R_{nl}(r)|^2 \left|Y_m^l(\theta,\phi)\right|^2$$

4 The Monte Carlo Method

The method of calculation for our probability density values comes from the Monte Carlo technique. In general, the Monte Carlo technique is a method of calculation that relies on random sampling to predict the probability of different outcomes. Many industries use this technique such as finance and engineering. A helpful way to understand this method is exampled in Lobo et al. (2019) which uses the Monte Carlo technique to find the area of a circle using darts. By throwing a large amount of darts at a circle, the area of the circle can be determined using the ratio of the darts within and outside the circle.

The same principle can be applied to the wave function of the hydrogen atom. Instead of darts and the area of a circle, we will have randomized coordinates x, y, z and our probability density $|\psi_{nlm}(r,\theta,\phi)|^2$. The random coordinates will be converted to spherical coordinates as,

$$r = \sqrt{x^2+y^2+z^2}$$

$$\theta = \text{atan}(\frac{y}{x})$$

which will be plugged into the radial and angular components to find ψ. Since we will be looking at 2D plots of our orbitals, we can set $z = 0$, and not worry about ϕ. Our coordinates are generated within specified bounds of (Δx,Δy,Δz) where we estimate and upper limit for the highest potential value of the probability density given our quantum numbers.

To get this upper limit, we define a 'grid' and calculate $|\psi_{nlm}(r,\theta,\phi)|^2$ between our coordinate bounds and angle θ from 0 to π. We then take the maximum value for $|\psi_{nlm}(r,\theta,\phi)|^2$ on this grid as a means to guess what the highest value will be for our Monte Carlo-generated

values. By setting this limit, we prevent the code from generating values way outside where the probability exists, which would take a very long time to compute.

The final step in this process is to decided how many points we want, and to generate a random variable w between 0 and our upper-limit, and only record points that satisfy $w \leq |\psi_{nlm}(r, \theta, \phi)|^2$. This creates a truly random sampling method where we can take the recorded coordinates to plot the orbital diagrams of the atom.

4.1 A note on Legendre Polynomials

As per the radial equation, we will need to solve the Legendre Polynomials. To start, we write the Rodrigues' formula for the Legendre polynomial as [5]l,

$$P_n(x) = \frac{1}{2'n!}\frac{d^n}{dx^n}\left(x^2 - 1\right)^n$$

From this form, we can derive several recurrence relations that are satisfied by Legendre polynomials. In particular, we can look at the relation

$$(n+1)P_{n+1} = (2n+1)xP_n - nP_{n-1}$$

which can be rearranged simply as

$$P_{n+1} = \frac{(2n+1)xP_n - nP_{n-1}}{(n+1)}$$

By setting $n = n - 1$, we can get a new representation of the n_{th} Legendre polynomial [2],

$$P_n = \frac{(2n-1)xP_{n-1} - (n-1)P_{n-2}}{n}$$

Where we take advantage of the first two known Legendre polynomials

$$P_0(x) = 1$$
$$P_1(x) = x$$

to solve for our first set of P_{n-2} and P_{n-2} terms at $n = 2$. By running a loop as written in `polynomials.cxx`, we are able to numerically determine the Legendre polynomial for each value. It should also be noted here that n is not the quantum number, but the subscript of the function. For the wave function, our subscript is the quantum number l.

The wave function also requires we solve for the *associated* Legendre polynomials defined as,

$$P_n^m(x) = \left(1 - x^2\right)^{m/2}\frac{d^m P_n}{dx^m}$$

where n, m are subscripts. From Wikipedia, we find the three-term recurrence relation for the associated Legendre polynomials as [1],

$$2mxP_n^m(x) = -\sqrt{1-x^2}\left[P_n^{m+1}(x) + (n+m)(n-m+1)P_n^{m-1}(x)\right]$$

Such as with our original polynomials, we can set $m = m - 1$ and solve for P_n^m.

$$2mxP_n^{m-1}(x) = -\sqrt{1-x^2}\left[P_n^m(x) + (n+m-1)(n-m)P_n^{m-2}(x)\right]$$

$$P_n^m(x) = \frac{2mx}{-\sqrt{1-x^2}}P_n^{m-1}(x) - (n+m-1)(n-m)P_n^{m-2}(x)$$

Where we can define another recurrence relation to solve for the first derivative of the Legendre polynomial,

$$\left(1-x^2\right)P_n' = n\left(P_{n-1} - xP_n\right)$$

$$P_n' = \frac{n\left(P_{n-1} - xP_n\right)}{(1-x^2)}$$

Unfortunately, these equations pose a problem due to the first term within the associated functions that creates complex numbers within our code when $x > 1$,

$$-\sqrt{1-x^2}$$

Due to the time constraints of this project, it would be difficult to write our code to handle the complex numbers produced in this term. As a way around this, we take advantage of the relation,

$$P_n^0 = P_n$$

and only solve for conditions when $m = 0$.

Parallelization

As part of this project, we aim to show the improvements of code runtime through parallelizing the code with MPI. The Monte Carlo method provided an excellent setup for where parallel computing can shine. Since our values are randomly generated, there is no particular order that our data needs to be collected and therefore can be run on many cores simultaneously. Additionally, our main compilation code operates as one large loop, so it was quite easy to break into different components.

We also paralleled the initial for-loop for calculating *nlm* values; however, this had no noticeable effect on our runtime, and we removed it.

Testing

To make sure everything was working as expected, we ran/implemented tests throughout the code. In the parallelization part, we checked that the number of points we wanted to generate could be divisible by the number of threads we wanted to run on. Boundary conditions were also checked often within the Legendre polynomial code as well. These tests became permanent installations to make sure our code ran consistently, but many were also used temporarily to help debug and fix the code as we went along.

Results

From the output of the code, we were able to generate orbitals of the hydrogen atom at different nlm values. Figure 1 depicts the oribtals at $nlm = 200$ for twenty-thousand points and cut along the x-y axis for $z = 0$. The two distinct sub-shells can be seen. Fig. 2 and 3 presents more examples of the orbital plots for other quantum number values.

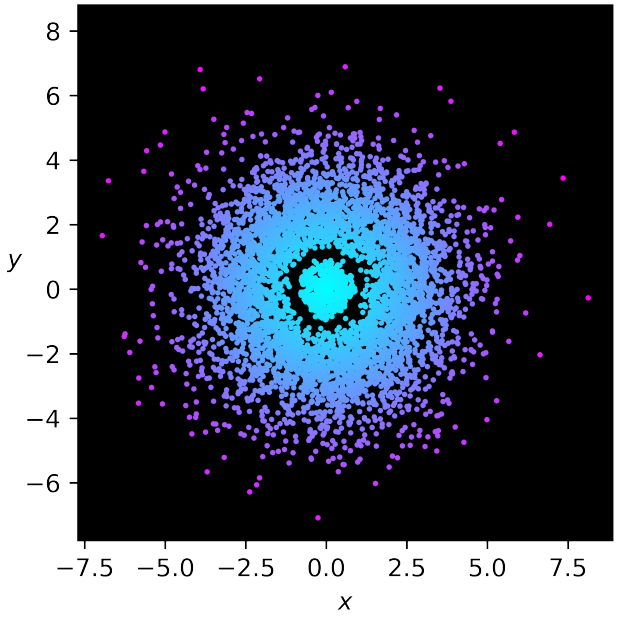

Figure 1: x-y plane of hydrogen atom orbital with 20,000 points for quantum numbers n, l, m = 2, 0, 0. Coordinates are in units of angstroms $\mathring{A}$.

Another way to represent our data is through plotting the radial distribution function which depicts the probability density as a function of the radius. Through this method, we are able to see the different nodes and how the orbital are distributed. Two sets of these plots are depicted in figures 4 and 5.

We can compare these plots to the orbital diagrams found in Fig. 2. For the case of (100), we see a continuous, smooth line going to zero as the radius goes to infinity. This agrees with the orbital diagram with presents a continuous circle with no breaks. On the other hand, Fig. 4 represents a probability distribution where the radius goes to zero at four distinct points. These points line up exactly with the orbitals found in Fig. 2.

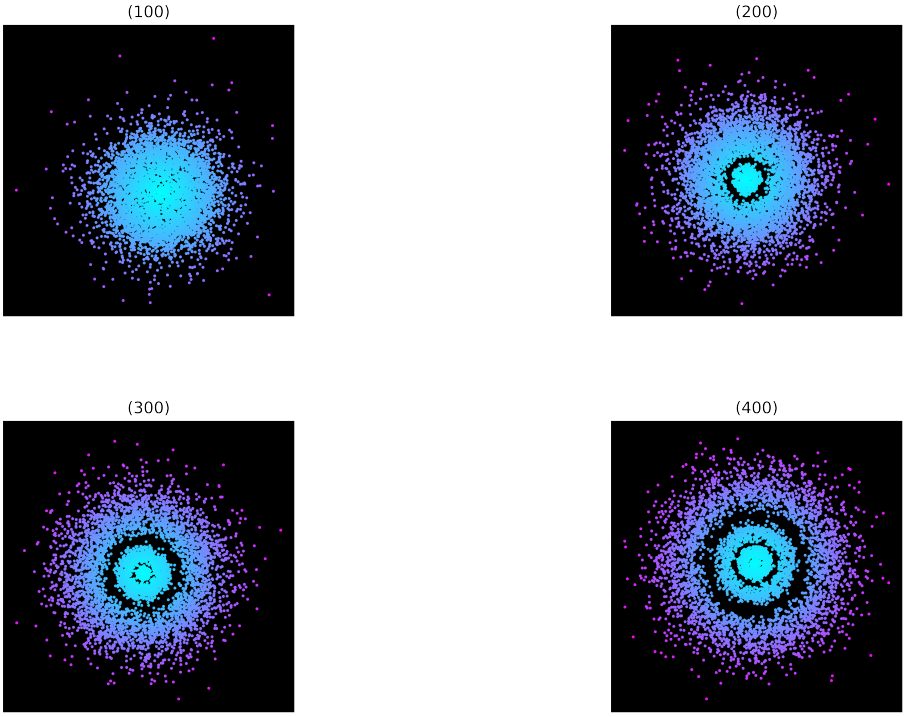

Figure 2: x-y plane of hydrogen atom orbitals with 20,000 points for quantum number $n =$ 1, 2, 3, and 4.

Figure 3: x-y plane of hydrogen atom orbitals with 20,000 points. Quantum numbers (n, l, m) denoted by top values.

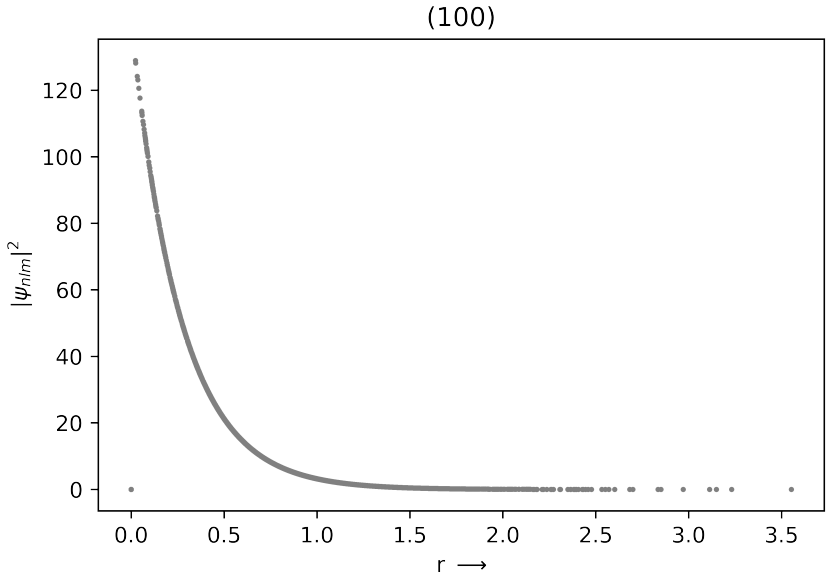

Figure 4: Radial probability distribution 10,000 points for n, l, m = 1,0,0. Radius is in units of angstroms $\mathring{A}$.

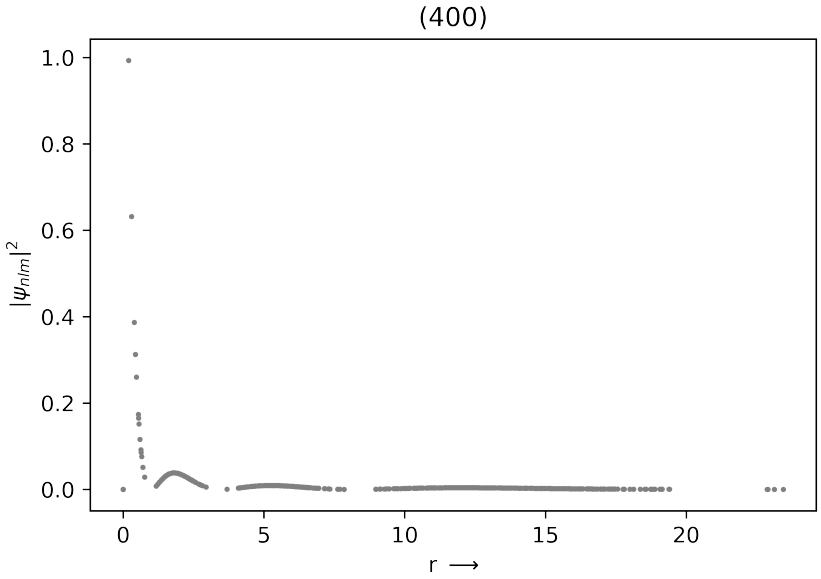

Figure 5: Radial probability distribution 10,000 points for n, l, m = 4,0,0. Radius is in units of angstroms $\mathring{A}$.

Aside from an alternative representation of our orbitals, the probability distribution plot can be used to verify the values of $|\psi_{nlm}|^2$ we calculated numerically. For a hydrogen atom with quantum numbers n, l, m = 2,0,0, we have the normalized wave function [4].

$$\psi_{2,0,0}(r,\theta,\phi) = \frac{1}{4\sqrt{2\pi}a_0^{3/2}}\left(2 - \frac{r}{a_0}\right)e^{-r/2a_0} \tag{1}$$

In Fig. 6, we plot this equation against our numerical values calculated with the same quantum numbers. Our values almost perfectly line up with the theoretical equation as expected. Our results in Fig. 6 provides us with a good indication that our numerical methods have high accuracy.

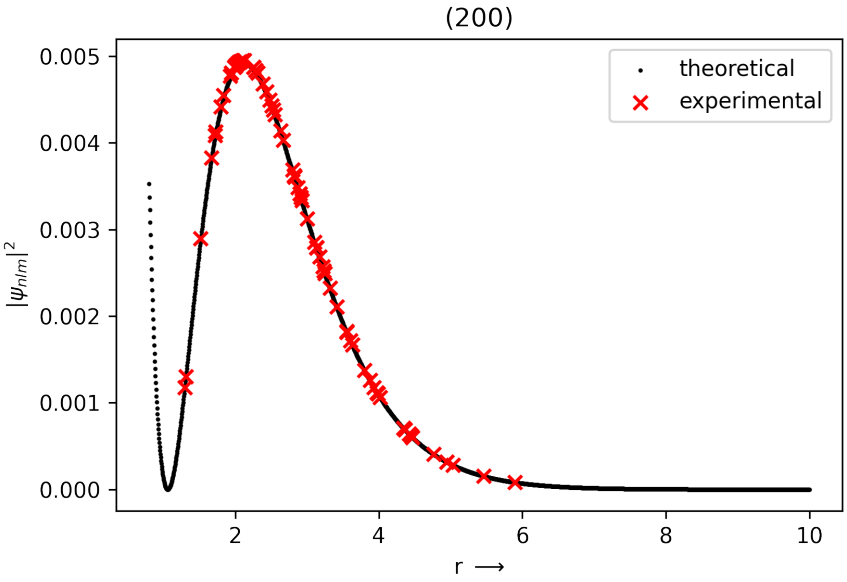

Figure 6: Comparison of numerical and theoretical values for the radial probability distribution for n, l, m = 4,0,0. Fifteen-thousand points of data were calculated, where we found the theoretical values through the wave equation. Radius is in units of angstroms $\mathring{A}$.

Parallelization

To run a small scability test, we ran our code with ten thousand points for various nlm values at different numbers of threads. Table 1. presents the raw data from one of these runs for the runtime and total time for each N number of threads. The runtime is the time it took to run the code for one combination of numbers, where the total runtime is the time the code took to run start to finish.

For one thread (essentially non-paralleled code), it takes about a half hour to run ten different wave functions. As we increase the number of threads, our runtime drops. By the time we reach ten threads, our total runtime is about a third of the initial runtime. This is

great improvement and verifies the advantages of parallel computing over regular computing.

In Fig. 7, we plot of few of our wave functions and threads vs runtime and see plots in the shape of a decaying exponential. Throughout all of our plots, the runtime drops with the increased number of threads.

nlm	$t_{N=1}$ (s)	$t_{N=2}$ (s)	$t_{N=4}$ (s)	$t_{N=5}$ (s)	$t_{N=8}$ (s)	$t_{N=10}$ (s)	$t_{N=20}$ (s)
100	438.843	352.326	231.818	247.013	237.507	214.943	174.200
200	71.301	58.774	35.829	35.362	39.176	29.643	29.994
210	405.044	245.258	179.595	149.265	171.530	125.519	151.895
300	60.950	34.345	23.434	22.566	20.033	15.810	15.581
310	324.577	165.744	137.433	111.110	77.596	80.260	82.445
320	70.781	38.665	29.517	22.996	18.222	18.903	21.593
400	42.006	24.914	16.904	16.545	10.688	10.685	12.501
410	244.196	131.393	102.050	86.087	58.621	61.182	66.333
420	55.968	29.342	23.889	19.407	9.393	5.677	6.901
430	34.568	17.175	11.168	7.286	4.407	3.039	1.830
Total	1748.234	1097.936	791.638	717.636	647.173	565.662	563.273

Table 1: Run-times for N threads of different n,l,m values with simulation of 10,000 points.

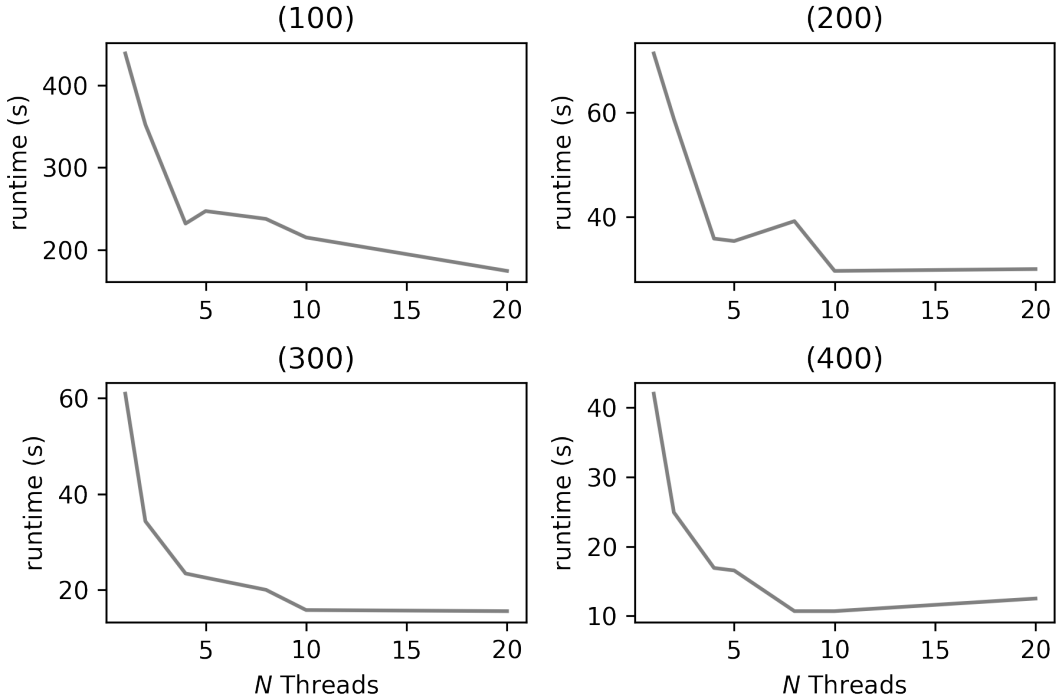

Figure 7: Plots of simulation runtime against number of threads for four different quantum number combinations: 100, 200, 300, 400, with 10,000 points.

We also considered algorithmic scalability. To analyze this, we ran the same number of threads but changed the amount of points in each run. Our results are plotted in Table 2.

We see that there is not a consistent increase when the number of points double. A second trial was run and performed about the same. Though it did not perform as consistently as expected, it seems the algorithm scales about 3/2 per double the data points.

Pts	Total Runtime (s)
100	0.64
500	2.88
1000	4.91
5000	24.64
10000	548.96
15000	848.27

Table 2: Run-times 10 variations of quantum numbers on 4 cores for different numbers of points.

In this report, we outlined and implemented a method for plotting the hydrogen atom using the Monte Carlo technique. By making the code run in parallel, we were also able to show the affects of parallel computing and perform a brief analysis on the scalability of our code.

Appendix

The source code for Fig. 7, while not in the ipynb, was written as follows:

```
# plotting code source:
# https://matplotlib.org/stable/gallery
# /subplots_axes_and_figures/subplots_demo.html

t1 = [438.842573,352.32614,231.818459,247.012522,237.5065,214.942725,174.200338]
t2 = [71.301312,58.773942,35.829318,35.361901,39.176354,29.642879,29.994143]
t3 = [60.949613,34.345205,23.433553,22.56561,20.03303,15.810138,15.580875]
t4 = [42.00643,24.914439,16.904283,16.545326,10.688244,10.685316,12.500596]

threads = [1,2,4,5,8,10,20]

fig, axs = plt.subplots(2,2, constrained_layout=True)
#fig.tight_layout()

axs[0,0].plot(threads, t1, c = 'gray')
axs[0,1].plot(threads, t2, c = 'gray')
axs[1,0].plot(threads, t3, c = 'gray')
axs[1,1].plot(threads, t4, c = 'gray')

axs[0,0].set_title('(100)')
axs[0,1].set_title('(200)')
axs[1,0].set_title('(300)')
axs[1,1].set_title('(400)')

for ax in axs.flat:
    ax.set(ylabel = 'runtime (s)')

axs[1,0].set(xlabel = '$N$ Threads')
axs[1,1].set(xlabel = '$N$ Threads')
fig.savefig('runtimeplot.png', dpi=300)
```

References

[1] Wikipedia: Associated legendre polynomials, Apr 2021.

[2] Wikipedia: Legendre polynomials, Apr 2021.

[3] V. De Aquino Manoel, V. Aguilera-Navarro, M. Goto, and H. Iwamoto. Monte carlo image representation. *American Journal of Physics - AMER J PHYS*, 69:788–792, 07 2001.

[4] P. H. Lobo, E. Arashiro, A. Silva, and C. Saraiva Pinheiro. A smooth path to plot hydrogen atom via monte carlo method. *Revista Brasileira de Ensino de Física*, 41, 01 2019.

[5] K. F. Riley, M. P. Hobson, and S. J. Bence. *Mathematical Methods for Physics and Engineering: A Comprehensive Guide*. Cambridge University Press, 3 edition, 2006.