Schrödinger's Equation

Two Electrons in a 3D Harmonic Oscillator Well

by

Thomas Bolden

2016

Contents

Introduction

Why? *The Background*

The Schrödinger equation for one confined electron is a relatively simple example of a particle in a potential well. The two-electron case however, becomes much more challenging due to the repulsive interaction between the electrons.

Why? *The Motivation*

- Schrödinger's equation is difficult to solve analytically

Why? *The Motivation*

- Schrödinger's equation is difficult to solve analytically
- Numerical methods work on more complex systems

Why? *The Motivation*

- Schrödinger's equation is difficult to solve analytically
- Numerical methods work on more complex systems
- Computational methods ...

The Process

Two methods of computing eigenvalues of a matrix were implimented and compared: Jacobi rotation and Armadillo's *eig_sym* Since the one-electron solution is simple to solve analytically, it provides a reasonable standard to check the algorithms.

Methods

The Single Electron Case

The radial Schrödinger equation for one electron is given by

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}-\frac{\ell(\ell-1)}{r^2}\right)R(r)+V(r)R(r)=ER(r) \quad , \quad V(r)=\frac{m\omega^2 r^2}{2}$$

The energy solutions can be quantized

$$E_{nl}=\hbar\omega\left(2n+l+\frac{3}{2}\right)$$

This is not yet helpful!

The Single Electron Case

If, in spherical coordinates, a dimensionless variable is introduced such that $\rho = r/\alpha$, with α having dimensions of length, the equation becomes

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho) + \left(V(\rho) + \frac{l(l+1)}{\rho^2}\frac{\hbar^2}{2m\alpha^2}\right)u(\rho) = Eu(\rho).$$

If we neglect angular momentum by setting $l = 0$, and making $\omega = \alpha\rho$, we get

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho) + \frac{k}{2}\alpha^2\rho^2 u(\rho) = Eu(\rho)$$

The Single Electron Case

If both sides of the previous equation are multiplied by $\frac{2m\alpha^2}{\hbar^2}$ while fixing $\alpha = \left(\frac{\hbar^2}{mk}\right)^{1/4}$ the equation becomes

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2 u(\rho) = \lambda u(\rho).$$

Discrete energy eigenvalues are given by

$$\lambda = \frac{2m\alpha^2}{\hbar^2}.$$

The first three eigenvalues are therefore $\lambda_0 = 3$, $\lambda_1 = 7$, $\lambda_2 = 11$.

The Single Electron Case

In order to solve this, the standard approximation of the second derivatve with step length h will be used.

$$u'' = \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2} + O(h^2),$$

If we let

$$\begin{aligned} u_i &= u(\rho) \\ u_{i\pm1} &= u(\rho \pm h) \\ \rho_i &= ih \\ V_i &= V(\rho_i) \end{aligned}$$

we can discretize the second derivative to

$$-\frac{u_{i+1} + u_{i-1} - 2u_i}{h^2} + V_i u_i = \lambda u_i$$

The Single Electron Case

The previous equation resembles a set of linear equations. The finial tridiagonal matrix which is the solution to $\mathbf{Au} = \lambda\mathbf{u}$ is then

$$\mathbf{A} = \begin{pmatrix} \frac{2}{h^2} + V_1 & -\frac{1}{h^2} & 0 & \dots & 0 & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2} + V_2 & -\frac{1}{h^2} & \dots & 0 & 0 \\ 0 & -\frac{1}{h^2} & \cdots & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \frac{2}{h^2} + V_{n-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{n-1} \end{pmatrix}$$

with $\mathbf{u} = (u_1\ u_2\ \dots\ u_{n-1})^T$

The Single Electron Case

Now that I have shown how the Schrödinger equation can be written as a set of linear equations, I can now talk about how it might be solved computationally.

Jacobi's Rotation Algorithm

This linear equation solver uses a rotation matrix to solve for the eigenvalues of **A**. Given a square matrix, Jacobi's algorithm rotates the matrix until only the only remaining non-zero elemets in the matrix exist on the diagonal. These numbers are the eigenvalues.

Jacobi's Rotation Algorithm

The first step in Jacobi's algorithm is finding the largest off-diagonal element in the matrix. This is implemented in the code that follows.

Jacobi's Method in C++ Code

```
1  double MaxOffDiagonal(mat A, int *k, int *l, int n){
2
3      double maxOD;
4      double a_ij;
5
6      for (int i = 0; i < n; i++){
7          for (int j = i+1; j < n; j++){
8              a_ij = fabs(A(i,j));
9              if (a_ij > maxOD){
10                 maxOD = a_ij;
11                 *k = i;
12                 *l = j;
13             }
14         }
15     }
16     return maxOD;
17 }
```

Jacobi's Rotation Algorithm

Once the largest off-diagonal matrix element is found, the rotation matrix **R** (the standard sine, cosine rotation matrix) can be applied to **A**.

$$\mathbf{B} = \mathbf{R}^T\mathbf{A}\mathbf{R}$$

The elements of **B** are equations involving $\sin\theta$ and $\cos\theta$.

$$\begin{aligned}
B_{ik} &= A_{ik}\cos\theta - A_{il}\sin\theta, i \neq k, i \neq l \\
B_{il} &= A_{il}\cos\theta + A_{ik}\sin\theta, i \neq k, i \neq l \\
B_{kk} &= A_{kk}\cos^2\theta - 2A_{kl}\cos\theta\sin\theta + A_{ll}\sin^2\theta \\
B_{ll} &= A_{ll}\cos^2\theta + 2A_{kl}\cos\theta sin\theta + A_{kk}\sin^2\theta \\
B_{kl} &= (A_{kk} - A_{ll})\cos\theta\sin\theta + A_{kl}(\cos^2\theta - \sin^2\theta)
\end{aligned}$$

The following slides show this implemented in C++.

Jacobi's Rotation in C++

```
1  void Rotate(mat &A, mat &B, int k, int l, int n){
2
3      double t, tau, sine, cosine;
4
5      if ( A(k,l) != 0.0){
6          tau = ( A(l,l) − A(k,k) )/( 2*A(k,l) );
7          if (tau >= 0.0) {
8              t = 1.0/(fabs(tau) + sqrt(1.0 + tau*tau));
9          }
10         else {
11             t = −1.0/(fabs(tau) + sqrt(1.0 + tau*tau));
12         }
13         cosine = 1.0/ sqrt (1.0 + t*t);
14         sine = t*cosine;
15     }
16     else {sine = 1.0; cosine = 0.0;}
```

Jacobi's Rotation in C++

```
 1      double A_ik;
 2      double A_il;
 3      double B_ik;
 4      double B_il;
 5      double A_kk = A(k,k);
 6      double A_ll = A(l,l);
 7
 8      A(k,k) = A_kk*cosine*cosine - 2*A(k,l)*cosine*sine
 9               + A_ll*sine*sine;
10      A(l,l) = A_ll*cosine*cosine + 2*A(k,l)*cosine*sine
11               + A_kk*sine*sine;
12      A(k,l) = 0.0;
13      A(l,k) = 0.0;
```

Jacobi's Rotation in C++

```
for (int i = 0; i < n; i++){
        if (i != k && i != l){

            A_ik = A(i,k);
            A_il = A(i,l);
            A(i,k) = A_ik*cosine - A_il*sine;
            A(k,i) = A(i,k);
            A(i,l) = A_il*cosine + A_ik*sine;
            A(l,i) = A(i,l);
        }
        B_ik = B(i,k);
        B_il = B(i,l);
        B(i,k) = B_ik*cosine - B_il*sine;
        B(i,l) = B_il*cosine + B_ik*sine;

    } return; }
```

Results

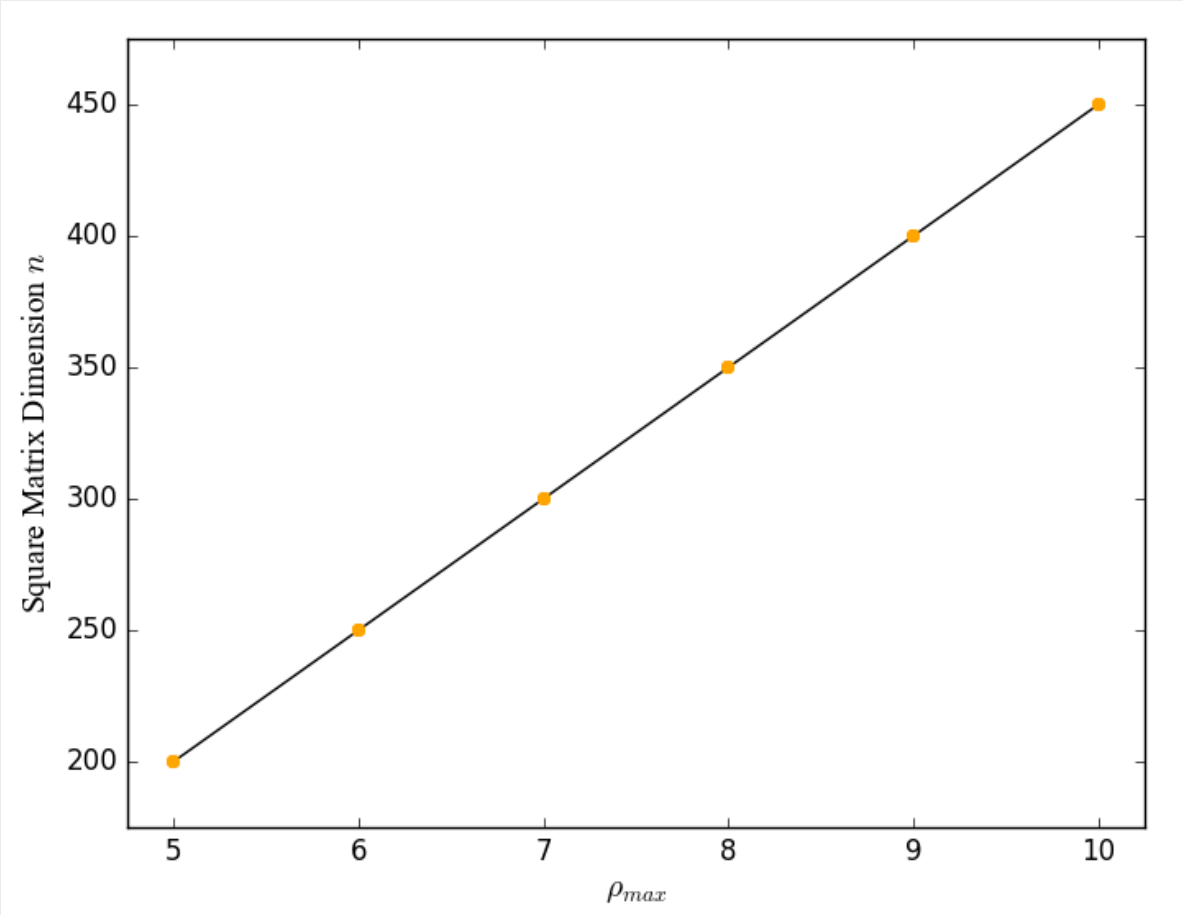
Evaluating ρ Dependency

When the dimensionality of the matrix is changed, the dimensionless variable ρ needed to accurately determine the lowest three eigenvectors also changes.

Minimum ρ_{max}	Matrix Dimensions n
5	200
6	250
7	300
8	350
9	400
10	450

Evaluating ρ Dependency

With a step size of 50 for the matrix dimensinality, the relationship between ρ and n is perfectly linear.



Jacobi vs. Armadillo

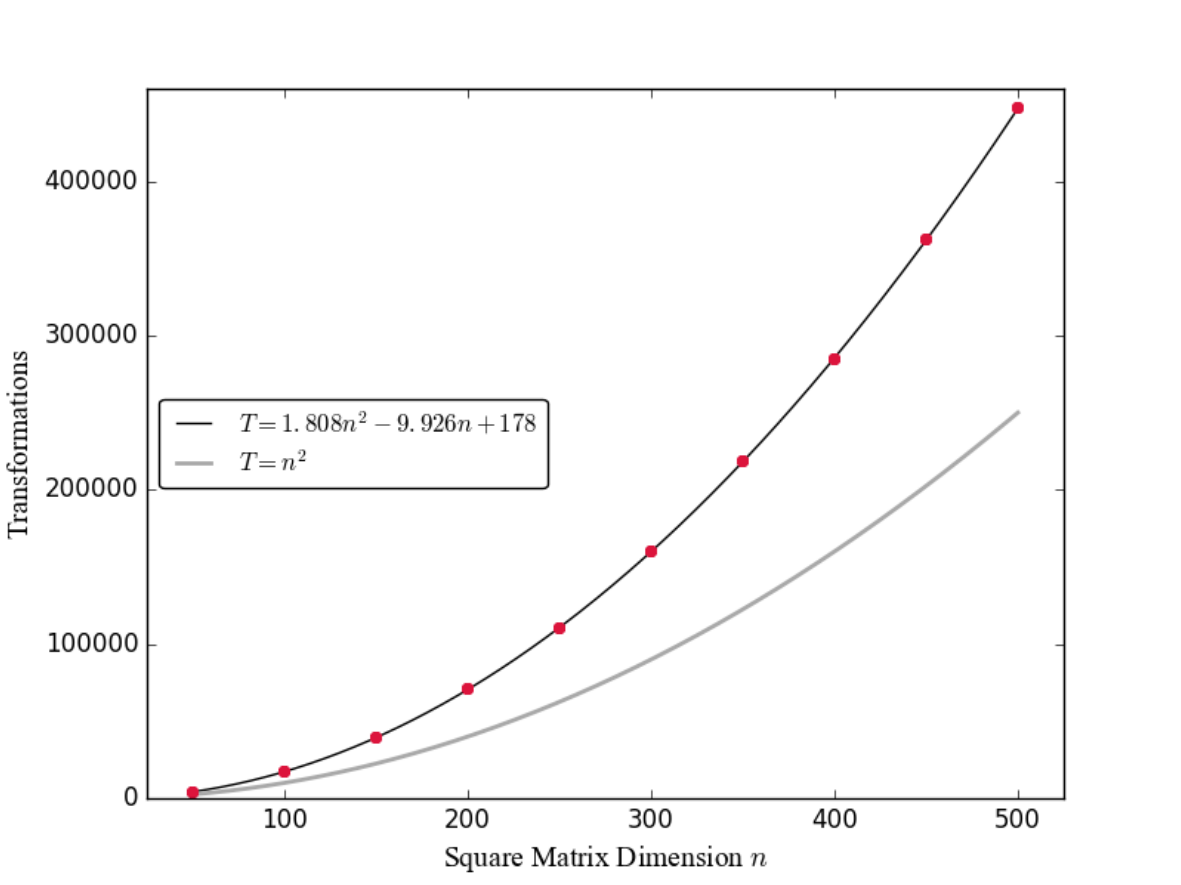
The time needed to compute the lowest three eigenvalues for Jacobi's rotation algorithm and Armadillo's *eig_sym* function were compared. The Armadillo library function was many orders of magnitude faster than the brute force Jacobi method.

Jacobi vs. Armadillo

Dimensionality n	Eigenvalues	$t_{\text{Armadillo}}$ (s)	t_{Jacobi} (s)
50	$(2.99687, 6.98434, 10.9619)$	0.000617	0.08572
100	$(2.99922, 6.99609, 10.9907)$	0.003777	1.15983
150	$(2.99965, 6.99827, 10.9960)$	0.007561	5.89020
200	$(2.99980, 6.99903, 10.9978)$	0.012636	18.5741
250	$(2.99988, 6.99938, 10.9987)$	0.019315	49.3644
300	$(2.99991, 6.99957, 10.9991)$	0.032661	110.025
350	$(2.99994, 6.99968, 10.9994)$	0.061280	248.396
400	$(2.99995, 6.99976, 10.9996)$	0.086203	430.594
450	$(2.99996, 6.99981, 10.9997)$	0.094821	901.509
500	$(2.99997, 6.99985, 10.9998)$	0.123586	1394.52

Matrix Transformations

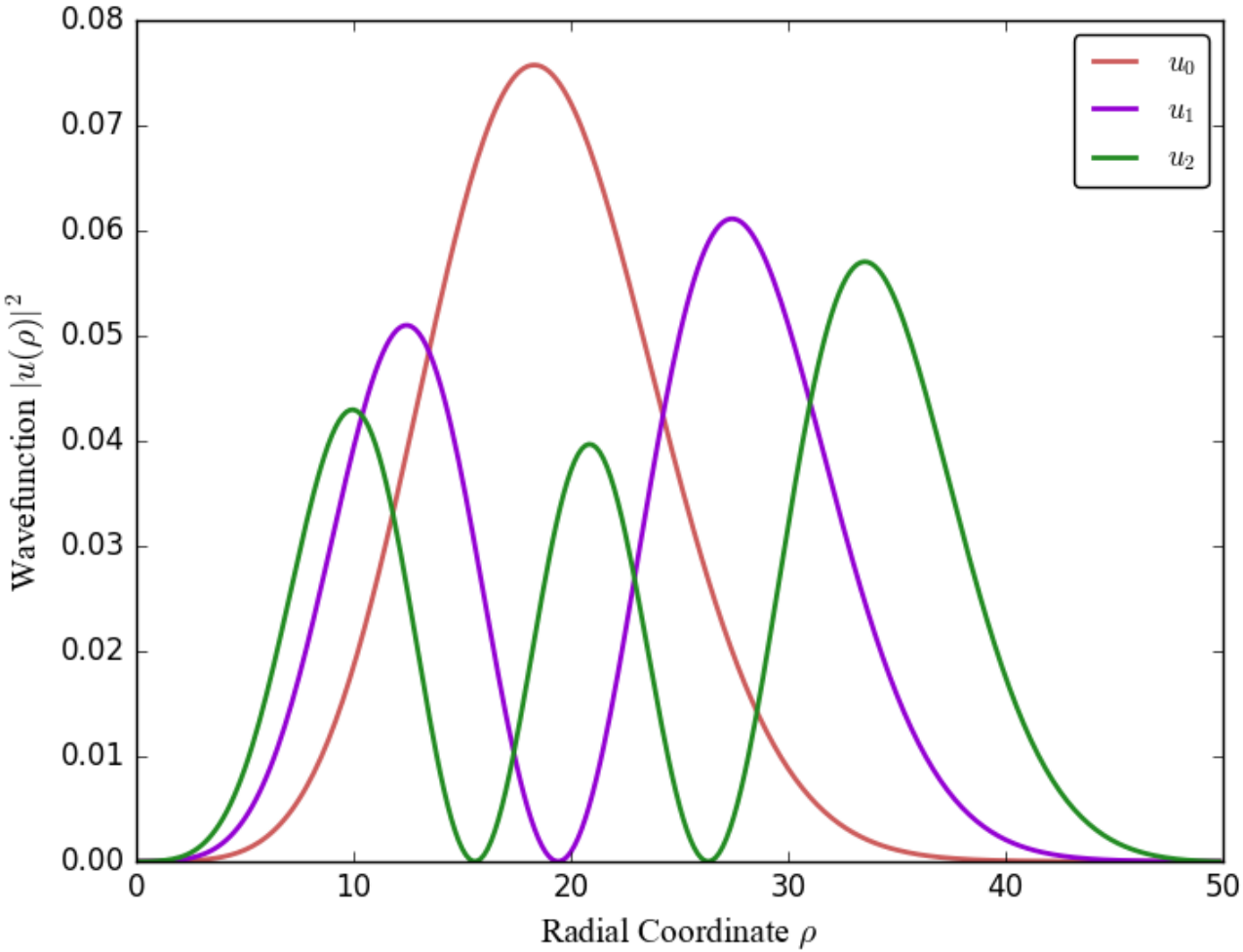
The transformations on the $n \times n$ matrix $\mathbf{A}$ grow as approximately n^2.



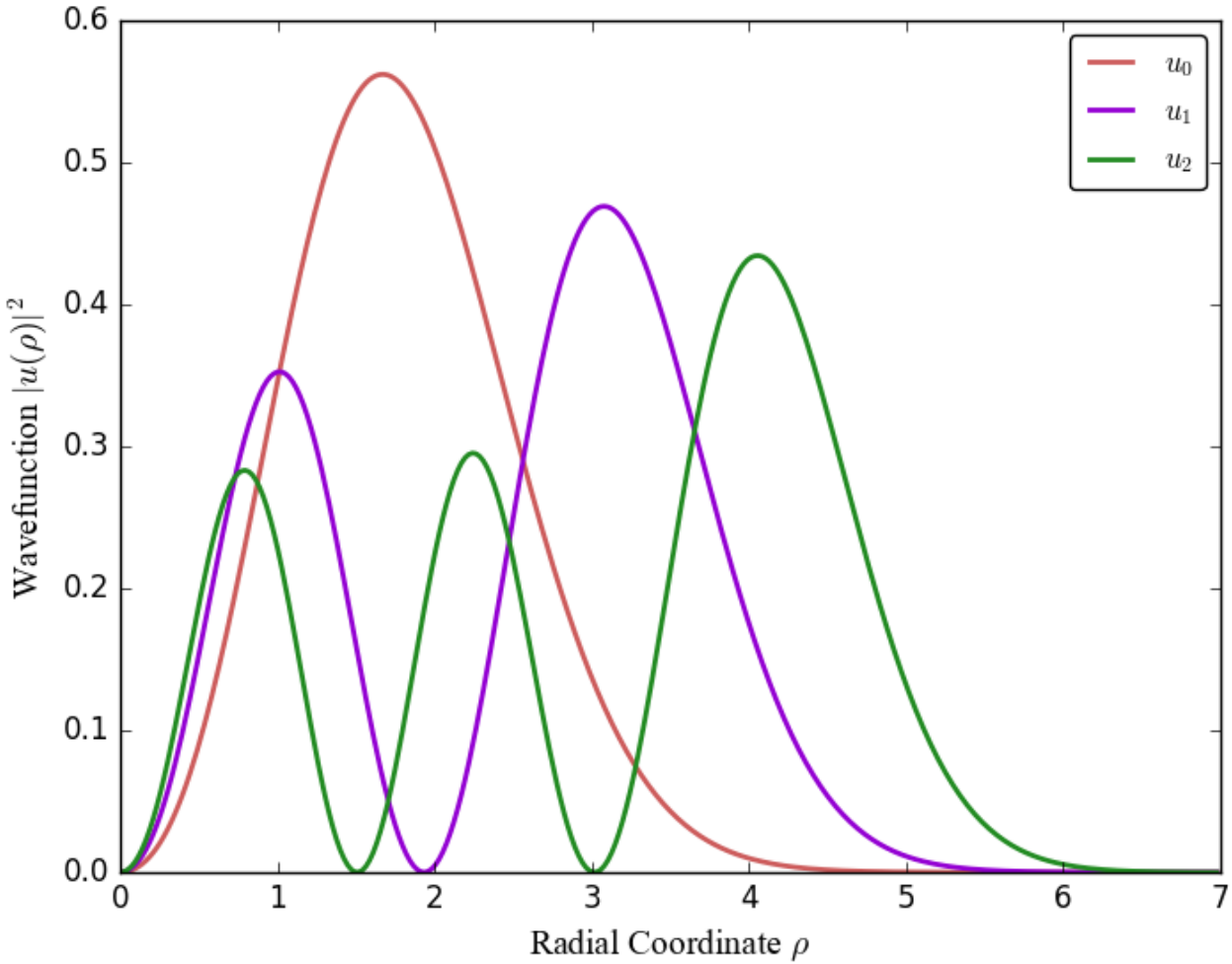
2 Electron Wavefunctions

By varying the strength of the harmonic oscillator potential, the effect of coulombic interaction was observed.
In the figures that follow, the strength of the oscillator potential is increased with each figure.
The first figure has $\omega_r = 0.01$
The second figure has $\omega_r = 0.50$
The third figure has $\omega_r = 1.00$
The fourth figure has $\omega_r = 5.00$

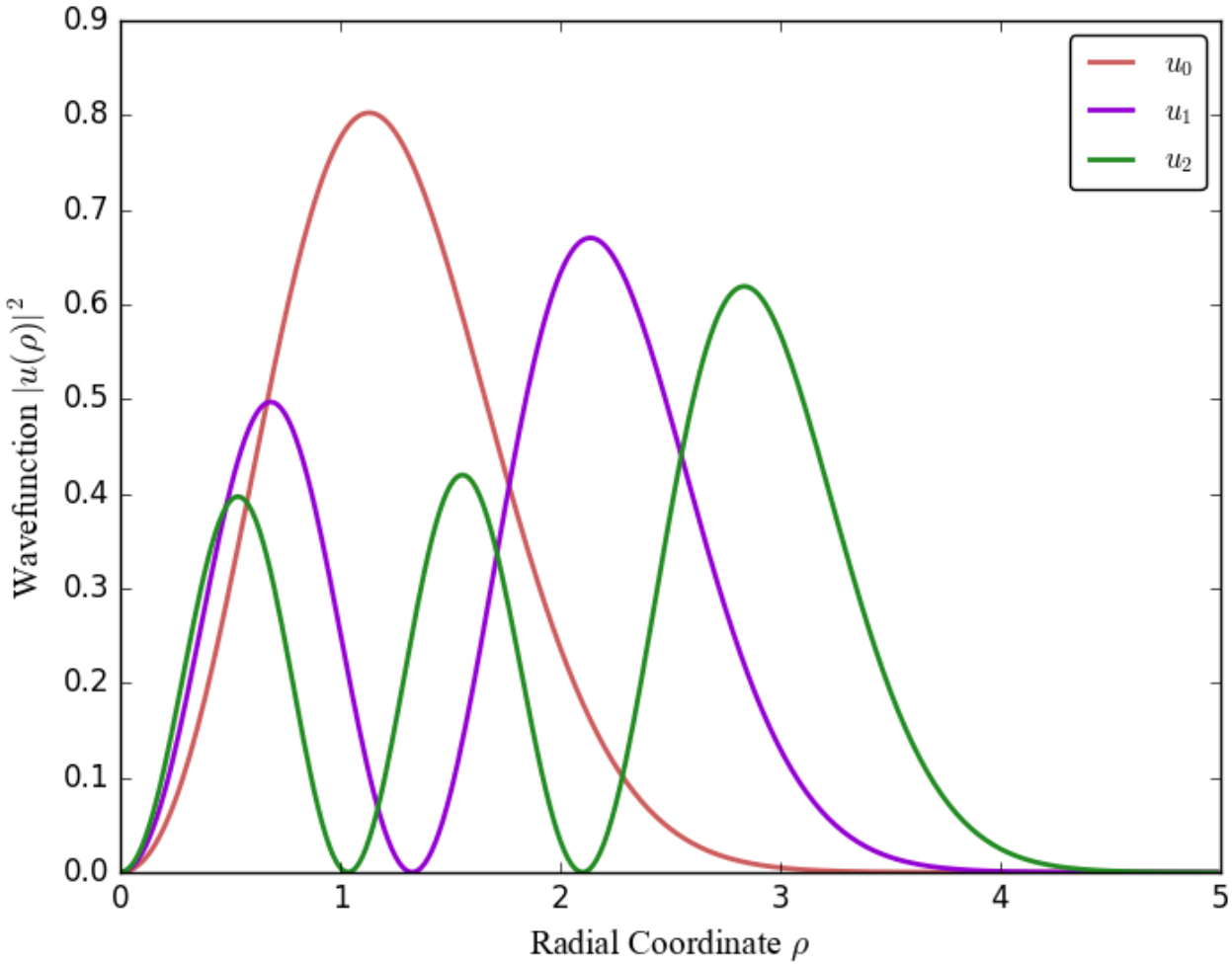
2 Electron Wavefunctions



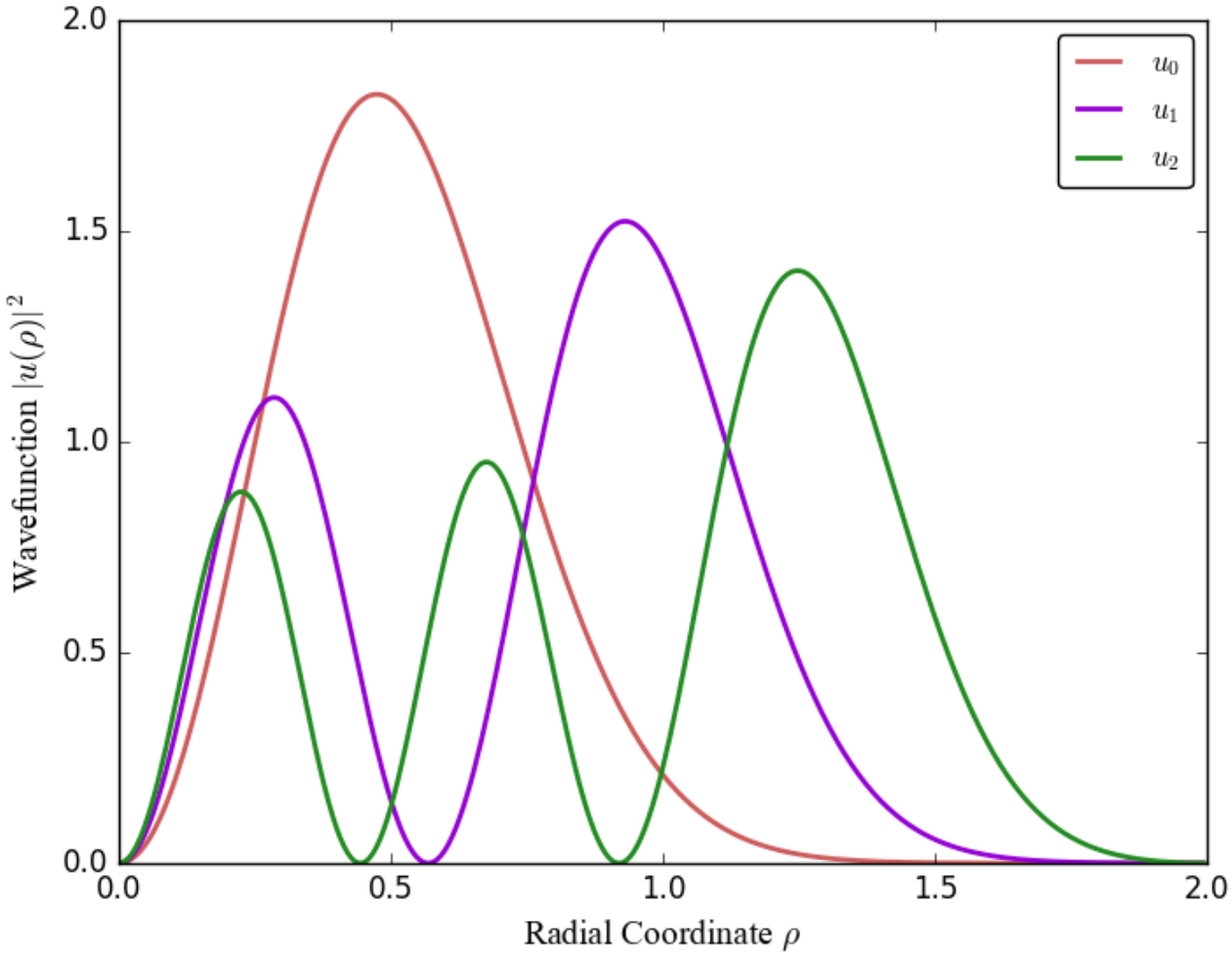
2 Electron Wavefunctions



2 Electron Wavefunctions



2 Electron Wavefunctions



Conclusions

Summary

The first task was to impliment algorithms that solve a one electron system in a three dimensional harmonic oscilator potential well. To do this, a square matrix was diagonalized using both a brute-force Jacobi rotation algorithm and Armadillo.

Closing Thoughts

Jacobi's method for diagonalizing matrices was found to be orders of magnitude slower than the Armadillo library's *eig_sym* function. This means that, while useful and simple to impliment, Jacobi's method is only practical for small matrices.
If however, Armadillo is unavailable, Jacobi's method can be improved upon! (parallelization)

References

Citations

- M. Hjorth-Jensen, *Computational Physics*, University of Oslo (2015).
- M. Taut, Phys. Rev. A 48, 3561 - 3566 (1993).
- C. Sanderson, *Armadillo*. **2010**. arma.sourceforge.net