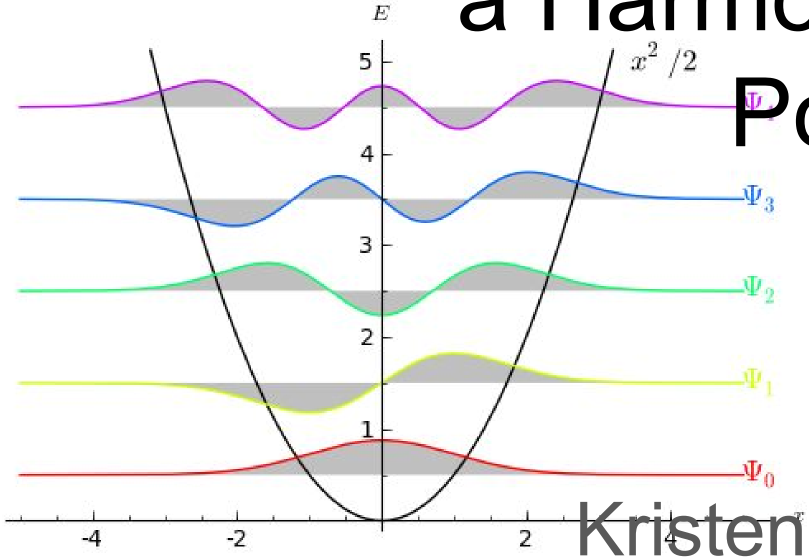
Solving Schrödinger's Equation for Two Particles in a Harmonic Oscillator Potential



https://physics.stackexchange.com/questions/170522/please-explain-the-following-graphs-that-describe-a-quantum-mechanical-harmonic

https://en.wikipedia.org/wiki/Erwin_Schr%C3%B6dinger

Kristen Parzuchowski
Michigan State University
May 3rd, 2017

Motivation

Two particles in a harmonic oscillator potential are not uncommon systems….

$\longrightarrow$ Qubits for quantum computing (trapped ions, electrons, quantum dots)

Theory

Schrödingers equation for two particle in a harmonic oscillator potential using COM coordinate and relative coordinate

$$\left(\underbrace{-\frac{\hbar^2}{m}\frac{d^2}{dr^2} - \frac{\hbar^2}{4m}\frac{d^2}{dR^2}}_{\text{Kinetic Energy}} \underbrace{+ \frac{1}{4}kr^2 + kR^2}_{\text{Potential Energy}}\right)\underbrace{u(r,R)}_{\text{Ansatz for separable wavefunction}} = \underbrace{E^{(2)}}_{E^{(2)} = E_r + E_R}u(r,R)$$

Kinetic Energy

Potential Energy

+

$$V(r_1, r_2) = \frac{\beta e^2}{|\mathbf{r}_1 - \mathbf{r}_2|} = \frac{\beta e^2}{r}$$

Ansatz for separable wavefunction

u(r,R) = Ψ(r)Φ(R)

$$E^{(2)} = E_r + E_R$$

Coordinates

$$\mathbf{R} = 1/2(\mathbf{r}_1 + \mathbf{r}_2)$$

$$\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$$

Jacobi Rotation Algorithm

$\longrightarrow$Used to find the eigenvalues and eigenvectors of hermitian matrices

Similarity Transformation using orthogonal matrix S

$$\mathbf{S}^{\mathbf{T}} = \mathbf{S}^{-1}$$

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

Goal: transform to a similar diagonal matrix

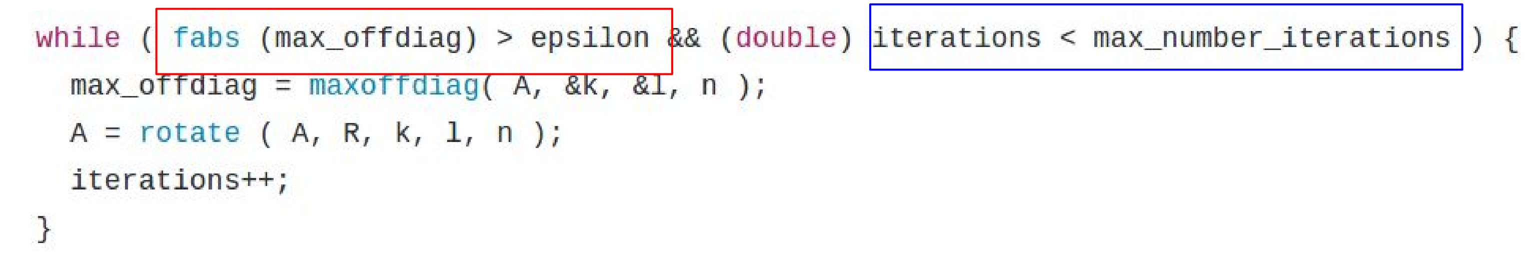
$$\begin{pmatrix} b_{kk} & 0 \\ 0 & b_{ll} \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix}$$

Eigenvalues on diagonal of B & **eigenvectors** are the similarity transformed basis vectors from initial case

Implementation of the Jacobi Rotation Algorithm

It is not necessary or computationally friendly☺ to run the algorithm until all off-diagonal elements equal exactly zero
⟶Set a nonzero tolerance, ε, which for our purposes is approximately zero

If a matrix is sufficiently complicated or unsolvable, it may never get to an answer
⟶Set a max number of iterations



```
while ( fabs (max_offdiag) > epsilon && (double) iterations < max_number_iterations ) {
  max_offdiag = maxoffdiag( A, &k, &l, n );
  A = rotate ( A, R, k, l, n );
  iterations++;
}
```

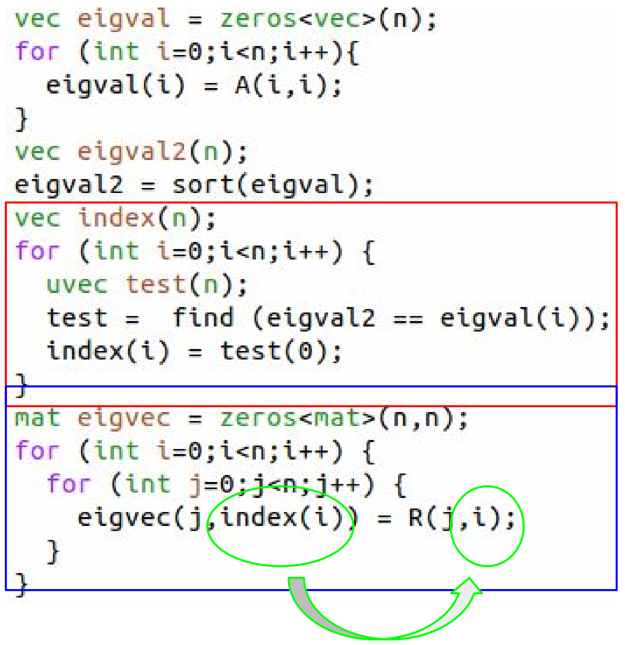
Implementation of the Jacobi Rotation Algorithm

The sort() function (built-in Armadillo function) used to sort the eigenvalues does not sort the eigenvectors as well

Solution:

- Match the sorted eigenvalue matrix to the unsorted eigenvalue matrix to find the original indexed location

-Find the eigenvector at that indexed location in the eigenvector matrix and correctly order a new matrix



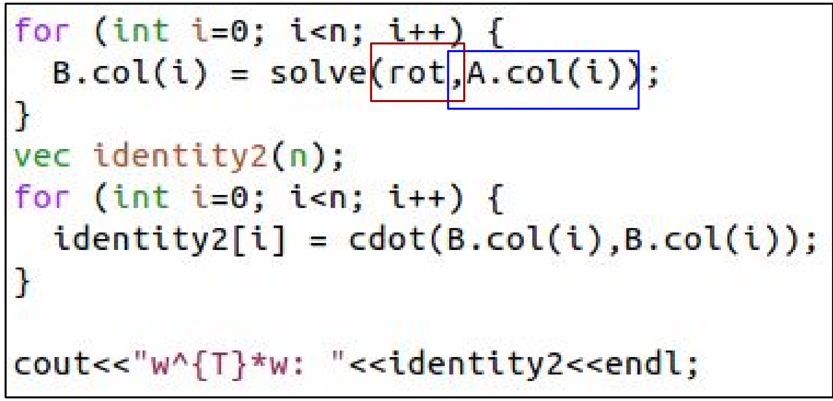
```
vec eigval = zeros<vec>(n);
for (int i=0;i<n;i++){
  eigval(i) = A(i,i);
}
vec eigval2(n);
eigval2 = sort(eigval);
vec index(n);
for (int i=0;i<n;i++) {
  uvec test(n);
  test =  find (eigval2 == eigval(i));
  index(i) = test(0);
}
mat eigvec = zeros<mat>(n,n);
for (int i=0;i<n;i++) {
  for (int j=0;j<n;j++) {
    eigvec(j,index(i)) = R(j,i);
  }
}
```

Unit Tests

1) Orthogonality Preservation under Similarity Transformation

Consider an orthogonal vector **v**, such as our eigenvectors, and a hermitian matrix **U**, such as our similarity transformation matrix :

$$(\mathbf{U}\mathbf{v}_j)^T(\mathbf{U}\mathbf{v}_i) = \mathbf{v}_j^T(\mathbf{U}^T\mathbf{U})\mathbf{v}_i = \mathbf{v}_j^T\mathbf{v}_i = \delta_{ij}$$



```
for (int i=0; i<n; i++) {
  B.col(i) = solve(rot,A.col(i));
}
vec identity2(n);
for (int i=0; i<n; i++) {
  identity2[i] = cdot(B.col(i),B.col(i));
}

cout<<"w^{T}*w: "<<identity2<<endl;
```

Unit Tests

1) Orthogonality Preservation under Similarity Transformation

Consider an orthogonal vector **v**, such as our eigenvectors, and a hermitian matrix **U**, such as our similarity transformation matrix :

$$(\mathbf{U}\mathbf{v}_j)^T(\mathbf{U}\mathbf{v}_i) = \mathbf{v}_j^T(\mathbf{U}^T\mathbf{U})\,\mathbf{v}_i = \mathbf{v}_j^T\mathbf{v}_i = \delta_{ij}$$

2) Checking the result for a simple matrix

Hard code a matrix with known eigenvalues that requires a number of Jacobi rotations to reach these values, check the accuracy throughout changes to code

Comparison to Theory

M. Taut, Phys. Rev. A. 48, 5 (Nov 1993).

Interacting case for ω = 0.25

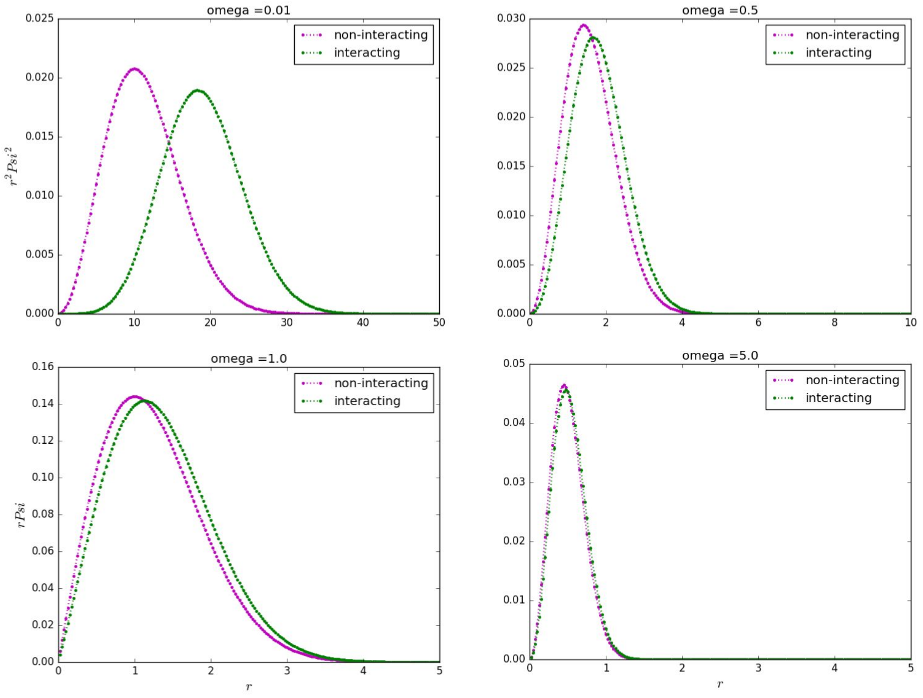
First Eigenvalue:
Taut: 1.250
Jacobi Algorithm: 1.24879

Interacting case

Frequency	Algorithm	Eigenvalues		
0.01	Jacobi	0.105773	0.141504	0.178008
	Arma	0.105773	0.141504	0.178008
0.5	Jacobi	2.22507	4.11139	6.017
	Arma	2.22507	4.1139	6.017
1	Jacobi	4.0372	7.81445	11.5845
	Arma	4.0372	7.81445	11.5845
5	Jacobi	16.9105	34.4305	49.9979
	Arma	16.9105	34.4305	49.9979

Ground State Wavefunction

Jacobi Rotation



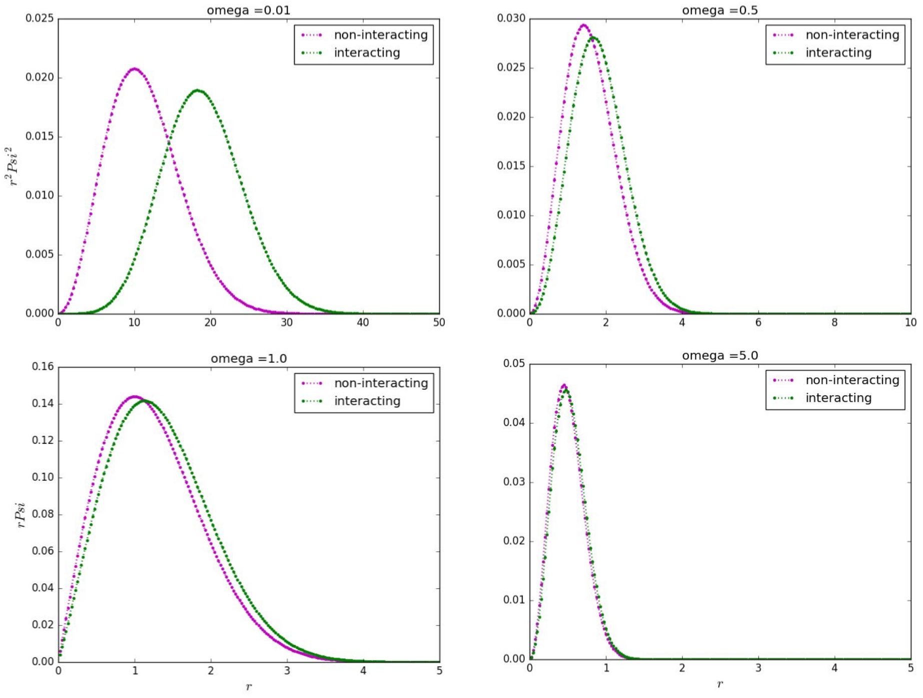
→Decreasing ω increases the prominence of the repulsive Coulomb potential

The oscillator potential disappears, so the particles will likely spread further apart - WHY so much repulsion??

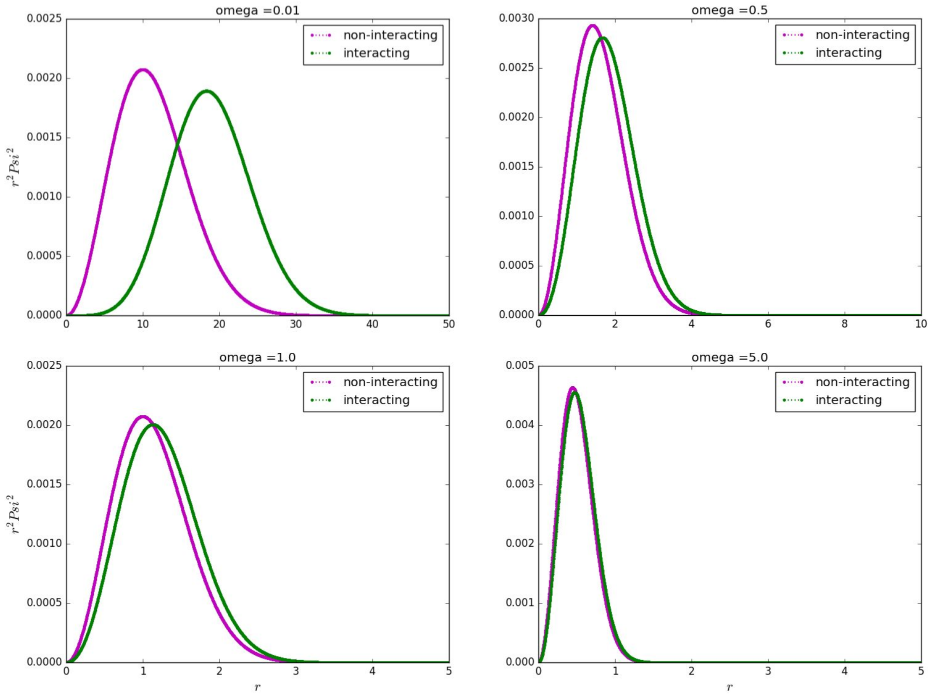
→The Coulomb potential has an infinite range

Ground State Wavefunction

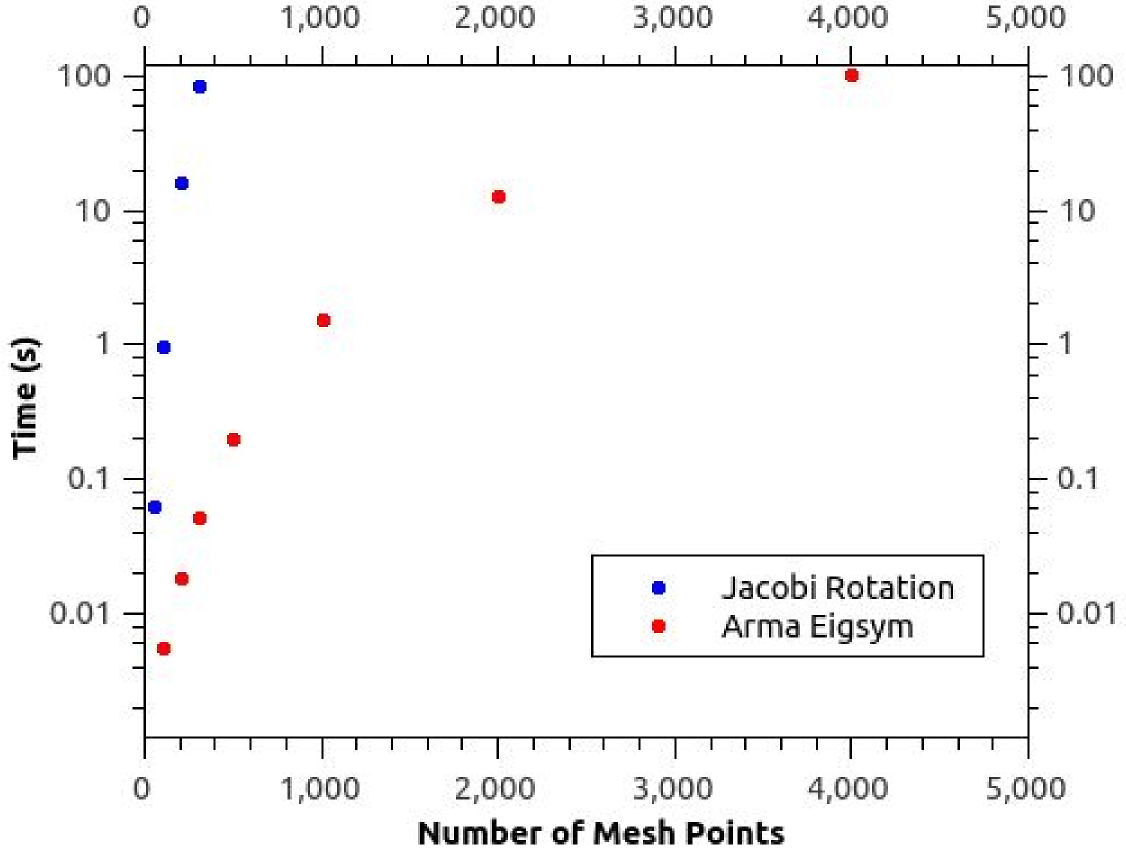
Jacobi Rotation



Armadillo Eigsym



Algorithm Efficiency

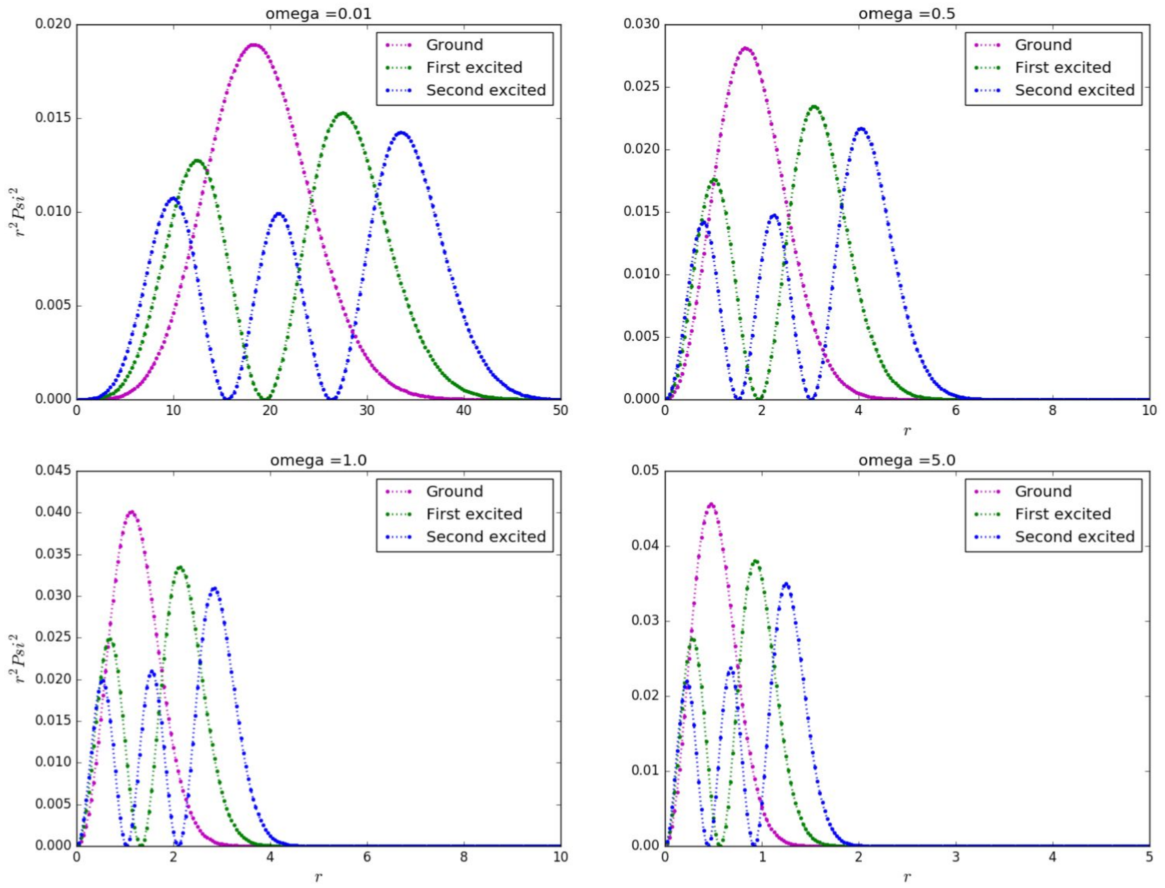


Arma Max: 4000 pts

Jacobi Max: 300 pts

Time for Arma Max ~
Time for Jacobi Max

Excited States



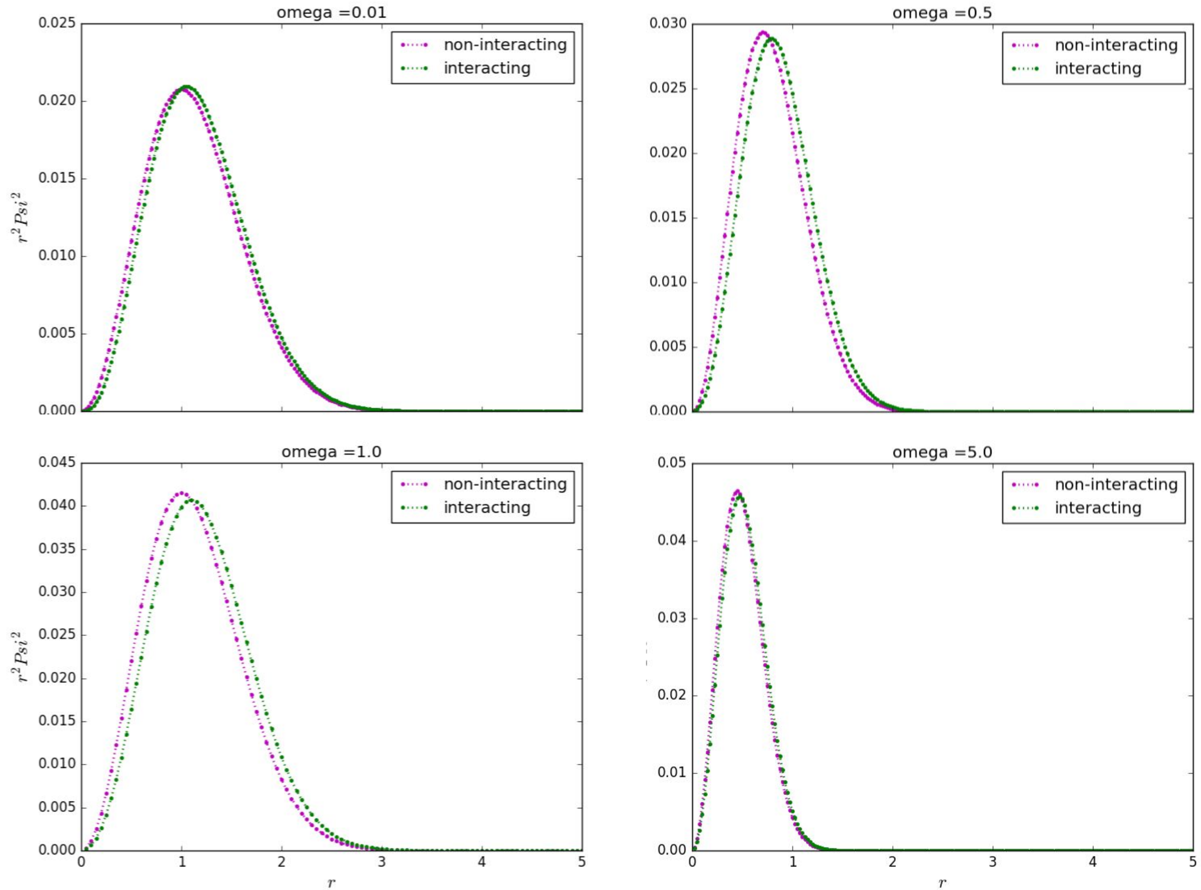
Algorithm: Jacobi Rotation
Mesh points: 200

-Highest amplitude peak for a state is furthest from origin, result of repulsion

-Scale of y-axis increases with increasing frequency $\longrightarrow$likelihood concentrated near the origin

Yukawa factor addition to Coloumb Potential

$$V = \frac{1}{2}m\omega^2 r^2 + \frac{e^{-r}}{r}$$



Conclusions

- Jacobi Rotation algorithm is producing accurate eigenvalues/eigenvectors
- Jacobi algorithm is not the most efficient technique
- The Coulomb potential has an infinite range which can be suppressed with a Yukawa factor

Thanks for listening!

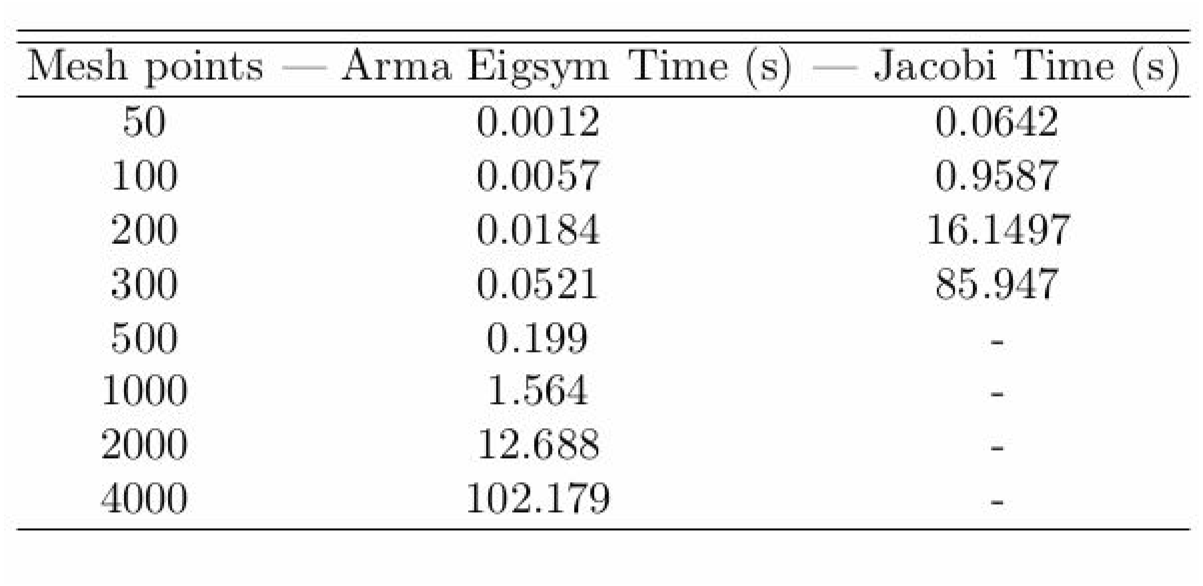
Thank you to Prof. Morten Hjorth-Jensen for teaching a great class!

Thanks to my partners Spencer Ammerman and Colin Gordon!

Non-interacting case

Frequency	Algorithm	Eigenvalues		
0.01	Jacobi	0.029998	0.0699903	0.109978
	Arma	0.029998	0.0699903	0.109978
0.5	Jacobi	1.4951	3.47541	5.43976
	Arma	1.4951	3.47541	5.43976
1	Jacobi	2.98033	6.9009	10.7562
	Arma	2.98033	6.9009	10.7562
5	Jacobi	14.4926	32.3417	48.0986
	Arma	14.4926	32.3417	48.0986

Algorithm Efficiency



Mesh points	Arma Eigsym Time (s)	Jacobi Time (s)
50	0.0012	0.0642
100	0.0057	0.9587
200	0.0184	16.1497
300	0.0521	85.947
500	0.199	-
1000	1.564	-
2000	12.688	-
4000	102.179	-