Project 2

Kristen Parzuchowski, Spencer Ammerman and Colin Gordon[1]

[1] *Michigan State University, East Lansing, MI*

We present our Jacobi eigenvalue solver for two particles in a 3-dimensional harmonic oscillator. The precision of this algorithm is determined through comparison of eigenvalues found using alternative methods. We analyze the speed of this algorithm and the effect of the number of mesh points used.

INTRODUCTION

We present our Jacobi rotation eigenvalue\eigenvector solver as well as the armadillo built-in eigenvalue\eigenvector solver. These algorithms are performed to solve the Schrodinger equation for two particles in a 3-dimensional harmonic oscillator. Initially we analyze the case of non-interacting particles and then we move to the interacting case. We compare the speed of the algorithms as well as the resulting eigenvalues\eigenvectors.

THEORY

One electron in a 3-dimensional Harmonic Oscillator Potential Well

We study the Schrödinger equation for one electron. We assume spherical symmetry and thus the radial equation contains all of the interesting information. The radial equation reads:

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}-\frac{l(l+1)}{r^2}\right)R(r)+V(r)R(r)=ER(r). \tag{1}$$

We consider only s-waves, $l = 0$. For a 3-dimensional harmonic oscillator:

$$V=\frac{1}{2}kr^2 \tag{2}$$

where $k = m\omega^2$. The radial equation can be simplified through the substitution of the wavefunction $u(r) = rR(r)$, the constants $\alpha = (\frac{\hbar^2}{mk})^{1/4}$ and $\lambda = \frac{2m\alpha^2}{\hbar^2}E$ and the parameter $\rho = \frac{r}{\alpha}$. The resulting equation reads:

$$-\frac{d^2}{d\rho^2}u(\rho)+\rho^2u(\rho)=\lambda u(\rho) \tag{3}$$

Two Electrons in a 3-dimensional Harmonic Oscillator Potential Well

The radial Schrödinger equation for s-waves of two electrons in a 3-dimensional harmonic oscillator potential is written as:

$$\left(-\frac{\hbar^2}{2m}\frac{d^2}{dr_1^2}+\frac{1}{2}kr_1^2-\frac{\hbar^2}{2m}\frac{d^2}{dr_2^2}+\frac{1}{2}kr_2^2\right)u(r_1,r_2)=Eu(r_1,r_2). \tag{4}$$

We introduce the relative coordinate $\mathbf{r} = \mathbf{r_1} - \mathbf{r_2}$ and the center-of-mass coordinate $\mathbf{R} = \frac{1}{2}(\mathbf{r_1} + \mathbf{r_2})$. Using these coordinates, the radial equation reads:

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2}+\frac{1}{4}kr^2-\frac{\hbar^2}{4m}\frac{d^2}{dR^2}+kR^2\right)u(r,R)=Eu(r,R). \tag{5}$$

We use the ansatz for separable equations, $u(r, R) = \Psi(r)\Phi(R)$. With this, energy can be written $E = E_r + E_R$. We will ignore center of mass energy. The coloumb interaction of two electrons is given by the potential:

$$V_c=\frac{\beta e^2}{r} \tag{6}$$

where $\beta e^2 = 1.44$eVnm. This potential is added to the radial equation for the relative coordinate:

$$\left(-\frac{\hbar^2}{m}\frac{d^2}{dr^2}+\frac{1}{4}kr^2+\frac{\beta e^2}{r}\right)\Psi(r)=E_r\Psi(r). \tag{7}$$

We define the new frequency:

$$\omega_r^2=\frac{1}{4}\frac{mk}{\hbar^2}\alpha^4 \tag{8}$$

where $\alpha = \frac{\hbar^2}{m\beta e^2}$. We use the previously defined variable ρ and constant λ to rewrite Schrödinger's equation as:

$$\left(-\frac{d^2}{d\rho^2}+\omega_r^2\rho^2+\frac{1}{\rho}\right)\Psi(r)=\lambda_r\Psi(r). \tag{9}$$

The repulsive term can be removed to consider the relative energy of non-interacting electrons

Unitary Transformations

Consider a basis vector $\mathbf{v}_i$:

$$\mathbf{v}_i=\begin{bmatrix}v_{i1}\\ \cdots\\ \cdots\\ v_{in}\end{bmatrix}$$

We take this basis to be orthogonal, $\mathbf{v}_j^T \mathbf{v}_i = \delta_{ij}$ Upon transformation by a unitary matrix, $\mathbf{U}$ ($\mathbf{U} = \mathbf{U}^T = \mathbf{U}^{-1}$), the dot product of the basis remains the same and the orthogonality remains:

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

ALGORITHMS AND METHODS

Composing a matrix eigenvalue problem

For discretized calculations, we can approximate the second derivative according to:

$$u'' = \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2} + O(h^2) \tag{10}$$

where h is the step size and we can ignore terms quadratic in h and higher due to their small contribution. We define minimum and maximum values for ρ, $\rho_{min} = 0$ and ρ_{max}, where ρ_{max} would ideally be set to infinity, but is instead a large finite value convenient for calculations.

Given a number, N, of mesh points, the step size for discretized calculations is given by:

$$h = \frac{\rho_N - \rho_0}{N} \tag{11}$$

The value of ρ at point i is then given by:

$$\rho_i = \rho_0 + ih \qquad i = 1, 2, ..., N \tag{12}$$

With this, the discretized Schrödinger equation can be written as:

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + V_i u_i = \lambda u_i \tag{13}$$

where in the case of the harmonic oscillator, $V_i = \rho_i^2$. This equation in matrix form gives diagonal elements:

$$d_i = \frac{2}{h^2} + V_i \tag{14}$$

and first non-diagonal elements:

$$e_i = -\frac{1}{h^2} \tag{15}$$

We can set up the matrix eigenvalue problem:

$$\begin{bmatrix} d_0 & e_0 & 0 & 0 & \dots & 0 & 0 \\ e_1 & d_1 & e_1 & 0 & \dots & 0 & 0 \\ 0 & e_2 & d_2 & e_2 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots e_{N-1} & d_{N-1} & e_{N-1} \\ 0 & \dots & \dots & \dots & \dots & e_N & d_N \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ \dots \\ \dots \\ u_N \end{bmatrix} = \lambda \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ \dots \\ \dots \\ u_N \end{bmatrix}$$

Jacobi's Rotation Algorithm

For Jacobi's rotation algorithm, it is convenient to define the quantities $tan\theta = s/c$, where $s = sin\theta$ and $c = cos\theta$ and

$$cot2\theta = \tau = \frac{a_{ll} - a_{kk}}{2a_{kl}} \tag{16}$$

We use an orthogonal (NxN) transformation matrix, $\mathbf{S}$. The nonzero elements of the matrix are given by:

$$s_{kk} = s_{ll} = c, s_{kl} = -s_{lk} = -s, s_{ii} = -s_{ii} = 1, i \neq k, l \tag{17}$$

A similarity transformation using this matrix, $\mathbf{B} = \mathbf{S^T A S}$, results in:

$b_{ii} = a_{ii}, i \neq k, l$
$b_{ik} = a_{ik}c - a_{il}s, i \neq k, l$
$b_{il} = a_{il}c + a_{ik}s, i \neq k, l$
$b_{kk} = a_{kk}c^2 - 2a_{kl}cs + a_{ll}s^2$
$b_{ll} = a_{ll}c^2 + 2a_{kl}cs + a_{kk}s^2$
$b_{kl} = (a_{kk} - a_{ll})cs + a_{kl}(c^2 - s^2)$

We choose θ that sets the highest non-diagonal matrix elements to zero. This leads to the quadratic equation:

$$t^2 + 2\tau t - 1 = 0 \tag{18}$$

which results in:

$$t = -\tau \pm \sqrt{1 + \tau^2} \tag{19}$$

This equation is used to solve for theta, where the smallest root is chosen as this ensures minimization of the difference between matrix A and B. In doing so, we minimize the norm of the off-diagonal matrix elements in A. This method diagonalizes the matrix, giving the eigenvalues and eigenvectors. In the implementation of this algorithm, rotations are performed until all off diagonal elements are less than a chosen tolerance, ϵ, as reaching exactly zero for all off diagonal elements can be computationally consuming.

```
// Function to find the values of cos and sin
mat rotate ( mat A, mat R, int k, int l, int n ) {
  double s, c;
  if ( A(k,l) != 0.0 ) {
    double t, tau;
    tau =( A(l,l) - A(k,k))/(2*A(k,l));
    if ( tau > 0 ) {
      t = tau + sqrt(1.0 + tau*tau);
    } else {
      t = -tau + sqrt(1.0 + tau*tau);
    }
    c = 1/sqrt(1+t*t);
    s = c*t;
  } else {
    c = 1.0;
```

```
    s = 0.0;
  }
  double a_kk, a_ll, a_ik, a_il, r_ik, r_il;
  a_kk = A(k,k);
  a_ll = A(l,l);
  // changing the matrix elements with indices k
      and l
  A(k,k) = c*c*a_kk - 2.0*c*s*A(k,l) + s*s*a_ll;
  A(l,l) = s*s*a_kk + 2.0*c*s*A(k,l) + c*c*a_ll;
  A(k,l) = 0.0; // hard-coding of the zeros
  A(l,k) = 0.0;
  // and then we change the remaining elements
  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) = c*a_ik - s*a_il;
      A(k,i) = A(i,k);
      A(i,l) = c*a_il + s*a_ik;
      A(l,i) = A(i,l);
    }
    // Finally, we compute the new eigenvectors
    r_ik = R(i,k);
    r_il = R(i,l);
    R(i,k) = c*r_ik - s*r_il;
    R(i,l) = c*r_il + s*r_ik;
  }
  return A;
}
```

FIG. 1: Function that rotates the given matrix in order to minimize the specified off-diagonal matrix element. Computes the rotated eigenvectors

```
vec jacobi_method ( mat A, mat &R, int n ) {
  //setting up the eigenvector matrix
  for ( int i = 0; i < n; i++ ) {
    for ( int j = 0; j < n; j++ ) {
      if ( i == j ) {
   R(i,j) = 1.0;
      } else {
   R(i,j) = 0.0;
      }
    }
  }
  int k, l;
  double epsilon = 1.0e-10;
  double max_number_iterations = (double) n *
      (double) n * (double) n;
  int iterations = 0;
  //find max off diagonal element
  double max_offdiag = maxoffdiag ( A, &k, &l, n );
  while ( fabs (max_offdiag) > epsilon && (double)
      iterations < max_number_iterations ) {
    max_offdiag = maxoffdiag( A, &k, &l, n );
    //perform the rotation to minimize the max off
        diagonal
    A = rotate ( A, R, k, l, n );
    iterations++;
  }
  //organize eigenvalues and eigenvectors
  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 temp(n);
    temp = 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,i) = R(j,index(i));
    }
  }
  R = eigvec;
  return eigval;
}
```

FIG. 2: Jacobi Rotation algorithm that produces a diagonal matrix revealing the eigenvalues of the original tridiagonal matrix. This function also organizes the eigenvalues and eigenvectors in ascending order.

RESULTS AND DISCUSSIONS

Unit Tests

In order to check the functionality of the discussed code, multiple unit tests are implemented throughout.

The first test applies the Jacobi algorithm to a 3x3 matrix. This test returns the expected eigenvalues and eigenvectors, which can be determined by hand or through the use of the Armadillo built-in Eigsym function. These values should be checked after changes are made to the code to ensure that functionality remains.

The next test proves that orthogonality remains after applying a rotation matrix to an orthogonal vector. It also shows that the dot product is conserved through these unitary transformations (Sec. Theory). This test uses a simple matrix such as the one used in the unit test mentioned above. The code with unit tests included can be found on github.

Jacobi Algorithm vs. Armadillo Eigensym

The Jacobi Algorithm was rigorously compared the Eigensym function to test for differences. The eigenvalues found with each method are in good agreement.

In Table 1, we see strong differences in the efficiency of each algorithm. For a given number of mesh points, this table presents an average time of completion for the algorithm, averaging over different frequencies and non-

interacting and interacting cases. The Eigsym algorithm is over an order a magnitude faster than the Jacobi algorithm for the lowest number of mesh points analyzed. When we use the highest number of mesh points that the Jacobi algorithm can handle in a reasonable amount of time, the Eigsym algorithm is over 3 orders of magnitude faster. 4ooo mesh points using the Eigsym function can be completed in approximately the same amount of time as 300 mesh points using the Jacobi algorithm.

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	-

TABLE I: Time for completion of both eigenvalue algorithms for a variety of mesh points. These times were averaged over interacting and non-interacting cases for a variety of frequencies. The Jacobi algorithm is increasingly less efficient when compared to the Armadillo Eigensym function for higher numbers of mesh points. At 500 mesh points, the Jacobi algorithm used more time than reasonably available.

CONCLUSIONS

We built a Jacobi Rotation Algorithm to solve for the eigenvalues and eigenvectors of tridiagonal matricies. We have applied this technique to the case of two electrons in a harmonic oscillator potential. The resulting energy eigenvalues and wavefunction eigenvectors were found in good agreement with the built-in Armadillo eigenvalue solver, Eigensym and with theory[2]. The Jacobi Rotation Algorithm is limited in its efficiency, as it takes longer than a minute to solve matrices for mesh points larger than 200.

Wavefunction for two Electrons in a Harmonic Oscillator Potential

Using the Armadillo function Eigsym, we were able to plot the wavefunction of two electrons in a Harmonic Oscillator Potential for the non-interacting and interacting cases for a variety of frequencies (fig3). We see that as frequency increases, the repulsive Coulomb potential from the interaction of two electrons becomes less prominent. It can be concluded that as the frequency decreases or as the potential well becomes shallower allowing for the electrons to be further apart, a stronger Coulomb interaction is in effect. This is due to the long range quality of the Coulomb interaction.

REFERENCES

[1] C. Sanderson, and R. Curtin, Armadillo, Journal of Open Source Software **1**(2). 2016.

[2] M. Taut, Phys. Rev. A. **48**, 5 (Nov 1993).

[3] M. Hjorth-Jensen, Computational Physics, Lecture Notes. Aug 2015

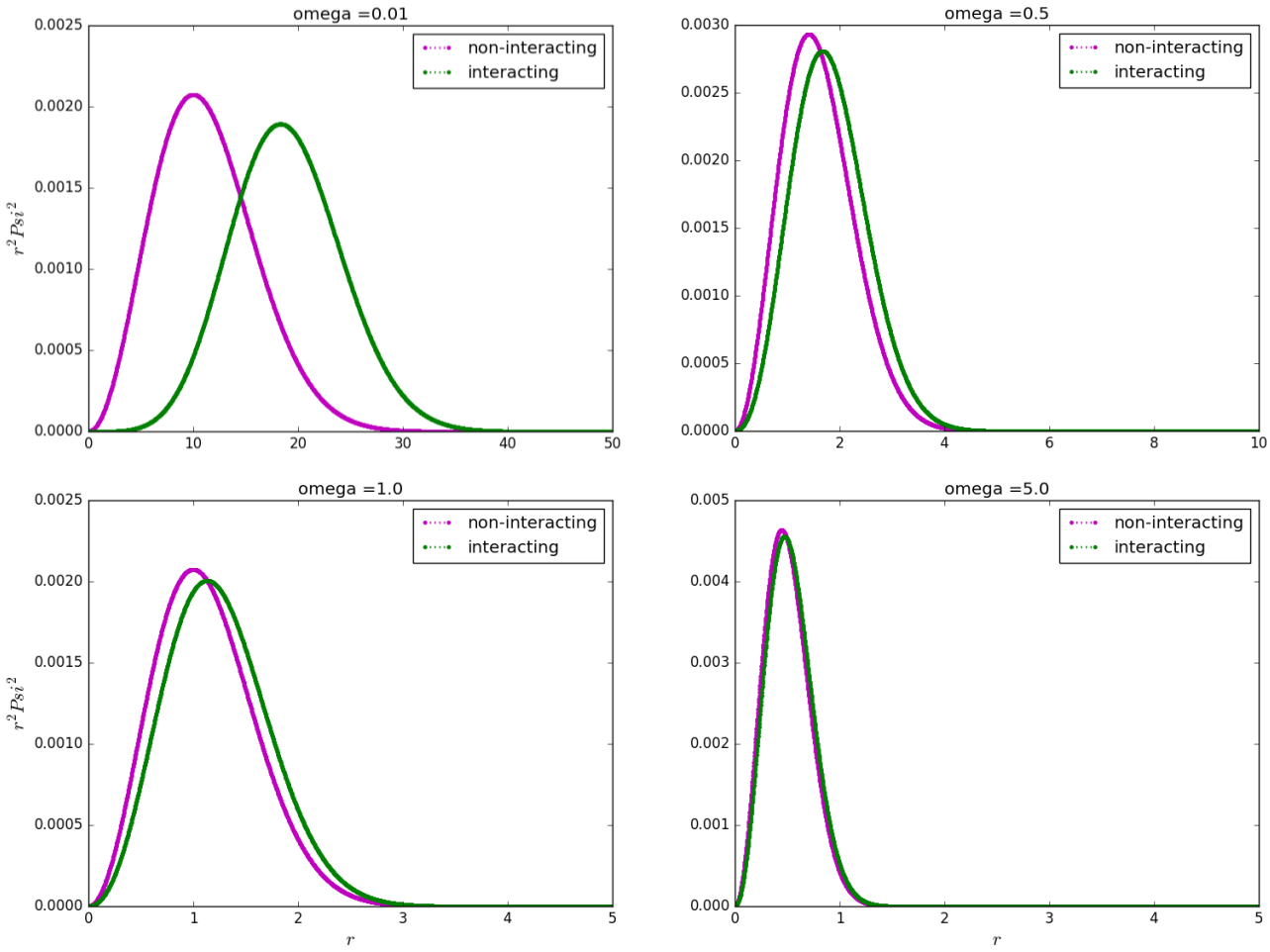

FIG. 3: Ground state wave function for the non-interacting and interacting cases at various frequencies. We see that as the frequency increases, the repulsive term in the potential becomes less These plots were made using the eigenvectors produced by the Armadillo Eigensym function.