Lars Christian Hauge

Project 2

Abstract

In this project I have used Jacobi's algorithm to find the eigenvalues of a harmonic oscillator well with one and later two electrons. I have looked at the efficiency of Jacobi's method by comparing it to armadillo's eigenvalue solver. Lastly I found eigenvectors for the three lowest energy levels of the oscillator system and plotted them against the relative distance between the electrons.

Introduction

The aim for this project is to solve Schrödinger's equation for two electrons in a three-dimensional harmonic oscillator well. Doing this we get an eigenvalue problem that we are going to solve using Jacobi's method. We are first going to study just one electron in a harmonic oscillator well, to see how efficient and precise Jacobi's method is. I am going to compare it with armadillo's way of finding eigenvalues. Then I am going to put a new electron in the potential to see how this affects the system.

Methods

The first thing I am asked to do is to write a function which implements Jacobi's rotation algorithm. Jacobi's rotation algorithm goes like this; if we have a symmetric matrix $\mathbf{A}$ and a rotation matrix $\mathbf{S}$. Where the rotation matrix's elements that differ from zero are: $\mathbf{S}_{kk} = \mathbf{S}_{ll} = \cos\theta, \mathbf{S}_{kl} = -\mathbf{S}_{lk} = \sin\theta and \mathbf{S}_{ii} = 1, i \neq k, l$. Then $\mathbf{B} = \mathbf{S}^\intercal \mathbf{A}\mathbf{S}$ is symmetric and similar and has entries:

$\mathbf{B}_{kk} = c^2\mathbf{A}_{kk} - 2cs\mathbf{A}_{kl} + \mathbf{A}_{ll}s^2$
$\mathbf{B}_{ll} = c^2\mathbf{A}_{ll} + 2cs\mathbf{A}_{kl} + \mathbf{A}_{kk}s^2$
$\mathbf{B}_{ik} = \mathbf{A}_{ik}c - \mathbf{A}_{il}s, i \neq k, l$
$\mathbf{B}_{il} = \mathbf{A}_{il}c + \mathbf{A}_{ik}s, i \neq k, l$
$\mathbf{B}_{kl} = (\mathbf{A}_{kk} - \mathbf{A}_{ll})cs + \mathbf{A}_{kl}(c^2 - s^2)$
where $c = \cos\theta$ and $s = \sin\theta$.

We want to choose θ so that all non-diagonal elements $\mathbf{B}_{kl} = \mathbf{B}_{lk} = 0$. This gives $(\mathbf{A}_{kk} - \mathbf{A}_{ll})cs + \mathbf{A}_{kl}(c^2 - s^2) = 0$ and after some algebra we have:

$$\cot 2\theta = \tau = \frac{\mathbf{A}_{ll} - \mathbf{A}_{kk}}{2\mathbf{A}_{kl}}$$

and after using $\cot 2\theta = frac12(\cot\theta - \tan\theta$ we end up with the quadratic equation

$$t^2 + 2\tau t - 1 = 0$$

which gives

$$t = \tan\theta = -\tau \pm \sqrt{1 + \tau^2}.$$

C and s I obtain by $c = 1\sqrt{1+t^2}$ and $s = ct$. Now we have Jacobi's method, it is just to repeat this many times till all non-diagonal elements are essentially zero, and we will eventually get the eigenvalues in the diagonal. Here is how I have implemented Jacobi's method in c++. First we have the main part that loops through the functions.

```
double maxnondiag = 1.0;
int maxiter = n*n*n;
double tolerance = 1.0E-10;
int iterations = 0;

while(maxnondiag > tolerance && iterations <= maxiter)
{
        int p, q;
        maxnondiag = offdiag(A, &p, &q, n);
        jacobi_rotate(A, R, p, q, n);
        iterations ++;
}
```

Then we have the part that does the actual work and rotates the matrix, and find the new entries to our matrix.

```
void jacobi_rotate(mat &A, mat &R, int k, int l, int n)
{
        double s, c;
        if (A(k,l) != 0.0){
        // defining tan, cos and sin of theta for the rotation
                double t, tau;
                tau = (A(l,l) - A(k,k))/(2*A(k,l));

                if (tau >= 0){
                        t = 1.0/(tau + sqrt(1.0 + tau*tau));}
                else {
                        t = -1.0/(-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;
        // Defining the new matrix elements for A.
        a_kk = A(k,k);
        a_ll = A(l,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;
        A(l,k) = 0.0;
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);
        }
        // THe eigenvectors also change so these we change here.
        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_il;
```

Lastly we have the part that searches after the largest non-diagonal element in the matrix so we can rotate around that.

```
double offdiag(mat &A, int *p, int *q, int n)
{
        double max = 0.0;
        int i, j;
        for(int i = 0; i < n; ++i){                 // for all elements in A
                for(int j = i+1; j < n; ++j){   // we search for the largest
                        double aij = fabs(A(i,j));       // and its indices
                        if ( aij > max)
                        {
                                max = aij; *p = i; *q = j;
                        }
                }
        }
        return max;
}
```

It could also be interesting to see how I made my matrix for in the last part.

```
for(i=0; i < n; i++){
        rho(i) = (i+1)*h;
        for(j=0; j < n; j++){
        if(j == i-1){
        A(i,j) = -1./pow(h,2);}
        else if(j == i+1 and j < n){
```

```
A(i,j) = -1./pow(h,2);}
else if(j == i){
A(i,j) = 2./pow(h,2) + pow(omega,2)*pow(rho(i),2) + 1./rho(i);}
}
}
```

To find eigenvalues to compare with to see if I have found the right ones, I used the armadillo function *eig_sym*. I also used this to find the eigenvectors in the last exercise. When I first tried this for $n = 100$ and $\rho_{max} = 1$, I could not get the right eigenvalues(3, 7, 11...). It was not until I changed ρ_{max} to 4.5 that I began to get real precision for the eigenvalues. As ρ_{max} is supposed to be about infinite it is somewhat unfortunate that the step size changes when ρ_{max} is changed. I think I have found a nice value that is not too low.

Results

We are asked in the text why we should choose $t = \tan\theta = -\tau \pm \sqrt{1+\tau^2}$ to be the smaller of the roots. By choosing the smaller of the roots we make sure that $|\tan\theta| \leq 1$ which again makes $|\theta| \leq \frac{\pi}{4}$. This again makes sure that c becomes as large as possible which minimizes

$$||\mathbf{B} - \mathbf{A}||_F^2 = 4(1-c) \sum_{i=1, i\neq k,l}^{n} (a_{ik}^2 + a_{il}^2) + \frac{2a_{kl}^2}{c^2}.$$

and makes sure that each step in the Jacobi's method gets you closer to a diagonal matrix and the eigenvalues instead of setting you back.

I never got all eigenvalues with four leading digits for the same n, with $n = 100$ and $\rho_{max} = 4.5$ I got 2.99938, 6.99702 and 10.9981 as the three lowest eigenvalues. I got the exact same thing with the armadillo function and the Jacobi rotation I implemented myself, what confused me after testing the program was that for higher n(=200) the precision went down for the third eigenvalue, but increased for the two lowest. It was the exact same thing for the armadillo function so I assume it is the initial matrix that is not so precise.

You needed many similarity transformations before all non-diagonal elements became essentially zero. We see that we need just over n^2 transformations before we get a nice matrix.

n	nr. of transf.	
10	161	$1.6n^2$
50	4329	$1.7n^2$
100	18439	$1.8n^2$
200	71624	$1.8n^2$

When comparing time usage I found, not surprisingly, that the armadillo function was much faster than Jacobi's method. I found the time usage for n=100 and n=200.

n	Jacobi	armadillo
100	1.313s	0.020s
200	19.172s	0.073s

When I added another electron I did not know what the eigenvalues should be, so I checked against Taut's article with oscillator frequency $\omega_r = 0.05$. My program did not give anything similar to what he got no matter how I chose the ρ_{max}. In the end I just went for $\rho_{max} = 5.0$ and n = 100, because I could not find a fault. The eigenvalue for the lowest energy state was:

ω_r	ϵ_0
0.01	0.84
0.5	2.23
1.0	4.06
5.0	17.43

We see that the energy increases with higher oscillator frequency just as in the article to Taut, which must mean I have done something right.

Lastly I have generated a number of plots that shows the wave function for different frequencies and energies. I have included both the square of the wave function and the wave function alone for some cases and have also for all energy levels plotted with and without the repulsion between the electrons. All the plots are collected on the last two pages of this report.

We see that it is mainly for $\omega_r = 0.5$ that we have any difference for the electron repulsion. We see that without the electron repulsion it becomes the inverse of the graph with repulsion.

Conclusion

In this exercise we have seen that Jacobi's method is not very fast as an eigenvalue method compared to armadillo's method for finding eigenvalues. They are about equally precise, so there are faster algorithms which gives the same result faster. Unfortunately they are harder to implement, so choosing Jacobi's method is not that idiotic.

We have seen how we can solve a harmonic oscillator problem numerically and seen how its energies behaves when putting in electrons for different oscillator frequencies.

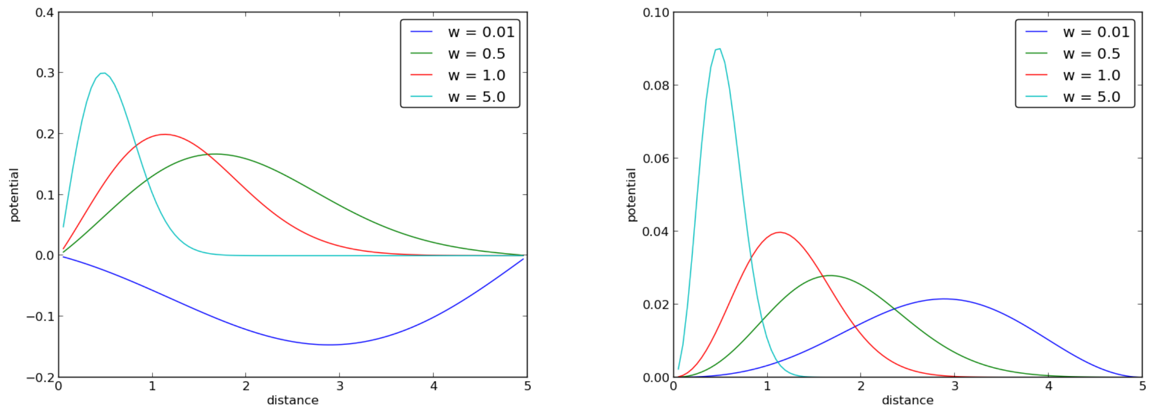

Figure 1: Ground state of the well, the one to the right is the square of the other

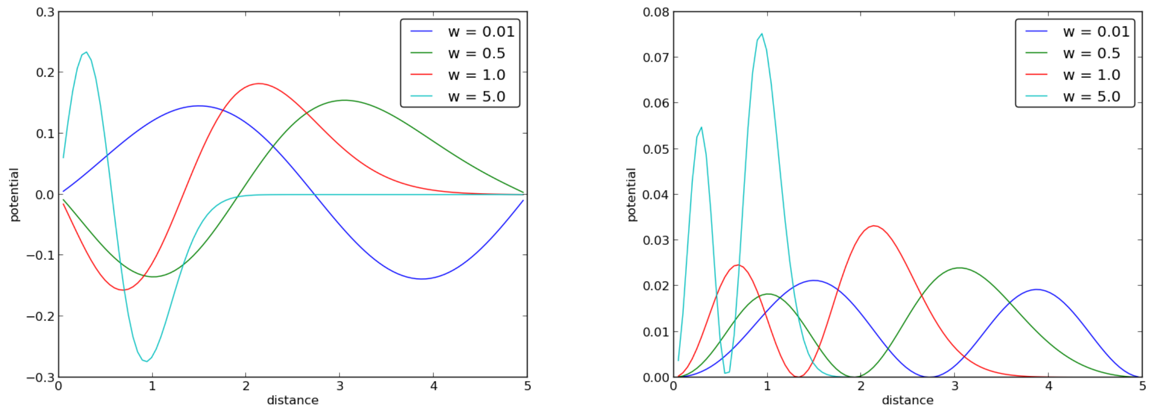

Figure 2: First excited state of the well, the one to the right is the square of the other

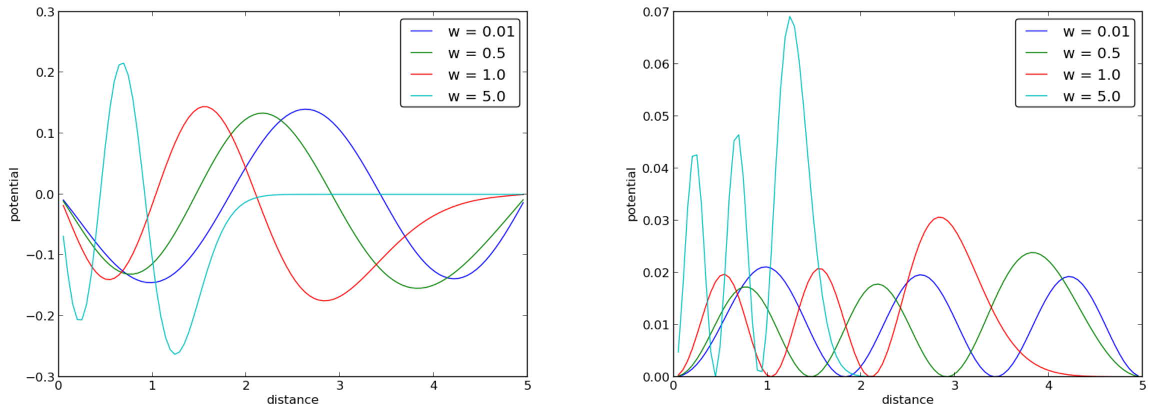

Figure 3: Second excited state of the well, the one to the right is the square of the other

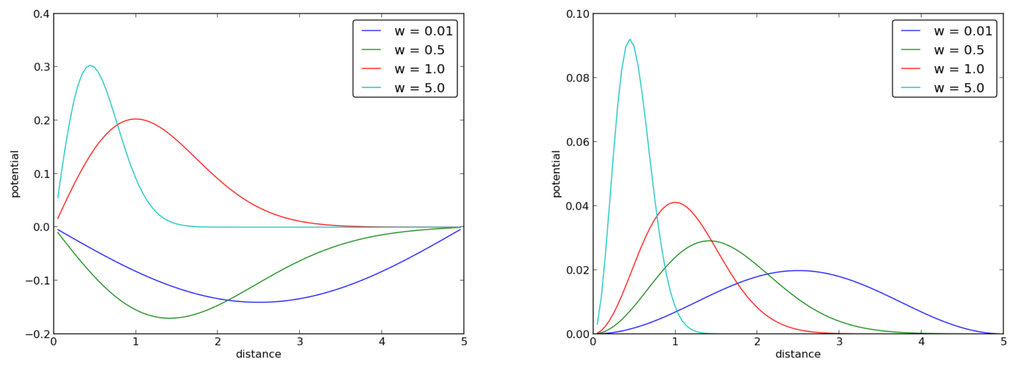

Figure 4: Ground state of the well without electron repulsion, the one to the right is the square of the other

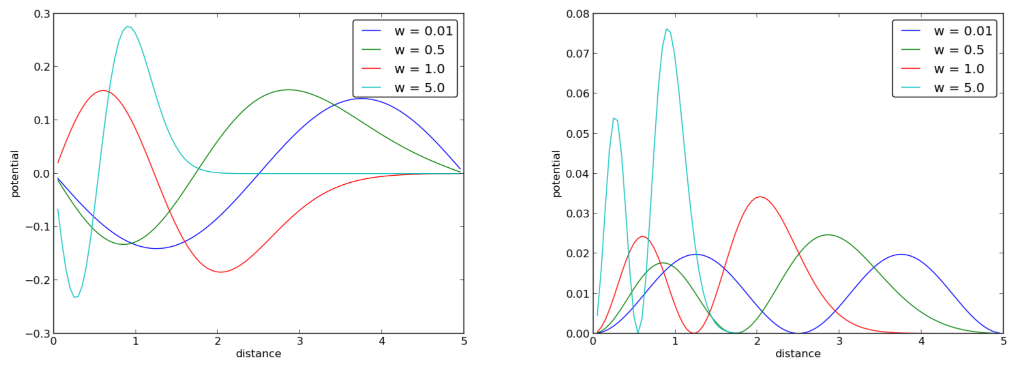

Figure 5: First excited state of the well without electron repulsion, the one to the right is the square of the other

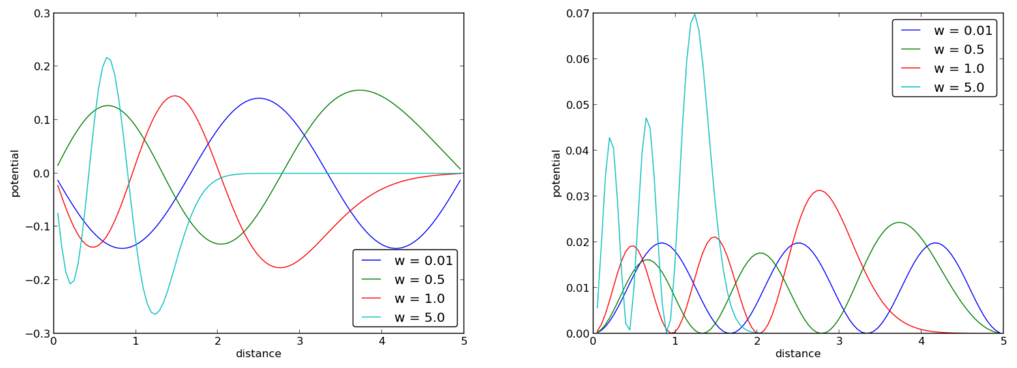

Figure 6: Second excited state of the well without electron repulsion, the one to the right is the square of the other