PHY 480 - Computational Physics
Project 2: Schrödinger's Equation for 2 electrons in a 3D Well

Thomas Bolden

March 4, 2016

Github Repository at https://github.com/ThomasBolden/PHY-480-Spring-2016

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

Introduction

In physics, there is a known solution to Schrödinger's equation for a single electron in a spherically symmetric three-dimensional well. However, this equation becomes more complicated when another elctron is added. In addition to the energy term, there are Coulombic interactions between the electrons that must be accounted for.

For one electron, the radial part of the Schrödinger equation reads

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

Methods

.

Results

.

Conclusions

.p

Code

../Code/EigensolverTests.cpp

```
/*
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        Project 2 – Schrodinger's Equation for two electrons in
                    a three–dimensional harmonic oscillator well

        Part a)   – Jacobi's rotation algorithm "brute force".

        Part b)   – Compare to Armadillo to find most efficient
                    algorithm.
                    Checking for n and rho dependency.

        Results are saved to a text file.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
*/

#include <iostream>
#include <fstream>
#include <iomanip>

#include <string>
#include <time.h>
```

```
22
23 #include <cmath>
24 #include <armadillo>
25
26 using namespace std;
27 using namespace arma;
28
29 ofstream myfile;
30
31 /*
32   ======= Function to return indices of largest off-diagonal element =======
33     Arguments:
34     mat A - matrix
35     int n - size of square matrix
36     int *k - new row
37     int *l - new column
38 */
39
40 double MaxOffDiagonal(mat A, int *k, int *l, int n){
41
42     double maxOD;
43     double a_ij;
44
45     for (int i = 0; i < n; i++){
46
47         for (int j = i+1; j < n; j++){
48
49             a_ij = fabs(A(i,j));
50
51             if (a_ij > maxOD){
52
53                 maxOD = a_ij;
54                 *k = i;
55                 *l = j;
56
57             }
58         }
59     }
60     return maxOD;
61 }
62
63 /*
64   ============ Function implimenting Jacobis rotation algorithm ============
65     Arguments:
66     mat &A - reference matrix A
67     mat &B - reference matrix B
68     int k - new row index
69     int l - new column index
70     int n - dimensionality of matrix
71 */
72
73 void Rotate(mat &A, mat &B, int k, int l, int n){
74
75     double t;
```

```
76      double tau;
77      double sine;
78      double cosine;
79
80      if ( A(k,l) != 0.0){
81
82          tau = ( A(l,l) - A(k,k) )/( 2*A(k,l) );
83
84          if (tau >= 0.0) {
85
86              t = 1.0/(fabs(tau) + sqrt(1.0 + tau*tau));
87
88          }
89
90          else {
91
92              t = -1.0/(fabs(tau) + sqrt(1.0 + tau*tau));
93
94          }
95
96          cosine = 1.0/ sqrt (1.0 + t*t);
97          sine = t*cosine;
98
99      }
100
101     else {
102
103         sine = 1.0;
104         cosine = 0.0;
105
106     }
107
108     double A_ik;
109     double A_il;
110     double B_ik;
111     double B_il;
112     double A_kk = A(k,k);
113     double A_ll = A(l,l);
114
115     A(k,k) = A_kk*cosine*cosine - 2*A(k,l)*cosine*sine + A_ll*sine*sine;
116     A(l,l) = A_ll*cosine*cosine + 2*A(k,l)*cosine*sine + A_kk*sine*sine;
117     A(k,l) = 0.0;
118     A(l,k) = 0.0;
119
120     for (int i = 0; i < n; i++){
121         if (i != k && i != l){
122
123             A_ik = A(i,k);
124             A_il = A(i,l);
125             A(i,k) = A_ik*cosine - A_il*sine;
126             A(k,i) = A(i,k);
127             A(i,l) = A_il*cosine + A_ik*sine;
128             A(l,i) = A(i,l);
129
```

```
130         }
131
132         B_ik = B(i,k);
133         B_il = B(i,l);
134         B(i,k) = B_ik*cosine - B_il*sine;
135         B(i,l) = B_il*cosine + B_ik*sine;
136
137     }
138     return;
139 }
140
141 int main(){
142
143     int n;
144     int loops;
145     string filename;
146     double rho;
147     double h;
148     double e;
149     double jacobi_time;
150     double armadillo_time;
151     rowvec N;
152
153     int maxloops = 1e8;
154     double epsilon = 1e-10;
155
156     filename = "Comparisons8.txt";
157     rho = 8.0;
158
159 //  cout << "Enter a name for the file: ";
160 //  cin >> filename;
161 //  cout << "\nEnter a value (double) for rho: ";
162 //  cin >> rho;
163
164     N << 50 << 100 << 150 << 200 << 250 << 300 << 350 << 400 << 450 << 500;
165
166     myfile.open(filename);
167     myfile << setiosflags(ios::showpoint | ios::uppercase);
168     myfile << "rho: " << rho << endl;
169     myfile << "Tolerance: " << epsilon << endl;
170     myfile << right << setw(4) << setfill(' ') << " n: ";
171     myfile << right << setw(30) << setfill(' ') << "  Eigenvalues: ";
172     myfile << right << setw(16) << setfill(' ') << "  Arma Time: ";
173     myfile << right << setw(16) << setfill(' ') << "  Jacobi Time: ";
174     myfile << right << setw(12) << setfill(' ') << "  Transforms: " << endl;
175
176     for (int j = 0; j < 10; j++){
177
178         n = N(j);
179         loops = 0;
180         h = rho / n;
181         e = -1.0/(h*h);
182
183         vec d(n-1);
```

```
184         vec p(n+1);
185         vec eigenvalues(n-1);
186         mat A(n-1,n-1);
187         A.zeros();
188         mat M(n-1,n-1);
189         M.eye();
190
191         for (int i = 0; i <= n; i++){
192
193             p(i) = i*h;
194
195         }
196
197         for (int i = 0; i < n-1; i++){
198
199             d(i) = 2/(h*h) + p(i+1)*p(i+1);
200
201         }
202
203         for (int i = 0; i < n-1; i++){
204
205             A(i,i) = d(i);
206
207             if (i != n-2){
208
209                 A(i,i+1) = e;
210                 A(i+1,i) = e;
211
212             }
213
214         }
215
216         mat B = A;
217
218         clock_t start_Jacobi , end_Jacobi;
219         int k;
220         int l;
221         double MaxOffDiag = MaxOffDiagonal(A, &k, &l, n-1);
222
223         start_Jacobi = clock();
224
225         while (MaxOffDiag > epsilon && loops < maxloops){
226
227             MaxOffDiag = MaxOffDiagonal(A, &k, &l, n-1);
228             Rotate(A, M, k, l, n-1);
229             loops ++;
230
231         }
232
233         end_Jacobi = clock();
234         jacobi_time = (end_Jacobi - start_Jacobi)
235                       /(double)CLOCKS_PER_SEC;
236
237         for (int i = 0; i < n-1; i++){
```

```
238
239             eigenvalues(i) = A(i,i);
240
241         }
242
243         eigenvalues = sort(eigenvalues);
244
245         clock_t start_Armadillo , end_Armadillo;
246         start_Armadillo = clock();
247         mat eigenvector;
248         vec eigenvalue;
249         eig_sym(eigenvalue , eigenvector , B);
250         end_Armadillo = clock();
251         armadillo_time = (end_Armadillo - start_Armadillo)
252                          /(double)CLOCKS_PER_SEC;
253
254         myfile << right << setw(4) << setfill(' ') << n;
255         myfile << right << setw(6) << setfill(' ') << "(" << eigenvalues(0)
256                << ", ";
257         myfile << right << setw(6) << setfill(' ') << eigenvalues(1) << ", ";
258         myfile << right << setw(6) << setfill(' ') << eigenvalues(2) << ")";
259         myfile << right << setw(16) << setfill(' ') << armadillo_time << " s";
260         myfile << right << setw(16) << setfill(' ') << jacobi_time << " s";
261         myfile << right << setw(12) << setfill(' ') << loops << endl;
262
263     }
264
265     myfile.close();
266     return 0;
267
268 }
```

../Code/2eWavefunctions.cpp

```
1  /*
2  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
3          Project 2 - Schrodinger's Equation for two electrons in
4                      a three-dimensional harmonic oscillator well
5
6          Part d)   - Solving the 2-electron wavefunction of coordinate
7                      rho with varying omega_rho
8
9          Results are saved to a text file.
10 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
11 */
12
13 #include <iostream>
14 #include <fstream>
15 #include <iomanip>
16
17 #include <string>
18 #include <time.h>
19
20 #include <cmath>
```

```cpp
#include <armadillo>

using namespace std;
using namespace arma;

ofstream myfile;

int main(){

    int n;

    cout << "Dimensionality of matrix (n): ";
    cin >> n;

    double omega;

    cout << "Value for Omega_r: ";
    cin >> omega;

    string filename;

    cout << "Enter a name for the file: ";
    cin >> filename;

    double rho = 5.0;
    double h;
    double el;

    h = rho / n;
    el = -1.0/(h*h);

    mat A(n-1,n-1);
    vec p(n+1);
    vec z(n+1);

    for (int i = 0; i < n; i++){

        p(i) = i*h;

    }

    for (int i = 0; i < n-1; i++){

        z(i) = pow(omega,2)*pow(p(i+1),2) + 1.0/p(i+1) + 2.0/(pow(h,2));

    }

    for (int i = 0; i < n-1; i++){

        A(i,i) = z(i);

        if (i != n-2){

            A(i,i+1) = el;
```

```
            A(i+1,i) = el;

        }
    }

    vec eigenvalues;
    mat eigenvectors;

    eig_sym(eigenvalues,eigenvectors,A);

    myfile.open(filename);
    myfile << setiosflags(ios::showpoint | ios::uppercase);
    myfile << right << setw(10) << setfill(' ') << "rho" << endl;

    for (int i = 0; i < n-1; i++){

        myfile << setw(15) << setfill(' ') << setprecision(10) << p(i+1);
        myfile << setw(15) << setfill(' ') << setprecision(10)
               << eigenvectors.col(0);
        myfile << setw(15) << setfill(' ') << setprecision(10)
               << eigenvectors.col(1);
        myfile << setw(15) << setfill(' ') << setprecision(10)
               << eigenvectors.col(2) << endl;

    }

    myfile.close();
    return 0;


    //eigenvectors.col(0)

}
```

References

[1] M. Hjorth-Jensen, *Computational Physics*, University of Oslo (2015).