Computational Physics

Project 2

Hunter Wilhelm Phillips (*hunterp*)

October 3, 2018

GitHub Repository at

https://github.com/robolux/Computational_Physics

Abstract

This project explores the Schrödinger's equation to find the wave function of a system composed by an electron in a harmonic oscillator well. The equations were discretized into an eigenvalue problem so that the Jacobi method to diagonalize a matrix could be used with differing integration points. By analyzing the eigenstates and eigenvectors the wave function and energy of the particle could be observed. After the general equation algorithm was developed, two special cases with the harmonic oscillator potential and repulsive Coulomb interaction were added in separate function definitions. The results confirmed a successful discretized solution for electron(s) in a three- dimensional harmonic oscillator well using Schrödinger's equation.

1. Discussion of Problem Statement

In this project, the goal is to analyze a 3-dimensional harmonic oscillator potential with spherical symmetry that acts on a system of electrons. It is assumed that there are no angular components for the particles, only radially dependent movement. The two particles interact with respect to Coulomb's force acting upon the bodies.

1.1 One Particle System

The first step to explaining a system containing one particle is to look at the radial Schrödinger's equation (1):

$$-\frac{h^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2\frac{d}{dr}\right)\varphi(r) + V(r)\,\varphi(r) = E\,\varphi(r) \tag{1}$$

where $\varphi(r)$ is the time independent wave function defined as the spatial second order derivative written in spherical coordinates
$V(r)$ is the harmonic oscillator potential
E is the energy of the harmonic oscillator in three dimensions

By taking this system and displaying the closed form for the energy $E(r)$ that are eigenstates of the above wave function.

$$E_n = h_\omega\left(2n + \frac{3}{2}\right)$$

Using n = 0, 1, 2 …. until an endpoint is desired the simplification of (1) can be simplified with the discretization of $u\,(r) = r\,\varphi(r)$ with the coupled boundary conditions of $u\,(0) = 0$ and $u\,(\infty) = 0$. This results in (2) being formed:

$$-\left(\frac{h^2}{2m}\right)\left(\frac{d^2}{dr^2}\right)u(r) + V(r)\,u(r) = E\,u(r) \tag{2}$$

The introduction of a dimensionless variable $\rho = \left(\frac{1}{\alpha}\right) r$ where the constant α has a dimensional length given forms (3):

$$-\left(\frac{h^2}{2m\alpha^2} \right)\left(\frac{d^2}{d\rho^2} \right) u(\rho) + \left(\frac{k}{2} \right) \alpha^2 \rho^2 \, u(\rho) = E \, u(\rho) \tag{3}$$

To make the potential coefficient equal to 1 the constant value must be fixed with the decision resulting in the following:

$$\alpha = \sqrt[4]{\frac{h^2}{mk}}$$

This allows the grounding equation to become dimensionless, however resulted in a change of the energy eigenvalues in the original wave function. By defining the following, we can subsequently rewrite Schrödinger's equation:

$$\lambda = \frac{2m\alpha^2}{h^2} E$$

$$-\left(\frac{d^2}{d\rho^2} \right) u(\rho) + \rho^2 \, u(\rho) = \lambda \; u(\rho) \tag{4}$$

This final equation (4) is the derived equation to solve numerically and the associated three-dimensional closed form for the eigenvalues, $\lambda_n = 3 + 4n$, can be coupled for a solution index.

1.2 Two Particle Solution

The second step is the explanation of two electron system in a harmonic oscillator well that interact with each other with a repulsive Coulomb interaction. By considering no repulsive Coulomb interactions, the following Schrödinger's equation in spherical coordinates can be derived:

$$\left(-\frac{h^2}{2m}\frac{d^2}{dr_1^2} - \frac{h^2}{2m}\frac{d^2}{dr_2^2} + \frac{1}{2}kr_1^2 + \frac{1}{2}kr_2^2 \right) \varphi(r_1, r_2) = E \, \varphi(r_1, r_2) \tag{5}$$

By separating the two-electron energy E from the wave function in (5) through the consideration of the relative coordinates $r = r_1 - r_2$ and the coupled center-of-mass coordinate $R = \frac{1}{2}(r_1 + r_2)$ the coordinate shifted Schrödinger's equation has become:

$$\left(-\frac{h^2}{m}\frac{d^2}{dr^2} - \frac{h^2}{4m}\frac{d^2}{dR^2} + \frac{1}{4}kr^2 + kR^2 \right) \varphi(r,R) = E^{(2)}\, \varphi(r,R) \tag{6}$$

By adding the Coulomb's potential (7) and the following dimensionless variable (8) the related definitions can be defined (9), (10), and (11):

$$V(r_1, r_2) = \frac{\beta e^2}{| r_1 - r_2 |} = \frac{\beta e^2}{r} \tag{7}$$

$$\rho = \frac{r}{\alpha} \tag{8}$$

$$\omega^2 = \frac{1}{4}\frac{mk}{h^2}\alpha^4 \tag{9}$$

$$\alpha = \frac{h^2}{m\beta e^2} \tag{10}$$

$$\lambda = \frac{m\alpha^2}{h^2}E \tag{11}$$

These terms coupled produce a compact version of Schrödinger's equation:

$$-\left(\frac{d^2}{d\rho^2} \right) u(\rho) + \left(\omega^2\rho^2 + \frac{1}{\rho} \right) u(\rho) = \lambda\, u(\rho) \tag{12}$$

This final equation (12) has concluded in a two-dimensionless differential equation to solve. It represents two systems for the eigenvalue problems, but can still be solved with the same single-electron system algorithm with the minor compact addition.

2. Discussion of Methods

2.1 Preservation of Orthogonality

A key aspect of this method is the preservation of the associated orthogonality and this starts with the transformation:

$$w_i = Uv_i \tag{13}$$

Where U is an orthogonal matrix, is defined as a unitary transformation

We will show that the unitary transformation of (13) preserves the orthogonality of the vector, which can be tested with the dot product $v_j^T v_i = \delta_{ij}$. The dot product after the unitary transformation $w_j^T w_i$ should stay the same.

$$w_j^T w_i = (Uv_i)^T (Uv_j) = (v_i^T U^T)(Uv_j) = v_i^T I v_j = v_i^T v_j = \delta_{ij}$$
$$(Uv_i)^T = v_i^T U^T$$
$$U^T U = I$$
$$\therefore \text{ U is orthogonal}$$

This verifies that orthogonality is preserved and the systems holds with the expected results.

2.2 Jacobi Method for Diagonalization

The simplest form to describe the matrix needed for solving is as follows:

$$Au = \lambda u \tag{14}$$

Where *u* is the eigenvector representing the wave function corresponding to its eigenvalue energy level, and *A* is the matrix containing the discretized Schrödinger's equation.

The most important first step in algorithmic mathematical development is to discretize the second derivative as followed in [2]. The defined domain of 0 to $+\infty$ needs to be restricted with a maximum value of ρ decided to be 7, which should suffice for this case. The step size is fixed by $h = \frac{\rho_{max}}{n_{step}}$ and this allows the discretized form of Schrödinger's equation to be written:

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + V_i u_i = \lambda u_i \tag{15}$$

Where V_i is the harmonic oscillator potential either alone or with the addition of the Coulomb potential term

By writing out the matrix through the use of (15), the potential terms can be formed into a tridiagonal matrix

$$A = \begin{bmatrix} \frac{2}{h^2} + V_1 & -\frac{1}{h^2} & 0 & \dots & \dots & 0 \\ -\frac{1}{h^2} & \frac{2}{h^2} + V_2 & -\frac{1}{h^2} & 0 & \dots & \dots \\ 0 & -\frac{1}{h^2} & \frac{2}{h^2} + V_3 & -\frac{1}{h^2} & 0 & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{n_{step}-2} & -\frac{1}{h^2} \\ 0 & \dots & \dots & \dots & -\frac{1}{h^2} & \frac{2}{h^2} + V_{n_{step}-1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_i \\ \dots \\ u_{n_{step}-1} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_i \\ \dots \\ u_{n_{step}-1} \end{bmatrix}$$

By applying rotations to the matrix in order to make it become diagonal the following equation (16) is sequenced:

$$(S^T A S)(S^T)x = \lambda (S^T)x \tag{16}$$

With the matrix *S* being in the form of:

$$S = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & \dots & 0 & \dots & \dots \\ 0 & 0 & 1 & \dots & 0 & \dots & 0 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \cos(\theta) & \dots & \sin(\theta) & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -\sin(\theta) & \dots & \cos(\theta) & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix}$$

This allows for the triple matrix product to be computed to derive a simple multiplication scheme. By calling $b_{i,j}$ the elements of the transformed matrix and let *k* and *l* be the indexes of the largest non-diagonal element the following equations are reduced:

$$b_{ii} = a_{ii}\ with\ i \neq k\ \&\&\ i \neq l$$
$$b_{ik} = a_{ik}cos\theta - a_{il}sin\theta$$
$$b_{il} = a_{il}cos\theta + a_{ik}sin\theta$$
$$b_{kk} = a_{kk}cos^2\theta - 2a_{kl}cos\theta sin\theta + a_{ll}sin^2\theta$$
$$b_{ll} = a_{ll}cos^2\theta + 2a_{kl}cos\theta sin\theta + a_{kk}sin^2\theta$$
$$b_{kl} = (a_{kk} - a_{ll})cos\theta sin\theta + a_{kl}(cos^2\theta - sin^2\theta)$$

Using these derived iteration-based equations, the next step is after every rotation, to also rotate the eigenvectors. Using (16) through the use of the orthonormal vectors reduce into the following explicit rotation results:

$$u'_{ik} = u_{ik}cos\theta - u_{il}sin\theta$$
$$u'_{il} = u_{il}cos\theta + u_{ik}sin\theta$$

This concludes the analysis of Jacobi Method for Diagonalization and allows the progression towards algorithmic development.

3. Discussion of Algorithms

3.1 Jacobi's Rotation for Tridiagonal Toeplitz Matrix

The core algorithm for this project involves a few key areas that are highlighted in their respective sections below.

Solve Matrix

```
def solve(A, R, tol):

    n = A.shape[0]
    iterations = 0
    maximum, k, l = find_max(A)
    while maximum > tol:
        iterations += 1
        rotate(A, R, k, l)
        maximum, k, l = find_max(A)

    return iterations, tol
```

It can be seen that the matrix is rotated and subsequently fed into the find_max function until the desired tolerance precision is reached. The find_max and rotate functions can be seen below. The iteration count is also outputted per the request of the prompt for later analysis.

Find Max Non-Diagonal

```
def find_max(A):
    n = A.shape[0]
    maximum = abs(A[0,1])
    max_k=0
    max_l=1
    for i in xrange(n):
        for j in xrange(i+1, n):
            if abs(A[i,j]) > maximum:
                maximum = abs(A[i,j])
                max_k = i
                max_l = j
    return maximum, max_k, max_l
```

Rotate Matrix

```
def rotate(A, R, k, l):
    n = A.shape[0]
    tau = (A[l,l] - A[k,k])/(2*A[k,l])
    if tau > 0:
        t = 1./(tau + math.sqrt(1 + tau**2))
    else:
        t = 1./(tau - math.sqrt(1 + tau**2))
    c = 1 / math.sqrt(1+t**2)
    s = c*t
    a_kk = A[k,k]
    a_ll = A[l,l]
    A[k,k] = c**2*a_kk - 2*c*s*A[k,l] + s**2*a_ll
    A[l,l] = s**2*a_kk + 2*c*s*A[k,l] + c**2*a_ll
    A[k,l] = 0
    A[l,k] = 0
    for i in xrange(n):
        if i != k and 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]
        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
```

To solve the Part B case (one particle system) the basic matrix generation is showcased as follows:

Part B: One Particle System

```
def make_matrix_case_b(n, omega, p_max):

    A = np.zeros(shape=(n,n), dtype=np.float64)

    rho_0 = 0
    rho_n = p_max
    rho = np.linspace(rho_0, rho_n, n+2)[:-1]   # quickfix
    h = rho[1]-rho[0]
    V = np.zeros(n+1)
    V[1:] = omega**2*rho[1:]**2
    d = 2/h**2 + V
    e = -1/h**2

    A[range(n), range(n)] = d[1:]
    A[range(1, n), range(n-1)] = e
    A[range(n-1), range(1, n)] = e
    return A, rho
```

3.2 Unit Testing

To ensure that our algorithm was working correctly in every execution, unit tests were devised. The first test compares the resulting eigenvalues with the built in numpy .eigh function to ensure they are equivalent to 8 decimal points. The second test uses the numpy .eig function and implements a sorting function to match the results and compares to make sure they are equivalent to the same precision as .eigh and the original algorithm that was developed. If these tests do or do not pass, the user it notified in the command line or text file of the pass or failure.

3.3 Modifications for Specific Mathematical Cases

To solve parts d and e of the prompt, the harmonic oscillator potential and repulsive Coulomb interaction needed to be added respectively. This is fairly trivial and the excerpts from the modified make_matrix functions are shown below:

Make Matrix for Specific Cases

```
# Case D - Added Harmonic Oscillator Potential
V[1:] = omega**2*rho[1:]**2 + rho[1:]**2
d = 2/h**2 + V
e = -1/h**2

# Case E - Repulsive Coulomb Interaction
V[1:] = omega**2*rho[1:]**2 + 1/rho[1:]
d = 2/h**2 + V
e = -1/h**2
```

4. Discussion of Results

4.1 System Specifications

A few notes about the system running the programs before analyzing the results should be helpful in gauging performance between machines as seen in *Table 1*.

Operating System	Mac OS X
Processor	i7 - 2.2 GHz
Memory	8 GB 1600 MHz DDR3
Hard drive	128 GB SSD

Table 1: System Specifications

With the test conditions being established we can now dive into the data recorded.

4.2 Part B

The returned eigenvalues come out as expected within rounding error for the test case of n = 4, omega = 1, and rho max = 2 with a tolerance of (1E-10) and can be seen in in *Table 2*.

	E_1	E_2	E_3	E_4
Numerical	3.4363526473	9.9109886642	17.624738783	23.827919904
Expected	3.43635	9.91099	17.62470	23.82800

Table 2: Numerical vs Expected – Part B

The relationship between matrix dimension and then computational time with the amount of transformational iterations needed for the solution tolerance can be seen below in *Table 3*.

n	Computation Time (s)	Iterations	Iterations / Element
4	0.000360	20	1.25
16	0.037828	440	1.72
64	5.868330	7408	1.81
128	90.74790	30064	1.83

Table 3: Comparing Relationships for Growing n x n Matrix Size in Part B

The tolerance that was selected (1E-10) to be zero for this algorithm and this resulted in the values shown in *Table 3* being produced. It is interesting to note that the larger n grows, the less iterations per element are needed, it is just a case of so many iterations being needed that grows the computational time so large.

4.3 Part C

The implementation of the unit tests described in Section 3.2 result in the following output either on the command line or file, depending on the script

```
Part B Sol     numpy .eig     numpy .eigh
****************************************
3.4363526473  3.4363526473  3.4363526473
9.9109886642  9.9109886642  9.9109886642
17.624738783  17.624738783  17.624738783
23.827919904  23.827919904  23.827919904
The Unit Tests have PASSED
```

It allows the end user to individually see the comparison and get an explicit Passed or Not Passed at the end of the output to validate their visual observation.

4.4 Part D

Through the exploration of quantum dots in three dimensions for one electron the slightly modified algorithm resulted in the following expected vs numerical results in *Table 4*. The same initial conditions used in Section 4.2 were used for these values.

	E_1	E_2	E_3	E_4
Numerical	4.3444801411	11.172505935	18.858022617	25.224991306
Expected	4.35	11.18	18.86	25.25

Table 4: Numerical vs Expected – Part D

The relationship between matrix dimension and then computational time with the amount of transformational iterations needed for the solution tolerance can be seen below in *Table 5*.

n	Computation Time (s)	Iterations	Iterations / Element
4	0.000288	16	1
64	5.524490	7407	1.81
128	77.13050	29884	1.82

Table 5: Comparing Relationships for Growing n x n Matrix Size in Part D

The algorithm is slightly faster for this case when compared to Part B, but that is just an observation for an unrelated facet.

4.5 Part E

Through the continued divergence of quantum dots in three dimensions for two electrons the slightly modified algorithm resulted in the following varying ω value results in *Table 6*. To quantify this difference the results in [1] are used to compare. The same initial conditions used in Section 4.2 were used for these values.

	E_1	E_2	E_3	E_4
$\omega = 0.01$	3.5300584925	10.040939887	17.789212852	23.848602101
$\omega = 0.50$	3.8334295107	10.360535871	18.108435348	24.105932602
$\omega = 1.00$	4.6851803654	11.320699653	19.057912350	24.944540964
$\omega = 5.00$	15.911354332	30.837117967	49.957335918	78.502525115

Table 6: Varying ω

Through the comparison it can be seen the increasing ω affects the Energy directly and can be correlated with verification.

5. Conclusion

Through this project, the differing solution spaces for Schrödinger's equation have been understood. It also verifies that some quantum mechanical systems can be solved through the use of diagonalization algorithms. The highly stable Jacobi algorithm displayed its reliable use, but with the cost of computation time. Typically, solution spaces less than n = 64 do not occur and above that the algorithm would take forever to solve large data sets. The behavior discovered in the modified special cases is expected and provides a broader scope with the use of eigenvector algorithm solvers. The project concludes in a more complete understanding of eigenvalue problems and the associated algorithm development to solve them.

6. References

[1] Taut, M. "Two Electrons in an External Oscillator Potential: Particular Analytic Solutions of a Coulomb Correlation Problem." *Physical Review A*, vol. 48, no. 5, 1993, pp. 3561–3566., doi:10.1103/physreva.48.3561.

[2] Hjorth-Jensen, Morten. "Project 2." *Computational Physics FYS3150*, Fall 2018. University of Oslo. Lectures.