# FYS3150-Project2
# Eigenvalue problems, from the equations of a buckling beam to Schroedinger's equation for two electrons in a threedimensional harmonic oscillator well

Chenghao Liu

October 1, 2018

## 1. Introduction

The aim of this project is to develop your own code for solving eigenvalue problems. The matrix to diagonalize is the same as the one we encountered in project one, a so-called tridiagonal Toeplitz matrix. This matrix has analytical eigenpairs (eigenvalues and eigenvectors) and gives us an excellent testing ground for our algorithms. In this project we will develop an eigenvalue solver based on Jacobi's method. The project will also introduce you to units tests and we will compare our results against other eigenvalue solvers (from LAPACK and/or numpy).

This project aims also at introducing to you the concept of scaling of equations.

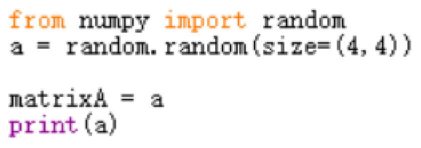
This means often either making various variables dimensionless or introducing

units which are more convenient.

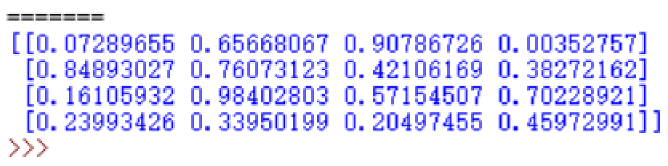
## 2. Method

I used Python for programing the code, for example, I used python numpy to create a random matrix for me to find the eigenvalues and eigenvectors. And also I can use python numpy to make a symmetric or nondiagonal matrix that is suitable for our project, with the parameter of a1, a2, a3, I can change the elements of the matrix as my choice.

The results of the code for random matrix is shown as follow:

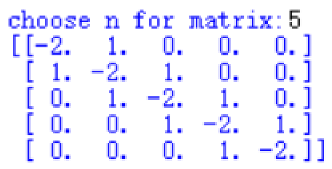


```
from numpy import random
a = random.random(size=(4,4))

matrixA = a
print(a)
```



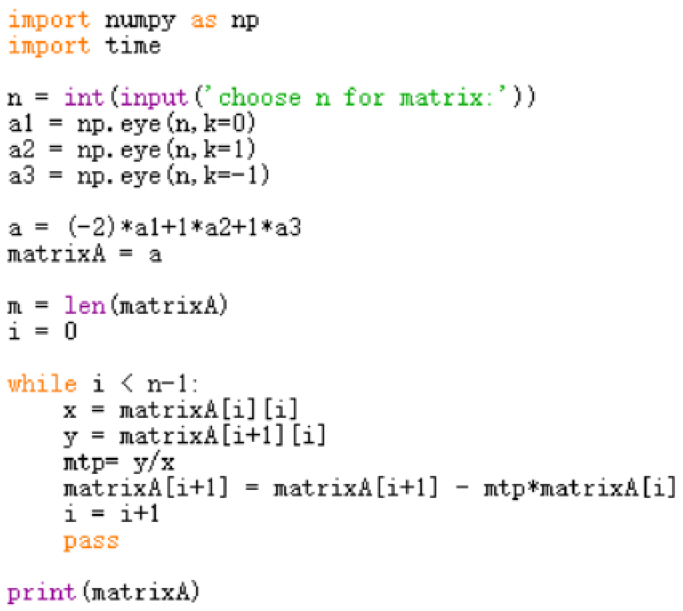
```
=======
[[0.07289655 0.65668067 0.90786726 0.00352757]
 [0.84893027 0.76073123 0.42106169 0.38272162]
 [0.16105932 0.98402803 0.57154507 0.70228921]
 [0.23993426 0.33950199 0.20497455 0.45972991]]
>>>
```

For creating symmetric matrix, I use numpy.eye function to make it. If I choose a 5*5 matrix, and the nondiagonal elements is 1, then we can get the matrix:



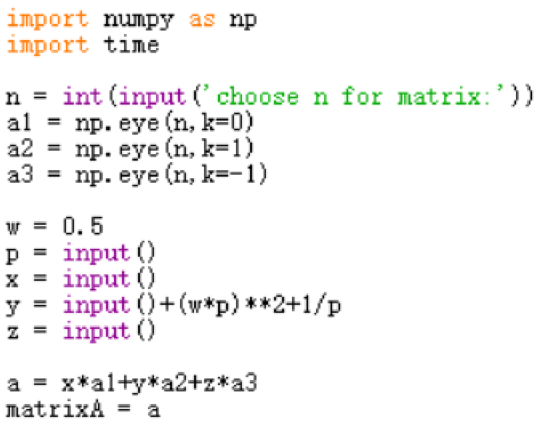
```
choose n for matrix:5
[[-2.  1.  0.  0.  0.]
 [ 1. -2.  1.  0.  0.]
 [ 0.  1. -2.  1.  0.]
 [ 0.  0.  1. -2.  1.]
 [ 0.  0.  0.  1. -2.]]
```

Finally we can simplify the matrix, used the code bellow:



```
import numpy as np
import time

n = int(input('choose n for matrix:'))
a1 = np.eye(n,k=0)
a2 = np.eye(n,k=1)
a3 = np.eye(n,k=-1)

a = (-2)*a1+1*a2+1*a3
matrixA = a

m = len(matrixA)
i = 0

while i < n-1:
    x = matrixA[i][i]
    y = matrixA[i+1][i]
    mtp= y/x
    matrixA[i+1] = matrixA[i+1] - mtp*matrixA[i]
    i = i+1
    pass

print(matrixA)
```

For the Quantum dots in three dimensions, two electrons. We need to add the potential $w^2p^2$+1/p to the diagonal elements, so the code change to this bellow:



```
import numpy as np
import time

n = int(input('choose n for matrix:'))
a1 = np.eye(n,k=0)
a2 = np.eye(n,k=1)
a3 = np.eye(n,k=-1)

w = 0.5
p = input()
x = input()
y = input()+(w*p)**2+1/p
z = input()

a = x*a1+y*a2+z*a3
matrixA = a
```

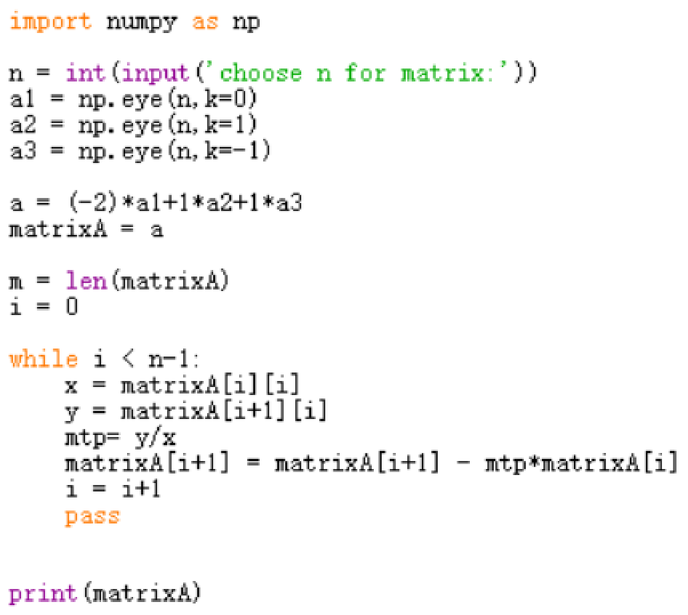
## 3. Implementation

Project a)

For question one, this is a linear algebra problem. That is, for a certain matrix A, it is not diagonalized, but we need to diagonalize it. We only need to first find the eigenvectors, orthogonalize the eigenvectors to obtain a new matrix S, and then use the calculation method of SAS^-1 to obtain the orthogonalization of the matrix A. The orthogonalized matrix diagonals The element is its eigenvector. So we can prove Wi = UVi,

Project b)

For question two, the problem is to simplify the non-diagonal elements of a matrix. Here I used python's numpy.eye function to create a non-diagonal matrix. For a matrix of N*N, it takes N operations to be matrixed into a diagonal matrix. The code is shown as follow:



```
import numpy as np

n = int(input('choose n for matrix:'))
a1 = np.eye(n,k=0)
a2 = np.eye(n,k=1)
a3 = np.eye(n,k=-1)

a = (-2)*a1+1*a2+1*a3
matrixA = a

m = len(matrixA)
i = 0

while i < n-1:
    x = matrixA[i][i]
    y = matrixA[i+1][i]
    mtp= y/x
    matrixA[i+1] = matrixA[i+1] - mtp*matrixA[i]
    i = i+1
    pass


print(matrixA)
```

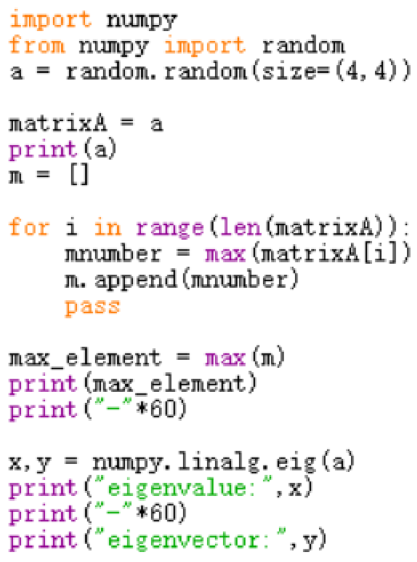
As can be seen from the above code, it is necessary to perform N calculations for simplifying an N*N matrix into a diagonal matrix.

Project2 c)

In this problem, we need to find the eigenvectors and eigenvalues of a particular matrix.

Existing python tools can help us achieve this. First, you can use numpy's random function to create a matrix of arbitrary order and arbitrary elements as the object of our test. Then we can use the numpy.linalg.eig function to get its eigenvectors and eigenvalues. Also, we can use the max function of python to find the largest nondiagonal element.

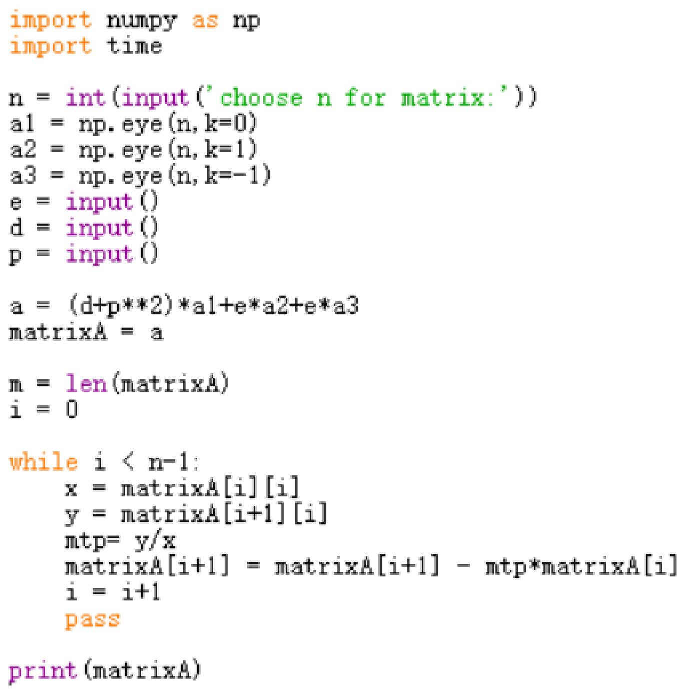
The code is shown as follow:



```
import numpy
from numpy import random
a = random.random(size=(4,4))

matrixA = a
print(a)
m = []

for i in range(len(matrixA)):
    mnumber = max(matrixA[i])
    m.append(mnumber)
    pass

max_element = max(m)
print(max_element)
print("-"*60)

x,y = numpy.linalg.eig(a)
print("eigenvalue:",x)
print("-"*60)
print("eigenvector:",y)
```

Project d):

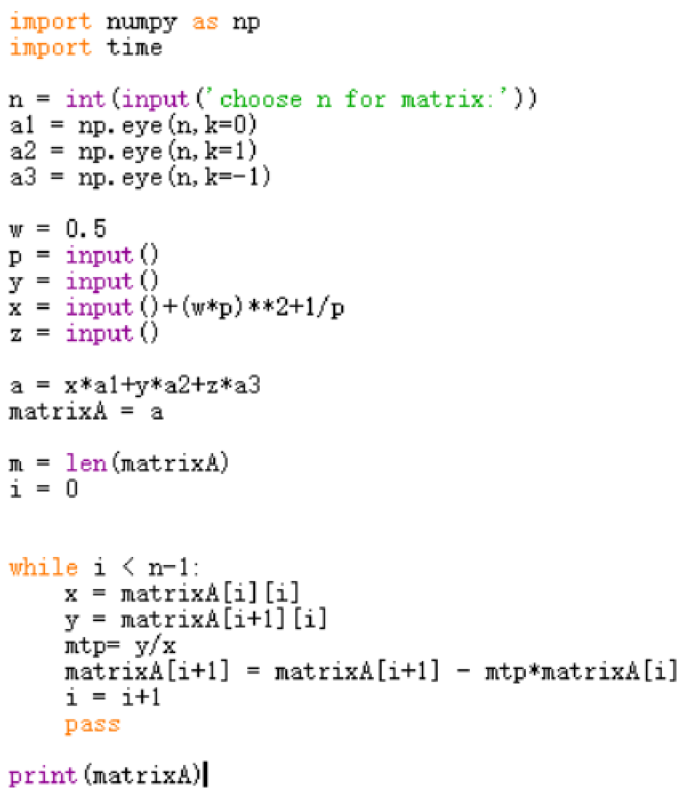
For question four, the problem is also to simplify the non-diagonal elements of a matrix, but now we need to add $p^2$ to the diagonal elements. Here I used python's numpy.eye function to create a non-diagonal matrix. Then multiply the diagonal matrix for $p^2$ times and plus it to original matrix. For a matrix of N*N, it takes N operations to be matrixed into a diagonal matrix. The code is shown as follow:



```
import numpy as np
import time

n = int(input('choose n for matrix:'))
a1 = np.eye(n,k=0)
a2 = np.eye(n,k=1)
a3 = np.eye(n,k=-1)
e = input()
d = input()
p = input()

a = (d+p**2)*a1+e*a2+e*a3
matrixA = a

m = len(matrixA)
i = 0

while i < n-1:
    x = matrixA[i][i]
    y = matrixA[i+1][i]
    mtp= y/x
    matrixA[i+1] = matrixA[i+1] - mtp*matrixA[i]
    i = i+1
    pass

print(matrixA)
```

Project e)

The question is kind of similar to question above, but now we need to add the element $(wp)^2$+1/p to each diagonal element, so the code change to bellow, and through this we can calculate quantum dots in three dimensions, two electrons. And we need to set the parameter w and l to get the ground state's results.



```
import numpy as np
import time

n = int(input('choose n for matrix:'))
a1 = np.eye(n,k=0)
a2 = np.eye(n,k=1)
a3 = np.eye(n,k=-1)

w = 0.5
p = input()
y = input()
x = input()+(w*p)**2+1/p
z = input()

a = x*a1+y*a2+z*a3
matrixA = a

m = len(matrixA)
i = 0


while i < n-1:
    x = matrixA[i][i]
    y = matrixA[i+1][i]
    mtp= y/x
    matrixA[i+1] = matrixA[i+1] - mtp*matrixA[i]
    i = i+1
    pass

print(matrixA)
```

## 4. Results

We have obtained several sets of matrices, and we can get their orthogonal form by mathematical verification of linear algebra. In addition, according to the code, we have realized the following functions. First, we have obtained a simplification method for non-diagonal matrices, which can be obtained by N operations for the reduced diagonal matrix. Second, we obtain the calculated matrix eigenvectors. And the code of the eigenvalues, so that we can quickly get the eigenvectors and eigenvalues of the matrix to calculate the orthogonalization of the matrix. Thirdly, by adding the diagonal element p^2, we can obtain the solution of the three-dimensional single-electron oscillator potential Schrödinger equation in the linearization of linear algebra; fourth, by adding the diagonal element (wp)^2 + 1/ p, we obtain the solution of the three-dimensional two-electron oscillator potential Schrödinger equation after matrix simplification of linear algebra.

## 5. Concluding Remarks

With important samples and exponential distribution, the results are good and accurate. Using these methods, we can get the solution to our project. But the speed of Python is still not as quick as C++, so the memory and velocity is very limited, which may cause some errors or shortages.

## 6. References

Hjort-Jensen,M., 2015. Computational physics.

Theprogramcan be found in https://github.com/lch172061365/Computational-Physics.git