Jacobi's Rotation Algorithm to Solve eigen value problems-buckling beam and Schrödinger one electron equation

Jeyalakshmi Thoppe Subrmanian

September 2019

1 Abstract

Abstract

In this project we study the Jacobi rotation algorithm to solve eigen value problems first for a buckling beam supported on both ends and then for one electron in the Schrödinger equation.The eigen values for buckling beam problem is compared with the analytical solution as the matrix is a tridiagnal matrix and found matching. The results for the Schrödinger wave equation is compared with the results using builtin Linalg results in numpy and found matching. The jacobi rotation algorithm is found very time consuming and time for different discretion points (n= 200) and for maximum discretion limit is studied . The Jacobi rotation algorithm is found to be very slow with increasing number of discretion points. This can be addressed with parallelization methods to some extend. Git hub link for the source code : `https://github.com/jeyalakt/4150COMP_PHY`

2 Introduction

We start with the with the two point boundary value problem of a buckling beam fastened at both ends which is a wave function in one dimension. By approximating the second derivative with a three point formula, the problem is shown to be equivalent to an eigenvalue problem for a tridiagonal matrix. This is implemented by the use of Jacobi's rotation algorithm first, and then compared with the standardized algorithms in the Numpy library. The three lowest energy eigenvalues and eigenvectors are found and compared to exact results. Then the problem is extended to involve one electron in a harmonic oscillator potential of Schrödinger wave equation.

3 Theory

3.1 The Buckling Beam Problem

From the bucking beam problem we have the following equation:

$$\gamma \frac{d^2u(x)}{dx^2} = -Fu(x)$$

where $u(x)$ is the displacment in Y direction We also have the Dirichlet boundary condition which is $u(0) = 0$ and $u(L) = 0$ for x $\in [0, L]$.
Here L is the beam length and F is the force that apply at the (L,0) toward the origin and γ is a constant that define by properties like rigidity of the beam.
We change the scale of the equation by defining dimensional variable ρ where ρ $\in [0, 1]$

$$\rho = \frac{x}{L}$$

Now by reordering the equation , we have the following equation:

$$\frac{d^2u(\rho)}{d\rho^2} = -\frac{FL^2}{R}u(\rho) = -\lambda u(\rho)$$

With discretizing the equation the problem becomes an eigenvalue problem. Now we can use the expression for second derivative :

$$u'' = \frac{u(\rho + h) - 2u(\rho) + u(\rho - h)}{h^2} + O(h^2), \tag{1}$$

where h is our step length. And we also should define the maximum and minimum value for the variable ρ, ρ_{min} and ρ_{max}. We consider $\rho_{min} = 0$ and $\rho_{max} = 1$. Now by considering $\rho_{min} = \rho_0$ and $\rho_{max} = \rho_N$ and with given number of mesh point N we can define step length as:

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

And we can calculate ρ at the different point of i by using the following relation:

$$\rho_i = \rho_0 + ih \qquad i = 1, 2, \dots, N$$

Then we rewrite the differential equation with ρ_i:

$$-\frac{u(\rho_i + h) - 2u(\rho_i) + u(\rho_i - h)}{h^2} = \lambda u(\rho_i)$$

This equation also can be written as:

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

Now we can write this equation in the form of tridiagonal matrix for different value of i:

$$\begin{bmatrix} d & e & 0 & 0 & \dots & 0 & 0 \\ e & d & e & 0 & \dots & 0 & 0 \\ 0 & e & d & e & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & e & d & e \\ 0 & \dots & \dots & \dots & \dots & e & d \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} \tag{2}$$

As we can see the boundary condition u_0 and u_N are not included in the matrix. Now we have the diagonal element $d = \frac{-2}{h^2}$ and non diagonal elements $e = \frac{1}{h^2}$. The eigen pairs are calculated by Jacobi rotaiton algorith and eigen values are compared with analytical values using the following equation:

$$\lambda_j = d + 2a\cos\left(\frac{j\pi}{N+1}\right) \qquad j = 1, 2, \dots N-1 \tag{3}$$

3.2 Schrödinger equation for one electron

The radial part of Schrödinger equation for one electron is given by .

$$-\frac{\hbar^2}{2m}\left(\frac{1}{r^2}\frac{d}{dr}r^2 - \frac{l(l+1)}{r^2}\right)R(r) + V(r)R(r) = ER(r) \tag{4}$$

In above equation $V(r) = \frac{1}{2}kr^2$ is harmonic oscillator potential and $k = m\omega^2$ and E is the energy of harmonic oscillator in three dimension. ω is the oscillator frequency. Energies are calculated by the following equation for $n = 0, 1, 2..$ and $l = 0, 1, 2...$ n and l are energy quantum number and orbital momentum quantum number respectively.

$$E_{nl} = \hbar\omega\left(2n + l + \frac{3}{2}\right)$$

Then we assume that $R(r) = (1/r)u(r)$ and rewrite the equation 4:

$$-\frac{\hbar^2}{2m}\frac{d^2}{dr^2}u(r) + \left(V(r) + \frac{l(l+1)}{r^2} - \frac{\hbar^2}{2m}\right)u(r) = Eu(r) \tag{5}$$

The boundary condition for this equation are $u(0) = 0$ and $u(\infty) = 0$.
Now we define a dimensionless variable $\rho = (1/\alpha)r$ where α is a constant and then rewrite the equation:

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho) + \left(V(\rho) + \frac{l(l+1)}{\rho^2}\frac{\hbar^2}{2m\alpha^2}\right)u(\rho) = Eu(\rho) \tag{6}$$

In this project for easier calculation we set $l = 0$ and substitute $V(\rho) = \frac{1}{2}k\alpha^2\rho^2$ in equation (6):

$$-\frac{\hbar^2}{2m\alpha^2}\frac{d^2}{d\rho^2}u(\rho) + \frac{1}{2}k\alpha^2\rho^2 u(\rho) = Eu(\rho) \tag{7}$$

Now we multiply the both side of equation(7) by $2m\alpha^2\rho^2/\hbar^2$ and define some constant as below:

$$\frac{mk}{\hbar^2}\alpha^4 = 1 \qquad \lambda = \frac{2m\alpha^2}{\hbar^2}E$$

Then we can rewrite the Schrödinger equation as:

$$-\frac{d^2}{d\rho^2}u(\rho) + \rho^2 u(\rho) = \lambda u(\rho) \tag{8}$$

Just like the previous section we can solve this equation by discretizing the equation. First we define the minimum and maximum value for the ρ,$\rho_{min} = 0$ and ρ_{max}. Since we cannot set $\rho_{max} = \infty$ we can solve the problem for different value of ρ_{max}. By setting $\rho_{min} = \rho_0$ and $\rho_{max} = \rho_N$ and having the given number of mesh point N, the step length can define as:

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

The value of ρ for different value of i:

$$\rho_i = \rho_0 + ih \qquad i = 1, 2, \ldots, N$$

Now we can rewrite the Schrödinger equation for value of ρ_i:

$$-\frac{u(\rho_i + h) - 2u(\rho_i) + u(\rho_i - h)}{h^2} + \rho_i^2 u(\rho_i) = \lambda u(\rho_i)$$

Or it can rewrite in more simple way:

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

In the above equation $V_i = \rho_i^2$ is the harmonic oscillator potential.
We define the diagonal and non-diagonal matrix element as $d_i = \frac{2}{h^2} + V_i$ and $e_i = \frac{1}{h^2}$ respectively. Now by using these simplification the Schrödinger equation can be writen as:

$$d_i u_i + e_{i-1}u_{i-1} + e_{i+1}u_{i+1} = \lambda u_i$$

Now we can write this equation as matrix eigenvalue problem:

$$\begin{bmatrix} d_0 & e_0 & 0 & 0 & \ldots & 0 & 0 \\ e_1 & d_1 & e_1 & 0 & \ldots & 0 & 0 \\ 0 & e_2 & d_2 & e_2 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & \ldots & e_{N-1} & d_{N-1} & e_{N-1} \\ 0 & \ldots & \ldots & \ldots & \ldots & & d_N \end{bmatrix} \begin{bmatrix} u_0 \\ u_2 \\ u_3 \\ \ldots \\ u_{N-1} \\ u_N \end{bmatrix} = \lambda \begin{bmatrix} u_0 \\ u_2 \\ u_3 \\ \ldots \\ u_{N-1} \\ u_N \end{bmatrix} \tag{9}$$

4 Jacobi eigenvalue algorithm

Eigenvalue problem is given as

$$Ax = \lambda x$$

. Jacobi's rotation algorithm applies a series of similarity transformations to both sides of this equation while preserving the eigenvalues and the initially chosen orthogonality of eigenvectors. An orthogonal transformation matrix is represented of the form:

$$S = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \dots & c & \dots & -s & \dots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \dots & s & \dots & c & \dots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 & \dots & 1 \end{bmatrix} \tag{10}$$

where $c = cos\theta$, $s = sin\theta$ appear intersection of i th and j th rows and columns.

More specifically we will apply S with $S^T = S^{-1}$ as an Euclidean rotation matrix around one of the basic axes until the non-diagonal elements of A have been transformed away, with only the eigenvalues on the diagonal. Say that the axis of rotation is orthogonal to the plane spanned by e_k and e_l, where $e_\{i\}$ denotes the Euclidean set of basis vectors.

We can then take the non-zero elements of S to be parametrized by an angle θ: $S_{kk} = S_{ll} = cos\theta$, $S_{kl} = -S_{lk} = -sin\theta$ and $S_{ii} = 1$ for $i \neq k$ and $i \neq l$. Then the iterative process B=$S^T AS$is applied until the Frobenius norm becomes smaller than some tolerance ϵ. The Frobenius norm is defined as $|A| = \sqrt{\sum_i^n \sum_j^n a_{ij}^2}$ where i$\neq$ j

We define $\tan\theta = t = \frac{s}{c}$, $\sin\theta = s$ and $\cos\theta = c$.

$$\cot 2\theta = \tau = \frac{a_{ll} - a_{kk}}{2a_{kl}}.$$

Now we can define the angel of θ in order that the non-diagonal element of the matrix become zero. Given the facts $\cot 2\theta = 1/2(\cot\theta - \tan\theta)$ and $\cot\theta = \frac{1}{\tan\theta}$ and since $\tan\theta = t$:

$$\tau = 1/2(\frac{1}{t} - t) \qquad \Rightarrow \qquad t^2 + 2\tau t - 1 = 0$$

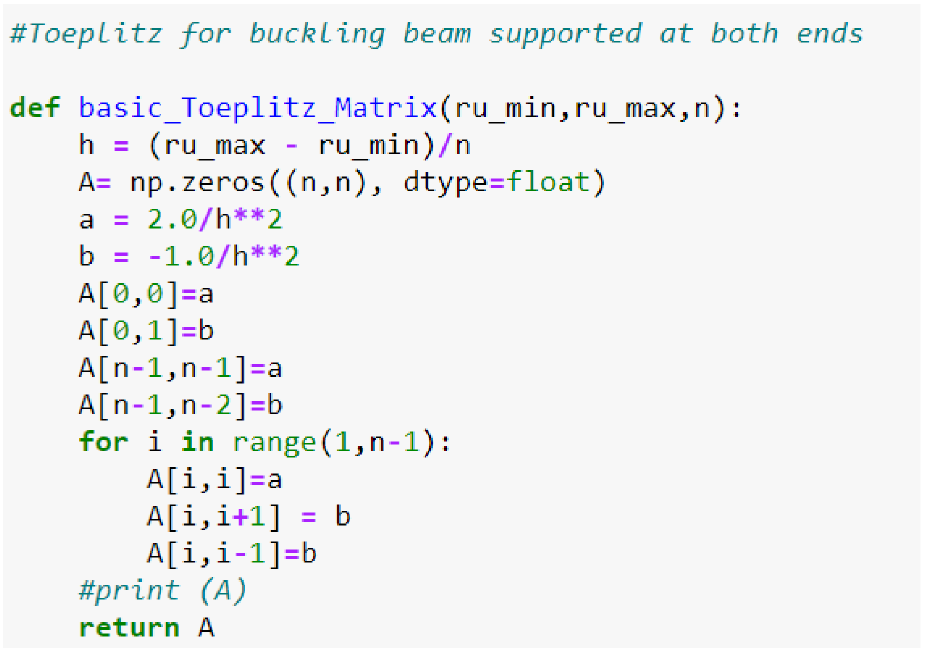
Solving above equation:

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

Now the c and s obtain easily:

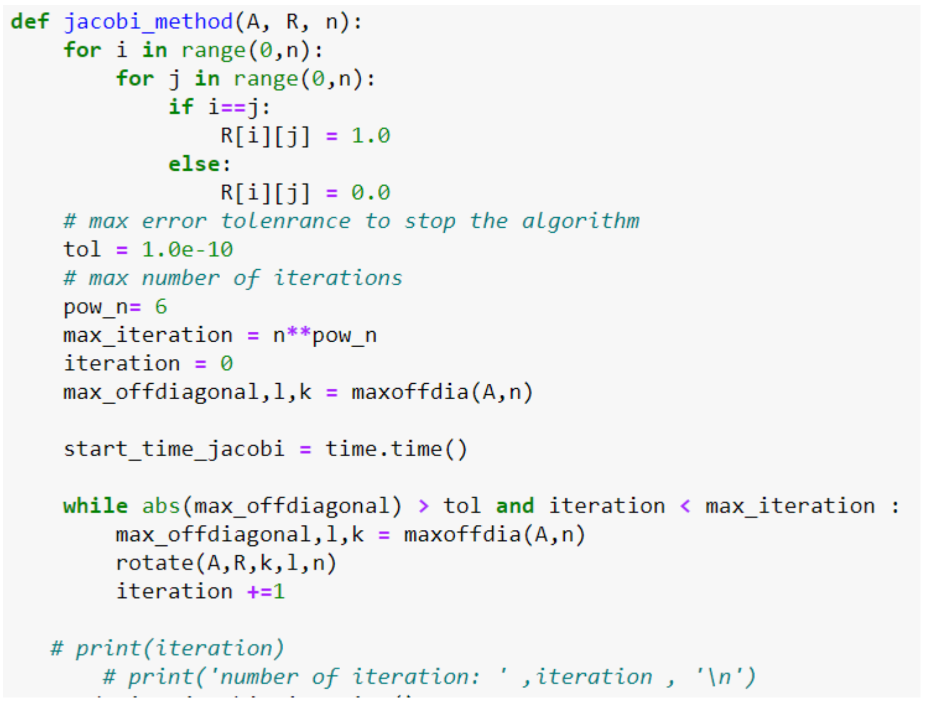
$$c = \frac{1}{\sqrt{1+t^2}}, \qquad s = tc$$

B has: $b_{ii} = a_{ii}$, $i \neq k$, $i \neq l$
$b_{ik} = a_{ii}c\theta - a_{il}s\theta$, $i \neq k$, $i \neq l$
$b_{il} = a_{il}c\theta + a_{ik}s\theta$, $i \neq k$, $i \neq l$
$b_{kl} = sc(a_{kk} - a_{ll}) + (c^2 - s^2)a_{kl} +$
$b_{kk} = c^2 a_{kk} - 2sca_{kl} + s^2 a_{ll}$
$b_{ll} = c^2 a_{ll} + 2sca_{kl} + s^2 a_{kk}$
where $c = cos\theta$, $s = sin\theta$ The following figures shows creation on Toeplitz matrix and implementation of the Jacobi's rotational algorithm in Python



```
#Toeplitz for buckling beam supported at both ends

def basic_Toeplitz_Matrix(ru_min,ru_max,n):
    h = (ru_max - ru_min)/n
    A= np.zeros((n,n), dtype=float)
    a = 2.0/h**2
    b = -1.0/h**2
    A[0,0]=a
    A[0,1]=b
    A[n-1,n-1]=a
    A[n-1,n-2]=b
    for i in range(1,n-1):
        A[i,i]=a
        A[i,i+1] = b
        A[i,i-1]=b
    #print (A)
    return A
```

Figure 1: Toeplitz matrix for buckling beam supported at both ends



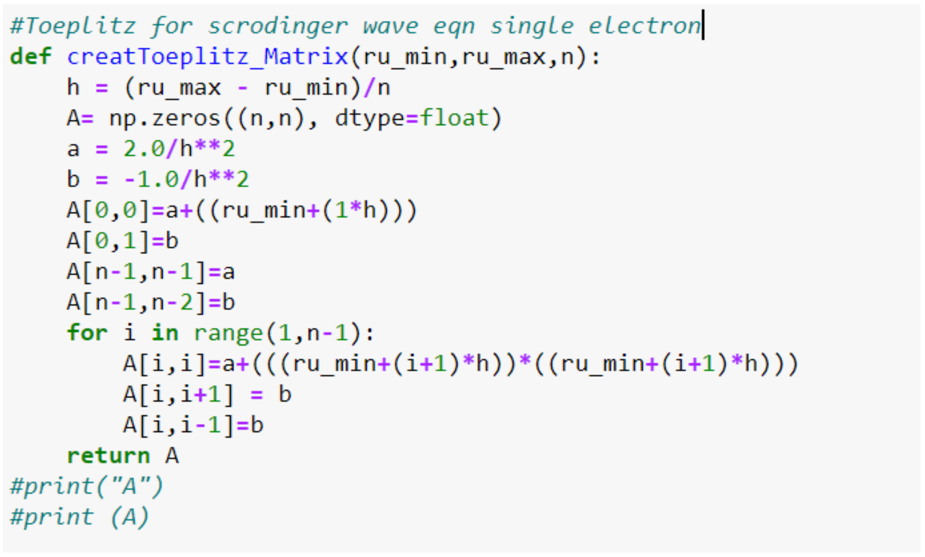
```
def jacobi_method(A, R, n):
    for i in range(0,n):
        for j in range(0,n):
            if i==j:
                R[i][j] = 1.0
            else:
                R[i][j] = 0.0
    # max error tolenrance to stop the algorithm
    tol = 1.0e-10
    # max number of iterations
    pow_n= 6
    max_iteration = n**pow_n
    iteration = 0
    max_offdiagonal,l,k = maxoffdia(A,n)

    start_time_jacobi = time.time()

    while abs(max_offdiagonal) > tol and iteration < max_iteration :
        max_offdiagonal,l,k = maxoffdia(A,n)
        rotate(A,R,k,l,n)
        iteration +=1

   # print(iteration)
       # print('number of iteration: ' ,iteration , '\n')
```

Figure 2: Jacobis rotation algorithm

4.1 adaptation for Schrödinger equation for one electron

The jacobi rotation algorithm is further extended to solve the Schrödinger equation for one electron with only adaptation to set the Toeplitz Matrix The matrix is set as follows:

$$\begin{bmatrix} \frac{2}{h^2}+V_1 & \frac{-1}{h^2} & 0 & 0 & \dots & 0 & 0 \\ \frac{-1}{h^2} & \frac{2}{h^2}+V_2 & \frac{-1}{h^2} & 0 & \dots & 0 & 0 \\ 0 & \frac{-1}{h^2} & \frac{2}{h^2}+V_3 & \frac{-1}{h^2} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \frac{-1}{h^2} & \frac{2}{h^2}+V_{nstep-2} & \frac{-1}{h^2} \\ 0 & \dots & \dots & \dots & \dots & & \frac{2}{h^2}+V_{nstep-1} \end{bmatrix} \tag{11}$$



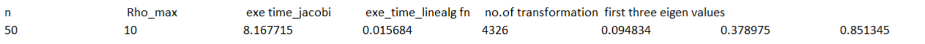
```python
#Toeplitz for scrodinger wave eqn single electron
def creatToeplitz_Matrix(ru_min,ru_max,n):
    h = (ru_max - ru_min)/n
    A= np.zeros((n,n), dtype=float)
    a = 2.0/h**2
    b = -1.0/h**2
    A[0,0]=a+((ru_min+(1*h)))
    A[0,1]=b
    A[n-1,n-1]=a
    A[n-1,n-2]=b
    for i in range(1,n-1):
        A[i,i]=a+(((ru_min+(i+1)*h))*((ru_min+(i+1)*h)))
        A[i,i+1] = b
        A[i,i-1]=b
    return A
#print("A")
#print (A)
```

Figure 3: Toeplitz Matrix for scrodinger equation for single electron

The Jacobi eigenvalue method repeatedly performs rotation until the matrix becomes almost diagonal. Then the diagonal elements are approximations of the eigenvalues of S.

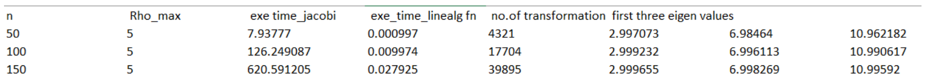
5 Results and Discussion

Jacobi's rotation algorithm is studied for different Rho-max values and for different discretisation values and the results are given as below:



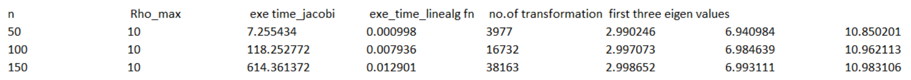
n	Rho_max	exe time_jacobi	exe_time_linealg fn	no.of transformation	first three eigen values		
50	10	8.167715	0.015684	4326	0.094834	0.378975	0.851345

Figure 4: Jacobi's algorithm results for buckling beam problem - n 50 and Rho-max 10



n	Rho_max	exe time_jacobi	exe_time_linealg fn	no.of transformation	first three eigen values		
50	5	7.93777	0.000997	4321	2.997073	6.98464	10.962182
100	5	126.249087	0.009974	17704	2.999232	6.996113	10.990617
150	5	620.591205	0.027925	39895	2.999655	6.998269	10.99592

Figure 5: Jacobi's algorithm results for scrodinger equation for single electron - n til 150 and Rho-max 5



n	Rho_max	exe time_jacobi	exe_time_linealg fn	no.of transformation	first three eigen values		
50	10	7.255434	0.000998	3977	2.990246	6.940984	10.850201
100	10	118.252772	0.007936	16732	2.997073	6.984639	10.962113
150	10	614.361372	0.012901	38163	2.998652	6.993111	10.983106

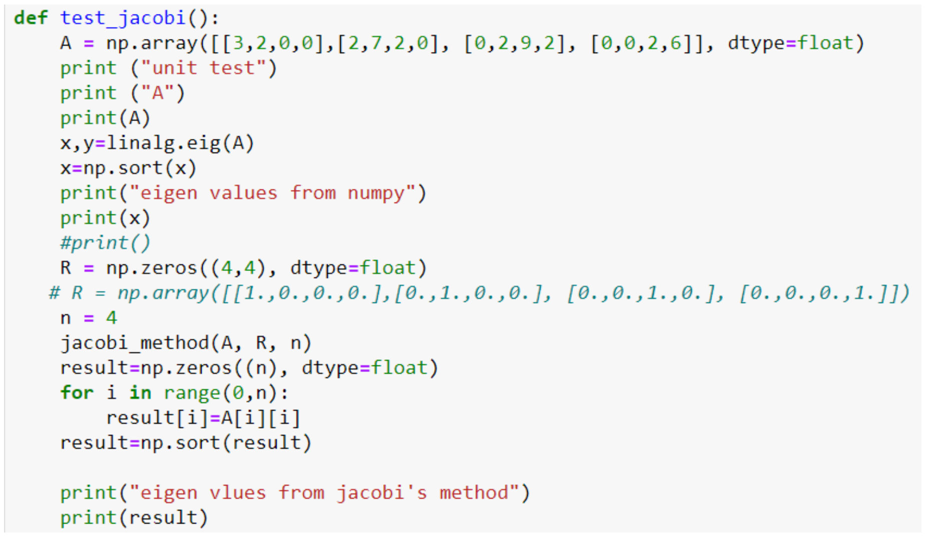
Figure 6: Jacobi's algorithm results for scrodinger equation for single electron - n til 150 and Rho-max 10

The results are provided under project 2 of git repositary.

Jacobi's rotation method is implemented to find eigen values for the bucking beam problem and found to be same with the analytical solution.Similarly Schrödinger equation for one electron is solved for the eigenvalues and found to be same as of built in Numpy function linalg.eig it is observed that for getting first three eigen values close to the approximation of 4 digits , number og discretion points should be at least n =200 at Rho-max is 10. Even though Jacobi's rotation algorithm gives correct solutions , it consumes lot of time as shown , hence practically not usable for n > 250 .Parallelisation techniques can be used to speed up the execution of the algorithm to certain extend. So other efficient algorithms as Lanzcos algorithm need to be studied and used .

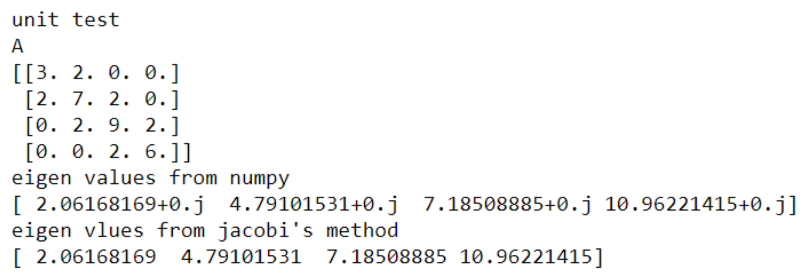
6 Unit Test

In this section i have implemented a unit test for the buckling beam problem using Jacobi's rotation algorithm as follows and the results are found to be same as from numpy fucntion



```
def test_jacobi():
    A = np.array([[3,2,0,0],[2,7,2,0], [0,2,9,2], [0,0,2,6]], dtype=float)
    print ("unit test")
    print ("A")
    print(A)
    x,y=linalg.eig(A)
    x=np.sort(x)
    print("eigen values from numpy")
    print(x)
    #print()
    R = np.zeros((4,4), dtype=float)
   # R = np.array([[1.,0.,0.,0.],[0.,1.,0.,0.], [0.,0.,1.,0.], [0.,0.,0.,1.]])
   n = 4
    jacobi_method(A, R, n)
    result=np.zeros((n), dtype=float)
    for i in range(0,n):
        result[i]=A[i][i]
    result=np.sort(result)

    print("eigen vlues from jacobi's method")
    print(result)
```

Figure 7: unit test for Jacobi's algorithm -buckling beam problem



```
unit test
A
[[3. 2. 0. 0.]
 [2. 7. 2. 0.]
 [0. 2. 9. 2.]
 [0. 0. 2. 6.]]
eigen values from numpy
[ 2.06168169+0.j  4.79101531+0.j  7.18508885+0.j 10.96221415+0.j]
eigen vlues from jacobi's method
[ 2.06168169  4.79101531  7.18508885 10.96221415]
```

Figure 8: Result of unit test for Jacobi's algorithm -buckling beam problem

7 Conclusion

The Jacobi rotation method or eigen value problem is very slow method to find eigenvalues compared to the built in method in the python `numpy` library. For systems where a relatively small number of grid points n is sufficient, the Jacobi method may be the preferred method.