


Time Dependent Quantum Chemistry
Professor. Atanu Bhattacharya
Department of Inorganic and Physical Chemistry
Indian Institute of Science Bengaluru
Module 04 Lecture 31
Python Tutorial 4 (Eigenvalue and Eigenfunction)

Welcome back to module 4 Python tutorial 4 of this course Time Dependent Quantum Chemistry. In this module, we have -- In this tutorial, we have learned how to represent a matrix and then how to get them eigenvalue, eigenvectors. And Python has a particular way of presenting eigenvalues and eigenvector and we have to collect that information specifically. So, we look at it how to collect it.

(Refer Slide Time: 00:52)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors



Steps to Find Eigenvalues and Eigenvectors

Example 1: Find eigenvalues and eigenvectors of $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

(c) Step 3: Finding Eigenvalues and Eigenvectors

```

from scipy import zeros
from scipy.linalg import eig
A=zeros((2,2))
A[0,0]=2
A[0,1]=1
A[1,0]=1
A[1,1]=2
E,V=eig(A)
print(E[0])
print(V[:,0])
print(E[1])
print(V[:,1])

(3+0j)
[0.70710678 0.70710678]

(1+0j)
[-0.70710678 0.70710678]
```

Time dependent Quantum Chemistry

```

C:\Program Files (x86)\Python38-32>python test1.py
[[0. 0.]
 [0. 0.]]

C:\Program Files (x86)\Python38-32>python test1.py
[[2. 1.]
 [1. 2.]]


C:\Program Files (x86)\Python38-32>python test1.py
[3.+0.j 1.+0.j]
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

C:\Program Files (x86)\Python38-32>python test1.py
(3+0j)
[0.70710678 0.70710678]
(1+0j)
[-0.70710678 0.70710678]

C:\Program Files (x86)\Python38-32>
```

```

File Edit Format View Help
#zeros functionality is imported from scipy module
from scipy import zeros
from scipy.linalg import eig
# a(2x2) null matrix is constructed
A=zeros((2,2))
A[0,0]=2
A[0,1]=1
A[1,0]=1
A[1,1]=2
E,V=eig(A)
print(E[0])
print(V[:,0])
print(E[1])
print(V[:,1])
```



Python will be clubbed every eigenvalue and eigenvectors in a particular arrangement and if we want to collect it the way it is presented here is that this particular eigenvalue is corresponding to the first column of this matrix, this eigenvalue matrix, V matrix. And the second eigenvalue corresponds to the column of this second column of this matrix.

So, we will check it how to get that we will first write down specifically instead of printing E will just print E[0], the first eigenvalue. And corresponding V, if we want to corresponding eigenvector, if you want to print, then it is going to be colon comma 0, so entire column has to be printed. Similarly, there are two eigenvalues we have.

So, second eigenvalue is going to be, it is going to be one because index starts from 0 and the corresponding eigenvector is going to be now 1, that is the way we can pull up the information specifically. And if we run this program, we get the following results. Corresponding to the eigenvalue three, I have this eigenvector 0.7, 0.7. And this is going to be the column vector actually. So, I will go back to the slide right now.

(Refer Slide Time: 02:47)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 1: Find eigenvalues and eigenvectors of $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \Rightarrow \begin{pmatrix} 3 \\ 1 \end{pmatrix}$

(c) Step 3: Finding Eigenvalues and Eigenvectors

```

from scipy import zeros
from scipy.linalg import eig
A=zeros((2,2))
A[0,0]=2
A[0,1]=1
A[1,0]=1
A[1,1]=2
E,V=eig(A)
print(E[0])
print(V[:,0])
print(E[1])
print(V[:,1])

```

Handwritten notes:

- $3 \Rightarrow \begin{pmatrix} 0.7 \\ 0.7 \end{pmatrix}$ (right normalized eigenvector)
- $1 \Rightarrow \begin{pmatrix} -0.7 \\ 0.7 \end{pmatrix}$ (Column matrix)
- Matrix: $\begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$

Output from code:

```

(3+0j)
[0.70710678 0.70710678]
(1+0j)
[-0.70710678 0.70710678]

```

Time dependent Quantum Chemistry

So, corresponding to the eigenvalue 3, I have now this eigenvector, the right normalized eigenvector. Right normalized eigenvector is nothing but a column matrix, column matrix is represented by 0.7 by 0.7. And corresponding to this eigenvalue I have another column matrix which is minus 0.7 and 0.7. So, these are two matrices we have, the eigenvectors we have.

$$3 \rightarrow \begin{bmatrix} 0.7 \\ 0.7 \end{bmatrix}$$

$$1 \rightarrow \begin{bmatrix} -0.7 \\ 0.7 \end{bmatrix}$$

And if we compare these two eigenvectors, we will see that the same results we have got, basically we represent -- When indeed analytically, we have got the eigenvectors as square root 1 by square root which is nothing but the value which we have got here. And another eigenvector was my minus 1 by square root of 2, 1 by square root of 2. And that is exactly what we have found here.

$$\begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \leftarrow \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \rightarrow \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

So, numerically we get the same result, only one point to be remembered here is that the way this array has been represented, it looks like a row matrix. This is just a visualization that is the way Python will print. Python is printing like a row does not mean that it is a row matrix, it is actually our column matrix, we have to remember that. So, it is just a visualization and one can change the visualization by changing certain, by giving certain command in this construct.

We will just skip that because that details we do not need immediately. But we will remember that this is just a printing issue of Python. The Python will print it like this way, but it is actually this E V, this construct will give me a only right normalized eigenvector that we have to remember, it is going to be always right normalized eigenvector. Right normalized eigenvector, it is nothing but a column matrix like this.

So, corresponding to a particular eigenvalue how to get the eigenvector that we have already found. And if we have these results, then one can actually go ahead and test, one can now test the properties of these eigenvectors.

(Refer Slide Time: 05:55)



Steps to Find Eigenvalues and Eigenvectors

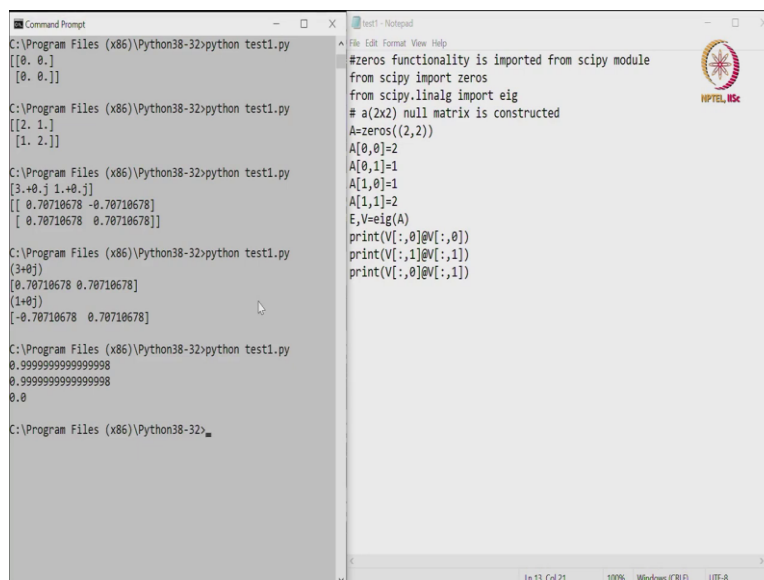
Example 1: Find eigenvalues and eigenvectors of $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

Testing Properties of the Eigenvectors

```

from scipy import zeros
from scipy.linalg import eig
A=zeros((2,2))
A[0,0]=2
A[0,1]=1
A[1,0]=1
A[1,1]=2
E,V=eig(A)
print(V[:,0]@V[:,0]) #inner product between eigenvectors V[:,0] and V[:,0]
print(V[:,1]@V[:,1]) #inner product between eigenvectors V[:,1] and V[:,1]
print(V[:,0]@V[:,1]) #inner product between eigenvectors V[:,0] and V[:,1]
    
```

Handwritten notes:
 - eig()
 - $A * B$
 - $A @ A$
 - $\begin{matrix} \downarrow & \downarrow \\ 1 & 1 \\ \downarrow & \downarrow \\ 1 & 1 \\ \downarrow & \downarrow \\ 0 & 0 \end{matrix}$



Properties such as we know that if I have eigenvectors, there are two eigenvectors if I have then inner product with itself because it is normalized it should give me 1, inner product with the other eigenvector, is going to be an orthonormal, each vector would be orthonormal. So, if I take the inner product with itself, it is going to be 1. So, both would be 1, this one also should be 1.

But if I take the inner product with the other vector, it is going to be 0 because it is orthogonal to each other. So, let us prove that we have situations like this and to get the inner product is a very simple way to get the inner product. Inner product can be used with this at the rate sign, this is the vectorial product. So, in general scalar product is used by this star.

So, A multiplied by B this is the scalar product, but if I want to take a vectorial product, it is going to be, the inner product is going to be A at the rate sign B or A. So, we will take a look

at it whether these vectors which you have created already, they follow the properties which we are familiar with, they should be orthonormal.

So, I will be able to take the inner product as follows, I have this eigenvector and if I multiply this eigenvector vector product inner product, so then I can also take the inner product for the other eigenvector and I can take them between two eigenvectors. So, if we do that, then I get back following information. I will go back to the slide. I have the first inner product between these two eigenvectors giving me 1, it is close to 1.

So, this is a numerical value and it means that this vector is normalized. On the other hand, if I take the other vector inner product that is also normalized. And if we take the inner product between two vectors, then what I get is 0 which means orthonormal. So, what we are getting is this eig functionality is giving me ortho normal right eigenvectors corresponding to a particular eigenvalue. We have proved this one.

(Refer Slide Time: 08:50)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 2: Dealing with Tridiagonal Matrix

(a) Automated Construction of Matrix

```

from scipy import zeros, arange
A=zeros((3,3))
for i in arange(3):
    A[i,i]=2
for i in arange(2):
    A[i,i+1]=-1
    A[i+1,i]=-1
print(A)
[[ 2. -1.  0.]
 [-1.  2. -1.]
 [ 0. -1.  2.]]

```

Time dependent Quantum Chemistry



Steps to Find Eigenvalues and Eigenvectors

Example 1: Find eigenvalues and eigenvectors of $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

Testing Properties of the Eigenvectors

```

from scipy import zeros
from scipy.linalg import eig
→ A=zeros((2,2))
A[0,0]=2
A[0,1]=1
A[1,0]=1
A[1,1]=2
E,V=eig(A)
print(V[:,0]@ V[:,0]) #inner product between eigenvectors V[:,0] and V[:,0]
print(V[:,1]@ V[:,1]) #inner product between eigenvectors V[:,1] and V[:,1]
print(V[:,0]@ V[:,1]) #inner product between eigenvectors V[:,0] and V[:,1]

```

Handwritten notes: eig(), $A * B$, $A @ A$, $\begin{matrix} 1 \\ 1 \\ 0 \end{matrix}$

We will move on and we have already seen the general procedure to construct the matrix. Now, we will move to this tridiagonal matrix. Tridiagonal matrix is something which we will be dealing with in quantum mechanics very frequently, because the kinetic energy part adopts this tridiagonal form. So, if it is adopting tridiagonal form, we have to learn how to get that eigenvalue eigenvector of the tridiagonal matrix.

$$\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

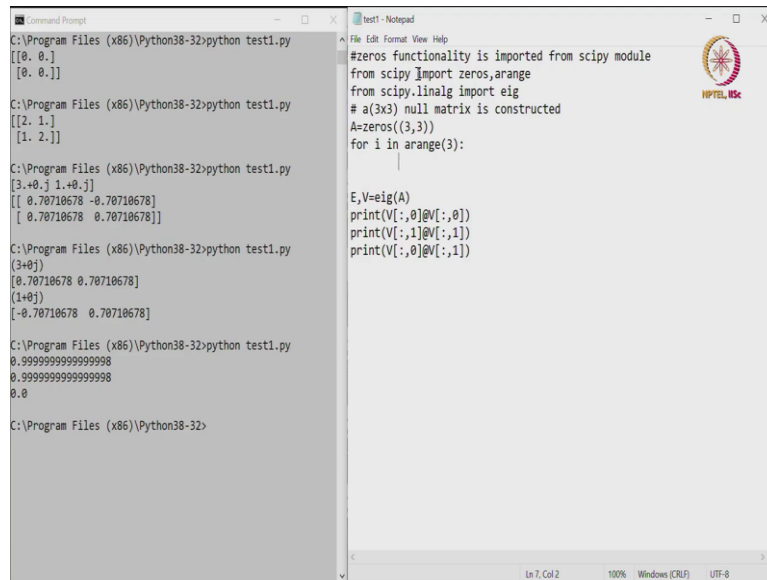
Now, if you look at the tridiagonal matrix, something is very interesting to note its diagonal elements all diagonal elements are actually 2, then upper diagonal elements is minus 1 lower diagonal element is also minus 1. So, instead of 3 by 3 matrix, if somebody is asking me to construct let us say 3000 by 3000 method matrix, two-dimensional matrix, in that case I have to write down 2, 2, 2 in the diagonal form minus 1 minus 1 like this way in the upper diagonal form, and then minus 1 minus 1 in the lower diagonal part also.

And this needs to be written many, many times. So, previously what we have done, we have manually, once we have created the null matrix, we have manually entered the values of the each element, we have reassigned the values of each element manually. We can automate the process matrix construction by using a for loop.

And that is exactly what we are going to learn. For a matrix of let us say 3 by 3 dimension or 2 by 2 dimension, we may not need a for loop, we can manually enter them, re-enter the values of each element. But if you are constructing a large matrix, very large matrix, then for

loop would be the only option and one should make use of this automated option. So, we will look at how to automate the process.

(Refer Slide Time: 11:21)



```
Command Prompt
C:\Program Files (x86)\Python38-32>python test1.py
[[0, 0.]
 [0, 0.]]

C:\Program Files (x86)\Python38-32>python test1.py
[[2, 1.]
 [1, 2.]]

C:\Program Files (x86)\Python38-32>python test1.py
[[3.+0.j 1.+0.j]
 [0.70710678 -0.70710678]
 [0.70710678 0.70710678]]

C:\Program Files (x86)\Python38-32>python test1.py
(3+0j)
[0.70710678 0.70710678]
(1+0j)
[-0.70710678 0.70710678]

C:\Program Files (x86)\Python38-32>python test1.py
0.9999999999999998
0.9999999999999998
0.8

C:\Program Files (x86)\Python38-32>

test1 - Notepad
File Edit Format View Help
#zeros functionality is imported from scipy module
from scipy import zeros,arange
from scipy.linalg import eig
# a(3x3) null matrix is constructed
A=zeros((3,3))
for i in arange(3):
    |
    |
    |
E,V=eig(A)
print(V[:,0]@V[:,0])
print(V[:,1]@V[:,1])
print(V[:,0]@V[:,1])
```

The first step is as usual, we will first create the zero matrix. And that we will keep it as it is zero matrix. And in this case, we will make 3 by 3 matrix. Because our desired matrix each dimension is 3 by 3. So, we will call it is zeros 3 by 3. That is the way we will mention and then instead of assigning manually, we will assign through automate the assignment process.

So, we will write down as for i in arange, arange functionality we will use. So, we will import this arange functionality, previously we have seen arange functionality can produce a list of elements. So, arange 3, this is something which we are using for the first time and I will explain what does it mean by arange 3. Previously, I will go back to the slide.

(Refer Slide Time: 12:22)



Steps to Find Eigenvalues and Eigenvectors

Example 2: Dealing with Tridiagonal Matrix

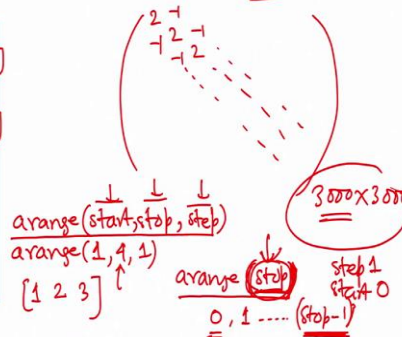


(a) Automated Construction of Matrix

```

from scipy import zeros, arange
A=zeros((3,3))
for i in arange(3):
    A[i,i]=2
for i in arange(2):
    A[i,i+1]=-1
    A[i+1,i]=-1
print(A)

[[ 2. -1.  0.]
 [-1.  2. -1.]
 [ 0. -1.  2.]]
    
```



So, previously, we have always used this arange functionality as start, then stop, then step. So, it will give me, it will. So, if the range functional, if I am giving this to be 1, then 4, then 1, it will create a list of element like 1, then 2, then 3, and it will stop at 3, because the stop is not included in the list in the sequence. That is the way arange functionality works.

And that is the construct which we have used before. It is giving me the list of elements which I need, but there is another very frequently used construct for arange functionality and that is called arange, just write down arange stop. So, what it does, if I do not use three input for the arange functionality, if I use only one input, it will be considered to be as stop, stop input.

And then by default, it will consider 0, 1 like this, and then stop minus 1 because it will exclude stop, and it will print up to stop minus 1. So, by default is taking the step size to be 1, step to be 1 and starting point to be 0. So, that is the default input it will take. So, arange functionality works with both constructs one construct where we give three inputs together, where we can control where to start and when to stop and what is the step size.

But if I use only arange functionality with only one input, that one input would be considered to be the stop input. And default by default, it will start from 0 and it will end at stop minus 1 value because always it will exclude the stop value. And that is exactly what we have done here. If I say arange, arange 3, it means I am getting a list of following lists 0, 1, 2 following list, and arange 2 I will get list 0, 1, 2 will be excluded. So, these are the list time getting. And this functionality we are going to use right now.

(Refer Slide Time: 15:41)


```

C:\Program Files (x86)\Python38-32>python test1.py
[[0. 0.]
 [0. 0.]]

C:\Program Files (x86)\Python38-32>python test1.py
[[2. 1.]
 [1. 2.]]

C:\Program Files (x86)\Python38-32>python test1.py
[[3.+0.j 1.+0.j]
 [ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

C:\Program Files (x86)\Python38-32>python test1.py
(3+0j)
[0.70710678 0.70710678]
(1+0j)
[-0.70710678  0.70710678]

C:\Program Files (x86)\Python38-32>python test1.py
0.9999999999999999
0.9999999999999999
0.0

C:\Program Files (x86)\Python38-32>python test1.py
[[ 2. -1.  0.]
 [-1.  2. -1.]
 [ 0. -1.  2.]]

C:\Program Files (x86)\Python38-32>

```

```

File Edit Format View Help
#zeros functionality is imported from scipy module
from scipy import zeros,arange
from scipy.linalg import eig
# a(3x3) null matrix is constructed
A=zeros(3,3)
for i in arange(3):
    A[i,i]=2
for i in arange(2):
    A[i,i+1]=-1
    A[i+1,i]=-1
print(A)

```

So, we have for i in $\text{arange } 3$, I have to specify now, i comma i . So, what I am going to do right now, diagonal elements I am specifying and I know that diagonal elements is going to be $A[i, i]$, that is 2 always. Next, I am going to specify off diagonal, upper diagonal and lower diagonal elements. i in $\text{arange } 2$ I am going to use 2 and then $A[i, i+1]$ is going to be minus 1, $A[i+1, i]$ it is also going to be minus 1.

So, upper diagonal elements will be x expressed by $i+1$, lower diagonal elements will be expressed by $i+1$ and both values are 1. So, we will be able to get the matrix will do one thing. After construction, we will just print the matrix to check what we have constructed. So, if we run the program, we get the desired matrix where diagonal elements are 2, upper diagonal elements are minus 1, lower diagonal elements is minus 1. And so, we have constructed the matrix which we want to do.

(Refer Slide Time: 17:27)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 2: Dealing with Tridiagonal Matrix

(a) Automated Construction of Matrix

```
from scipy import zeros, arange
A=zeros((3,3))
for i in arange(3):
    A[i,i]=2
for i in arange(2):
    A[i,i+1]=-1
    A[i+1,i]=-1
print(A)
```

Handwritten notes and diagrams:

- Matrix elements: $A[0,0]=2$, $A[1,1]=2$, $A[2,2]=2$, $A[0,1]=-1$, $A[1,0]=-1$, $A[1,2]=-1$, $A[2,1]=-1$.
- Matrix structure: $\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$
- Diagram of a 3x3 tridiagonal matrix with dashed lines indicating the off-diagonal elements.
- Diagram of a 3000x3000 matrix with a tridiagonal structure.
- Diagram of a 3x3 matrix with a tridiagonal structure.
- Diagram of a 3x3 matrix with a tridiagonal structure.
- Diagram of a 3x3 matrix with a tridiagonal structure.

Time dependent Quantum Chemistry

So, what will happen in the first, we will go back to the slide right now. In the first loop, it will collect the value for A 0, 0 that is going to be 2, second loop it is going to be 1, 1 that is going to be 2 and third loop it will collect the value 2, 2 that is going to be 2. So, I have first row first column, second row second column, third row third column, third row third column is given by A 2, 2 which is the element 2. So, this is the, this is all about this iteration.

For this iteration, for the first iteration, I will have A, i value will be taken from arange and look at this for the first iteration, I have set the arange to be 3 which means that I have only index value to be, i value to be selected from 0, 1, 2 only. So, three values are selected. And these are the indices which you have selected from the arange functionality.

On the other hand, for the next one, I have only option two 0, 1. And that is why I will write down first one is 0, and then second one is going to be 1, that is going to be minus 1. So, what it does in the first loop, I am reassigning as minus 1. So, this is going to be the first row second column, first row second column is going to be this one, then second row first column is going to be this one. So, this two has been assigned.

In the second iteration, this is the first iteration, second iteration I will have A next number is 1, so it is going to be 1, 2, that is going to be minus 1. And 2, 1 is going to be minus 1. So, I have these values and these values. And then I do not have any other numbers so that is why the for loop will terminate and will go to the next command line.

So, for loop we will be running for two iterations here. And here for loop will be running for three iterations and that is the way we are replacing automating the construction of this matrix. So, we have got back the python representation of this matrix.

(Refer Slide Time: 20:12)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 2: Dealing with Tridiagonal Matrix $\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$ }
3x3

(b) Directly Finding Eigenvector and Eigenvalues of the Tridiagonal Matrix

```

from scipy.linalg import eig
from scipy import zeros, arange
A=zeros((3,3))
for i in arange(3):
    A[i,i]=2
for i in arange(2):
    A[i,i+1]=-1
    A[i+1,i]=-1
E,V=eig(A)
print(E)
print(V)

```

```

(3.41421356+0.j, 2., 0., 0.58578644+0.j)
[[-5.00000000e-01 -7.07106781e-01  5.00000000e-01]
 [ 7.07106781e-01  4.05925293e-16  7.07106781e-01]
 [-5.00000000e-01  7.07106781e-01  5.00000000e-01]]

```

Time dependent Quantum Chemistry

```

C:\Program Files (x86)\Python38-32>python test1.py
[[ 0.  1.]
 [ 2.  1.]
 [ 1.  2.]]

C:\Program Files (x86)\Python38-32>python test1.py
[[ 3.+0.j  1.+0.j]
 [ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

C:\Program Files (x86)\Python38-32>python test1.py
[[ 3+0j]
 [ 0.70710678  0.70710678]
 [ 1+0j]
 [-0.70710678  0.70710678]]

C:\Program Files (x86)\Python38-32>python test1.py
0.9999999999999998
0.9999999999999998
0.0

C:\Program Files (x86)\Python38-32>python test1.py
[[ 2. -1.  0.]
 [-1.  2. -1.]
 [ 0. -1.  2.]]

C:\Program Files (x86)\Python38-32>python test1.py
[[ 3.41421356+0.j  2.      +0.j  0.58578644+0.j]
 [-5.00000000e-01 -7.07106781e-01  5.00000000e-01]
 [ 7.07106781e-01  4.05925293e-16  7.07106781e-01]
 [-5.00000000e-01  7.07106781e-01  5.00000000e-01]]

C:\Program Files (x86)\Python38-32>

```

```

#zeros functionality is imported from scipy module
from scipy import zeros, arange
from scipy.linalg import eig
# a(3x3) null matrix is constructed
A=zeros((3,3))
for i in arange(3):
    A[i,i]=2
for i in arange(2):
    A[i,i+1]=-1
    A[i+1,i]=-1
E,V=eig(A)
print(E)
print(V)

```

Next, what we will do, we will just follow the usual procedure to find out the eigenvalue eigenvectors of this matrix. We will just write down E comma V equals the first one would, be the first option is going to be eigenvalue second option is going to be eigenvector which can be represented and then A. So, if we do that and then if we print E, so this is Python will be printing on its own way one can extract each one, each eigenvalue and eigenvector following a construct I have already shown.

So, if I print these two, then what I get is that, it has three eigenvalues, we will go back to the previous to the slides right now. So, I have now, this is this is one eigenvector, sorry, the eigenvalue, then another eigenvalue and then another eigenvalue. So, there are two eigenvalues I have one is this one.

So, each one having this complex notation. I do not need this complex notation, each one is 0, that is why this is one eigenvalue, this is another eigenvalue and this is another eigenvalue, there's three eigenvalues I have. And corresponding eigenvectors are following, I have to construct this is one eigenvector then this is right eigenvector corresponding to this and corresponding to this this is another eigenvector.

So, there are three eigenvectors I have already found and these will be all in grid representation. So, we can directly calculate the eigenvalues and eigenvectors of a tridiagonal matrix. But this is not and the procedure which I am showing the direct procedure where I am using this, I am using making use of the entire matrix. This direct procedure can be useful for that lower dimension like let us say a 3 by 3 matrix I have in that case it can be very quick.

(Refer Slide Time: 22:50)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 3: Dealing with Large Tridiagonal Matrix

Time dependent Quantum Chemistry

But when we have very large very, very large tridiagonal matrix and if you are dealing with very large tridiagonal matrix one can see one can go for this elemental map of this matrix. If we carefully go over the elemental map, we see that all the elements are actually 0. And only I have this further tridiagonal matrix I have the diagonal matrix elements which are nonzero and upper and lower diagonal elements are nonzero.

So, there are procedures which one can follow to represent the difference storage form, one can use a different storage form for this kind of large diagonal matrix to reduce the computational time. So, if you are dealing with very large tridiagonal matrix, it is better not to directly use this matrix and find out the eigenvalue and eigenvector with the help of this construct. We should not do that.

We should use some other technique which would be more computationally time effective if we perform in a different way. So, we will present it how to deal with very large tridiagonal matrix and we will prove that if we directly do that calculations, instead of doing direct calculation, if we do it in a band storage form, then the computation would be much more efficient and computation time can be extraordinarily reduced.

So, in quantum mechanics, we often work with very large Hermitian matrix of tridiagonal form which is shown here. And a generic structure of the tridiagonal matrix shown here. A square matrix possessing nonzero elements only in the main diagonal is called a diagonal matrix. So, if I have a matrix where only diagonal elements having nonzero values, remaining part is 0 then is called diagonal matrix.

The main diagonal of a matrix consists of elements that lie on the diagonal that run from top left to the bottom right. A tridiagonal matrix which is presented here has nonzero elements on the main diagonal, this is the main diagonal and on the upper diagonal, this is upper diagonal and this is lower diagonal part and these values would be nonzero in a tridiagonal matrix.

So, in tridiagonal matrix is also called a band diagonal matrix because it is only nonzero elements are confined to a diagonal band. So, one can think about a band like this and all nonzero elements have been confined within this band, and that is why it is called also band diagonal matrix.

(Refer Slide Time: 26:13)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 3: Dealing with Large Tridiagonal Matrix

Band Storage Forms: With Upper Diagonal Elements

Hermitian Matrix
Self-adjoint $A_{ij} = A_{ji}^*$

$N \times N$

$2 \times N$

Band storage form. ✓

Time dependent Quantum Chemistry

Now, in module four, we have realized that the defining condition of a Hermitian matrix. this is an Hermitian matrix and defining condition of a Hermitian matrix is that it is self-adjoint. So, if it is an Hermitian matrix it has to be self-adjoint. And self-adjoint what does it mean? A_{ij} is going to be A_{ji} star. And due to this symmetry, because it is self-adjoint due to the symmetry of Hermitian matrix, a tridiagonal matrix $N \times N$ matrix can be stored as 2 by N band storage form. So, this is called band storage form.

So, the point I am trying to make here is that instead of dealing with very large tridiagonal matrix, one can store the entire tridiagonal matrix in its band storage form. And if we do that, then one can see that the actual matrix was N by N , let us say 3000 by 3000 the big matrix that can be stored as 2 by N which is two by 3000, let us say. So, the memory which will be used to store the matrix can be reduced significantly if we use this band storage form of the tridiagonal matrix.

There are two ways one can store the tridiagonal matrix in the band storage form one is called the upper diagonal elements with the help of upper diagonal elements. Upper diagonal elements, so we know that this is diagonal element and this part is upper diagonal element. So, with the help of only upper diagonal element, I can store it. And the way I can store it is following I have 0 then, so first to be 0, and then this upper diagonal elements and then the diagonal elements.

(Refer Slide Time: 28:49)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 3: Dealing with Large Tridiagonal Matrix

Band Storage Forms: With Lower Diagonal Elements

SciPy.linalg.eig_banded(B)

$$\begin{pmatrix} n & n & \dots & n & n \\ m_1 & m_1 & \dots & m_1 & 0 \end{pmatrix}_{2 \times N} = B$$

Band storage form.

Time dependent Quantum Chemistry

On the contrary, one can also store with the help of lower diagonal elements, and in that case, it is going to be the diagonal elements first, and then in the second row we will have the lower diagonal elements with 0 in the end. So, these are the two different representations of band storage form. So, what we get is that we actually store, we are actually storing the very large Hermitian tridiagonal matrix in the band storage form to reduce the memory consumption for the entire, when you are dealing with the matrix.

And if we do that this band storage form, then scipy linear algebra the sub module gives me one very efficient functionality to calculate the eigenvector and eigenvalues for this kind of matrix, if it is stored in the band storage form. So, and with the help of this. So, instead of only eig within bracket I have to use now underscore then banded then bracket.

So, what will, the bottom line of the entire discussion is that, if I have a very large Hermitian tridiagonal matrix, first thing I should do to reduce the memory consumption and to make the computation effective or faster, we have to convert this matrix form this large matrix to one of the band storage forms either upper diagonal with the help of upper diagonal elements or with the help of lower diagonal elements, their structures are different, but one can convert it.

The moment we convert it, we get this 2 by N matrix, and then this new matrix, I will name it let us say B. So, this was this was A, I will call it B and then I can find out the eigenvalue and eigenvector using this construct eig underscore banded B. So, we will take a look at it, how to do that.

(Refer Slide Time: 31:29)

```
Select Command Prompt
C:\Program Files (x86)\Python38-32>python test1.py
[[ 0. -1. -1.]
 [ 2.  2.  2.]]

C:\Program Files (x86)\Python38-32>python test1.py
[[ 2. -1.  0.]
 [-1.  2. -1.]
 [ 0. -1.  2.]]

C:\Program Files (x86)\Python38-32>

test1 - Notepad
#zeros functionality is imported from scipy module
from scipy import zeros,arange
from scipy.linalg import eig
# a(3x3) null matrix is constructed
A=zeros((3,3))
for i in arange(3):
    A[i,i]=2
for i in arange(2):
    A[i,i+1]=-1
    A[i+1,i]=-1
print(A)

#Construct (2x3) band storage form
A_band_up=zeros((2,3))
for i in arange(2):
    A_band_up[0,i+1]=-1.0
for i in arange(3):
    A_band_up[1,i]=2.0
print(A_band_up)
```

We will move to the laptop right now, and we have already constructed the tridiagonal matrix. And this tridiagonal matrix we have constructed now, we will construct 2 cross 3 band storage form. So, we have to now reconstruct the band storage form. And for that, what we will do we will use upper diagonal elements.

So, for that what we need to do we have to start with a null matrix of 2 by 3 dimension. So, remaining part, this part is not of use, we will just keep it as it is. So, this part is to show conventionally how to prepare the large matrix and then directly how to get the eigenvalues and eigenvector, we are not following that procedure.

So, these entire steps will be obsolete right now, we are following another steps. The steps is that by looking at the matrix which we are going to prepare, it is clear that is a tridiagonal

Hermitian matrix. And that is why we can store it in a band storage form. And in order to store the band storage form, first we have to prepare the null matrix.

So, we will prepare zeros, 2 by 3 and we have named the null matrix as A band up, up we are showing because we are going to use the upper diagonal elements and then we will automate the procedure for i in arange 2 this matrix this array values has to be specified and this is going to be 0 comma first row and i plus 1, this is going to be minus 1.0.

And for i in arange 3, I have to now reassign the elements it is going to be first row and this and that value is going to be 2.0. What we will do? We will not print to see what we are constructing. So, we have constructed the band storage form and now we run the program. So, what we have done, we are commanding to print the normal form of the tridiagonal matrix.

So, that has been printed here. And then with the help of the upper diagonal elements, we have printed this. So, that is the band storage form with the help of upper diagonal elements. So, take a look at it. Let us go back to the previous slide. So, we have shown in the slides.

(Refer Slide Time: 36:11)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 3: Dealing with Large Tridiagonal Matrix

Band Storage Forms: With Upper Diagonal Elements

Hermitian Matrix
Self-adjoint $A_{ij} = A_{ji}^*$

Band Storage Form. ✓

Time dependent Quantum Chemistry

So, what we see here is that if we use the upper diagonal elements, here we are using upper diagonal elements, the first element, the element at the first row first column is going to be 0, and then remaining part is going to be remaining the each element in this different column of the same row is going to be minus 1. And that is exactly what we have commanded here.

If we go back to the laptop, we will see that for this iteration the first iteration, therefore for the first loop, I have commented that for the zeroth row which will be 1, index 0 means it is a first row. For the first row and second column and onwards, first row second column, first row third column, first row fourth column like this way, all is going to be minus 1.

And how far it should go? It should go up to arrange 2, which means that 2 will be excluded. So, I have only 0 and 1. So, it should go up to 1 and 2. So, two elements I have, minus 1 minus 1. On the other hand, for the next for loop, I have set the row to be 1, which means the second row and for entire column, I have to use two values and I get this band storage form.

So, this is exactly the band storage form with the help of upper diagonal elements. Once we have that band storage form with the upper diagonal elements, one can actually calculate the eigenvalues and eigenvectors. So, I will put this to be, we will do both, directly we can calculate the eigenvalues of the entire matrix which is here or one can also calculate them eigenvalue eigenvector of the band storage form corresponding to this the matrix A, this is the band storage form.

And in that case, I have to use eigen banded. This is the construct. So, I have to also import the same functionality from scipy linear algebra sub module. So, if we run this program, and we have to print it. So, print E, print V. Here also we have to print, print -- We will name different, we will make it E1, V1 and this one as E2, V2. So, this print would be 1, 1 and this is going to be E2 and print V2.

So, if we do that, what we get is that we get the same results actually, if we run, if we try to calculate directly with the help of interest matrix and if we run the calculations with the band storage form, we see that the same results we are getting. In the band storage form the values are 0.5, 8, then second one is 0.2, third one is 3 point something and that is exactly what we get 0.5, 2 and 3.4 these are the three eigenvectors we get.

On the other hand, corresponding to 3.4 this eigenvalue I have this eigenvector minus 5, 7, minus 5 and what I get is 5, minus 7, 5. So, this part, this negative part is coming in to the fact that you have this, if we multiply each element by minus 1 it is going to give you this values. So, they are the same eigenvector.

On the other hand, for corresponding to 2, we have now minus 7, 4 and 7. And here I have 0.5. So, corresponding to 0.5 I have then 5 7 5. So, I have minus 5, minus 7, minus 5. So,

they are presenting the same eigenvectors which I have calculated through the band storage form.

(Refer Slide Time: 42:30)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 3: Dealing with Large Tridiagonal Matrix

(b) Step 2: Find Eigenvalues and Eigenvectors using eig_banded

```

from scipy.linalg import eig, eig_banded
from scipy import zeros, arange
#Constructing (2x3) band storage form using lower diagonal elements
A_band_lower=zeros((2,3))
for i in range(3):
    A_band_lower[0,i]=2.0
for i in range(2):
    A_band_lower[1,i]=-1.0
print(A_band_lower)
E,V=eig_banded(A_band_lower,lower=True)
print(E)
print(V)

[[ 2.  2.  2.]
 [-1. -1.  0.]]

[0.58578644  2.          3.41421356]

[[-5.00000000e-01 -7.07106781e-01  5.00000000e-01]
 [-7.07106781e-01  3.12250226e-16 -7.07106781e-01]
 [-5.00000000e-01  7.07106781e-01  5.00000000e-01]]

```

Time dependent Quantum Chemistry

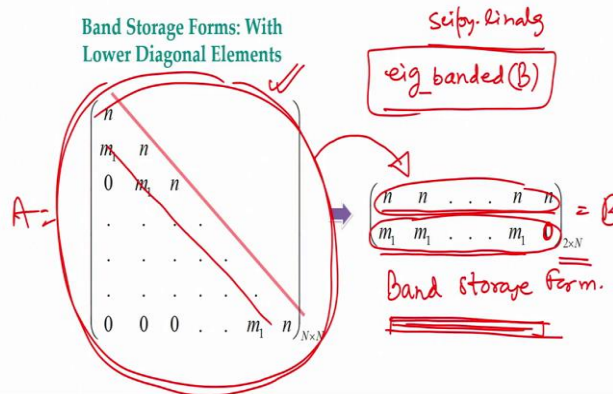
The screenshot shows a Python script being executed in a Command Prompt window. The script defines a 3x3 matrix A using band storage and uses `eig_banded` to find its eigenvalues and eigenvectors. The output in the Command Prompt shows the matrix A, the eigenvalues E, and the eigenvectors V. The Notepad window shows the code being executed, which includes comments and the same code as shown in the slide above.



Steps to Find Eigenvalues and Eigenvectors

Example 3: Dealing with Large Tridiagonal Matrix

Band Storage Forms: With Lower Diagonal Elements



So, now, one may want to use another convention, one can use the lower band for the band storage form. So, I will write down lower. And if it is lower, then this values would be 0, first 0 and then this is going to be 1 and this is going to be 2 and second one is the 1, i and that is going to be minus 1 and this one going to be 2 and this is going to be 3. Now, one can, this is going to be lower.

And if we now print this entire band storage form. And here also, we print A. What we see here? Now, we are actually using the lower diagonal matrix. So, if we look at, if we go back to the slides, we see that if we are using the lower diagonal elements, then the band storage form we use, then the diagonal elements comes in the first row, and the second row will be occupying the lower diagonal elements, the last value would be 0. And that is exactly what is going on here last value is 0.

So, we are getting back this matrix band storage form and one can now calculate the eigenvalues and eigenvectors by this. But in this case, because we are using lower diagonal elements, we have to by default, it is taking the upper diagonal elements, but if I specifically want to use lower diagonal elements, then I have to make lower to be true, then it can calculate otherwise not be able to calculate it, and then we have to print it, print E, print V.

So, it is giving me the same result 0.5, 2, 3.4 these are the eigenvalues. And so, I get back the same eigenvalues, it depends on what kind of storage procedure we are following and if we have a particular procedure to store we will just use it. So, if I do not use this lower equals true, then I will have some value but these values are wrong.

Because now, it does not know whether I have used a lower band storage form, it is accepting that I am using upper band storage form. And that is why they have taken like this way. So, it is going to be wrong, because by default is going to be upper band storage form. So, always we have to use this lower equals true if we are storing it in terms of lower diagonal elements, then I get the values 0.5, 2, 3.4. That is exactly what we had previously 0.5, 2, 3.4.

(Refer Slide Time: 47:19)

Python Tutorial 4: Matrix, Eigenvalues and Eigenvectors

Steps to Find Eigenvalues and Eigenvectors

Example 3: Dealing with Large Tridiagonal Matrix

```

from scipy.linalg import eig, eig_banded
from scipy import zeros, arange
#Constructing (2x3) band storage form using lower diagonal elements
A_band_lower=zeros((2,3))
for i in range(3):
    A_band_lower[0,i]=2.0
for i in range(2):
    A_band_lower[1,i]=-1.0
print(A_band_lower)
E,V=eig_banded(A_band_lower,lower=True)
print(E)
print(V)

```

↓
True
False

(b) Step 2: Find Eigenvalues and Eigenvectors using eig_banded

```

[[ 2.  2.  2.]
 [-1. -1.  0.]]

[0.58578644 2.    3.41421356]

[[-5.00000000e-01 -7.07106781e-01  5.00000000e-01]
 [-7.07106781e-01  3.12250226e-16 -7.07106781e-01]
 [-5.00000000e-01  7.07106781e-01  5.00000000e-01]]

```

Time dependent Quantum Chemistry

One more point to mention here is that this lower equals true, this is the Boolean value. So, it will always start with true like this or false with a capital letter, but this lower is the input or the argument that is going to be the small letter. So, we should remember this one when you are employing this.

(Refer Slide Time: 47:51)

Time Dependent Quantum Chemistry

Python Tutorial 4: Constructing Matrix and Finding its Eigenvalues and Eigenvectors

Summary

- (1) How to construct a desired matrix
- (2) How to find eigenvalues and eigenvectors
- (3) How to obtain eigenvalues and eigenvectors of tridiagonal matrix using band storage form

Time dependent Quantum Chemistry

So, we have come to end of this Python tutorial where we have learned how to represent a matrix, how to represent a tridiagonal matrix, kinetic energy matrix in grid representation is actually a tridiagonal matrix Hermitian tridiagonal matrix. And when you are dealing with a very large tridiagonal matrix, one can use this one of the band storage forms and once you represent in one of the band storage form, Python linear algebra sub module provides a particular functionality to get the eigenvalues and eigenvectors from this band storage form. We will meet again in next tutorial and modules.