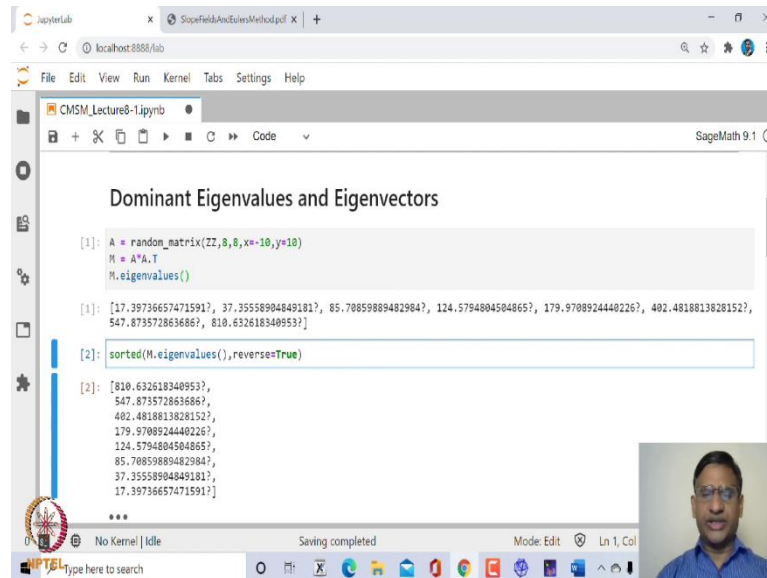


Computational Mathematics with SageMath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Lecture - 49
Numerical Eigenvalues

(Refer Slide Time: 00:15)



The screenshot shows a JupyterLab window with a SageMath 9.1 kernel. The title bar indicates the file is 'CMSM_Lecture8-1.ipynb'. The code cell contains the following Python code:

```
[1]: A = random_matrix(ZZ, 8, 8, x=-10, y=10)
M = A*A.T
M.eigenvalues()

[2]: sorted(M.eigenvalues(), reverse=True)
```

The output of the code is displayed in the cell:

```
[1]: [17.39736657471591?, 37.35558904849181?, 85.70859889482984?, 124.5794804504865?, 179.9708924440226?, 402.4818813828152?,
547.873572863686?, 810.632618340953?]

[2]: [810.632618340953?,
547.873572863686?,
402.4818813828152?,
179.9708924440226?,
124.5794804504865?,
85.70859889482984?,
37.35558904849181?,
17.39736657471591?]
```

The interface also shows a sidebar with icons for file management, a top bar with 'File Edit View Run Kernel Tabs Settings Help', and a bottom status bar indicating 'No Kernel | Idle' and 'Mode: Edit'.

Welcome to the 49th lecture on Computational Mathematics with SageMath. In this lecture, we shall look at how to find Dominant Eigenvalues and Eigenvector, corresponding eigenvector, numerically. We have seen use of dominant eigenvalue in Google search algorithm and it has many other applications.

So, let us start with an example. We know that Sage has inbuilt function to find eigenvalues and eigenvectors, and it can find eigenvalues of, eigenvectors of reasonably large matrix as well. So, for example, if I look at matrix A, which is a random matrix over integers of 8 cross 8, and let us say the entries lies between minus 10 and 10, and let us convert this A by multiplying by its transpose.

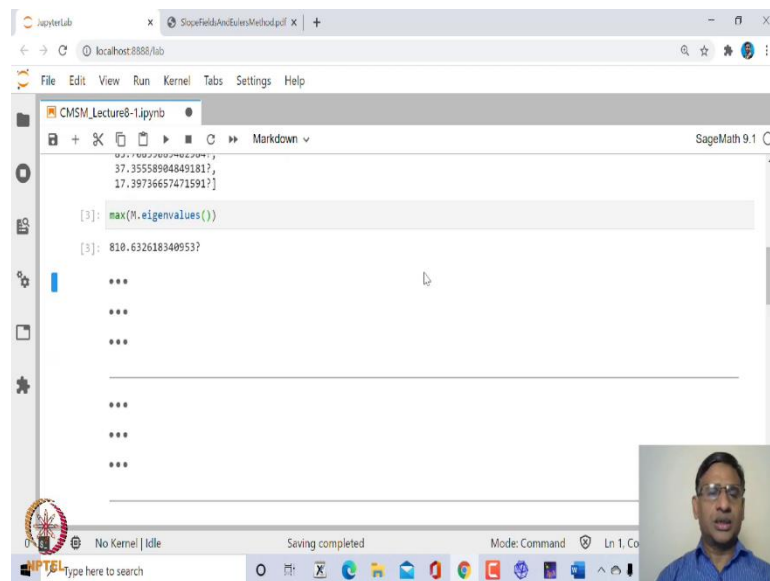
So, A into A transpose will be a symmetric matrix. So, all its eigenvalues will be real. So, let us find eigenvalues of this, so this is what we get. So, you can see here, these numbers which you are getting are in decimal, and there is a question mark at the end, this

simply means that these computations are done numerically. These are numerical eigenvalues. So, how does one find such eigenvalues?

So, we will be looking at an algorithm to find dominant eigenvalue of a matrix. So, in this case, for example, let us look at what is the dominant eigenvalues. For example, if you look at, here, this set of eigenvalues are by default arranged in increasing order of its magnitude. So, now, suppose we sort this eigenvalues and say reverse equal to true, it will be arranged in decreasing order of the magnitude.

So, Sage has inbuilt method in order to find the dominant eigenvalues of a matrix. So, if you want to extract this dominant eigenvalue, you can simply say, find the maximum of the eigenvalues of this.

(Refer Slide Time: 02:48)



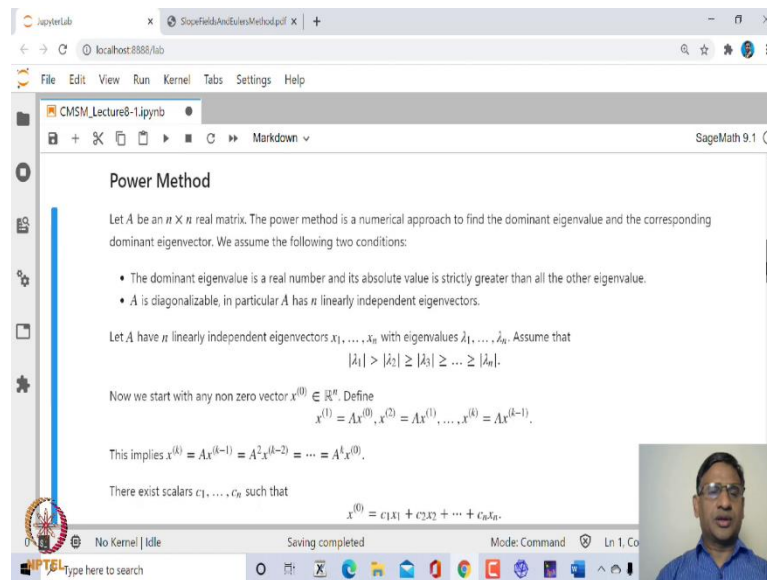
The screenshot shows a JupyterLab window with a SageMath 9.1 kernel. The code cell contains the following text:

```
37.35558904849181?,  
17.39736657471591?]  
  
[3]: max(M.eigenvalues())  
  
[3]: 810.632618340953?  
  
...  
...  
...  
  
...  
...  
...
```

The output shows the maximum eigenvalue as 810.632618340953. The interface includes a file explorer on the left, a command line at the bottom, and a video feed of the presenter in the bottom right corner.

So, let us look at, if I say max of M dot eigenvalues, then I will get the maximum of the eigenvalues. So, this is what we want to find out using an algorithm, which is known as power method.

(Refer Slide Time: 03:04)



Power Method

Let A be an $n \times n$ real matrix. The power method is a numerical approach to find the dominant eigenvalue and the corresponding dominant eigenvector. We assume the following two conditions:

- The dominant eigenvalue is a real number and its absolute value is strictly greater than all the other eigenvalues.
- A is diagonalizable, in particular A has n linearly independent eigenvectors.

Let A have n linearly independent eigenvectors x_1, \dots, x_n with eigenvalues $\lambda_1, \dots, \lambda_n$. Assume that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots \geq |\lambda_n|.$$

Now we start with any non zero vector $x^{(0)} \in \mathbb{R}^n$. Define

$$x^{(1)} = Ax^{(0)}, x^{(2)} = Ax^{(1)}, \dots, x^{(i)} = Ax^{(i-1)}.$$

This implies $x^{(i)} = Ax^{(i-1)} = A^2x^{(i-2)} = \dots = A^ix^{(0)}$.

There exist scalars c_1, \dots, c_n such that

$$x^{(i)} = c_1\lambda_1^i x_1 + c_2\lambda_2^i x_2 + \dots + c_n\lambda_n^i x_n.$$

So, let us see how it works, what is this algorithm. So, first of all, let us assume that A is an n cross n real matrix, and we shall also assume the two conditions on A , that A has a dominant eigenvalue which is real, and its absolute value is strictly greater than all other eigenvalues, right?

So, in case $\lambda_1, \lambda_2, \dots, \lambda_n$ are eigenvalues of A , then modulus of λ_1 is strictly bigger than modulus of λ_2 , which is bigger than equal to modulus of λ_3 , and so on, that is the assumption. Where $\lambda_1, \lambda_2, \dots, \lambda_n$ are eigenvalues arranged in decreasing order. Second assumption on A will be that A is diagonalizable, in particular A has n linearly independent eigenvectors, and it forms a basis of \mathbb{R}^n , right?

(Refer Slide Time: 04:08)

Multiplying both sides by A^k , we get

$$\begin{aligned} x^{(k)} &= A^k x^{(0)} = \sum_{i=1}^n c_i A^k x_i \\ &= \sum_{i=1}^n c_i \lambda_i^k x_i \\ &= \lambda_1^k \left[c_1 x_1 + \sum_{i=2}^n c_i \left(\frac{\lambda_i}{\lambda_1} \right)^k x_i \right] \end{aligned}$$

Since $\lambda_1 > \lambda_i$ for all $i > 1$, the ratio $\left| \frac{\lambda_i}{\lambda_1} \right| < 1$. Thus as $k \rightarrow \infty$, $\frac{\lambda_i}{\lambda_1} \rightarrow 0$. Hence $\frac{x^{(k)}}{\lambda_1^k} \rightarrow c_1 x_1$.

In case, $c_1 \neq 0$, we can approximate x_1 . Once we find an eigenvector x_1 , then the corresponding eigenvalue is given by

$$\lambda = \frac{x_1^T A x_1}{x_1^T x_1}.$$

The right hand side of the above equation is called the *Rayleigh quotient* of x_1 with respect to A .

This leads to one of the very important method of finding the dominant eigenvalue and eigenvector, namely the *Power Method*.

Now, what we do is, we start with a non-zero vector x_0 , you can start with unit vector, or start with some vector with dominant entries, or the entries of, the maximum value of the entries is 1, ok. So, let us start with a non-zero vector x_0 and define x_1 is equal to A times x_0 similarly and x_2 is equal to A times x_1 , and so on. In general x_k is equal to A times x_{k-1} . So, that is an iterative scheme.

So, in, if you, if you just simply simplify this, x_k would be nothing but $A^k x_0$. So, we are defining a sequence of vectors starting with x_0 as x_1, x_2, \dots, x_k , and so on, by x_k is equal to $A^k x_0$.

Now, since A has n linearly independent eigenvectors x_1, x_2, \dots, x_n , it forms a basis corresponding to eigenvalues λ_1 to λ_n , this x_0 vector x_0 can be written as linear combination of x_1 to x_n . So, let us write x_0 as $c_1 x_1 + c_2 x_2 + \dots + c_n x_n$.

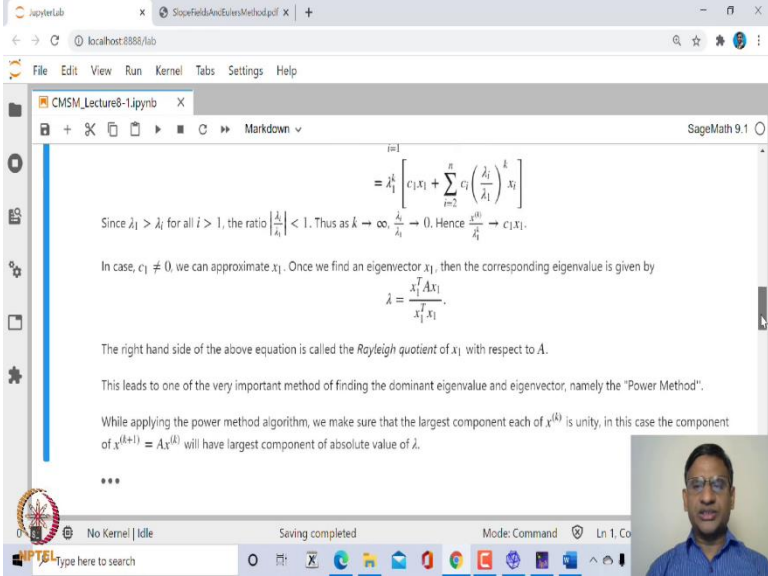
Now, apply A to the power k on both the sides. So, $A^k x_0$ is nothing but x_k , and that is going to be, on the right-hand side, it will be summation $c_i A^k x_i$, and since x_i 's are eigenvectors, $A^k x_i$ will be nothing, but $\lambda_i^k x_i$.

power $k \times i$, and then you take λ_1 to the power k out outside this bracket, what you will be left with? $c_1 x_1$ plus summation $c_i \lambda_i^k x_i$ by λ_1^k .

Now, if you notice this, since λ_1 is the largest, strictly bigger than all eigenvalues, λ_i by λ_1 for i bigger than equal to 2 will be strictly less than 1. Therefore, as k goes to infinity, as k becomes large and large, this quantity will go to 0. So, what it means is that, if you take x to the power k divided by λ_1^k , this will converge to $c_1 x_1$, right?

And what is x_1 ? x_1 is the eigenvector corresponding to eigenvalue λ_1 , which is the largest, right? So, this gives you, in case c_1 is non-zero, we can write x to power k divided by λ_1^k , that converges to x_1 . So, it gives you a way to find out x_1 , namely, the dominant eigenvector.

(Refer Slide Time: 07:12)



The screenshot shows a JupyterLab window with a SageMath 9.1 notebook. The notebook content is as follows:

$$= \lambda_1^k \left[c_1 x_1 + \sum_{i=2}^n c_i \left(\frac{\lambda_i}{\lambda_1} \right)^k x_i \right]$$

Since $\lambda_i > \lambda_1$ for all $i > 1$, the ratio $\left| \frac{\lambda_i}{\lambda_1} \right| < 1$. Thus as $k \rightarrow \infty$, $\frac{\lambda_i}{\lambda_1} \rightarrow 0$. Hence $\frac{x^{(k)}}{\lambda_1^k} \rightarrow c_1 x_1$.

In case, $c_1 \neq 0$, we can approximate x_1 . Once we find an eigenvector x_1 , then the corresponding eigenvalue is given by

$$\lambda = \frac{x_1^T A x_1}{x_1^T x_1}$$

The right hand side of the above equation is called the *Rayleigh quotient* of x_1 with respect to A .

This leads to one of the very important method of finding the dominant eigenvalue and eigenvector, namely the "Power Method".

While applying the power method algorithm, we make sure that the largest component each of $x^{(k)}$ is unity, in this case the component of $x^{(k+1)} = A x^{(k)}$ will have largest component of absolute value of λ .

...

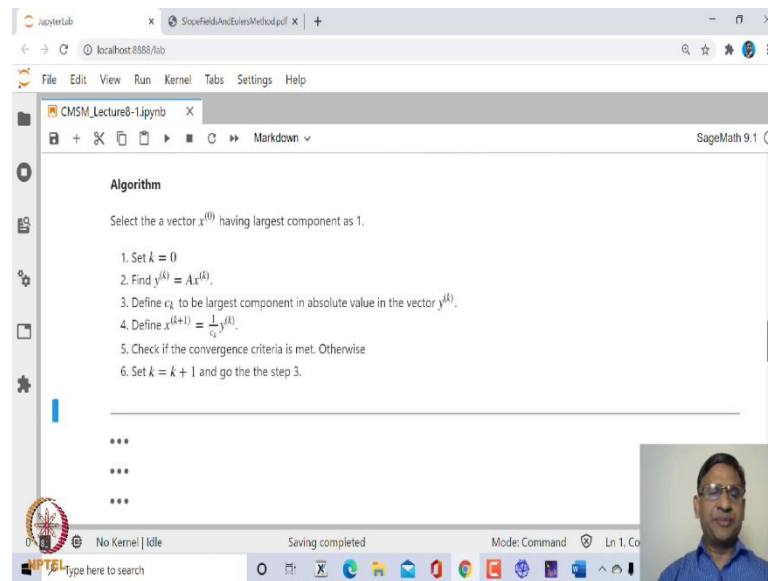
Eigenvector, with, corresponding to dominant eigenvalue is known as dominant eigenvector. So, and once you have obtained x_1 , namely an approximation of dominant eigenvector, you can find the approximation of dominant eigenvalue using this Rayleigh quotient formula which is $x_1^T A x_1$ divided by $x_1^T x_1$.

So, this is $A x_1$ is λx_1 , so, numerator will be $\lambda x_1^T x_1$, norm x_1 square, divided by norm x_1 square. Since x_1 is eigenvector, this will be non-zero, therefore, you can cancel this and you have the dominant eigenvalue. So, that is what is

the algorithm. So, in this algorithm, the only thing is that you, you need to do some scaling, so that this lambda 1 you get as an eigenvalue, right?

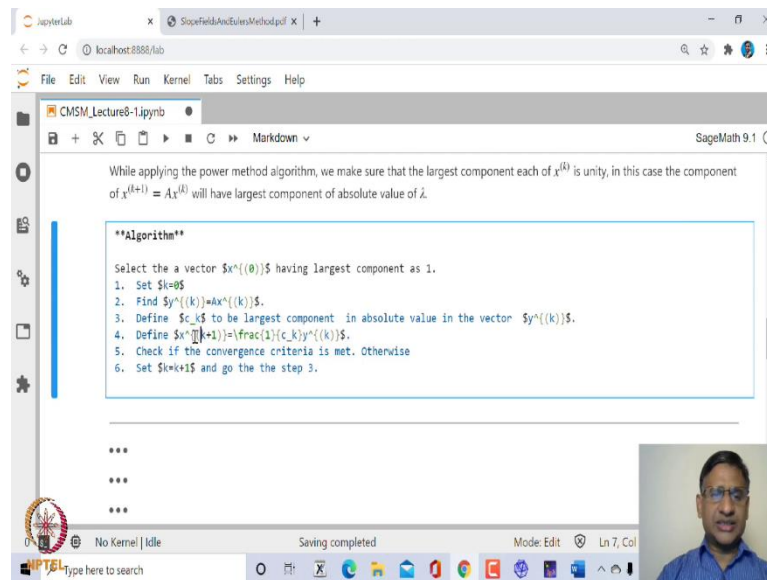
So, the scaling is done as follows. You can, you can choose x_0 at every stage, you choose x_k in such a way that the maximum entry is 1, right? So, you can divide this vector Ax_k by the maximum of the entry in x_k and go to the next iterate, ok? So, let us see how, how, what is the algorithm.

(Refer Slide Time: 08:39)



So, algorithm is as follows. Start with a non-zero vector x_0 having the largest component as 1. So, if its largest component is 1, this will be non-zero set k equal to 0, that is 0th iterate, and define y_k , vector y_k as Ax_k and then take c_k to be the maximum or the largest component of, of this y_k in absolute value.

(Refer Slide Time: 09:12)

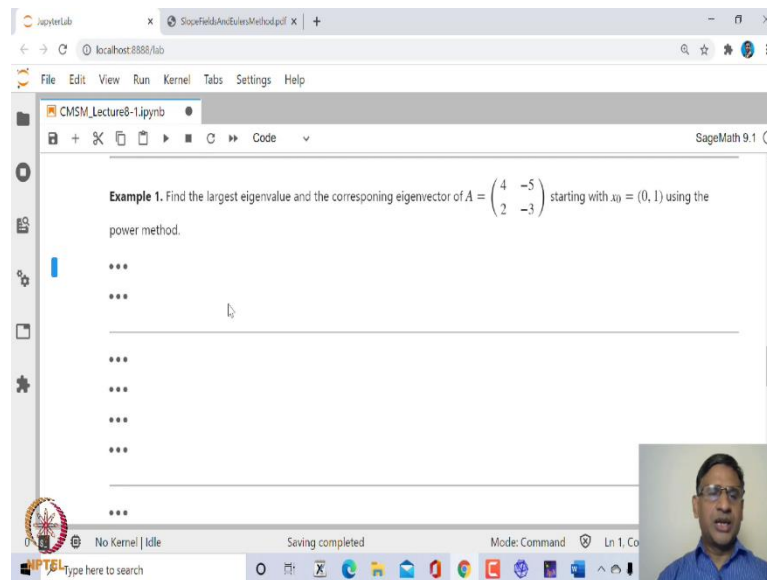


And then define x_k , x_{k+1} , it should be a bracket here, x_{k+1} is equal to $\frac{1}{c_k} y_k$, right? So, again, and then check whether this meets the convergence criteria. Now, what would be the convergence criteria?

The convergence criteria you can, you can take the, the, you can start with some arbitrary positive real number, and in case the difference between 2 consecutive x_k becomes less than epsilon, then you can stop, right, or if you want to stop after certain number of iterates, that itself becomes your convergence criteria.

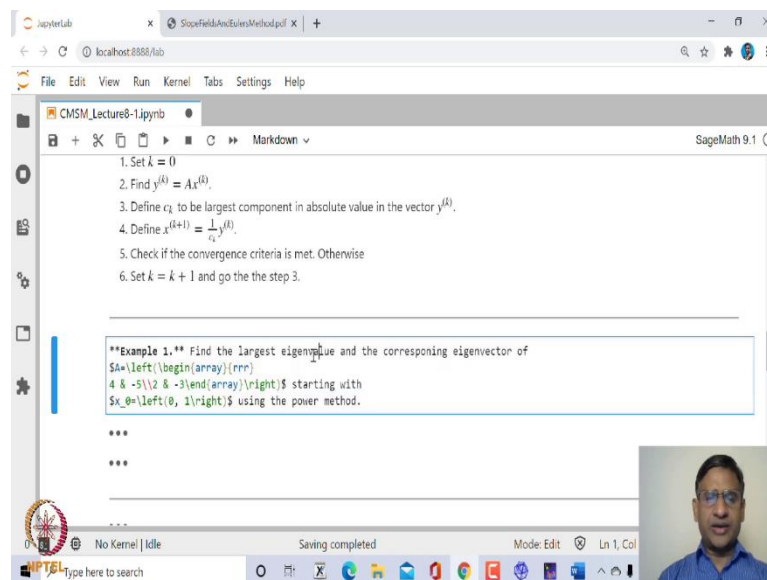
Now, let us, so in case the convergence criteria is met, you have already achieved what you wanted. That means, x_{k+1} will be the largest dominant eigenvector, and you can stop. Otherwise, you redefine k is equal to $k+1$, that is go to the next iterate, and then go to step 3, that is, again find out what is c_k and, and then, then find x_{k+1} and so on, right? So, let us implement this.

(Refer Slide Time: 10:31)



So, let us look at an example. Suppose we want to find the largest eigenvalue, there is spelling mistake here.

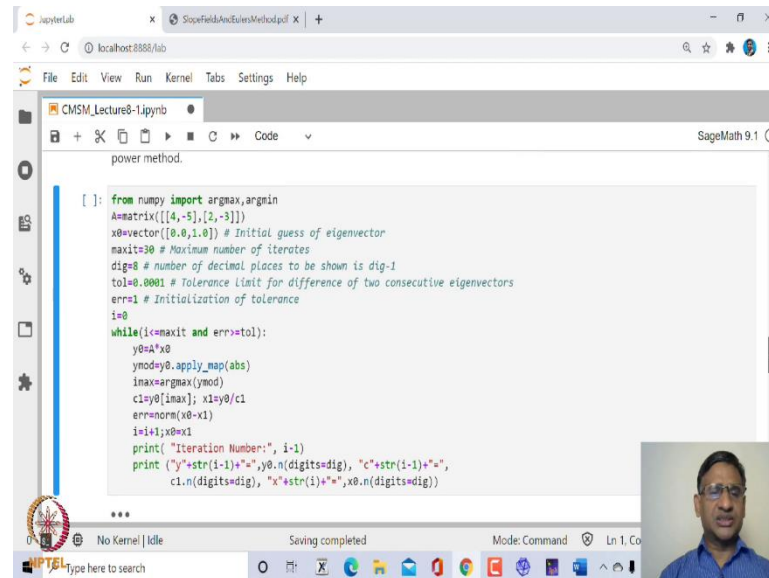
(Refer Slide Time: 10:36)



Eigenvalue, and the corresponding eigenvector of a matrix A which is 4 minus 5 2 minus 3, starting with, let us say, initial vector x₀ is equal to (0, 1), using the power method. So, this method is called power method, and why the name power method, because, you are

dealing with A to the power k for various values of k , that is how the iterates are defined, right?

(Refer Slide Time: 11:03)



```
power method.

In [ ]: from numpy import argmax, argmin
        A=matrix([[4,-5],[2,-3]])
        x0=vector([0.0,1.0]) # Initial guess of eigenvector
        maxit=30 # Maximum number of iterates
        dig=8 # number of decimal places to be shown is dig-1
        tol=0.0001 # Tolerance limit for difference of two consecutive eigenvectors
        err=1 # Initialization of tolerance
        i=0
        while(i<=maxit and err>=tol):
            y0=A*x0
            ymod=y0.apply_map(abs)
            imax=argmax(ymod)
            c1=y0[imax]; x1=y0/c1
            err=norm(x0-x1)
            i=i+1; x0=x1
            print("Iteration Number:", i-1)
            print("y"+str(i-1)+"=","y0.n(digits=dig), "c"+str(i-1)+"=","
                  c1.n(digits=dig), "x"+str(i)+"=","x0.n(digits=dig))
```

So, now let us write sage routine for this we will import two functions `argmax` and `argmin`. This will find the index for which a list has maximum value and similarly `argmin` will find the index for a list which is minimum right.

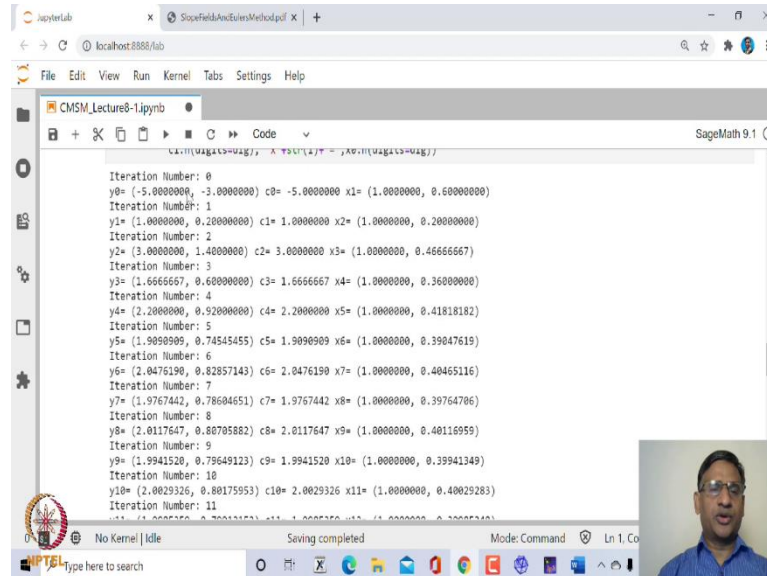
Now, so A is this matrix x naught is the starting vector. Let us look at we want to do maximum 30 number of iterates and since each computation is going to be numerical; and it will display so many decimal places. Let us restrict to only 8 decimal places in each of this coordinate and the tolerance limit is let us say 10 to the power minus 4 .

And so, I start with initial error which is 1 and initial iteration that is 0 and then, while the i is less than equal to maximum number of iterates which is 30 . And the error is bigger than equal to the tolerance limit you find out y naught is equal to A times x naught and find the maximum value of the y naught each coordinate of y naught. So, that is y naught dot apply map absolute value.

So, apply map apply underscore map will apply this function to each of entry of this list. And then define the maximum of the argument of this and then take the $c1$ to be the maximum value of this y naught and then define $x1$ to be y naught by $c1$ and then define the error term as x naught minus $x1$ in the norm.

And then increment this by iteration increment by 1 and re assign x naught x1 as x naught and then we are simply printing here this yi, ci and xi this is what is printed.

(Refer Slide Time: 13:08)



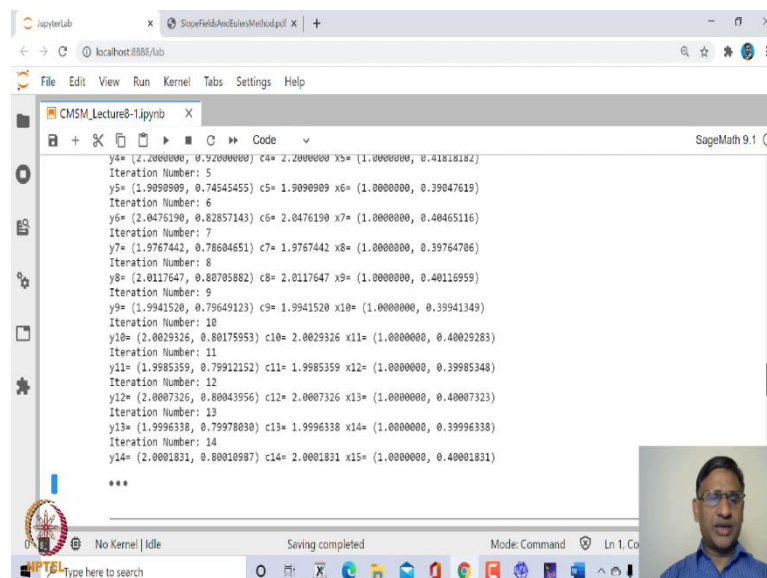
```

Iteration Number: 0
y0= (-5.0000000, -3.0000000) c0= -5.0000000 x1= (1.0000000, 0.6000000)
Iteration Number: 1
y1= (1.0000000, 0.2000000) c1= 1.0000000 x2= (1.0000000, 0.2000000)
Iteration Number: 2
y2= (3.0000000, 1.4000000) c2= 3.0000000 x3= (1.0000000, 0.4666667)
Iteration Number: 3
y3= (1.6666667, 0.6000000) c3= 1.6666667 x4= (1.0000000, 0.3600000)
Iteration Number: 4
y4= (2.2000000, 0.9200000) c4= 2.2000000 x5= (1.0000000, 0.4181818)
Iteration Number: 5
y5= (1.9090909, 0.7454545) c5= 1.9090909 x6= (1.0000000, 0.3904761)
Iteration Number: 6
y6= (2.0476190, 0.8285714) c6= 2.0476190 x7= (1.0000000, 0.4046511)
Iteration Number: 7
y7= (1.9767442, 0.7860465) c7= 1.9767442 x8= (1.0000000, 0.3976470)
Iteration Number: 8
y8= (2.0117647, 0.8070588) c8= 2.0117647 x9= (1.0000000, 0.4011695)
Iteration Number: 9
y9= (1.9941520, 0.7964912) c9= 1.9941520 x10= (1.0000000, 0.3994134)
Iteration Number: 10
y10= (2.0029326, 0.8017593) c10= 2.0029326 x11= (1.0000000, 0.4002928)
Iteration Number: 11
y11= (1.9985359, 0.7991215) c11= 1.9985359 x12= (1.0000000, 0.3998534)

```

So, let us run this program, and this is what you see, y 0 in the first iterate is this, and then the maximum value of that is minus 5, then you divide by minus 5 you get 1 comma 0.6.

(Refer Slide Time: 13:34)



```

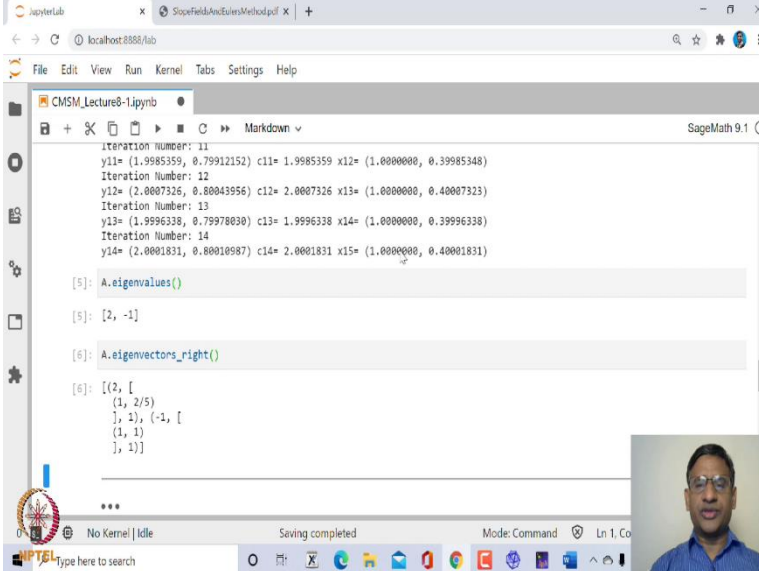
y4= (2.2000000, 0.9200000) c4= 2.2000000 y5= (1.9090909, 0.7454545)
Iteration Number: 5
y5= (1.9090909, 0.7454545) c5= 1.9090909 x6= (1.0000000, 0.3904761)
Iteration Number: 6
y6= (2.0476190, 0.8285714) c6= 2.0476190 x7= (1.0000000, 0.4046511)
Iteration Number: 7
y7= (1.9767442, 0.7860465) c7= 1.9767442 x8= (1.0000000, 0.3976470)
Iteration Number: 8
y8= (2.0117647, 0.8070588) c8= 2.0117647 x9= (1.0000000, 0.4011695)
Iteration Number: 9
y9= (1.9941520, 0.7964912) c9= 1.9941520 x10= (1.0000000, 0.3994134)
Iteration Number: 10
y10= (2.0029326, 0.8017593) c10= 2.0029326 x11= (1.0000000, 0.4002928)
Iteration Number: 11
y11= (1.9985359, 0.7991215) c11= 1.9985359 x12= (1.0000000, 0.3998534)
Iteration Number: 12
y12= (2.0007326, 0.8004305) c12= 2.0007326 x13= (1.0000000, 0.4000732)
Iteration Number: 13
y13= (1.9996338, 0.7997803) c13= 1.9996338 x14= (1.0000000, 0.3999633)
Iteration Number: 14
y14= (2.0001831, 0.8001097) c14= 2.0001831 x15= (1.0000000, 0.4000183)
***

```

And then next one is, define y1; y1 will be A times x1 and this is your y1, and then define c1, which is the maximum value, in this case 1, and so on.

So, this is what you get. It requires, in this case, 14 number of iterates. So, at 14th iterate, y_1 is this, and c_1 , c_k , that is, c_{14} is 2, which actually will be eigenvalue, and this is the x_{15} , that's x_k plus 1, this will be an eigenvector.

(Refer Slide Time: 13:57)



```

Iteration Number: 11
y1= (1.9985359, 0.79912152) c1= 1.9985359 x12= (1.0000000, 0.39985348)
Iteration Number: 12
y12= (2.0007326, 0.80043956) c12= 2.0007326 x13= (1.0000000, 0.40007323)
Iteration Number: 13
y13= (1.9996338, 0.79978030) c13= 1.9996338 x14= (1.0000000, 0.39996338)
Iteration Number: 14
y14= (2.0001831, 0.80010987) c14= 2.0001831 x15= (1.0000000, 0.40001831)

[5]: A.eigenvalues()

[5]: [2, -1]

[6]: A.eigenvectors_right()

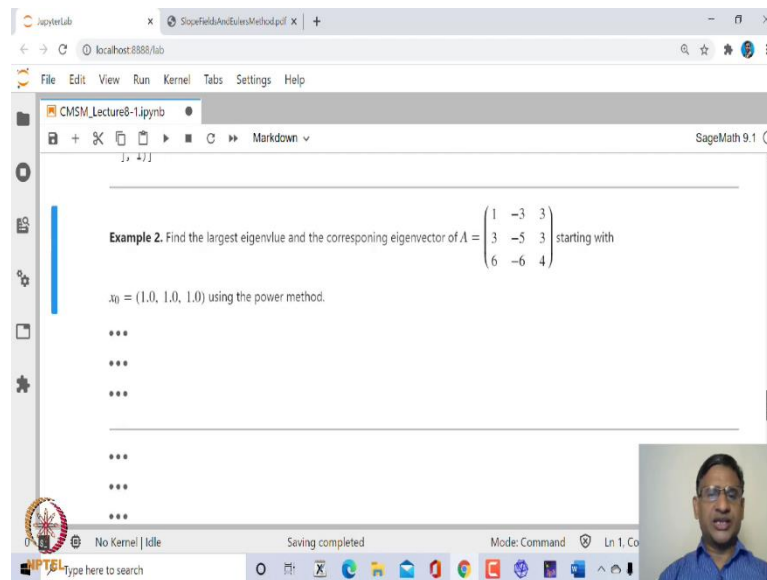
[6]: [(2, [(1, 2/5), (1, 1), (-1, 1), (1, 1)])]

```

So, that is and now, let us look at, what are the eigenvalues and eigenvectors of A . So, A eigenvalues are 2 comma minus 1, and that is what you are getting here c , c_k , and let us also find out, what are the eigenvectors. So, A dot eigen, eigenvectors right will give us the eigenvector; the eigenvector corresponding to eigenvalue 2 is 1 comma 2 by 5, 2 by 5 is 0.4.

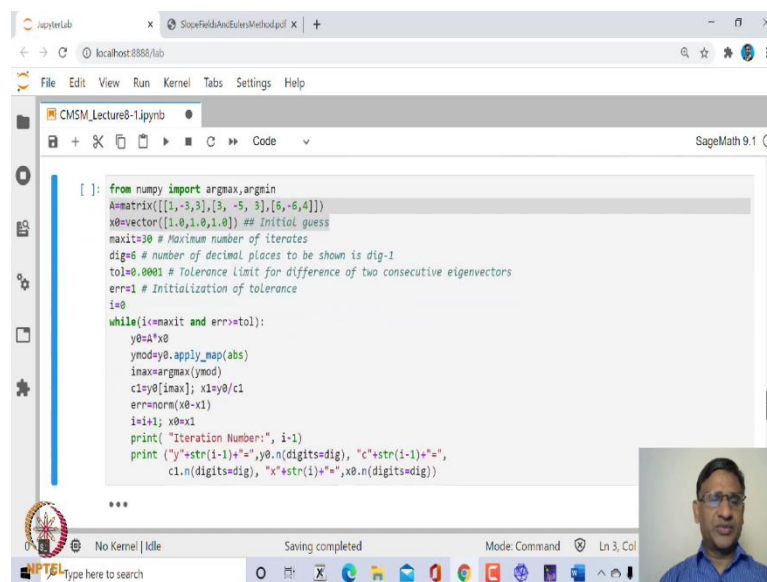
And so, that is what you can see here, this is 1 comma 0.4 approximately, right? So, that is how you can apply this method to find dominant eigenvalue and corresponding eigenvector.

(Refer Slide Time: 14:45)



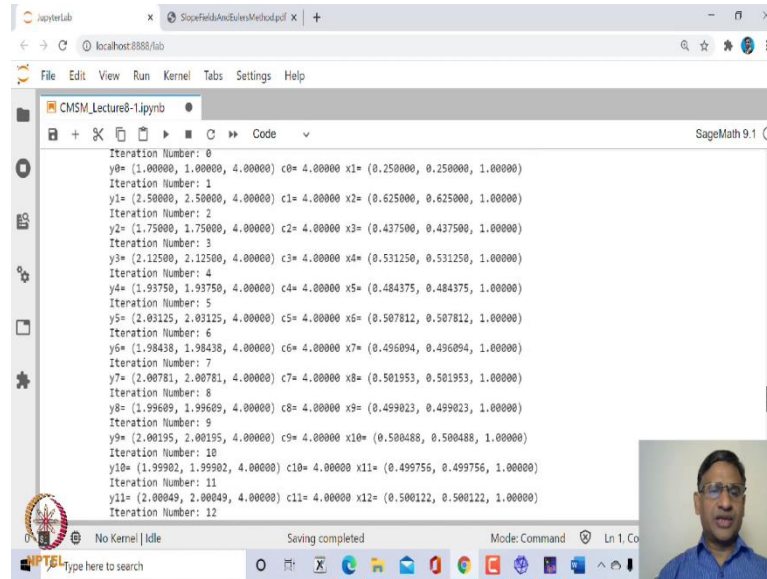
Let us look at one more example, slightly bigger example, 3 by 3. So, this is, matrix A in this case this is not symmetric, but one can check that it has all eigenvalues and dominant eigenvalues are, is strictly bigger than 1, right?

(Refer Slide Time: 15:01)



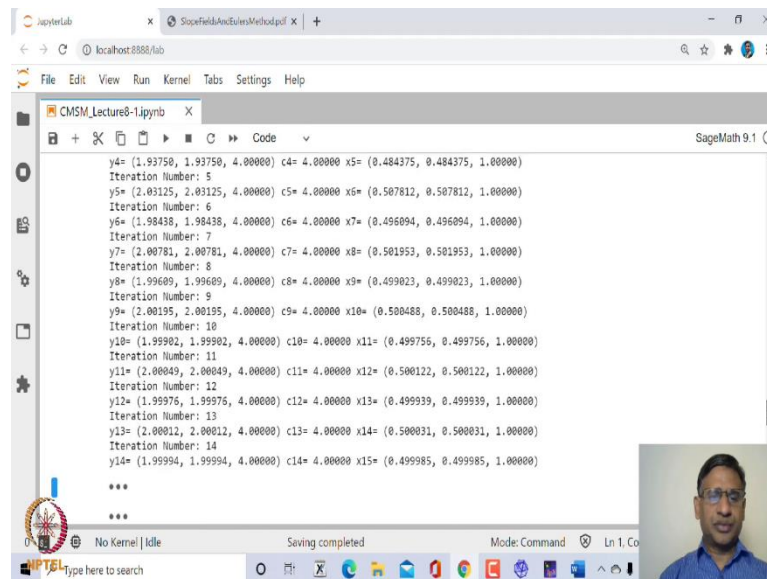
So, again we are just applying in, change in the previous syntax is just that we are changing the matrix A, and everything remains the same.

(Refer Slide Time: 15:14)



```
Iteration Number: 0
y0= (1.00000, 1.00000, 4.00000) c0= 4.00000 x1= (0.250000, 0.250000, 1.00000)
Iteration Number: 1
y1= (2.50000, 2.50000, 4.00000) c1= 4.00000 x2= (0.625000, 0.625000, 1.00000)
Iteration Number: 2
y2= (1.75000, 1.75000, 4.00000) c2= 4.00000 x3= (0.437500, 0.437500, 1.00000)
Iteration Number: 3
y3= (2.12500, 2.12500, 4.00000) c3= 4.00000 x4= (0.531250, 0.531250, 1.00000)
Iteration Number: 4
y4= (1.93750, 1.93750, 4.00000) c4= 4.00000 x5= (0.484375, 0.484375, 1.00000)
Iteration Number: 5
y5= (2.03125, 2.03125, 4.00000) c5= 4.00000 x6= (0.507812, 0.507812, 1.00000)
Iteration Number: 6
y6= (1.98438, 1.98438, 4.00000) c6= 4.00000 x7= (0.496094, 0.496094, 1.00000)
Iteration Number: 7
y7= (2.00781, 2.00781, 4.00000) c7= 4.00000 x8= (0.501953, 0.501953, 1.00000)
Iteration Number: 8
y8= (1.99609, 1.99609, 4.00000) c8= 4.00000 x9= (0.499023, 0.499023, 1.00000)
Iteration Number: 9
y9= (2.00195, 2.00195, 4.00000) c9= 4.00000 x10= (0.500488, 0.500488, 1.00000)
Iteration Number: 10
y10= (1.99902, 1.99902, 4.00000) c10= 4.00000 x11= (0.499756, 0.499756, 1.00000)
Iteration Number: 11
y11= (2.00049, 2.00049, 4.00000) c11= 4.00000 x12= (0.500122, 0.500122, 1.00000)
Iteration Number: 12
```

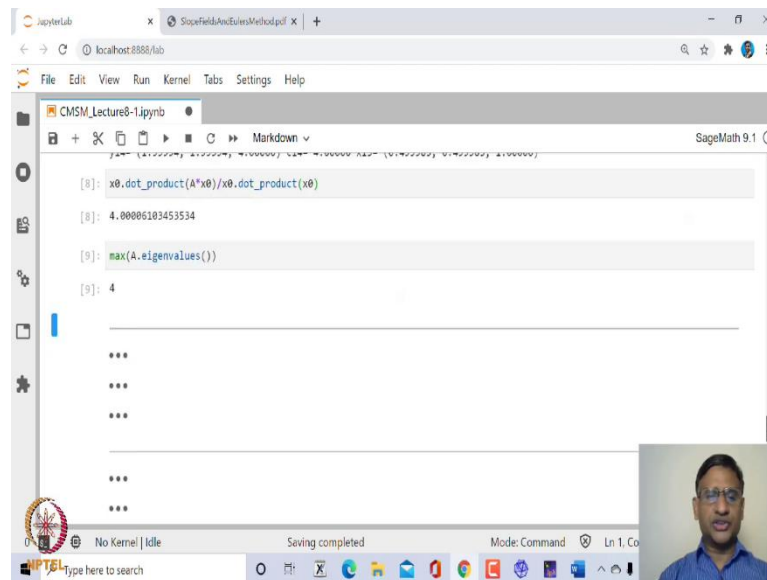
(Refer Slide Time: 15:17)



```
y4= (1.93750, 1.93750, 4.00000) c4= 4.00000 x5= (0.484375, 0.484375, 1.00000)
Iteration Number: 5
y5= (2.03125, 2.03125, 4.00000) c5= 4.00000 x6= (0.507812, 0.507812, 1.00000)
Iteration Number: 6
y6= (1.98438, 1.98438, 4.00000) c6= 4.00000 x7= (0.496094, 0.496094, 1.00000)
Iteration Number: 7
y7= (2.00781, 2.00781, 4.00000) c7= 4.00000 x8= (0.501953, 0.501953, 1.00000)
Iteration Number: 8
y8= (1.99609, 1.99609, 4.00000) c8= 4.00000 x9= (0.499023, 0.499023, 1.00000)
Iteration Number: 9
y9= (2.00195, 2.00195, 4.00000) c9= 4.00000 x10= (0.500488, 0.500488, 1.00000)
Iteration Number: 10
y10= (1.99902, 1.99902, 4.00000) c10= 4.00000 x11= (0.499756, 0.499756, 1.00000)
Iteration Number: 11
y11= (2.00049, 2.00049, 4.00000) c11= 4.00000 x12= (0.500122, 0.500122, 1.00000)
Iteration Number: 12
y12= (1.99976, 1.99976, 4.00000) c12= 4.00000 x13= (0.499939, 0.499939, 1.00000)
Iteration Number: 13
y13= (2.00012, 2.00012, 4.00000) c13= 4.00000 x14= (0.500031, 0.500031, 1.00000)
Iteration Number: 14
y14= (1.99994, 1.99994, 4.00000) c14= 4.00000 x15= (0.499985, 0.499985, 1.00000)
***
***
```

And then, when we run this program we will, we will get, we will get in this case again, we require 14th iterate, and 14 iterates, and in this case, the eigen, dominant eigenvector is 0.499 which is approximately 0.5; 0.5 and 1 and the eigenvalue is 4, right?

(Refer Slide Time: 15:37)



```
[8]: x0.dot_product(A*x0)/x0.dot_product(x0)
[8]: 4.00006103453534

[9]: max(A.eigenvalues())
[9]: 4

***

***

***

***

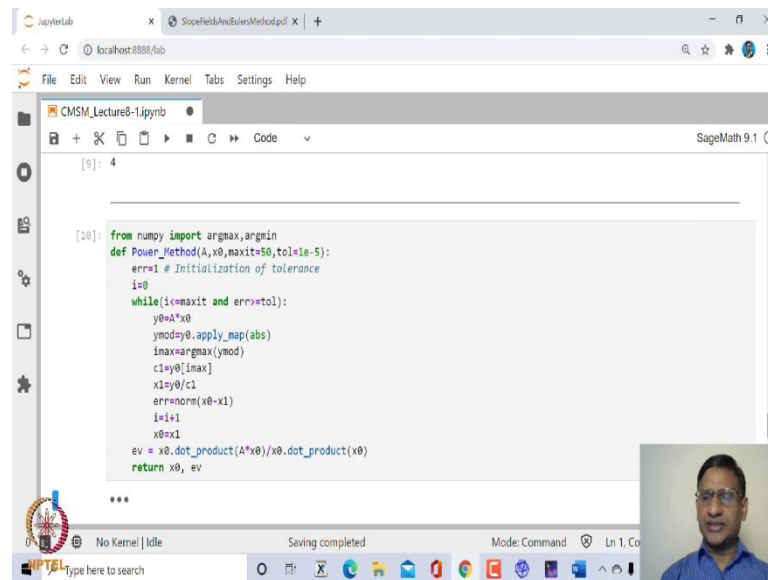
***
```

So, and this λ_k are giving you eigenvalue because, because we scaled at every stage, that, we looked at the maximum of that entry, and, ok. So, in case, this is your eigenvalue, eigenvector so, this eigenvector I have stored you can see here in x_0 , x_0 is, is redefined I mean x_1 is redefined in x_0 or stored in x_0 .

And therefore, we can find, you can, we can apply, let us say, this Rayleigh quotient, in order to compute eigenvalue, and in this case, the eigenvalue is 4.0061 and that is what you, you have here, right?

So, once you get eigenvector, then you can find corresponding eigenvalues using this Rayleigh quotient. And if you try to find out what is the maximum eigenvalue of A , then this is 4, right? So, this is another example, but if you, if you want, you can make a user-defined function for this power method.

(Refer Slide Time: 16:44)



```
[9]: 4

[10]: from numpy import argmax, argmin
def Power_Method(A, x0, maxit=50, tol=1e-5):
    err=1 # Initialization of tolerance
    i=0
    while(i<=maxit and err>=tol):
        y0=A*x0
        ymod=y0.apply_map(abs)
        imax=argmax(ymod)
        c1=y0[imax]
        x1=y0/c1
        err=norm(x0-x1)
        i=i+1
        x0=x1
    ev = x0.dot_product(A*x0)/x0.dot_product(x0)
    return x0, ev

***
```

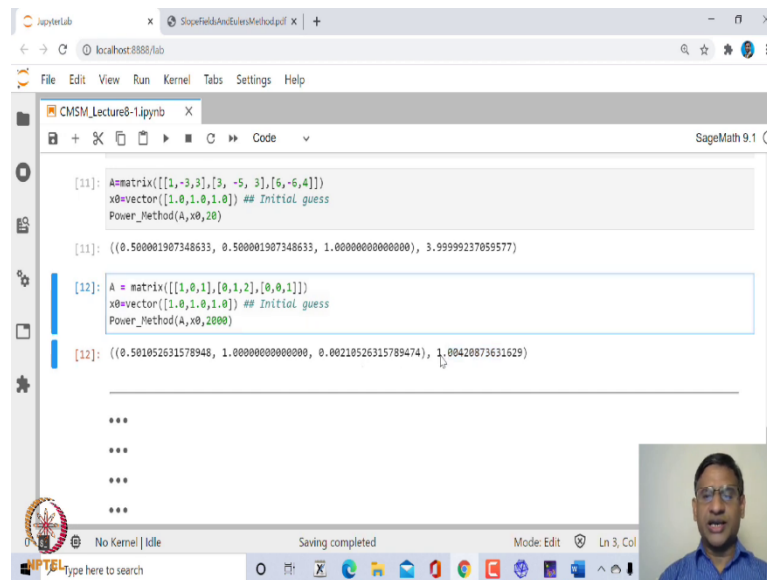
So, how do we do that? Everything remains same, except that instead of printing the values of y_k 's and c_k 's and x_k 's, we are just returning, at the end we are returning value of x_0 and the eigenvalue which is some eigenvalue which is computed using Rayleigh quotient.

Of course, in this case, we saw that this c_k itself will give me eigenvalue, so therefore, we can simply return eigenvalues as c_k , right? So, this is exactly same, except that I have put this in, in user-defined function, and the name of this is power underscore method.

So, input A, the matrix A, x_0 is the, the initial guess vector, and maximum number of iteration, iteration is less than 50, tolerance limit is $1e-5$, that is 10 to the power minus 5 , error, initial error is 1 , starting value is 0 .

Now, this is the maximum number of iterate; and as long as the maximum number of iterate is less than equal to, i is less than equal to maximum number of iterate, and the error is bigger than equal to the tolerance limit, you compute this, and let me run this.

(Refer Slide Time: 18:05)



```
[11]: A=matrix([[1,-3,3],[3,-5,3],[6,-6,4]])
x0=vector([1.0,1.0,1.0]) ## Initial guess
Power_Method(A,x0,20)

[11]: ((0.500001907348633, 0.500001907348633, 1.00000000000000), 3.99995237059577)

[12]: A = matrix([[1,0,1],[0,1,2],[0,0,1]])
x0=vector([1.0,1.0,1.0]) ## Initial guess
Power_Method(A,x0,2000)

[12]: ((0.501052631578948, 1.00000000000000, 0.00210526315789474), 1.00420873631629)

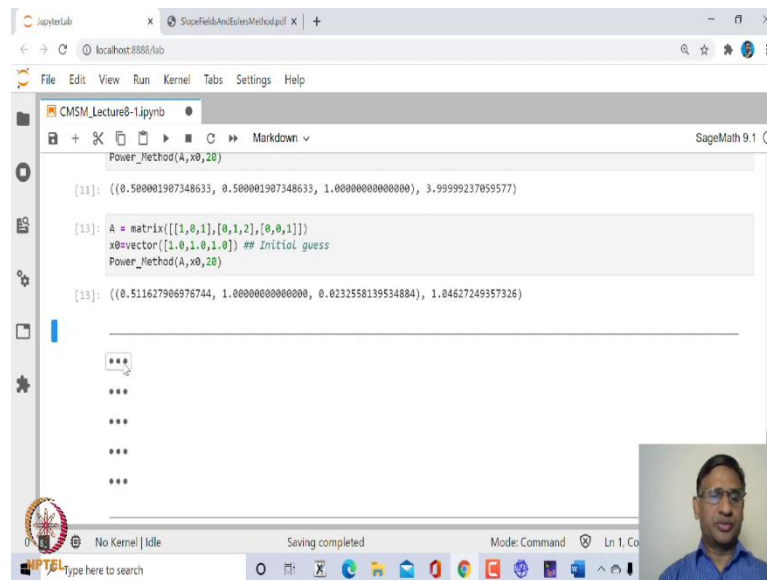
***
***
***
***
```

And let us call this function for a matrix A which is 1 minus 3 1, 3 minus 5 3, 6 minus 6 1, and so on. So, let us run this. So, in this case, the eigenvector is this, and the maximum eigenvalue is 3.999. So, this is the, what you get. And if you change this, let us say I have this matrix 1, 0, 1 and 0, 1, 2; 0, 0, 1.

So, this is a triangular, upper triangular matrix, and the diagonal entries are 1, 1, and 1. So, this means all eigenvalues, in this case, are 1. So, this does not have dominant eigenvalue, ok? So, in this case, if you apply this power method, it will still work, but however, the convergence will be very slow. So, in, actually in case of power method, the convergence depends upon the ratio of λ_2 by λ_1 .

So, if λ_2 by λ_1 , if λ_2 is small, much smaller than λ_1 , that will, will go to 0 much faster, right? So, λ_2 upon λ_1 to the power k will go to 0 quite quickly in case λ_2 is smaller, much smaller than λ_1 . So, and in this case, since all our λ s are same, the convergence will be very very slow. For example, you have to run here 2000 iterates, and still it is not very close to 1, right?

(Refer Slide Time: 19:35)



```
Power_Method(A,x0,20)

[11]: ((0.500001907348633, 0.500001907348633, 1.000000000000000), 3.9999923705577)

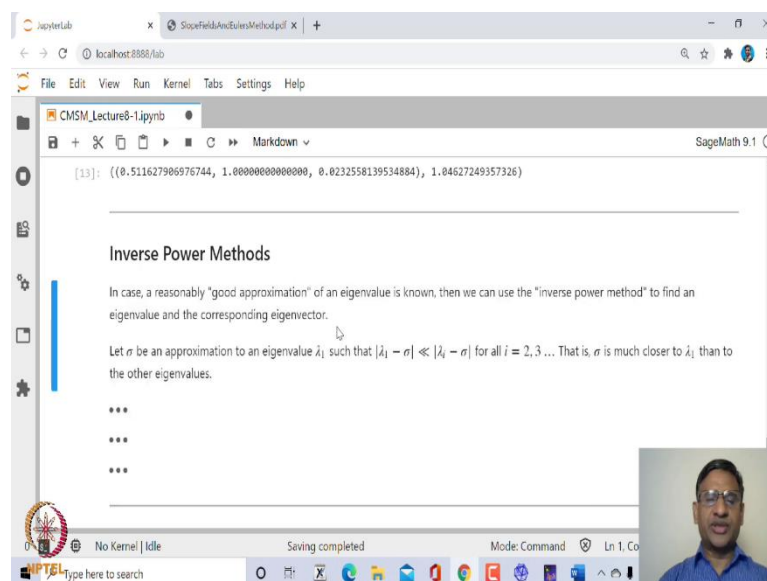
[13]: A = matrix([[1,0,1],[0,1,2],[0,0,1]])
      x0=vector([1,0,1,0]) ## Initial guess
      Power_Method(A,x0,20)

[13]: ((0.511627906976744, 1.000000000000000, 0.0232558139534884), 1.04627249357326)

***
***
***
***
***
```

If I take only 20 iterates, then you have 1.04 which is not very close to 1 numerically, right? So, this is how you compute dominant eigenvalue and eigenvectors, corresponding eigenvector, using power method. And as I said, this dominant eigenvalue of a matrix has several applications, including Google search, and even the twitter uses this, and several other algorithm in machine learning like, dimensionality reductions etcetera uses this, this particular notion.

(Refer Slide Time: 20:16)



```
[13]: ((0.511627906976744, 1.000000000000000, 0.0232558139534884), 1.04627249357326)

Inverse Power Methods

In case, a reasonably "good approximation" of an eigenvalue is known, then we can use the "inverse power method" to find an eigenvalue and the corresponding eigenvector.

Let  $\sigma$  be an approximation to an eigenvalue  $\lambda_1$  such that  $|\lambda_1 - \sigma| \ll |\lambda_i - \sigma|$  for all  $i = 2, 3 \dots$ . That is,  $\sigma$  is much closer to  $\lambda_1$  than to the other eigenvalues.

***
***
***
```

Now, suppose you wanted to find, let us say, the smallest eigenvalue. And let us assume that the matrix is invertible, therefore, the all eigenvalues are, are non-zero, therefore, it will have a smallest and largest eigenvalue, of course, in mod, and in case we assume that all eigenvalues are real so, therefore. So in, and you know that eigenvalue of A inverse is $1/\lambda$ upon eigenvalue of A ; if λ is an eigenvalue of A then $1/\lambda$ is an eigenvalue of A inverse.

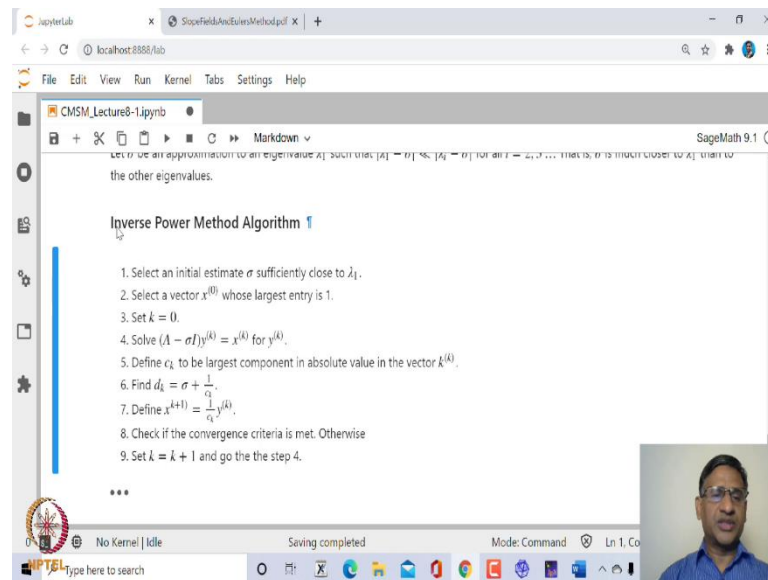
So, if you look at eigenvalues of, if you find dominant eigenvalue of A inverse, that will be nothing but the smallest eigenvalue of A , right? So, you can apply power method to A inverse in order to find the smallest eigenvalue of A .

However, finding inverse of a matrix, especially for large matrix is quite expensive task. So, in that case, finding inverse and finding the largest eigenvalue numerically will be somewhat quite expensive. So, in that case, one can apply what is called inverse power method, and how does it work?

So, in this case, what we look at is, first, we look at some approximation of eigenvalues, of eigenvalues, and then, then we, what do we do? Suppose σ_1 is an approximation of eigenvalue λ_1 which is the largest eigenvalue. And then we assume that this σ_1 is away, far away from remaining sigmas, that is why this modulus of $\lambda_1 - \sigma_1$ is less than equal to $|\lambda_i - \sigma_1|$ in mod for all i between 2 to n , right?

So, it simply means that σ_1 is much closer to λ_1 than any other, right? So, this inverse power method is based on this notion.

(Refer Slide Time: 22:23)



Now, if you look at, for example, if, look at, if λ_1 is an eigenvalue of A , then $\lambda_1 - \sigma$ will be an eigenvalue of $A - \sigma I$; A minus σ times identity.

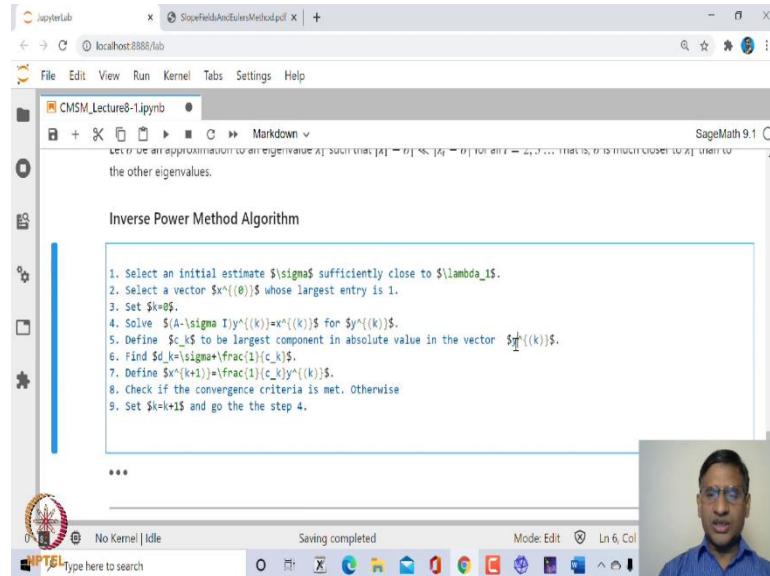
So, one has to apply this power method to, to $A - \sigma I$, the inverse of that actually, right? So, what is, how this algorithm works? So, let me tell you this, the inverse power method algorithm goes as follows. Start with, again initial estimate, let us say σ , it should be sufficiently close to λ_1 , right?

Now, in, in, in principle, how do we select such thing? That one has to do some experiment and then, then find out this, and one can look at what is the convergence criteria for convergence of these, these algorithms. Now, then you look at, start with x_0 , again whose largest entry is 1, a vector x_0 with the largest entry as 1, set k equal to 0, and then solve $(A - \sigma I)y_k = x_k$.

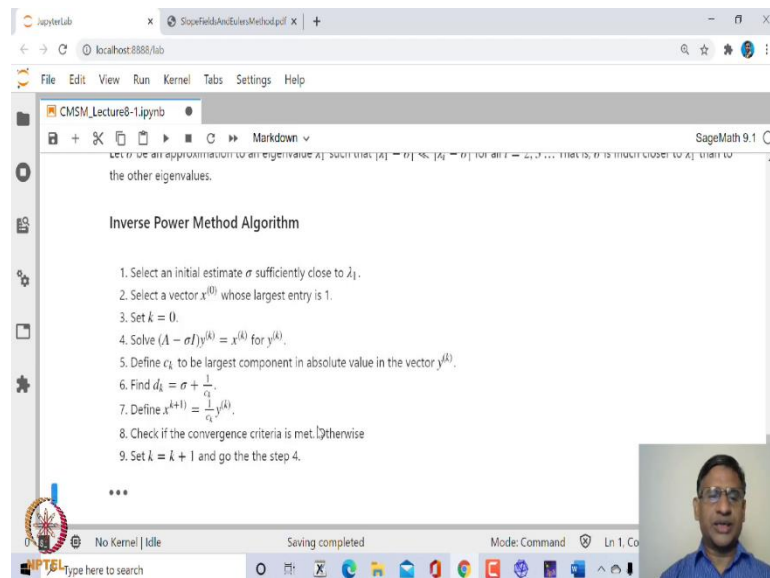
So, you are solving this particular equation at every stage, again you can see here, when we want to solve this, solving does not always involve finding inverse ok, solving can be done using other algorithms which does not require inverse, right? So, instead of finding inverse, we are just trying to solve $(A - \sigma I)y_k = x_k$, and this is, this you solve for y_k and

then define c_k to be again the largest component of the, the absolute value in the vector, in the vector it should be y_k , in the vector y_k .

(Refer Slide Time: 24:29)



(Refer Slide Time: 24:33)

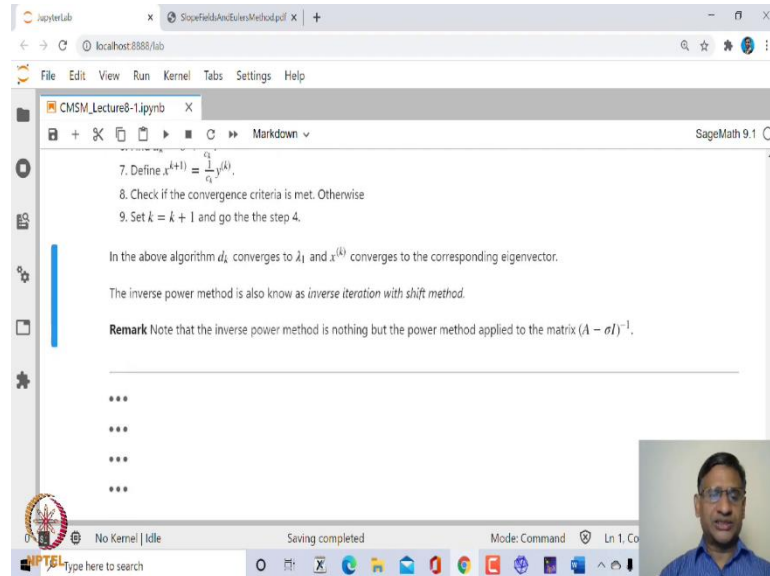


And then, define d_k to be σ plus 1 upon c_k , and this c_k as k goes to infinity, as k becomes larger and larger, this, this actually will converge to the largest eigen, the, the eigenvalue, right?

And define x_{k+1} as $\frac{1}{c_k} y_k$ and then check if the convergence criteria is met. In case that convergence criteria is met, you stop, otherwise, increment this iteration by 1,

k equal to $k + 1$, and go to step number 4, that is, again solve $(A - \sigma I)y = x_k$ for y . So, that is the algorithm.

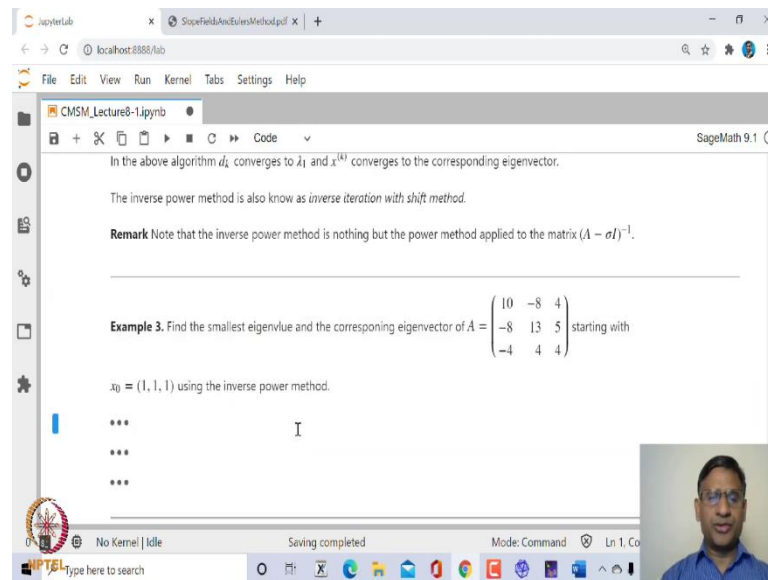
(Refer Slide Time: 25:16)



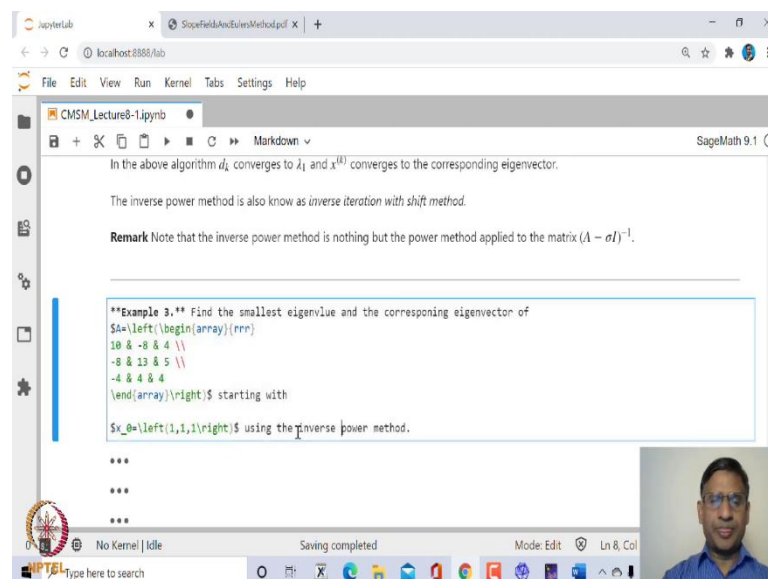
And this algorithm, let us look at how we can implement. So, in this case, this α_k will converge to λ_1 and this x_k will converge to the corresponding eigenvector, right? This is, since here we are using inverse of this, this matrix in some sense, that is why it is called inverse iteration with shift method; shift in the sense, here we are using, we are shifting λ by σ , minus σ actually, $\lambda - \sigma$. So, that is why this shift method, right?

And if you can notice that the inverse power method is nothing but the power method applied to the matrix $A - \sigma I$ to the power minus 1, that is inverse of this matrix, right? So, let us implement this in Sage.

(Refer Slide Time: 26:06)

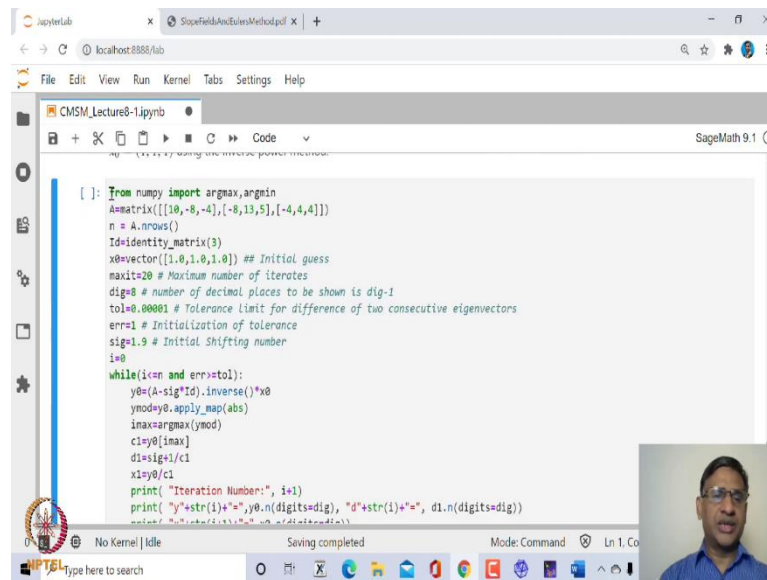


So, let us take a matrix A, which is this 10, minus 8, 4, minus 8, 13, 5, minus 4, 4, 4 and find out its, find out its smallest eigenvalues, in this case, we have to say smallest eigenvalue. (Refer Slide Time: 26:23)



Find out the smallest eigenvalue and the corresponding eigenvector of A right, using inverse power method; using inverse power method; using inverse power method, right, ok.

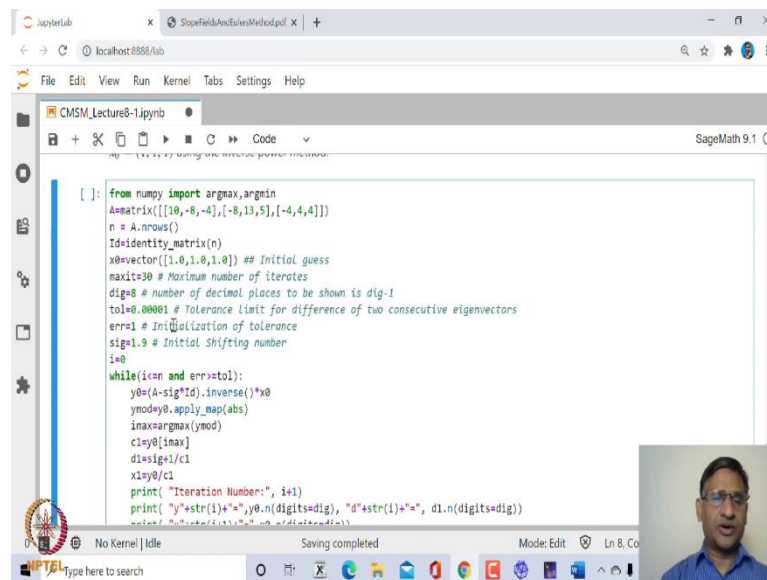
(Refer Slide Time: 26:41)



```
[ ]: from numpy import argmax, argmin
A=matrix([[10,-8,-4],[-8,13,5],[-4,4,4]])
n = A.nrows()
Id=identity_matrix(3)
x0=vector([1,0,1,0]) # Initial guess
maxit=20 # Maximum number of iterates
dig=8 # number of decimal places to be shown is dig-1
tol=0.00001 # Tolerance limit for difference of two consecutive eigenvectors
err=1 # Initialization of tolerance
sig=1.9 # Initial Shifting number
i=0
while(i<n and err>tol):
    y0=(A-sig*Id).inverse()*x0
    ymod=y0.apply_map(abs)
    imax=argmax(ymod)
    c1=y0[imax]
    d1=sig+i/c1
    x1=y0/c1
    print("Iteration Number:", i+1)
    print("y"+str(i)+"=", y0.n(digits=dig), "d"+str(i)+"=", d1.n(digits=dig))
    err=(c1-d1)/(d1-d1)
```

So, again the algorithm will be exactly very similar to what we, we saw, we, we import argmin and argmax, this is the matrix, let us say n is the number of rows in A and then define the identity matrix, in this case I should say, actually n here this is the initial vector which is 1, 1, 1 and maximum number of iterate we have taken as 20.

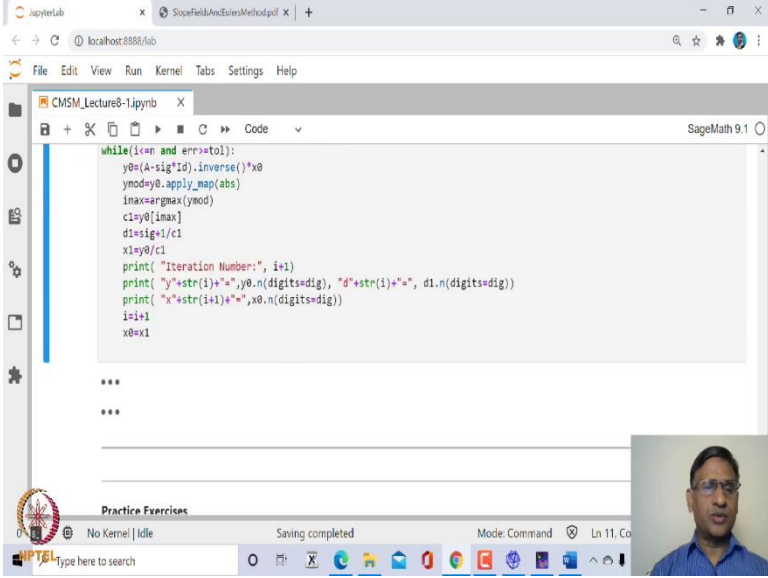
(Refer Slide Time: 27:06)



```
[ ]: from numpy import argmax, argmin
A=matrix([[10,-8,-4],[-8,13,5],[-4,4,4]])
n = A.nrows()
Id=identity_matrix(n)
x0=vector([1,0,1,0]) # Initial guess
maxit=30 # Maximum number of iterates
dig=8 # number of decimal places to be shown is dig-1
tol=0.00001 # Tolerance limit for difference of two consecutive eigenvectors
err=1 # Initialization of tolerance
sig=1.9 # Initial Shifting number
i=0
while(i<n and err>tol):
    y0=(A-sig*Id).inverse()*x0
    ymod=y0.apply_map(abs)
    imax=argmax(ymod)
    c1=y0[imax]
    d1=sig+i/c1
    x1=y0/c1
    print("Iteration Number:", i+1)
    print("y"+str(i)+"=", y0.n(digits=dig), "d"+str(i)+"=", d1.n(digits=dig))
    err=(c1-d1)/(d1-d1)
```

If you want you can take, let us say, 30, number of digits is 8, tolerance is 10 power minus 5, error, initial error is 1, and sigma value you start, with let us say, 1.9.

(Refer Slide Time: 27:22)



```
while(i<n and err>tol):
    y0=(A-sig*Id).inverse()*x0
    ymod=y0.apply_map(abs)
    imax=argmax(ymod)
    c1=y0[imax]
    d1=sig+1/c1
    x1=y0/c1
    print("Iteration Number:", i+1)
    print("y"+str(i)+"=", y0.n(digits=dig), "d"+str(i)+"=", d1.n(digits=dig))
    print("x"+str(i)+"=", x0.n(digits=dig))
    i=i+1
    x0=x1
    ...
    ...
```

And then, i is equal to, start with i equal to 0, and y_i is less than equal to n , and error bigger than equal to tolerance limit, you compute y is equal to A minus σ times identity the inverse of that into x_0 .

Now, here I, though I have written the inverse of, find the inverse, but you could apply some other method to solve this, ok. Since this is not a very big example, this is only 3 by 3 matrix, finding inverse may not be all that difficult.

But in principle, if you want to implement this for any generic matrix, you should avoid this, finding inverse, right; and then again define $y \bmod$ is the modulus of all the entries of y_i , maximum is the argument of the maximum, c_1 as y_1 that is y_0 upon the maximum of the y_i , define d_1 as $\sigma + 1$ upon c_1 , and x_1 as y_0 upon c_0 . And then print all these y_0 , c , y_0 , c_0 and x_1 , and then, then increment i by 1, and then redefine, or x_1 you store it in x_0 , and then continue this loop, ok?

(Refer Slide Time: 28:42)

```

print("Iteration Number:", i+1)
print("y"+str(i)+"=", y0.n(digits=dig), "d"+str(i)+"=", d1.n(digits=dig))
print("x"+str(i)+"=", x0.n(digits=dig))
i=i+1
x0=x1

Iteration Number: 1
y0= (3.0273924, -3.8029171, 13.486304) d0= 1.9741493
x1= (1.0000000, 1.0000000, 1.0000000)
Iteration Number: 2
y1= (1.7493556, -3.3806960, 10.247717) d1= 1.9975827
x2= (0.22447903, -0.28198365, 1.0000000)
Iteration Number: 3
y2= (1.6724019, -3.3366981, 10.017333) d2= 1.9998270
x3= (0.17070686, -0.32989747, 1.0000000)
Iteration Number: 4
y3= (1.6670698, -3.3335702, 10.001219) d3= 1.9999878
x4= (0.16695061, -0.33309245, 1.0000000)

[15]: A.eigenvalues()
[15]: [2, 3.321220124657091?, 21.67877987534291?]

```

So, let us run this, when you run this, in this case, it requires only 4 iterates, iterates and in this case we, we have obtained in 4 and 4th iterate this, this is the eigenvector and this should be eigenvalue. So, let us look at what are the eigenvalues of A, the smallest eigenvalues of A is 2, you can see here, and you can also find out what are the eigenvectors of A.

(Refer Slide Time: 29:11)

```

Iteration Number: 3
y2= (1.6724019, -3.3366981, 10.017333) d2= 1.9998270
x3= (0.17070686, -0.32989747, 1.0000000)
Iteration Number: 4
y3= (1.6670698, -3.3335702, 10.001219) d3= 1.9999878
x4= (0.16695061, -0.33309245, 1.0000000)

[16]: A.eigenvalues()
A.eigenvectors_right()

[16]: [(2, [(1, -2, 6), 1]), (3.321220124657091?, [(1, 1.084847484417864?, -1/2)], 1), (21.67877987534291?, [(1, -1.209847484417864?, -1/2)], 1)]

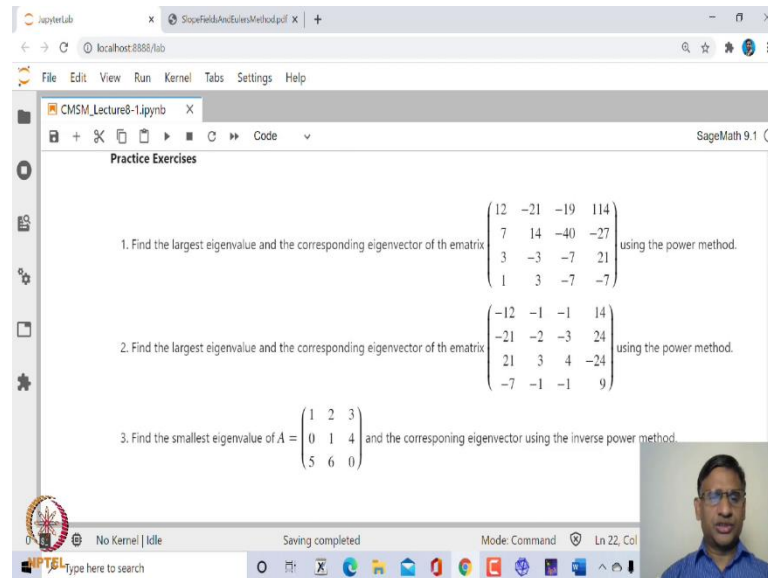
Exercise. Explore the following methods and create Sage routines.
• Rayleigh Quotient Iteration Algorithm
• QR Method

```

So, let us find out A dot eigenvectors, and so, the eigenvalue corresponding to eigenvector; eigenvector corresponding eigenvalue 2 is 1 minus 2 6. And in this case, actually this is,

since it is divided by 6, the maximum value you can see here, this is 1, and this will be minus 2 by 6, which is minus 1 by 3, and this is 1 by 6, right?

(Refer Slide Time: 29:49)



So, that is what you, you have got, right? There are other methods which one can use in order to find eigenvalues, eigenvector; one is Rayleigh quotient iterations algorithm. So, it uses Rayleigh quotient and in this case, in fact, one can also find other eigenvalues and eigenvectors. And then the other one is called QR method, which uses QR factorization and in this case, actually you can find all eigenvalues and eigenvectors, corresponding eigenvectors together.

Although, this, this method is quite expensive because at every stage you need to compute QR factorization of a matrix. So, that is quite expensive, so, but it, it can, in case of a small matrix, you can, you can try out.

So, I, I expect you all to go through this method, look at help on these methods, and try to implement it in Sage, right? Let me leave you with two-three simple exercises. Find the largest eigenvalue and corresponding eigenvector of this matrix using power method, and this is the second problem, next one is find the smallest eigenvalue of A and the corresponding eigenvector using inverse power method.

So, these are the three simple exercises we, we have already written user-defined function for each of this, though we have not written user-defined function for inverse power

method, but I am sure we can, you can just make small change in this, this syntax and convert into user-defined function, ok?

So, let me stop here, this is how we find numerically eigenvalues and eigenvector, which are dominant eigenvalues and eigenvectors, and it has several applications in science and engineering.

Thank you very much.