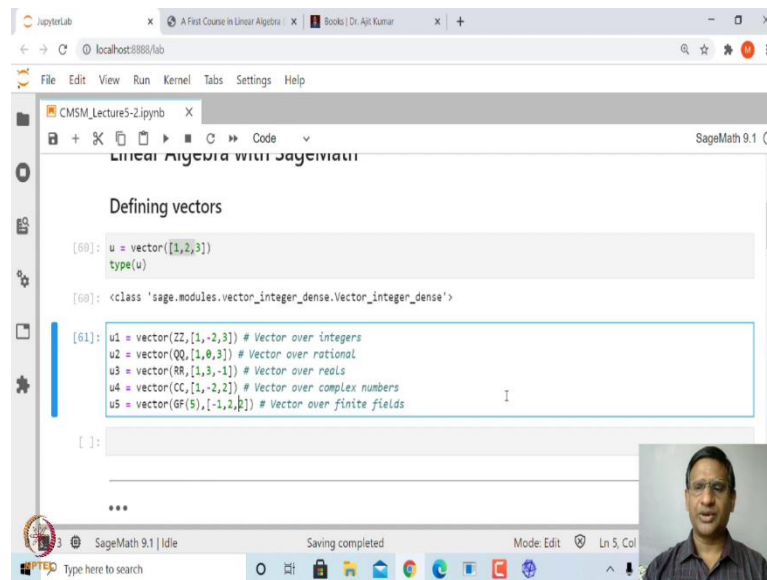


Computational Mathematics with SageMath
Prof. Ajit Kumar
Department of Mathematics
Institute of Chemical Technology, Mumbai

Linear Algebra with SageMath
Lecture – 29
Working with vectors in SageMath

Welcome to the 29th lecture on Computational Mathematics with SageMath. In this lecture, we will start exploring concepts in Linear Algebra. So, first, let us look at basic objects in linear algebra, for example, vectors, and matrices. So, let us see how we can define vectors in SageMath, and do various operations or computations using vectors. (Refer Slide Time: 00:47)



```
[60]: u = vector([1,2,3])
type(u)

[60]: <class 'sage.modules.vector_integer_dense.Vector_integer_dense'>

[61]: u1 = vector(ZZ,[1,-2,3]) # Vector over integers
u2 = vector(QQ,[1,0,3]) # Vector over rational
u3 = vector(RR,[1,3,-1]) # Vector over reals
u4 = vector(CC,[1,-2,2]) # Vector over complex numbers
u5 = vector(GF(5),[-1,2,0]) # Vector over finite fields

[ ]:
```

So, the easiest way of defining vectors in SageMath is to declare vector, in the square bracket, write their entries.

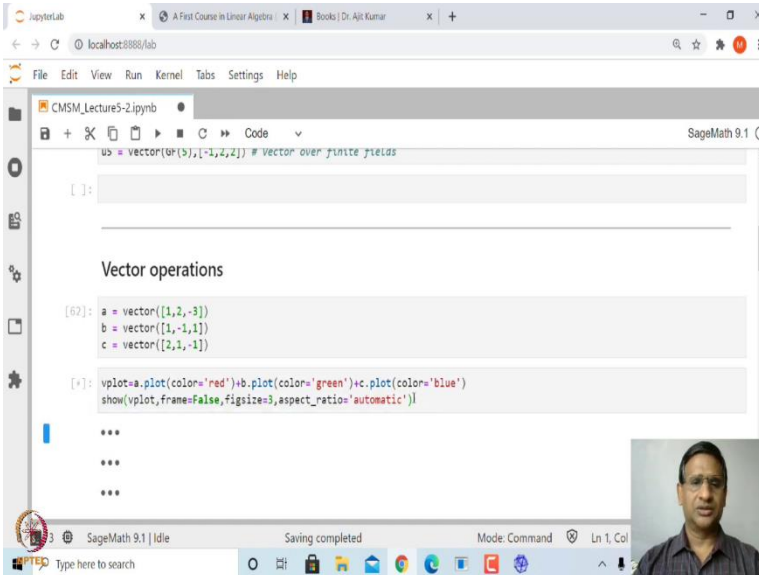
So, for example, in this case, 1, 2, 3 are the coordinates of this vector u , and here 1, 2, 3 are integers. So, if I declare this, and ask what is the type of u , it says that it is a vector coming from module called vector underscore integer, and dense, right? So, this is very similar to how we declare array in numpy, except that there we had array, now we have

vector, and everything else is the same. You can also mention the domain from which entries are taken.

So, by default, here all these entries are integers, so it says that it is vector, integers vectors, right? Next, you can declare entries coming from rational numbers, from real numbers, from complex number, even from finite fields. So, several options are there in order to give the domain from which the entries are defined for any vector, right?

So, these are, for example, if I, if I now, if I ask, we have declared u_5 as a vector over finite field F be 5. So, this will take all the integers modulo 5, it will give, only entries will be integers having entries modulo, in modulo 5, right? So, let us see various operations that can be, can be done on vectors.

(Refer Slide Time: 02:47)



The screenshot shows a JupyterLab window with a SageMath 9.1 kernel. The code in the cell is as follows:

```
u5 = vector(GF(5), [-1, 2, 2]) # vector over finite field F5

[ ]:
```

Below the code, the output shows the title "Vector operations" and the execution of three vectors:

```
[62]: a = vector([1, 2, -3])
      b = vector([1, -1, 1])
      c = vector([2, 1, -1])
```

The next line shows a plot command:

```
[*]: vplot=a.plot(color='red')+b.plot(color='green')+c.plot(color='blue')
      show(vplot, frame=False, figsize=3, aspect_ratio='automatic')]
```

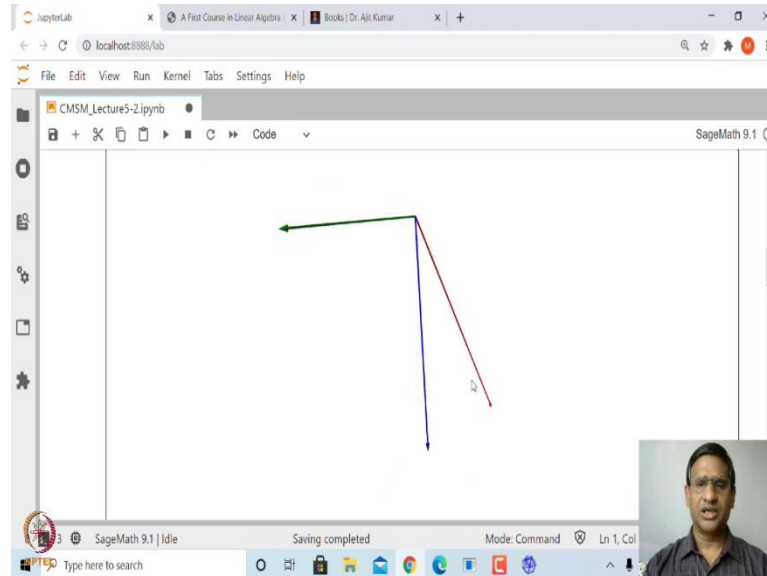
The output shows three asterisks (***) indicating the plot was displayed. A small video inset of a man is visible in the bottom right corner of the JupyterLab window.

So, for example, let us look at three vectors a , b , c . Let us declare these three vectors a, b, c . Again, I have not mentioned the domain from which each of these vectors are taken, and since, these are integers, it will be vectors over integers. Now, first thing you can do is, you can plot the graph of a vector.

So, any vector which is declared in two, and three dimensions, you can plot using same plot function. So, if I say a dot plot it will plot vector a , I am mentioning the colour red, b dot plot will plot vector b , and c dot plot will plot vector c , and then ask it to show this,

these three vectors. So, when you run this, it will plot graph of these three vectors you can rotate again, and see.

(Refer Slide Time: 03:33)



So, red one is the vector a , green vector b , and blue one is vector c . You can reduce the size of this vector, and rotate, ok? So, this is how we can plot graph of any vector, and visualize, right?

(Refer Slide Time: 03:55)

```
[64]: 2*a-3*b+4*c
[64]: (7, 11, -13)
[66]: a.norm(2) # Euclidean norm (length)
[66]: sqrt(14)
...
...
```

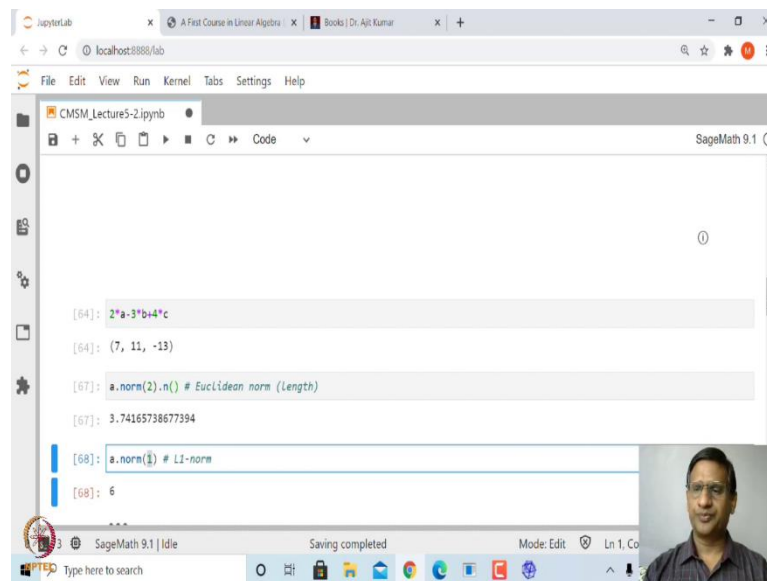
Now, suppose you want to do, let us say scalar multiple of a vector, and add to another vector. In particular, you can take any linear combination of vectors. So, here we are asking

for 2 times a plus 3 times b minus 4 times c, or let us say 2 times a minus, minus 3 times b plus 4 times c. This gives me this vector.

So, any linear combination, a scalar linear combination of vectors can be obtained. Similarly, you can find length of a vector using a function called norm. A method called norm on a vector, right? So, a dot norm so, for example, let me, let me first say a dot norm, if you look at the vector, a is 1, 2, minus 3.

So, the norm of this is going to be square root of 1 square, plus 2 square plus, minus 3 square. So, it should come out as square root 14, right? So, this is what is called Euclidean norm. So, you can even mention inside the bracket 2, that is a Euclidean norm, it is also known as L2 norm, right?

(Refer Slide Time: 05:11)

A screenshot of a JupyterLab web interface. The browser address bar shows 'localhost:8888/lab'. The JupyterLab window has a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help) and a toolbar. The main area displays a code editor with the following SageMath code:

```
[64]: 2*a-3*b+4*c  
[64]: (7, 11, -13)  
[67]: a.norm(2).n() # Euclidean norm (length)  
[67]: 3.74165738677394  
[68]: a.norm(1) # L1-norm  
[68]: 6
```

The output for the L1 norm is highlighted in blue. At the bottom right, there is a small video feed of a man. The Windows taskbar is visible at the bottom with various icons and the text 'SageMath 9.1 | Idle' and 'Saving completed'.

If you want to find the numerical values you know what needs to be done, you can say dot n. Similarly, you can find L1 norm, which is actually sum of the modulus of the each entry, this is called L1 norm. So, if you are, in the bracket if you mention 1, it will give you sum of the mod of each entry or each component.

(Refer Slide Time: 05:31)

```
[68]: 6
[69]: a.norm(infinity) # Sup norm
[69]: 3

Dot Product

[70]: a,b
[70]: ((1, 2, -3), (1, -1, 1))
[71]: a.dot_product(b) ## or a*b
[71]: -4
```

Similarly, you can find what is called Sup norm, which is by giving option infinity in the bracket. So, this is actually maximum of the modulus of each entry. So, that should be 3, right?

So, this is how you can find length of a vector, and you can verify various properties of length. For example, length of vectors satisfy triangle law of inequality; so, length of a plus b will be less than equal to length of a plus length of b, right? You can also find dot product of 2 vectors. So, if you have two vectors a, and b, which we have already declared; a is this, b is this. And if I look at what is dot product of a and b, so, you can simply say a dot, dot underscore product, in the bracket you write b. So, this is a dot b. (Refer Slide Time: 06:29)

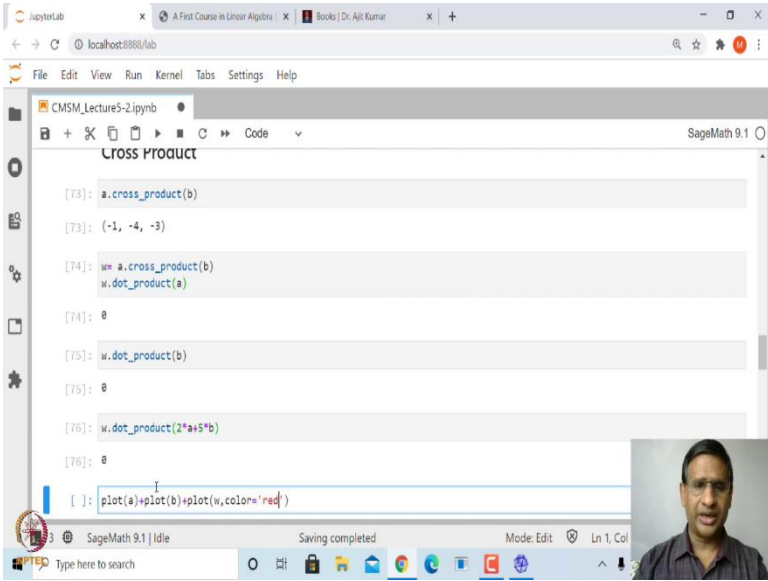
```
Dot Product

[70]: a,b
[70]: ((1, 2, -3), (1, -1, 1))
[71]: a.dot_product(b) ## or a*b
[71]: -4

[72]: a*b
[72]: -4
```

You could also write dot product as a star b. So, if I say a star b, it will give me dot product of a and b, right? So, you can mention a star b or a dot dot product. This is more natural, because many times when you write a star b, it could be just entry-wise multiplication in many other packages, but here gives me dot product. So, you see, Sage is somewhat intelligent. It understands the context, and based on that, it does appropriate operation. Similarly, you can find cross-product of two vectors.

(Refer Slide Time: 07:01)



The screenshot shows a JupyterLab window with a SageMath 9.1 kernel. The code in the cell is as follows:

```

[73]: a.cross_product(b)
[73]: (-1, -4, -3)

[74]: w = a.cross_product(b)
      w.dot_product(a)

[74]: 0

[75]: w.dot_product(b)

[75]: 0

[76]: w.dot_product(2*a+5*b)

[76]: 0

[ ]: plot(a)+plot(b)+plot(w,color='red')

```

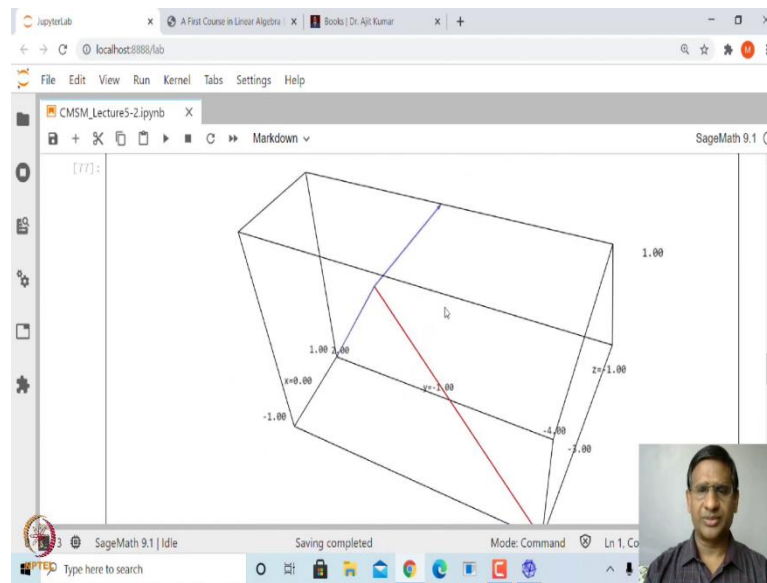
The output of the code shows the cross product of vectors a and b is (-1, -4, -3). The dot product of this result with a is 0, and with b is 0, confirming it is perpendicular to both. The final line of code is partially visible, showing a plot command.

So, a and b are, we already know what are a and b. If I say a dot cross underscore product, it will give me cross product of two, two vectors. So, for example, I am sure you know that if I take cross product of a and b, it is a vector quantity, and it lies along the direction which is perpendicular to a and b, and the direction is given by the right-hand thumb rule.

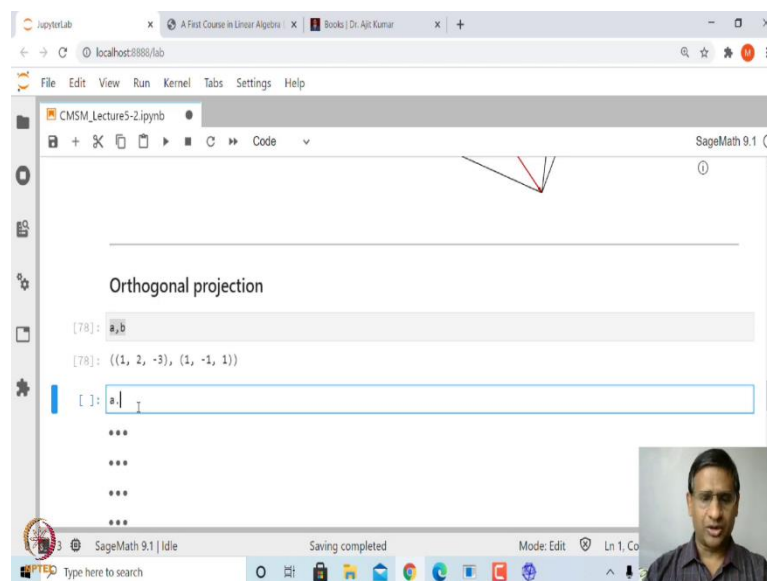
So, if suppose, if I store this cross product in w, and check whether w is perpendicular to a or not, the answer should be true. This is perpendicular, because this dot product of w with a is 0, and hence, w is perpendicular to a. Similarly, you can say w is perpendicular to b. So, cross product is perpendicular to a and b. You can again try to plot a, b, and c, and then, then see whether it is perpendicular to the, any linear combination of a, and b.

So, for example, let me, let us just look at, if I say, plot a plus plot b plus plot, let us say w, and let us put this color in red.

(Refer Slide Time: 08:29)

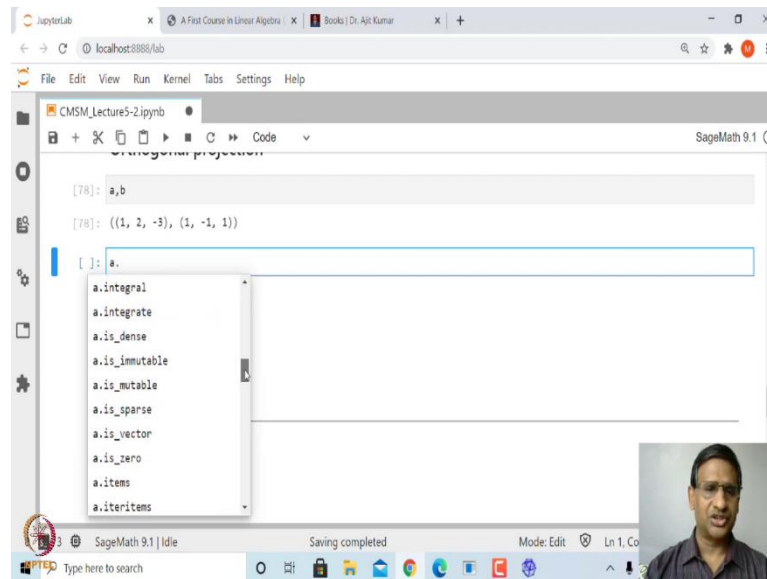


So, color is equal to red, and when you, when you run this, you can see here, this somewhat you can rotate in various form, and, but a and b are blue in color, and this is the perpendicular to, to a and b, right? (Refer Slide Time: 08:47)



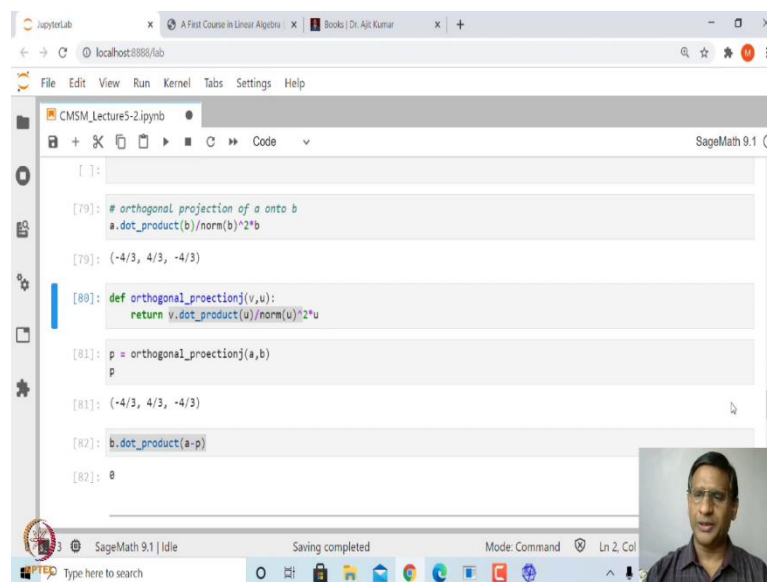
Similarly, you can find out orthogonal projection of a vector onto another vector. So, for example, we, we have declared two vectors a and b, and let us check whether there is some inbuilt function to find orthogonal projection of a onto b or not.

(Refer Slide Time: 09:17)



So, if I say a dot, and then press tab, and if you go through this list, just once second, if you go through this list a dot tab, if you go through this list you will not find anything which is related to orthogonal projection, but we already know what is orthogonal projection of a onto b.

(Refer Slide Time: 09:33)



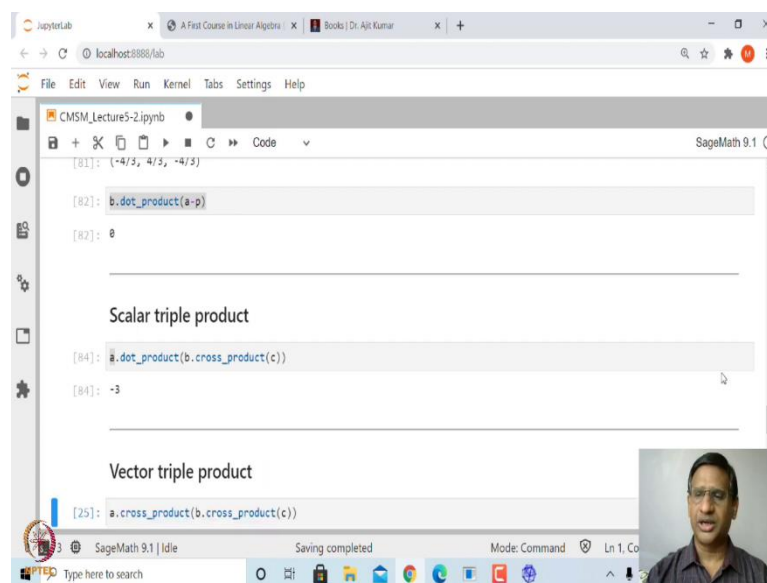
So, we can just compute that. So, orthogonal projection of a vector a onto b is given by a dot product with b divided by length of b square times b. So, it is a parallel to b, and that therefore, it will be some constant t times b, and that constant is given by this, this scalar

right, but we can declare our own function. So, this is our orthogonal projection of a onto b , we can declare our own function.

So, we, let us define orthogonal projection of v onto u as what it returns? v dot product with u divided by norm of u square times u . So, that is the orthogonal projection of v on to u . And let us now check that suppose, if I take orthogonal projection of a onto b , and store it in p , and then, so this is orthogonal projection, and suppose, if we subtract p from p minus a , and how is it related to b ?

So, if we take the dot product of p minus a , minus p with b , it should be 0, because this is how actually geometrically you define orthogonal projection of a vector onto another vector. So, a minus p is perpendicular to b , and in fact, using this only, you obtain this, this formula for orthogonal projection, right? So, this is a way to verify that p is orthogonal projection of a onto b , right?

(Refer Slide Time: 11:11)



```
[01]: (-4/3, 4/3, -4/3)

[02]: b.dot_product(a-p)

[02]: 0

Scalar triple product

[04]: a.dot_product(b.cross_product(c))

[04]: -3

Vector triple product

[25]: a.cross_product(b.cross_product(c))
```

Similarly, you can define a scalar triple product. So, you have three vectors a , b , c . So, how do I define? The scalar triple product of a , b , c is, take b cross c , and take its dot product with a , take its dot product with a . So, let me run this, this gives you 3, minus 3. So, and you must have studied some properties of a scalar triple product. So, actually it

gives you volume of a parallelepiped whose base is b , and c , and along a , or the volume of a parallelepiped spanned by three vectors a , b , and c , right?

Similarly, you can find vector triple products of three vectors a , b , c . So, if a , b , c are three vectors, vector triple product of a , b , c is given by a cross product with b cross product with c . So, it is $a \times (b \times c)$, right?

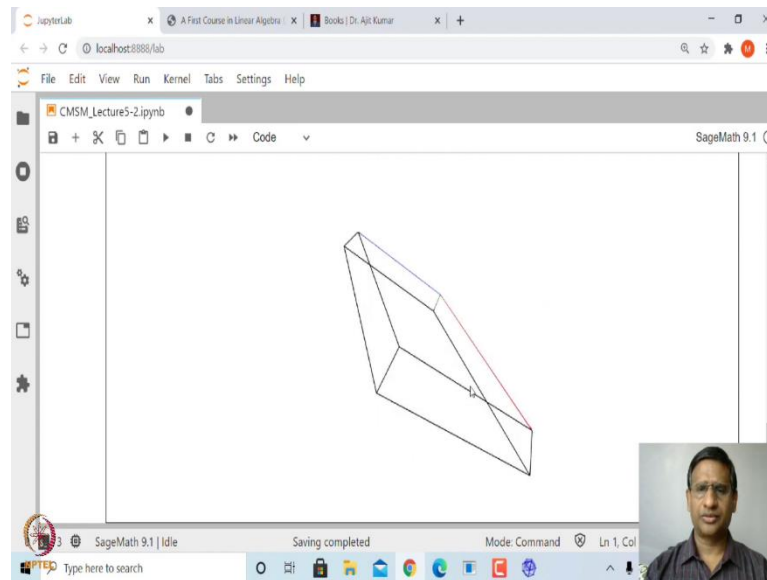
(Refer Slide Time: 12:05)

So, let me run this. It gives you a vector which is this, and you can again verify various properties of vector triple product, right?

(Refer Slide Time: 13:15)

Suppose, you want to plot the parallelepiped spanned by these vectors a , b , and c . It is very simple; you can just plot a , b , and c , and then plot the join the line from a to $a + b$, b to $a + b$, b to $a + b$ to $b + c$, c to $b + c$, and then $a + b$ to $a + b + c$, and things like that this is what is done here.

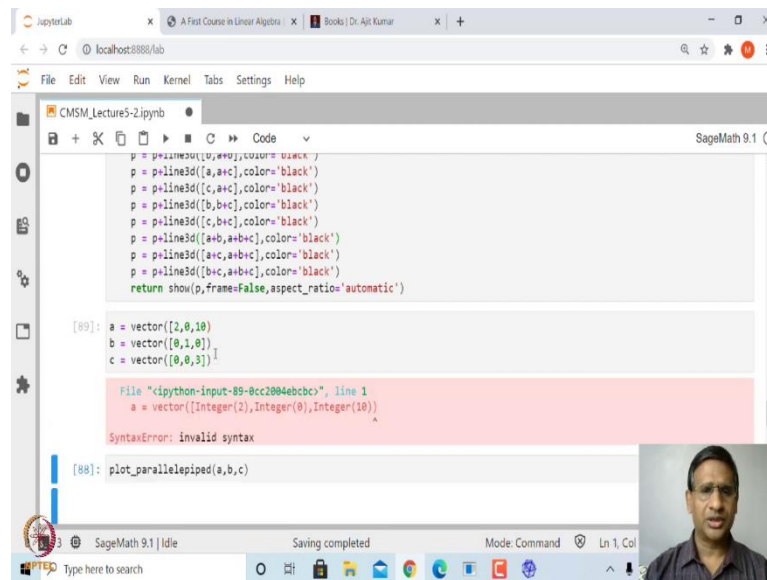
(Refer Slide Time: 13:27)



So, first these three vectors are plotted and then in that you add a line in 3D that is, because these three vectors are taken in \mathbb{R}^3 . So, a to $a + b$, b to $a + b$, a to $a + c$, c to $a + c$, b to $a + b + c$, c to $b + c$, $a + b$ to $a + b + c$, $a + c$ to $a + b + c$, $b + c$ to $a + b + c$ and then show at all these things together and do not use the frame, right? So, that is the user defined function to define a parallelepiped.

Now, let us take some vector a , b , c and let us plot the parallelepiped spanned by these three vectors using this inbuilt functions. So, this is how it looks like. So, you can go through this, this does not look very nice, but, yeah.

(Refer Slide Time: 13:39)



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

```
p = p1line3d([0,0,0],color='black')
p = p1line3d([a,a+c],color='black')
p = p1line3d([c,a+c],color='black')
p = p1line3d([b,b+c],color='black')
p = p1line3d([c,b+c],color='black')
p = p1line3d([a+b,a+b+c],color='black')
p = p1line3d([a+c,a+b+c],color='black')
p = p1line3d([b+c,a+b+c],color='black')
return show(p,frame=False,aspect_ratio='automatic')
```

Below the code, the input prompt [89]: is followed by the definition of vectors a, b, and c:

```
a = vector([2,0,10])
b = vector([0,1,0])
c = vector([0,0,3])
```

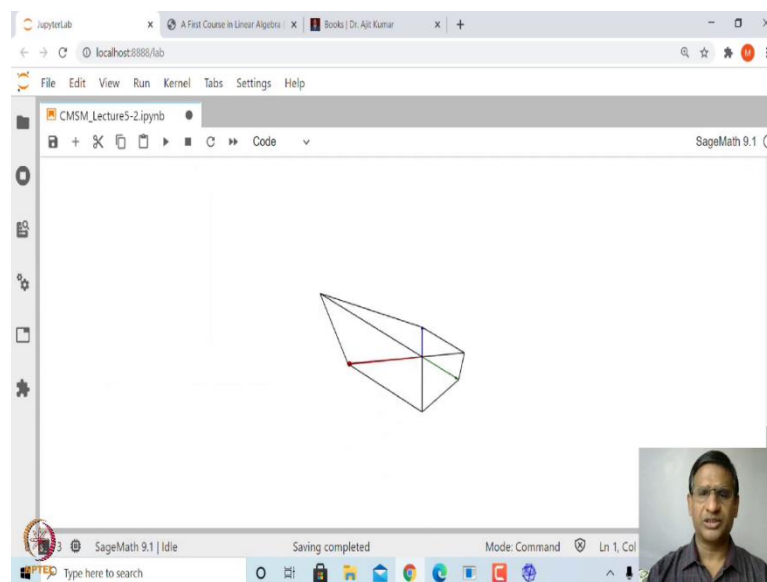
A red error message box appears, indicating a syntax error in the definition of vector a:

```
File "c:\python-input-89-0cc2004ebc", line 1
a = vector([Integer(2),Integer(0),Integer(10)])
SyntaxError: invalid syntax
```

The input prompt [88]: is followed by the command `plot_parallelepiped(a,b,c)`.

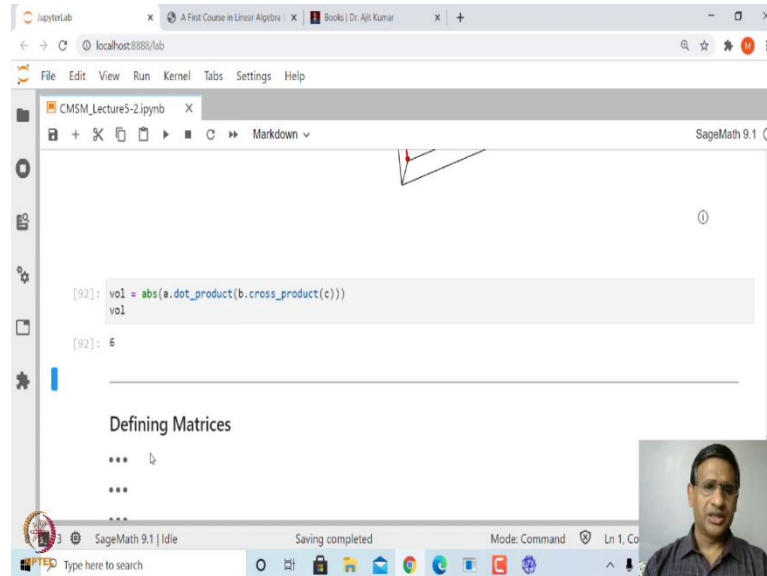
So, suppose, let me say I have this 2 comma 0 comma 0, this is, let us say, 0 comma 1 comma 2, and this is, let us say, 0 comma 1, or let us say 0 comma 3, I think somewhere I made a mistake, this is square bracket is missing, right?

(Refer Slide Time: 14:03)



Now, if I ask it to plot the parallelepiped this may look much better, yeah. So, that is the parallelepiped, right, ok?

(Refer Slide Time: 14:11)



```
[92]: vol = abs(a.dot_product(b.cross_product(c)))
vol
[92]: 6
```

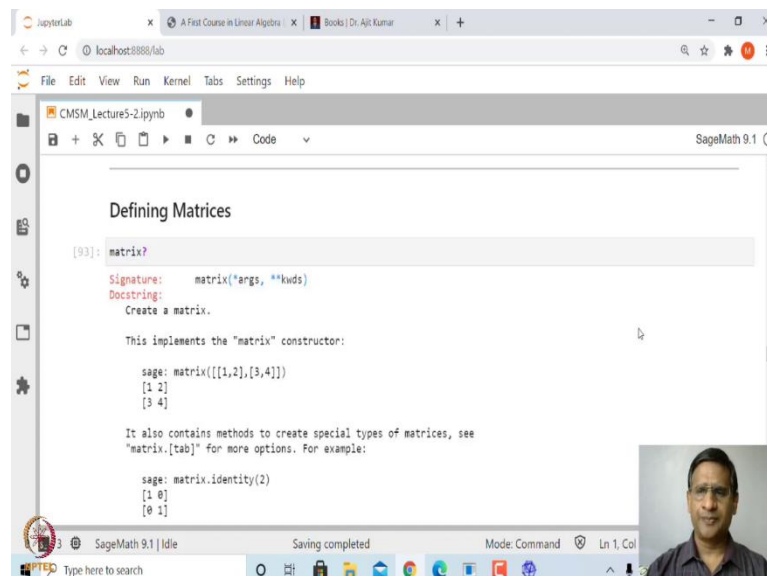
Defining Matrices

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

You can find the volume of this parallelepiped spanned by a , b , c . So, volume is given by the scalar triple product.

So, take the cross product of a and b , and then take its cross product with this, dot product with c , or you can take, let us say, this cross product with b and c , and then take dot product with a , dot product with a , this is 6, right? Now, let us look at how to define matrices.

(Refer Slide Time: 14:43)



```
[93]: matrix?
Signature: matrix(*args, **kws)
Docstring:
Create a matrix.

This implements the "matrix" constructor:

sage: matrix([[1,2],[3,4]])
[1 2]
[3 4]

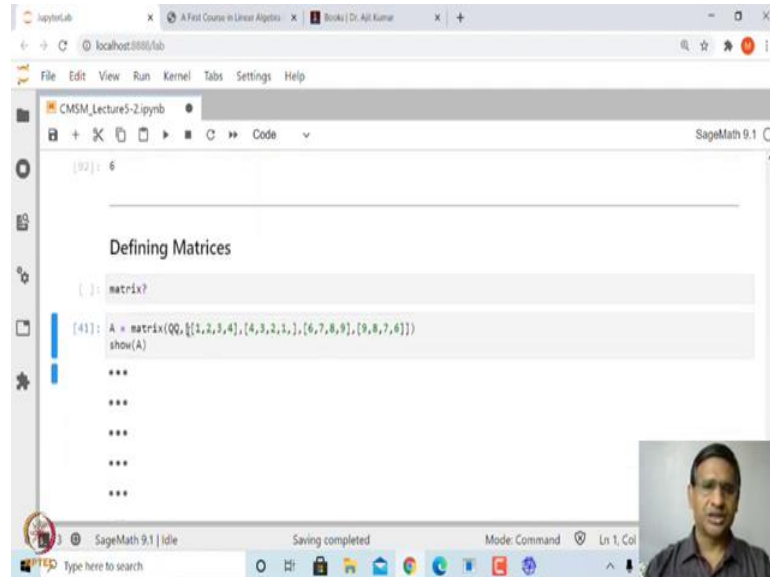
It also contains methods to create special types of matrices, see
"matrix.[tab]" for more options. For example:

sage: matrix.identity(2)
[1 0]
[0 1]
```

Defining Matrices

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

So, the easiest way of declaring the matrix is to use this function called matrix. So, you can take help on this matrix, and go through this help document, how this, what is the meaning of this function? How it is used, with so many examples. So, let me, let me remove this help document, right? (Refer Slide Time: 15:11)



```
[02]: 6
```

Defining Matrices

```
[ ]: matrix?
```

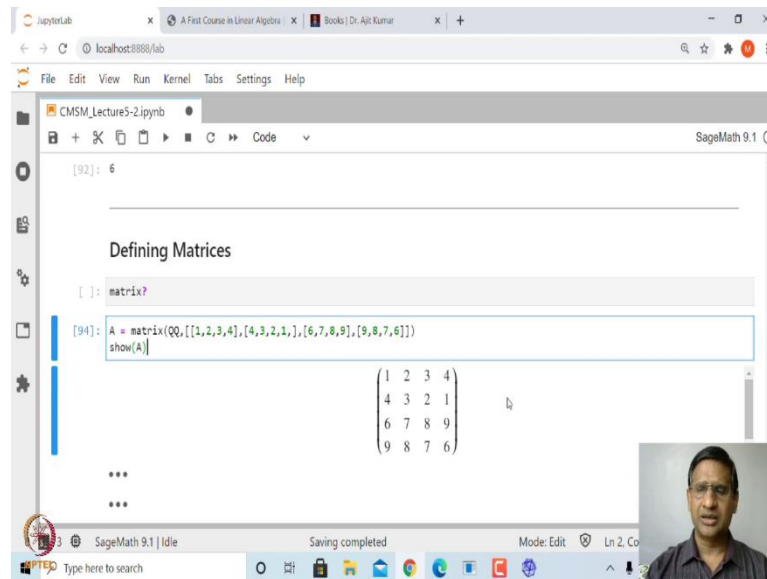
```
[41]: A = matrix(QQ, [[1, 2, 3, 4], [4, 3, 2, 1], [6, 7, 8, 9], [9, 8, 7, 6]])
      show(A)
```

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

And let us declare a matrix, say A, which is a 4 cross 4 matrix.

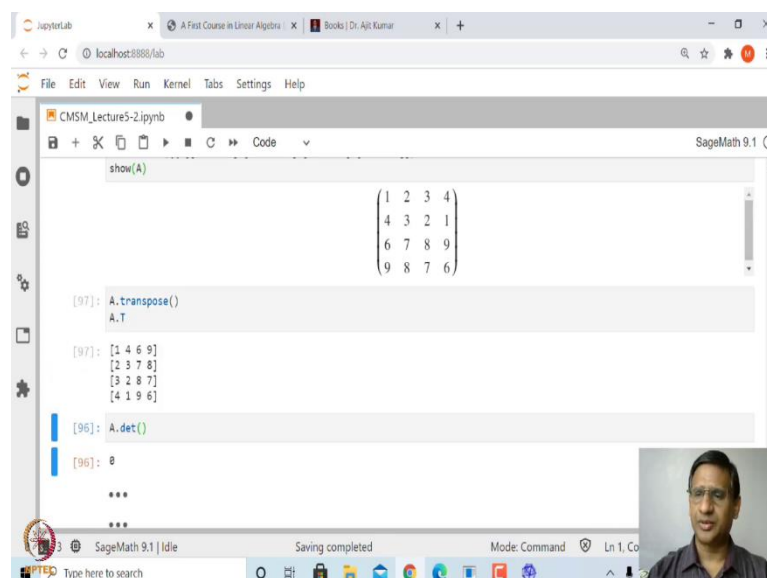
So, this again, very similar to how you declare a matrix in numpy using array command, except the array is replaced by matrix, and you can again mention the domain from which the entries are taken, like very similar to how we did for vectors. So, here we are saying that the matrix entries are from rational numbers. So, this is the first row, this is the second row, third row, fourth row, and if I asked you to show this matrix, this is how it looks like.

(Refer Slide Time: 15:11)

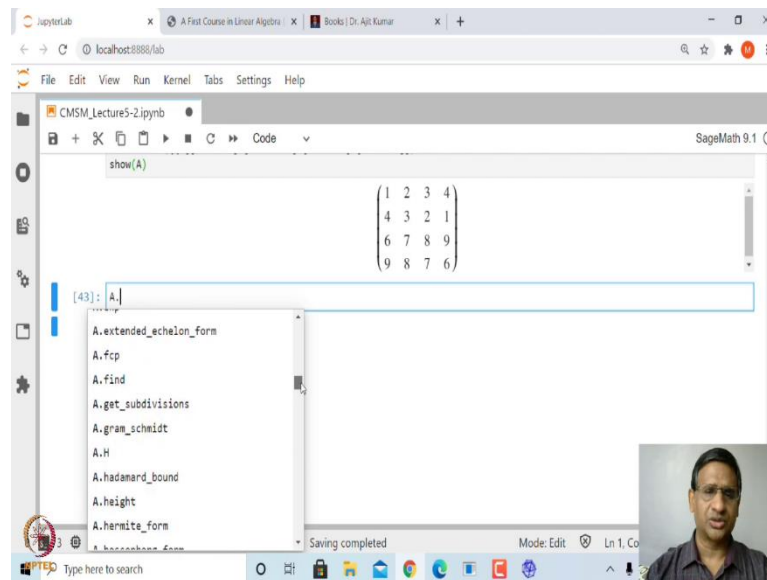


I can even ask for latex command of this.

So, if I say latex A, it will give me latex command of, command of this, and you can do various operation on, on matrix just by saying A dot, and then press tab, and if you go through this list, you will see all the standard operations on the, on a matrix are available here. You can find adjoint, you can find transpose, you can find determinant, you can find inverse, you can find eigenvalues, eigenvectors, rank, trace all these things are available. (Refer Slide Time: 15:57)



(Refer Slide Time: 16:01)



```
show(A)
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \\ 6 & 7 & 8 & 9 \\ 9 & 8 & 7 & 6 \end{pmatrix}$$

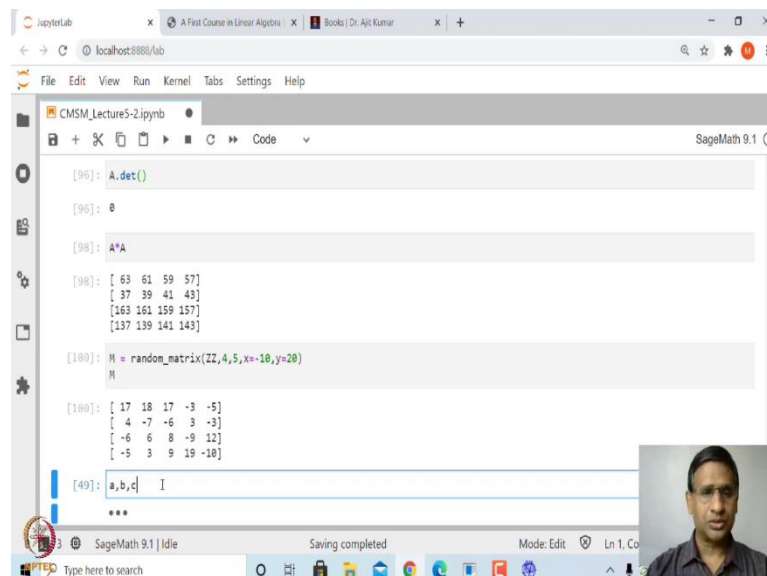
[43]: A

- A.extended_echelon_form
- A.fcp
- A.find
- A.get_subdivisions
- A.gram_schmidt
- A.H
- A.hadamard_bound
- A.height
- A.hermite_form

So, in principle, you do not need to actually remember all these commands, because there are, there is too many, nobody can remember, but dot tab will give you all these, these methods that can be applied on A. So, if I say A dot transpose, transpose, and then empty round bracket, this will give me transpose of this matrix. If I say A dot, dot det, this will give me determinant of this matrix.

By the way, transpose can also be obtained as A dot capital T. These two are the same thing, exactly the same thing we saw in case of numpy array, right?

(Refer Slide Time: 17:03)



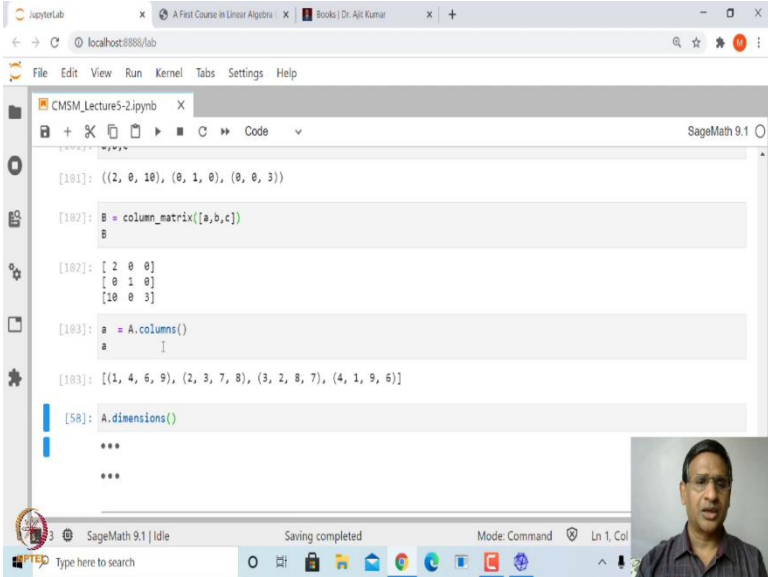
```
[95]: A.det()
[96]: 0
[98]: A*A
[98]: [ 63 61 59 57]
      [ 37 39 41 43]
      [163 161 159 157]
      [137 139 141 143]
[100]: M = random_matrix(22,4,5,x=-10,y=20)
M
[100]: [ 17 18 17 -3 -5]
      [ 4 -7 -6 3 -3]
      [-6 6 8 -9 12]
      [-5 3 9 19 -10]
[49]: a,b,c I
***
```


You can find square of a matrix, or if I say $A \star A$, it will give me matrix multiplication of A with itself.

Of course, the matrix should be compatible, and if I say, you can also generate a random matrix over integers or over rational numbers. So, for example, this gives you a random matrix, 4 cross 5 random matrix whose entries are coming from integers, varying between minus 10, and 20.

So, if I say this, let me call this as M , and so, let us see what is this M . Of course, it will keep changing. So, this is a 4 cross 5 matrix, right? You can also declare matrix from vector.

(Refer Slide Time: 17:49)



```
[101]: ((2, 0, 10), (0, 1, 0), (0, 0, 3))

[102]: b = column_matrix([a,b,c])
b
[102]: [ 2 0 0]
[ 0 1 0]
[10 0 3]

[103]: a = A.columns()
a
[103]: [(1, 4, 6, 9), (2, 3, 7, 8), (3, 2, 8, 7), (4, 1, 9, 6)]

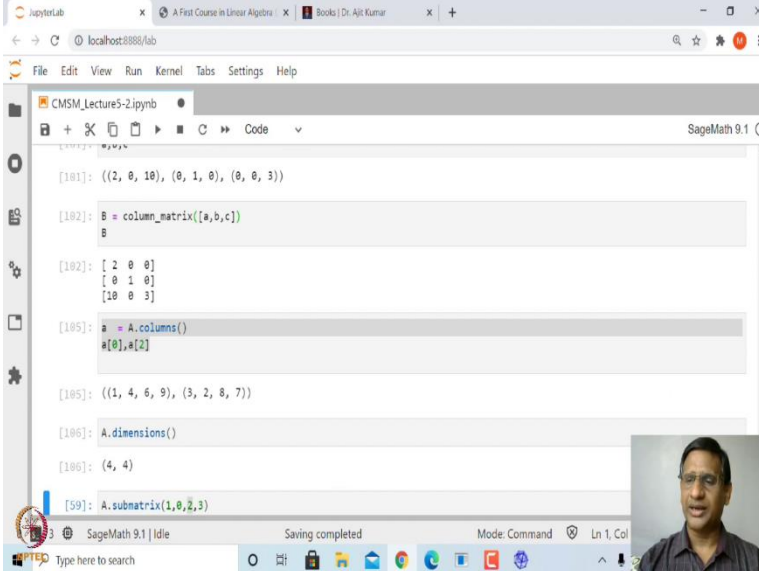
[58]: A.dimensions()
***
***
```

So, for example, we already have declared these three vectors a , b , c . Then we can define a matrix whose first column is a , second column is b , third column is c , by declaring column underscore matrix, and in the bracket, you write list of vectors.

So, this first column was 2 0 0, second column 0 1 0, third column 0 0 3. So, that is a column matrix. Next, you can extract columns of a matrix using A dot columns command.

So, suppose if I store this into a small a, then small a is list of columns, so, list of columns of a.

(Refer Slide Time: 18:37)



```
[181]: ((2, 0, 10), (0, 1, 0), (0, 0, 3))

[182]: B = column_matrix([a,b,c])
B
[182]: [ 2 0 0]
      [ 0 1 0]
      [10 0 3]

[185]: a = A.columns()
a[0],a[2]
[185]: ((1, 4, 6, 9), (3, 2, 8, 7))

[186]: A.dimensions()
[186]: (4, 4)

[189]: A.submatrix(1,0,2,3)
```

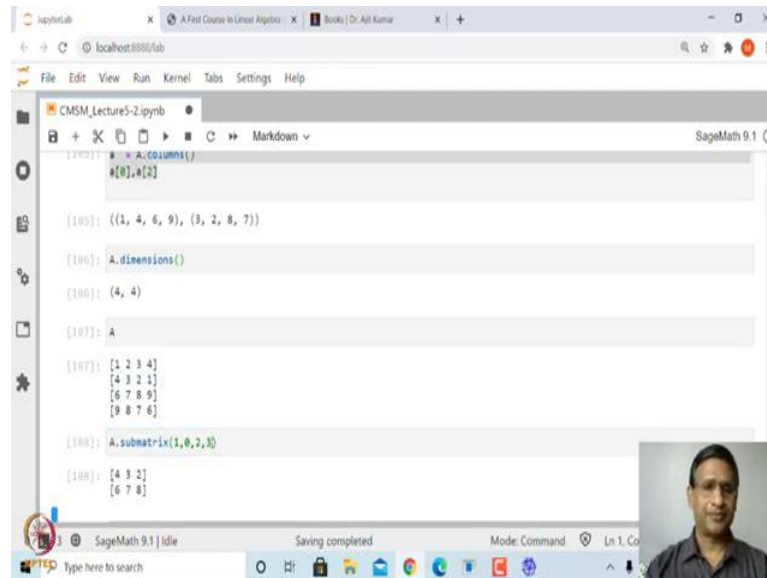
Now, if I, for example, if I say, what is a of 0, it will give me first column. If I say what is a of 2, it will give me third column.

So, many times we when, we do some computation with, with a, we work with columns. So, it will be easier to extract columns of a. You can look at what is dimension of a matrix. So, in this case it is 4 cross 4. You can even find out any sub matrix of a matrix. You can take any slice of a matrix, because it is actually a list of a list.

So, any slice of a matrix can be obtained exactly in the same way as we did in slicing of an array in Python. So, for example, if I say submatrix 0, 1, 2, 3. So, the first one is the

starting column, starting row, this is the starting column, this is the number of rows, this is the number of column

(Refer Slide Time: 19:33)

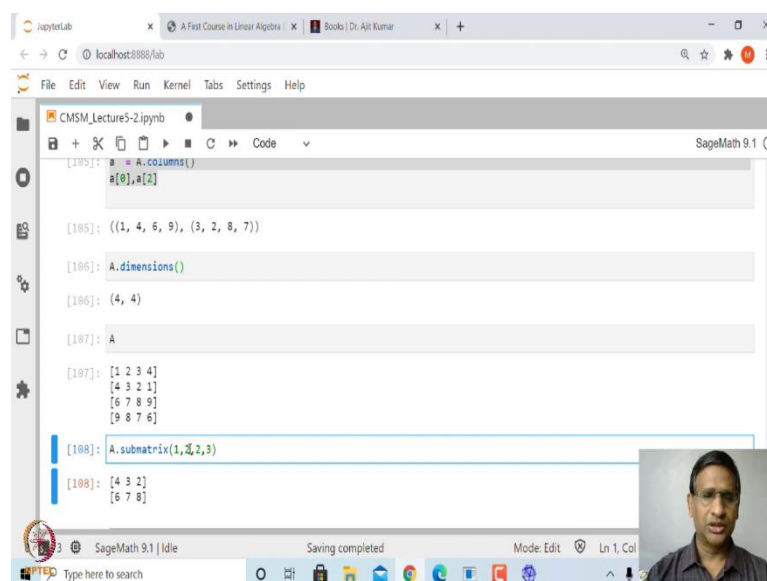


```
[105]: ((1, 4, 6, 9), (3, 2, 8, 7))
[106]: A.dimensions()
[106]: (4, 4)
[107]: A
[107]:
[1 2 3 4]
[4 3 2 1]
[6 7 8 9]
[9 8 7 6]
[108]: A.submatrix(1,0,2,3)
[108]:
[4 3 2]
[6 7 8]
```

So, it says that from A. So, let us just look at what is A. So, it will be easier to look at A, this is A, and this is the second row, and then first column. So, it will start from here, and then how many rows? Two rows, so, it will take the second, and third row, and 3 columns.

So, it will give you 4, 3, 2, 6, 7, 8, let us just see, that is correct.

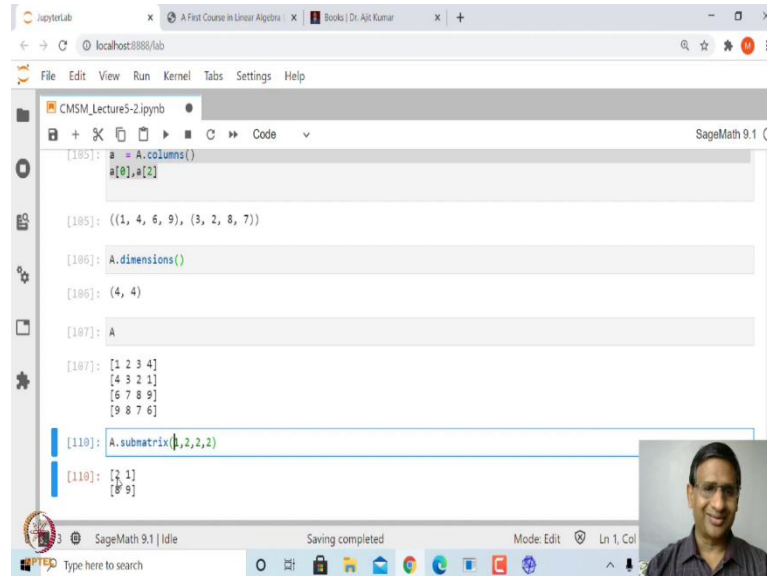
(Refer Slide Time: 19:59)



```
[105]: ((1, 4, 6, 9), (3, 2, 8, 7))
[106]: A.dimensions()
[106]: (4, 4)
[107]: A
[107]:
[1 2 3 4]
[4 3 2 1]
[6 7 8 9]
[9 8 7 6]
[108]: A.submatrix(1,0,2,3)
[108]:
[4 3 2]
[6 7 8]
```

If I, if I mention here, let us say 2 so, it will start with third column, and of course, it will give you error, because after that it, it does not have 3 columns.

(Refer Slide Time: 20:10)



```
[105]: a = A.columns()
       a[0],a[2]

[105]: ((1, 4, 6, 9), (3, 2, 8, 7))

[106]: A.dimensions()

[106]: (4, 4)

[107]: A

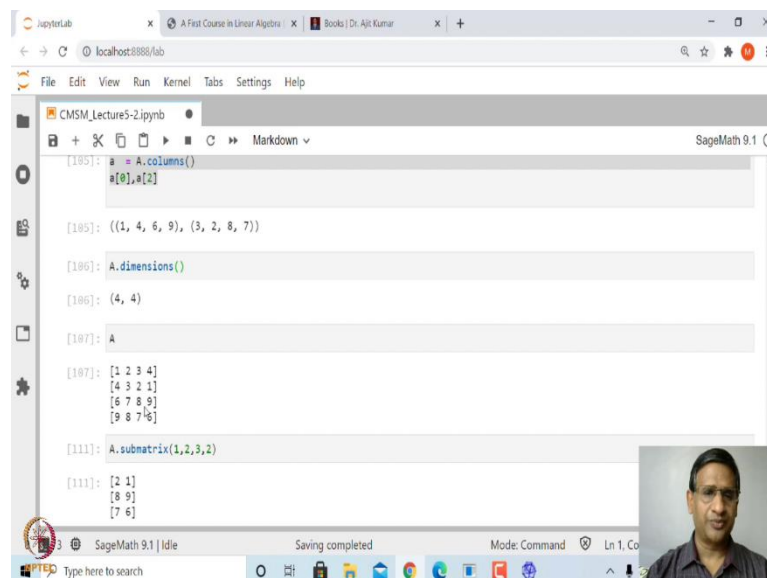
[107]: 
[[1 2 3 4]
 [4 3 2 1]
 [6 7 8 9]
 [9 8 7 6]]

[110]: A.submatrix(1,2,2,2)

[110]: 
[[2 1]
 [8 9]]
```

So, let us say only two columns. Now, it will give you, it is starting from first row, and third column. So, first row third column, and then the two rows, and then two columns.

(Refer Slide Time: 20:24)



```
[105]: a = A.columns()
       a[0],a[2]

[105]: ((1, 4, 6, 9), (3, 2, 8, 7))

[106]: A.dimensions()

[106]: (4, 4)

[107]: A

[107]: 
[[1 2 3 4]
 [4 3 2 1]
 [6 7 8 9]
 [9 8 7 6]]

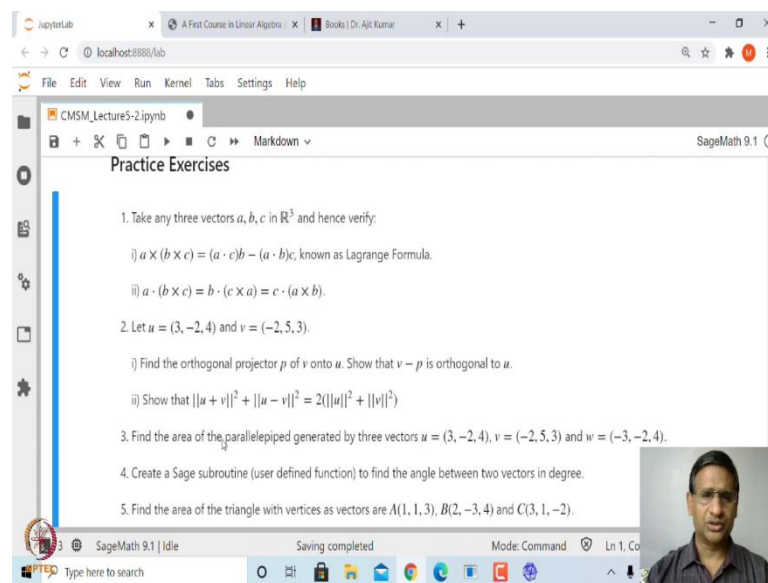
[111]: A.submatrix(1,2,3,2)

[111]: 
[[2 1]
 [8 9]
 [7 6]]
```

So, if I say, for example, take three rows, and two columns it will give me all the rows after that, right?

So, this is how you can take any sub matrix of any matrix, and try to explore other concepts that you have learned. For example, finding inverse, determinant, and verifying some of the properties of inverse, determinant, transpose, you can just try to explore, ok?

(Refer Slide Time: 20:49)



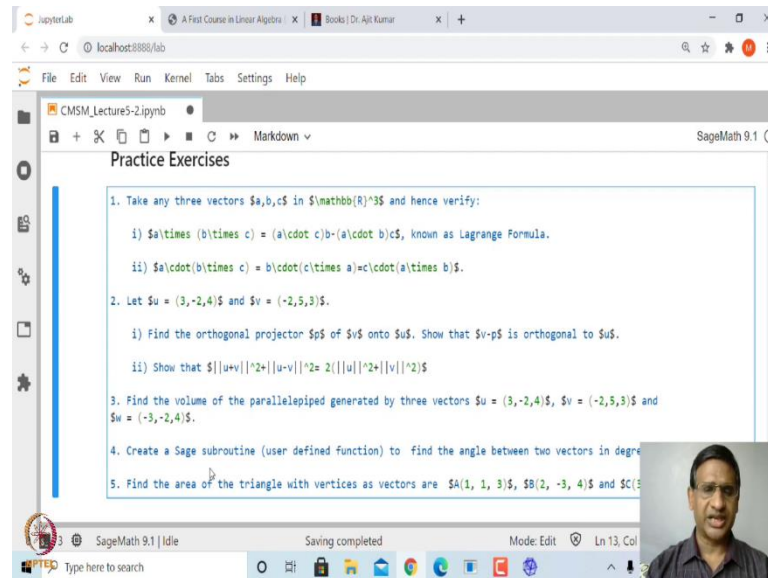
Let me leave you with few simple exercises. So, the first exercise is to verify these formula which is known as Lagrange formula; a cross b cross c which is a vector, is a dot c times b minus a dot b times c , and then a dot b cross c is equal to, is same as b dot c cross a is equal to, same as c dot a cross b , that is a cyclic kind of rule.

And if you have two vectors u and v , find orthogonal projection of let us say v onto u , and verify that v minus p , p is orthogonal projection, is perpendicular to u , and then verify this formula.

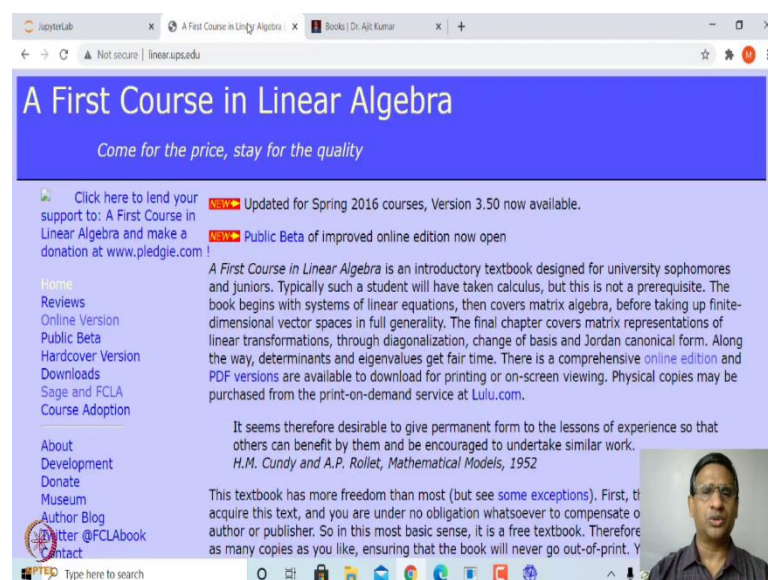
This is actually known as parallelogram law, which says that sum of the diagonals square, the length of the sum, sum of the diagonal square is equal to twice sum of the squares of

each side. And then, the next one is to find the area of parallelepiped generated by this. Actually, it should be volume of a parallelepiped.

(Refer Slide Time: 21:57)



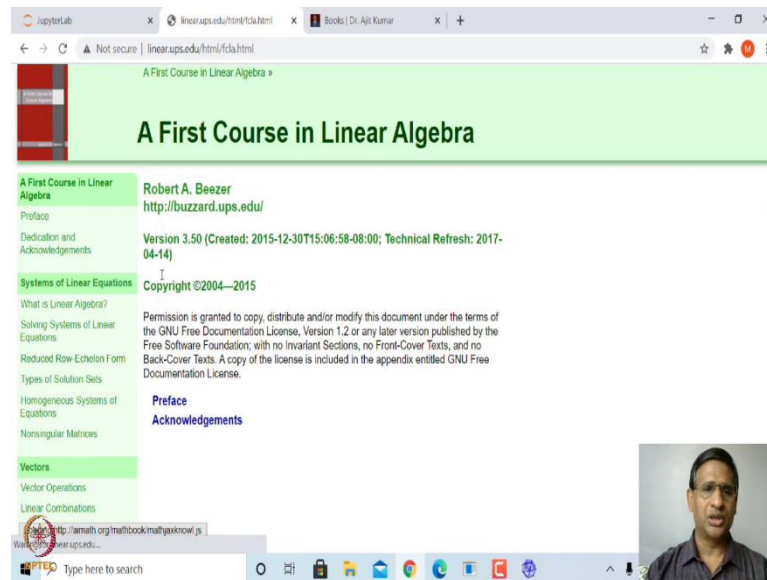
Let me just change it to volume, volume of the parallelepiped spanned by three vectors u , v , and w . The next exercise is to create a Sage subroutine to find angle between two vectors in degree, and the last one is to find the area of a triangle which is, whose vertices are A , B , C , the coordinates of A , B , C are given as vectors, ok? (Refer Slide Time: 22:29)



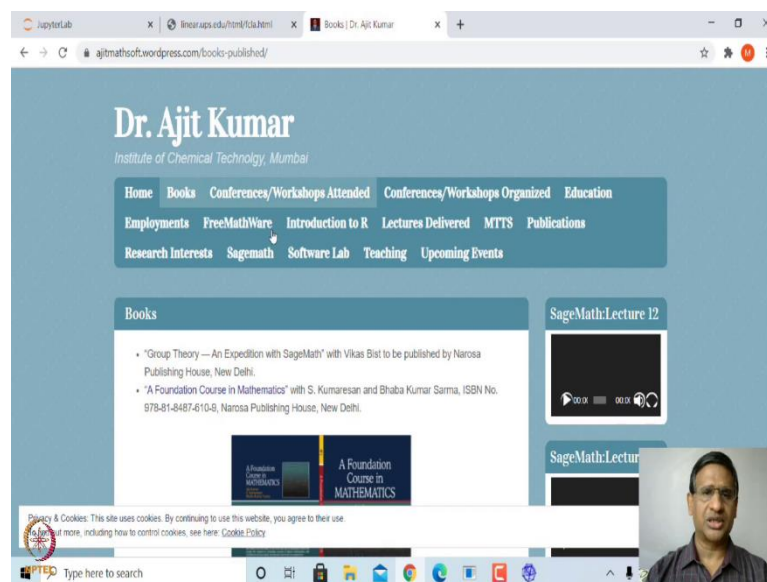
So, these are the few exercises. In case you want to look at some reference, there are two things you can look at. One, you can possibly go to this is linear dot ups dot edu a website.

This is an online book which is freely available, A First Course in Linear Algebra. This is by Robert Beezer, and you can download the PDF version of this, or you can use online version, both are available.

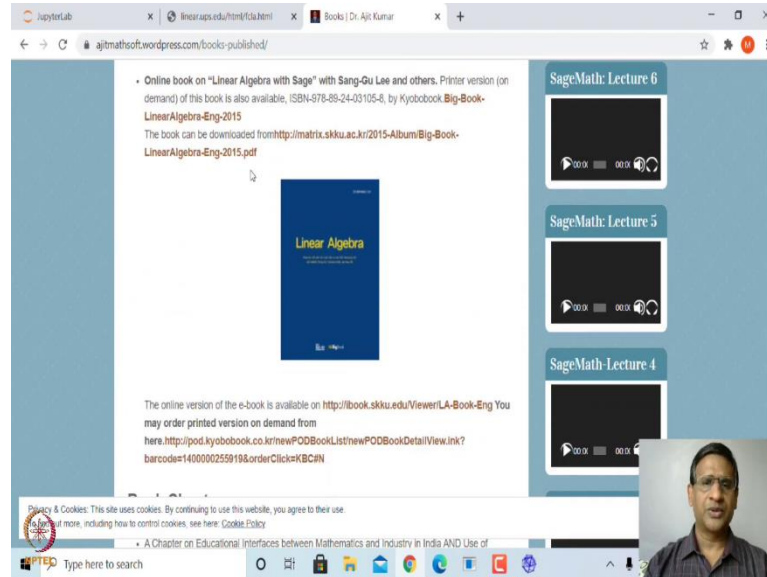
(Refer Slide Time: 22:49)



(Refer Slide Time: 22:57)



Online version again, you can see the topic-wise it is given or you can also go to my personal website, and go to the section book. This is [ajitmathsoft dot wordpress dot com](http://ajitmathsoft.wordpress.com), and for example, you have a book on Linear Algebra. (Refer Slide Time: 23:11)



This is along with Sang-Gu Lee, and a few others. This is freely available, you can download from this website.

So, you can make use of these, these things. So, let me stop here. In the next class, we will look at solving system of linear equations using various methods.

Thank you very much.