

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

Least Square Problems
Lecture – 40
Least Square Solution with SageMath

Welcome to the 40th lecture on Computational Mathematics with SageMath. In this lecture we will look at Least Square Problems as an application to linear algebra. We have already seen how to solve least square problems using calculus. Let us first start with a problem.

(Refer Slide Time: 00:38)

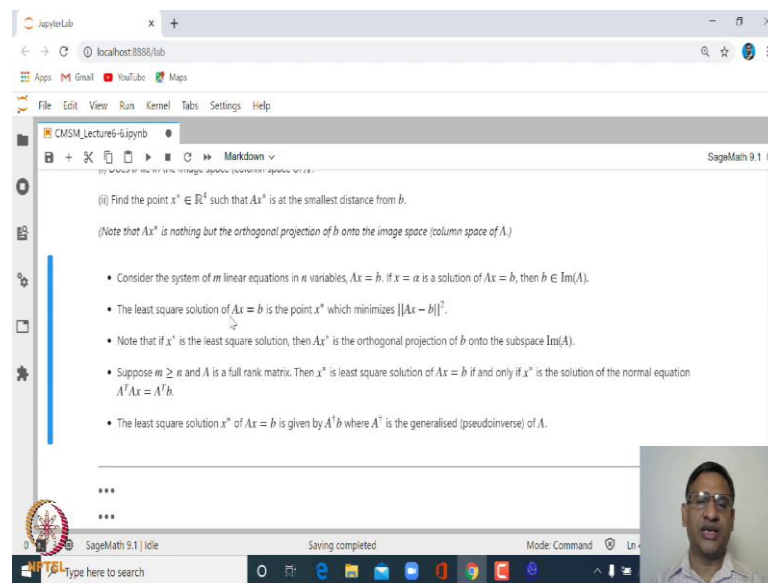
The screenshot displays a JupyterLab window with a SageMath 9.1 notebook. The notebook's title bar indicates it is 'CM5M_Lecture6-6.ipynb'. The main content area shows a slide titled 'Least Square Problems' from the 'Institute of Chemical Technology, Mumbai'. The slide text defines a problem: 'Problem Let A be a 6 x 4 matrix whose entries are between -10 and 10 of maximum rank. Then A is a linear map from \mathbb{R}^4 to \mathbb{R}^6 . Suppose $b = (1, 2, 3, 4, 5, 6)$. (i) Does b lie in the image space (column space) of A? (ii) Find the point $x^* \in \mathbb{R}^4$ such that Ax^* is at the smallest distance from b. (Note that Ax^* is nothing but the orthogonal projection of b onto the image space (column space) of A.)'. The bottom of the window shows a taskbar with various application icons and a small video feed of the professor.

Suppose you have a random 6 cross 4 matrix whose entries are let us say integer from minus 10 to 10. Then A can be thought of as a linear map from \mathbb{R}^4 to \mathbb{R}^6 . Now suppose there is a vector b in the codomain, which is 1, 2, 3, 4, 5, 6. Now, the question is, does b lie in this image of A, that is column space of A? If it lies, that is same as saying that Ax equal to b, has a solution.

However if it does not lie in the image space of A, in case b does not lie in the image space of A, then then we can ask, what is the point x star let us say in \mathbb{R}^4 , such that the image of x star under A, that is Ax star is at the smallest distance from b in the codomain. Such a x star is known as least square solution of this system of linear equation Ax equal to b.

You can note that this Ax^* is actually, nothing but the orthogonal projection of b onto the image space of A . So, that is what we will obtain. We will obtain the least square solution as an orthogonal projection of b onto image space of A .

(Refer Slide Time: 02:14)



Let us see what we are saying here. In this case, we have a system of m linear equations in n variables. Let us say that is Ax equal to b .

If x equal to α is a solution of Ax equal to b , then b lies in the image space of A , and the least square solution of Ax equal to b is x^* which minimizes this distance of Ax from b for all x . This is a minimization problem. That is how we have seen as a solution to this problem in calculus.

Note that if x^* is a least square solution of least square solution, then Ax^* is the orthogonal projection of b onto image space of A , that is what I already said. Now, suppose this m is bigger than equal to n , that is, you have more equations than the number of variables, and let us assume that A is a full rank. That means, the rank of A will be n here.

In this case solving this Ax equal to b , or finding the least square solution of this Ax equal to b , is same as, you just multiply both sides of Ax equal to b by A transpose. So,

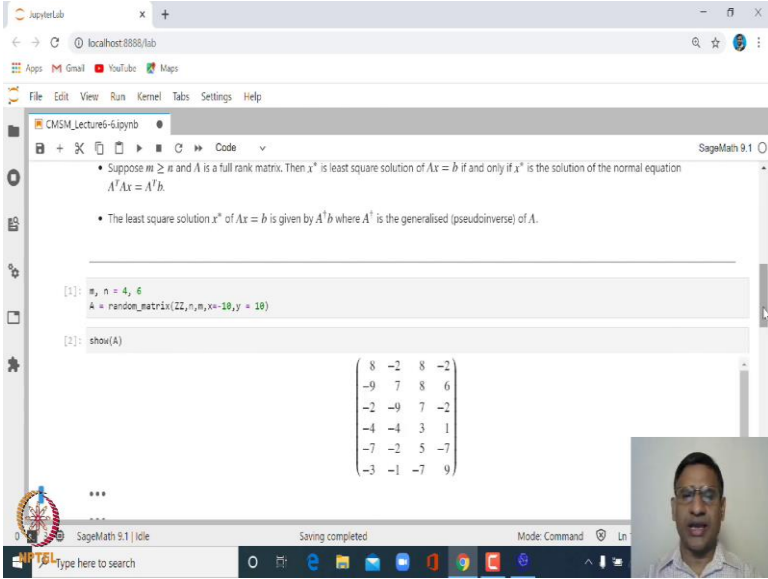
what you will get? $A^T A x$ is equal to $A^T b$. This is what is called the normal equation of this least square problem.

Now, the least square solution of Ax equal to b , is nothing but the solution of this normal equation $A^T A x$ is equal to $A^T b$. Since A is of full rank, that is rank of A is n , $A^T A$, which is n by n matrix will be invertible matrix and hence the inverse will exist. So, finding least squares solution of Ax equal to b is nothing, but solving this normal equation for x .

One can also obtain this least square solution namely, x^* as a solution to this Ax equal to b , But since A may not be invertible matrix, what we need to do is, we need to find the generalized inverse of A and then multiplied by b . Notice that if A is invertible, the solution x is $A^{-1} b$, but if even if A is not invertible, we can find its generalized inverse.

We will look at this, how we can solve this in using singular value decomposition, but Sage also has an inbuilt function to find generalized inverse of a matrix. So, in this case what will you get? You will get least square solution x^* , as generalized inverse of A times b . This is how we denote generalized inverse of A . Now, let us look at how we can solve this problem, which we have stated.

(Refer Slide Time: 05:23)



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

```
[1]: m, n = 4, 6
A = random_matrix(ZZ, n, m, x=-10, y=10)

[2]: show(A)
```

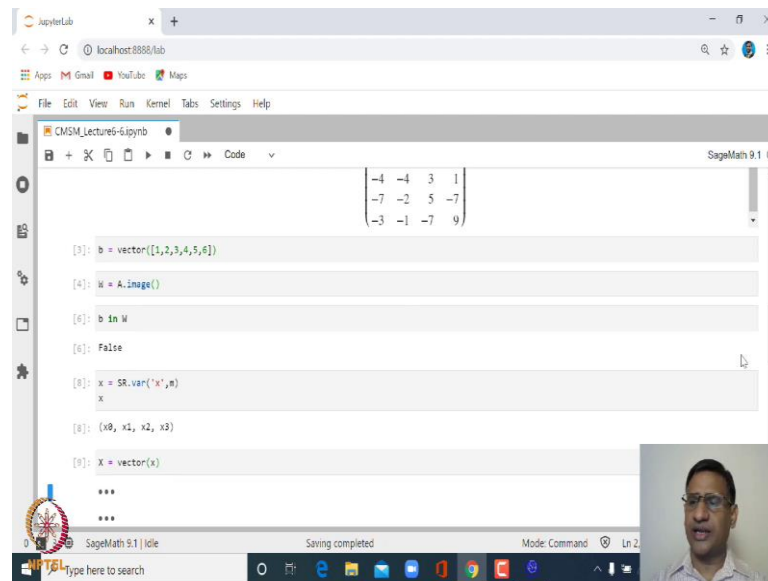
The output of the code is a 6x6 matrix A :

$$\begin{pmatrix} 8 & -2 & 8 & -2 & & \\ -9 & 7 & 8 & 6 & & \\ -2 & -9 & 7 & -2 & & \\ -4 & -4 & 3 & 1 & & \\ -7 & -2 & 5 & -7 & & \\ -3 & -1 & -7 & 9 & & \end{pmatrix}$$

At the bottom right of the JupyterLab window, there is a small video feed of a man speaking.

Let us start with m and n as 4 and 6 and define a random matrix over integers whose entries are coming from minus 10 to 10 right. Now, let us see what this matrix is. This is how this matrix looks like.

(Refer Slide Time: 05:44)



```

[3]: b = vector([1,2,3,4,5,6])
[4]: W = A.image()
[5]: b in W
[6]: False
[7]: x = SR.var('x',n)
[8]: (x0, x1, x2, x3)
[9]: X = vector(x)
***
***

```

The matrix A shown is:

$$\begin{bmatrix} -4 & -4 & 3 & 1 \\ -7 & -2 & 5 & -7 \\ -3 & -1 & -7 & 9 \end{bmatrix}$$

Next let us let us define a vector b which is 1, 2, 3, 4, 5, 6. Now, we can ask whether b lies in the image space of A . How do I how do check? You can define, let us say W , equals to A dot image, which is same as saying column space of A . Now, we can check if b in W ? The answer is not true. So, b does not lie in W .

Now we can ask what is the least square solution of this Ax equal to b . Let us define x as variable $x_0, x_1, x_2, x_3, x_4, x_5$. You can check, what is this x here; x_0, x_1, x_2, x_3 . Since m is 4, this is x_0, x_1, x_2, x_3 , we have seen this earlier. Now, let us define capital X as a vector which is going to be column vector x_0, x_1, x_2, x_3 .

(Refer Slide Time: 06:59)

```

[8]: x = SR.var('x',e)
x

[9]: (x0, x1, x2, x3)

[10]: X = vector(x)

[11]: err = A*X-b
f = sum([err[i]**2 for i in range(n)]) # Sum of error squares

[12]: gradf = f.gradient()
show(gradf)

(446*x0 - 56*x1 - 96*x2 - 96*x3 + 170, -56*x0 + 310*x1 - 76*x2 + 130*x3 + 94, -96*x0 - 76*x1 + 520*x2 - 154*x3 - 80, -96*x0 + 130*x1 - 154*x2 + 350*x3 - 54)

[13]: s = solve([gradf[i]==0.0 for i in range(n)],x,solution_dict=True)
show(s[0])

{x0: -347046565/967494899, x1: -441611213/967494899, x2: 97535076/967494899, x3: 37289026/138213557}

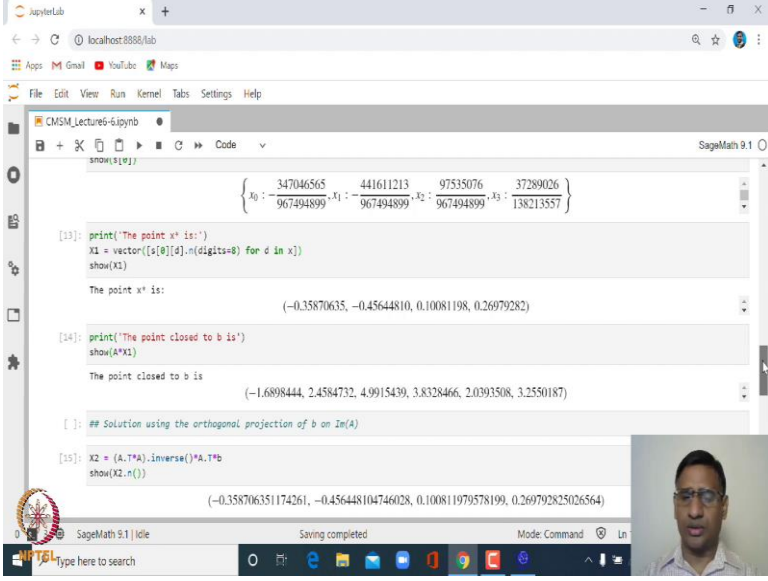
```

Now, if you recall in calculus, we solved this problem. First we define f as sum of the error squares and error in this case is nothing, but AX minus b . So, for any x_0, x_1, x_2, x_3 , AX is in the image and its error is going to be AX minus b . Then find the sum of the error square, that is, just look at the i th coordinate of this AX minus b . This will be a column vector, look at the i th coordinate of this. This is going to be the error, take the error square and take the sum, that is your f . So, f is going to be a function of x_0, x_1, x_2, x_3 and we want to minimize this function.

In calculus, how did we do? Let us quickly do that. We obtained the gradient of f with respect to x_0, x_1, x_2, x_3 and this is what we get. Then we solve this for each each coordinate of the gradients would be equal to 0. That is, the critical point and when you solve this for x , the solution is going to be given by, let us just wait it is taking time.

So, x_0 is this one, x_1 is this, x_2 is this, x_3 is this. This is how, we have already obtained using calculus. In calculus we wrote this as each coordinate, here we are writing this as a matrix, as a distance from b to AX .

(Refer Slide Time: 08:44)



```
show(s1[0])  

$$\begin{Bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{Bmatrix} = \begin{Bmatrix} -\frac{347046565}{967494899} \\ -\frac{441611213}{967494899} \\ \frac{97535076}{967494899} \\ \frac{37289026}{138213557} \end{Bmatrix}$$
  
[13]: print("The point x* is:")  
X1 = vector([s1[d].n(digits=8) for d in x])  
show(X1)  
The point x* is:  

$$(-0.35870635, -0.45644810, 0.10081198, 0.26979282)$$
  
[14]: print("The point closed to b is")  
show(A*X1)  
The point closed to b is  

$$(-1.6898444, 2.4584732, 4.9915439, 3.8328466, 2.0393508, 3.2550187)$$
  
[ ]: ## Solution using the orthogonal projection of b on Im(A)  
[15]: X2 = (A.T*A).inverse()*A.T*b  
show(X2.n())  

$$(-0.358706351174261, -0.456448104746028, 0.100811979578199, 0.269792825026564)$$

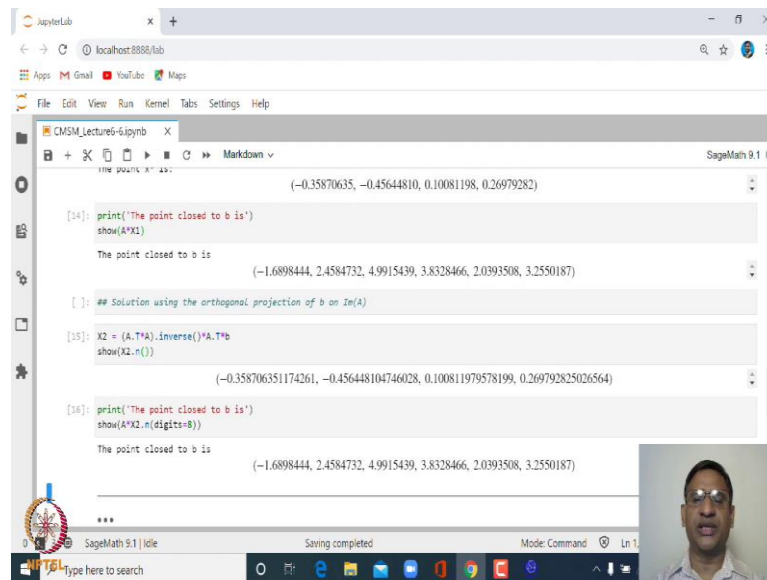
```

Now, in this case if you want to print the solution as a decimal value, this is what you get, right.

Next let us see the point which is closest to b, is A star X1, where X1 is this solution vector or the least square solution of A x equal to b.

Next this point AX1, which we obtained using calculus, is nothing but orthogonal projection of b onto image space of A. So, how does one find orthogonal projection of b onto image space of A? We have already seen that, this is nothing but take A transpose A, inverse of this and then multiply by A transpose b. That is your point X2, which is same as, what you have obtained using calculus.

(Refer Slide Time: 09:53)



```

[14]: print('The point closed to b is')
      show(A*X1)

The point closed to b is
(-1.6898444, 2.4584732, 4.9915439, 3.8328466, 2.0393508, 3.2550187)

[ ]: ## Solution using the orthogonal projection of b on Im(A)

[15]: X2 = (A.T*A).inverse()*A.T*b
      show(X2.n())

(-0.358706351174261, -0.456448104746028, 0.100811979578199, 0.269792825026564)

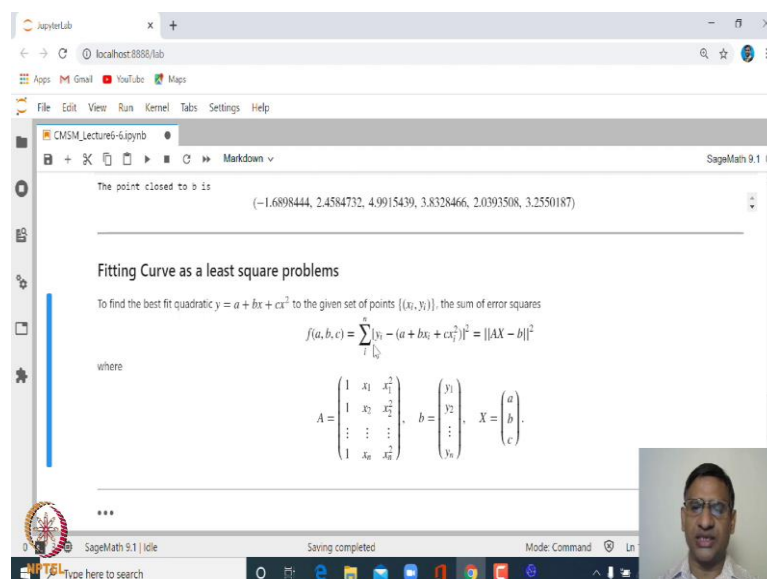
[16]: print('The point closed to b is')
      show(A*X2.n(digits=8))

The point closed to b is
(-1.6898444, 2.4584732, 4.9915439, 3.8328466, 2.0393508, 3.2550187)

```

Now, multiply this by AX_2 , this is what you get, as the least square solution. So, we have seen that orthogonal projection of b onto subspace W , which is obtained by the image space, or column vectors of basis of W , is given by A into A transpose A inverse times A transpose b . And that is what we have got. So, least squares solution can be thought of as an application to this orthogonal projection and as an application to linear algebra.

(Refer Slide Time: 10:36)



The point closed to b is

$(-1.6898444, 2.4584732, 4.9915439, 3.8328466, 2.0393508, 3.2550187)$

Fitting Curve as a least square problems

To find the best fit quadratic $y = a + bx + cx^2$ to the given set of points $\{(x_i, y_i)\}$, the sum of error squares

$$f(a, b, c) = \sum_{i=0}^n (y_i - (a + bx_i + cx_i^2))^2 = \|AX - b\|^2$$

where

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}, \quad b = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} a \\ b \\ c \end{pmatrix}.$$

Now, this again we have seen that you can fit a curve to a given set of points in the plane or also fit some kind of plane in this space and this can be generalized.

If you have to find, let us say best fit quadratic y equal to a plus bx plus cx^2 to a given set of points x_i, y_i , i going from 1 to n , then this problem, we define, what is the sum of the errors square as a function of a, b, c and this is what we got.

This is summation y_i minus a plus $b x_i$ plus $c x_i^2$, the whole square of this, that is the error of any point on this curve from this y_i . And this can be expressed as matrix norm square of AX minus b , where what is capital A ? Capital A is the column vector 1 1 and the next column is x_1, x_2, x_3 up to x_n . And the next column will be x_1^2, x_2^2, x_3^2 up to x_n^2 . And then b is y_1, y_2, \dots, y_n . So, if you take A as this and capital X is a, b, c , you take AX , what will you get? The first entry of AX will be a plus $b x_1$ plus $c x_1^2$, the second row is going to be a plus $b x_2$ plus $c x_2^2$ and so on. And this minus b , the first AX minus b , the norm squared is nothing but the first entry in this summand. So, that is how you are converting this best fit problem to a least square problem. That is minimizing this sum of the error square. So, I will not solve any problem using this, I will just leave it an exercise. We already know how to obtain this solution of this least square problem.

And the solution in this case is going to be X^* is going to be $A^T A$ inverse of that into $A^T b$ and then you multiply, this what you get X^* by A you get the least square solution.

(Refer Slide Time: 12:55)

Least square solution using QR decomposition.

$$\|Ax - b\|^2 = \|QRx - b\|^2 = \|R^T(Q^T b)\|^2.$$

Here $R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$, where R_1 is a $m \times m$ upper triangular matrix and $Q^T b = \begin{bmatrix} c \\ d \end{bmatrix}$, c is of size m . Thus

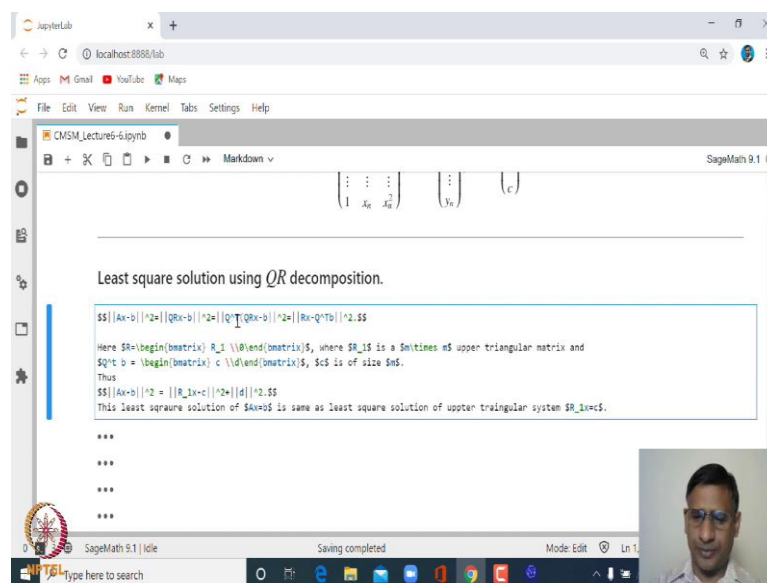
$$\|Ax - b\|^2 = \|R_1 x - c\|^2 + \|d\|^2.$$

This least square solution of $Ax = b$ is same as least square solution of upper triangular system $R_1 x = c$.

This least square solution, we can also obtain using QR factorization. So, how do we do that? Let us see this. We want to minimize, Ax minus b the whole square. Now suppose A has QR factorization, A is equal to Q times R . Then this norm of Ax minus b whole square is QR minus b the norm square.

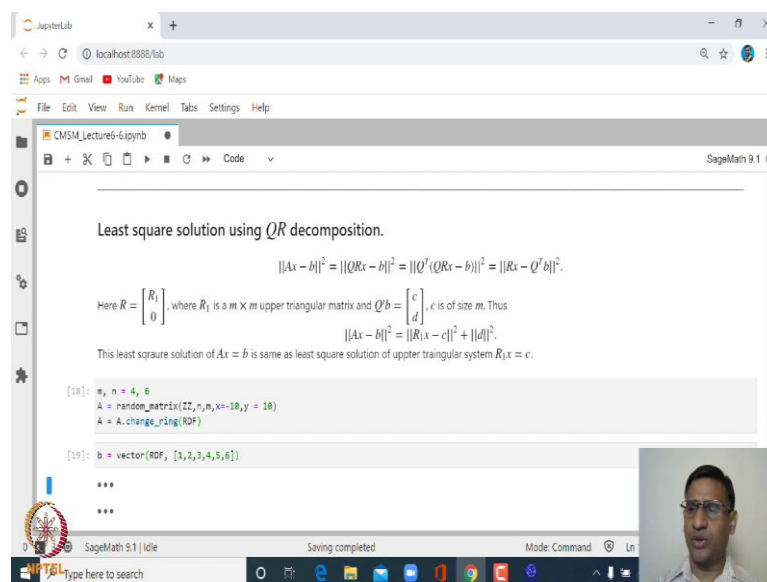
Now, we know that, since Q is orthogonal matrix the norm of Qx will be same as norm of x . So, if I take norm of this square, that is same as multiply this whole thing by Q transpose. Q is orthogonal therefore, Q transpose will also be orthogonal. But on the left hand side Q transpose Q is identity. So, this terms reduces to $R x$ this would be a small x and minus Q transpose b .

(Refer Slide Time: 14:02)



This let me change this to small x and let me also add this intermediate step here. This is equal to, we are multiplying this by Q transpose.

(Refer Slide Time: 14:23)



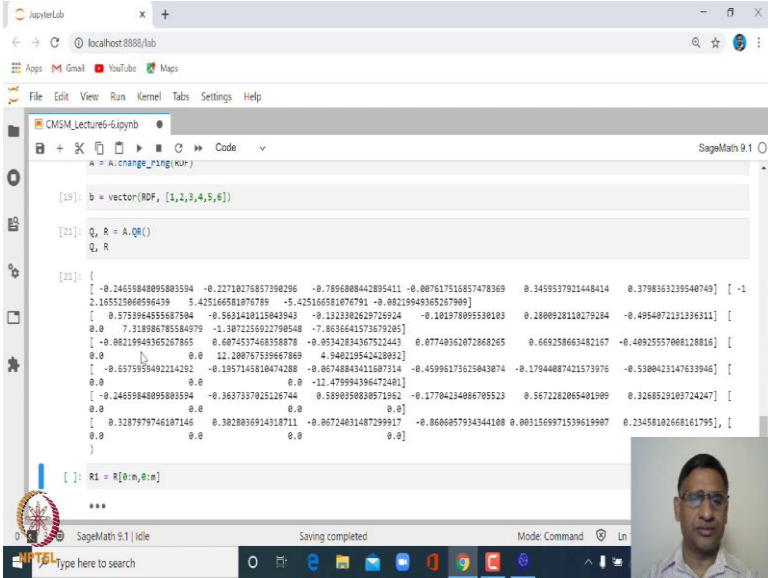
So, norm of Ax minus b whole square is same as norm of Q transpose Ax minus b, but A you replace it by QR. So, this now reduces to norm of Ax minus b whole square reduces, to norm of Rx minus Q transpose b the whole square. Now, this least square problem Ax equal to b has reduced to solving Rx is equal to Q transpose b. Since R is in an upper triangular matrix this is the same as solving this upper triangular system which

is much easier. Here this R is upper triangular, you split this R as R_1 and 0 , 0 is a 0 matrix, where R_1 is m cross m upper triangular matrix and Q transpose b you also split as c, d , where c is of size m and the d is going to be the size n minus m .

Therefore the norm of Ax minus b whole square, which is norm of Rx minus Q transpose b whole square will reduce to $R_1 x$ minus c , the norm square plus norm of d squared. But norm d is actually a fixed constant. So, minimizing norm of Ax minus b whole square is same as saying minimize this $R_1 x$ minus c whole square. So, the least solution of Ax equal to b is same as least square solution of $R_1 x$ equal to c .

Let us see how we can solve this. Again let us take a random matrix 4 cross 6 random matrix and over integer. Since we want to find QR factorization, you may have to change the underlying domain from which the entries are taken. So, let us change this ring to RDF, that is extended real field. The vector b is again 1, 2, 3, 4, 5, 6. Then what do we do? We define QR factorization of A , Q, R is equal to A dot QR .

(Refer Slide Time: 16:42)



```

In [10]: b = vector(RDF, [1,2,3,4,5,6])

In [11]: Q, R = A.QR()
          Q, R

In [12]:
          [
            [-0.24655948095893594, -0.22718275857390296, -0.7896888442895411, -0.087617516857478369, 0.3459537921448414, 0.3798363239548749], [-1
            2.165525608596439, 5.425166581876789, -5.425166581876791, -0.082194949365267989],
            [0.5733944553987594, -0.5624420113843943, -0.122302629728924, -0.181978095338189, 0.2808928118279284, -0.4894872191336311], [
            0.0, 7.218986785584979, -1.5872255612790548, -7.8636441573079285],
            [-0.082194949365267985, 0.6874537468358878, -0.85342834367512443, 0.87748362072668265, 0.669258663482167, -0.48925557908128816], [
            0.0, 0.0, 12.280767539667889, 4.948219542428932],
            [-0.6575959492214292, -0.195714518474288, -0.86748843411687314, -0.45996173625843874, -0.17944887421573976, -0.5388423147633946], [
            0.0, 0.0, 0.0, -12.479994386472481],
            [-0.24655948095893594, -0.3637337825126744, 0.5896939838571962, -0.17784234086785523, 0.5672282865401989, 0.3268529289724247], [
            0.0, 0.0, 0.0, 0.0, 0.0],
            [0.3287879746187146, 0.3828836914318711, -0.86724831487299917, +0.8686857934344188, 0.8831568971539619907, -0.23458182668161795], [
            0.0, 0.0, 0.0, 0.0, 0.0]
          ]

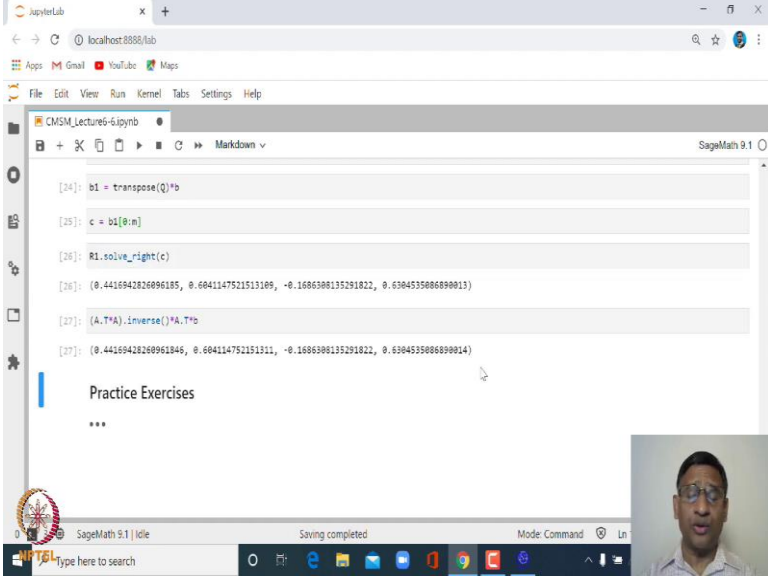
In [ ]: R1 = R[0:m,0:m]

***

```

So, we can print what are Q and R . So, this is the two matrices Q and R . It looks slightly complicated, but since this is a 4 cross 6 matrix it looks like this.

(Refer Slide Time: 17:15)



The screenshot shows a JupyterLab window with a browser address bar at localhost:8888/lab. The main editor displays a SageMath notebook with the following code and output:

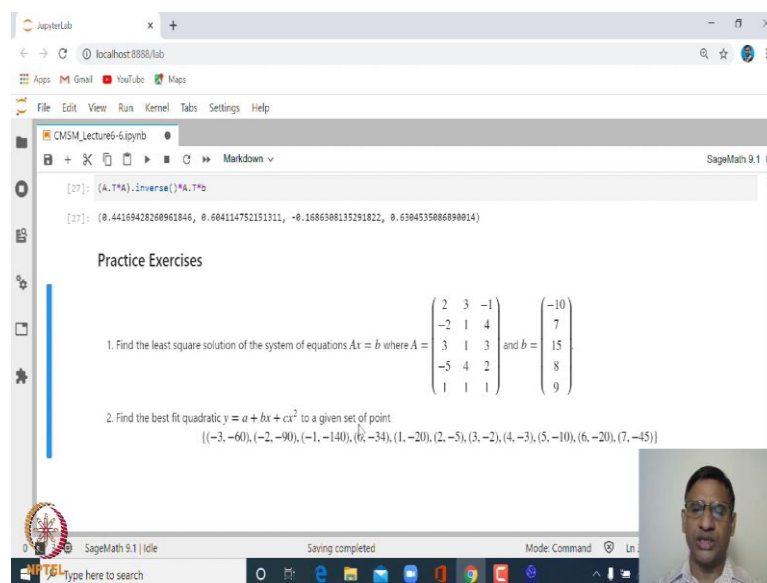
```
[24]: b1 = transpose(Q)*b
[25]: c = b1[0:m]
[26]: R1.solve_right(c)
[26]: (0.4416942826096185, 0.6041147521513189, -0.1686386135291822, 0.6304535886898913)
[27]: (A.T*A).inverse()*A.T*b
[27]: (0.44169428260961846, 0.6041147521513111, -0.1686386135291822, 0.6304535886898914)
```

Below the code, there is a section titled "Practice Exercises" followed by three asterisks. The bottom status bar indicates "SageMath 9.1" and "Saving completed". A small video feed of a person is visible in the bottom right corner.

Now, let us define $R1$. $R1$ is, from R you take out the first m rows and m columns that is your $R1$, and then define $b1$ as Q transpose b , that is $b1$. Again from $b1$ you define this as c and d . So for c , take the first m component of $b1$ that is your c . Now remember we need to solve $R1 x$ is equal to c , that is what we need to do. So, what we have to do? We need to solve $R1$ dot solve right and bracket c . That is the solution you are getting, so that is your x star. Now if you look at what is the solution using orthogonal projection.

So, it is A transpose A inverse of this into A transpose b . What you get here is same as what you get in this case. So, solving this least square problem is same as solving this system of linear equation $R1 x$ is equal to c . Since $R1$ is invertible matrix, solving it is quite easy you can obtain this using Gaussian elimination method or simply take inverse of $R1$ multiplied by c . So, that how you can solve this least square problem using QR factorization except that you need to do this computation of finding QR factorization of A . But once you have done this, then it is quite easy.

(Refer Slide Time: 18:59)



Now let us look at some exercises. Just two exercises here.

Find least squares solution of this system of linear equations which is 5 equations in 3 variables and b is this vector.

And find the best fit quadratic y equals to $a + bx + cx^2$ to a given set of points this. This problem, solve using minimizing norm of $AX - b$ whole square. We have already seen what should be A , what should be X what should be b in this case. So, that is what you have to use and once you have a b and capital X , then the least square solution will be $A^T A$ inverse of that into $A^T b$ that is it.

Instead of quadratic, you can try to fit cubic, you can try to fit n th degree polynomial. So, if you want to fit n th degree polynomial, this matrix A will have first column as all 1, 1, 1, second column will be x_1, x_2, x_n , third column will be x_1^2, x_2^2, x_n^2 and so on. Last column will be x_1 to the power n, x_2 to the power n, \dots, x_n to the power n . So, fitting n th degree polynomial to a given set of points is quite easy.

Similarly, if you have to fit a plane to a given set of points in R^3 , then plane can be written as $z = a + bx + c$. So, in this case the matrix A is going to be the first column will be all 1, 1, 1; second column will be coefficient of x , third column will

be coefficient of y . And b is going to be the coefficient for z . So, z_1, z_2, z_n will be the column vector will be b , and x_1, x_2, x_n will be the second column of A , y_1, y_2, y_n will be the third column of the matrix A . That is how you can solve this least square problem and as an application to linear algebra.

There are other methods also to solve these least square problems, but we will not get into all these things.

Thank you very much, we will see you in the next class. In the next class we will look at singular value decomposition which is again a concept which has several applications.

Thank you.