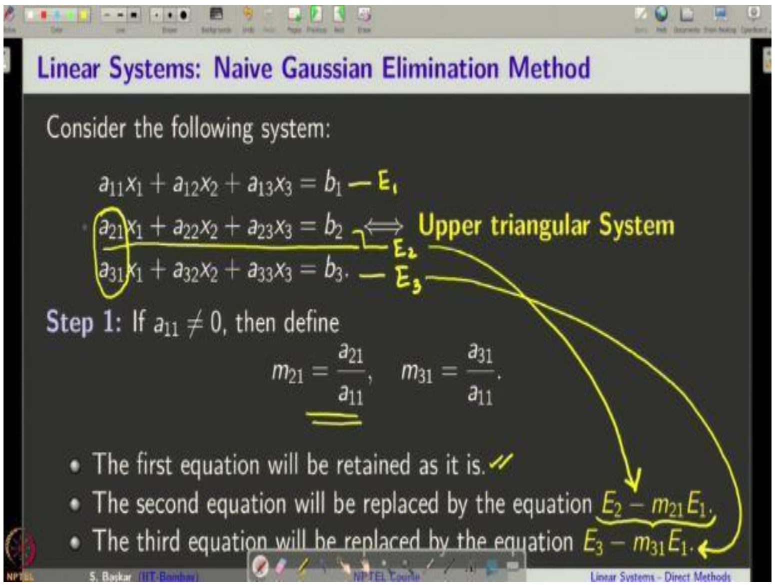
Numerical Analysis

Prof. S. Baskar
Department of Mathematics
Indian Institute of Technology-Bombay

Lecture-13
Linear System: Python Coding for Naive Gaussian Elimination Method

Hi, in this lecture, we will learn to implement Gaussian elimination method as a python code. We will start our discussion with a quick recall of the Naive Gaussian elimination method.

(Refer Slide Time: 00:34)



Linear Systems: Naive Gaussian Elimination Method

Consider the following system:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \quad - E_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \quad \Longleftrightarrow \text{Upper triangular System} \quad E_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3. \quad - E_3$$

Step 1: If $a_{11} \neq 0$, then define

$$m_{21} = \frac{a_{21}}{a_{11}}, \quad m_{31} = \frac{a_{31}}{a_{11}}.$$

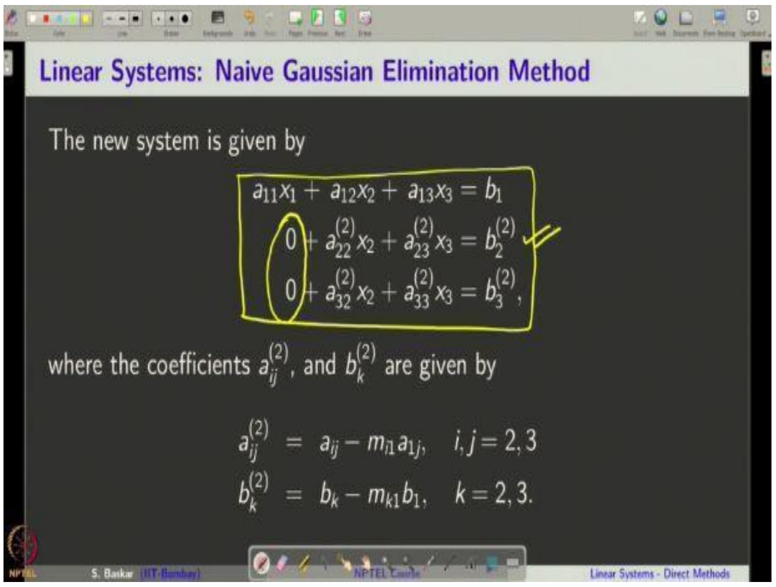
- The first equation will be retained as it is.
- The second equation will be replaced by the equation $E_2 - m_{21}E_1$.
- The third equation will be replaced by the equation $E_3 - m_{31}E_1$.

Remember, we will implement only the Naive Gaussian elimination method. We will not go to implement the modified Gaussian elimination method in this class. As we did in the theory part we will implement the Naive Gaussian elimination method only for a 3×3 system. Recall that the Naive Gaussian elimination method first does an elimination process in which we will convert the given system into a upper triangular system by doing certain elementary row operations, the elementary row operations are as follows.

First, we will define certain elements like m_{21}, m_{31} which are given like this. This should hold only if a_{11} is not equal to 0; therefore we have to first check whether, a_{11} is not equal to 0. If it is not equal to 0 then only you will go to calculate m_{21} and m_{31}. And then next what we will do is we will retain the first equation as it is, we will replace the second equation by this expression.

What is this? We will call the first equation as E_1, the second equation as E_2 and the third equation as E_3 and then we will replace E_2 by this equation, which is $E_2 - m_{21}E_1$. Similarly, E_3 is replaced by this equation that is$E_3 - m_{31}E_1$. This is what we have to do in step one and that will make these terms to be 0**.**

(Refer Slide Time: 02:38)



Linear Systems: Naive Gaussian Elimination Method

The new system is given by

$$\boxed{\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ 0 + a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 &= b_2^{(2)} \\ 0 + a_{32}^{(2)}x_2 + a_{33}^{(2)}x_3 &= b_3^{(2)}, \end{aligned}}$$

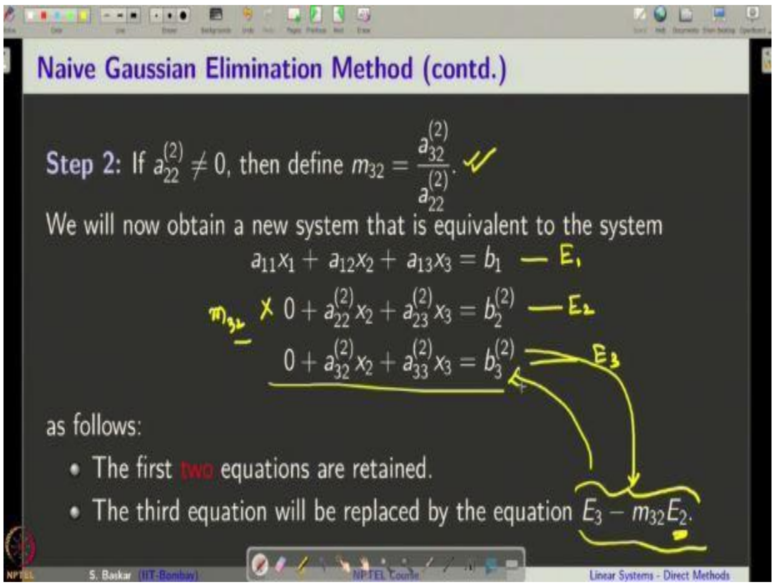
where the coefficients $a_{ij}^{(2)}$, and $b_k^{(2)}$ are given by

$$\begin{aligned} a_{ij}^{(2)} &= a_{ij} - m_{i1}a_{1j}, \quad i,j = 2,3 \\ b_k^{(2)} &= b_k - m_{k1}b_1, \quad k = 2,3. \end{aligned}$$

S. Baskar (IIT-Bombay) NPTEL Course Linear Systems - Direct Methods

And thereby you will get your reduced system like this, where the second equation and the third equation first coefficients will be 0.

(Refer Slide Time: 02:58)



Naive Gaussian Elimination Method (contd.)

Step 2: If $a_{22}^{(2)} \neq 0$, then define $m_{32} = \dfrac{a_{32}^{(2)}}{a_{22}^{(2)}}$.

We will now obtain a new system that is equivalent to the system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \quad — E_1 \\ m_{32} \times \quad 0 + a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 &= b_2^{(2)} \quad — E_2 \\ 0 + a_{32}^{(2)}x_2 + a_{33}^{(2)}x_3 &= b_3^{(2)} \quad — E_3 \end{aligned}$$

as follows:

- The first two equations are retained.
- The third equation will be replaced by the equation $E_3 - m_{32}E_2$.

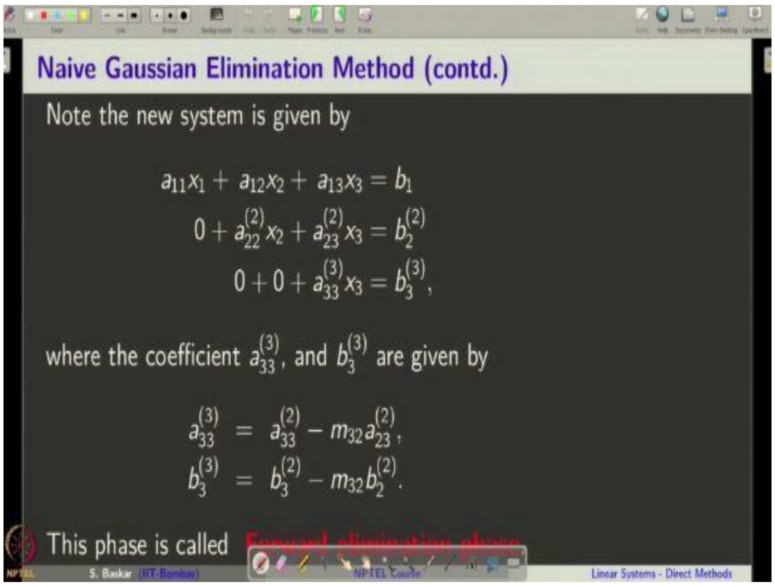
S. Baskar (IIT-Bombay) NPTEL Course Linear Systems - Direct Methods

And then we will go for the second step, where we will define m_{32} like this. For that once again we have to check whether $a_{22}^{(2)}$, what is this? this is nothing but the coefficient of x_2 in the second equation of the reduced system after step 1 that is this one. So, that you have to take and check, whether it is non zero. If it is non-zero then only you will go further into the algorithm, otherwise you have to stop your code.

Now, if it is non-zero then you will first find m_{32} and then you will go to do the elimination process with the third equation. How will you do that? Well, for the second step you will retain the first two equations as they are and then you will go to replace the third equation by the third equation – m_{32} into the second equation that is you will multiply this equation with m_{32} and then subtract with the third equation.

Recall, we still call the first equation as E_1, second as now E_2 and E_3 and then you will replace this new equation with the old third equation.

(Refer Slide Time: 04:22)



Naive Gaussian Elimination Method (contd.)

Note the new system is given by

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ 0 + a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 &= b_2^{(2)} \\ 0 + 0 + a_{33}^{(3)}x_3 &= b_3^{(3)}, \end{aligned}$$

where the coefficient $a_{33}^{(3)}$, and $b_3^{(3)}$ are given by

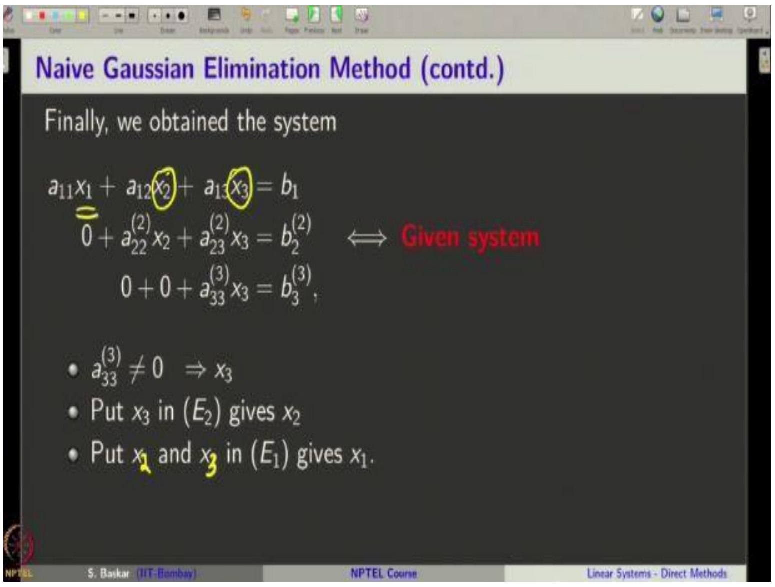
$$\begin{aligned} a_{33}^{(3)} &= a_{33}^{(2)} - m_{32}a_{23}^{(2)}, \\ b_3^{(3)} &= b_3^{(2)} - m_{32}b_2^{(2)}. \end{aligned}$$

This phase is called Forward elimination phase

S. Baskar (IIT-Bombay) NPTEL Course Linear Systems - Direct Methods

And thereby you will get an upper triangular matrix like this**.** This is what the Gaussian elimination method does. We have already learned it in one of our previous lectures. And now we will go to do the backward substitution process.

(Refer Slide Time: 04:40)



Naive Gaussian Elimination Method (contd.)

Finally, we obtained the system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ 0 + a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 &= b_2^{(2)} \\ 0 + 0 + a_{33}^{(3)}x_3 &= b_3^{(3)}, \end{aligned} \iff \text{Given system}$$

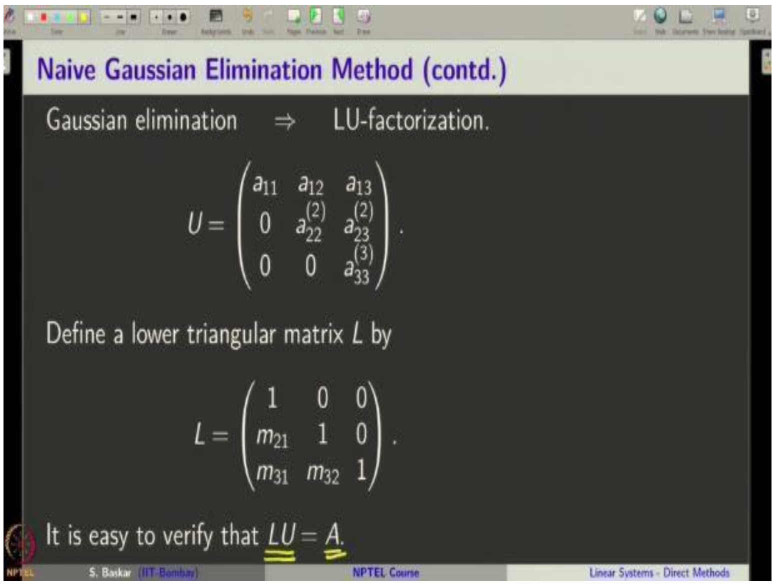
- $a_{33}^{(3)} \neq 0 \Rightarrow x_3$
- Put x_3 in (E_2) gives x_2
- Put x_2 and x_3 in (E_1) gives x_1.

S. Baskar (IIT-Bombay) NPTEL Course Linear Systems - Direct Methods

The elimination process is now over and we got an upper triangular matrix, which is known to be equivalent to the given system in the sense, that the solution of the given system is the same as the solution of the reduced upper triangular system**.** Now, at the end you have to go for the back substitution. For that we first have to check whether a_{33} is not equal to 0. If so then you can obtain x_3 from this equation and then once you have x_3 you will substitute x_3 into the second equation and get x_2 here.

And once you have this is x_2 and x_3 then we will go to substitute them in the first equation to get the value of x_1. So, that is the back substitution process.

(Refer Slide Time: 05:57)



Naive Gaussian Elimination Method (contd.)

Gaussian elimination $\Rightarrow$ LU-factorization.

$$U = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{pmatrix}.$$

Define a lower triangular matrix L by

$$L = \begin{pmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & m_{32} & 1 \end{pmatrix}.$$

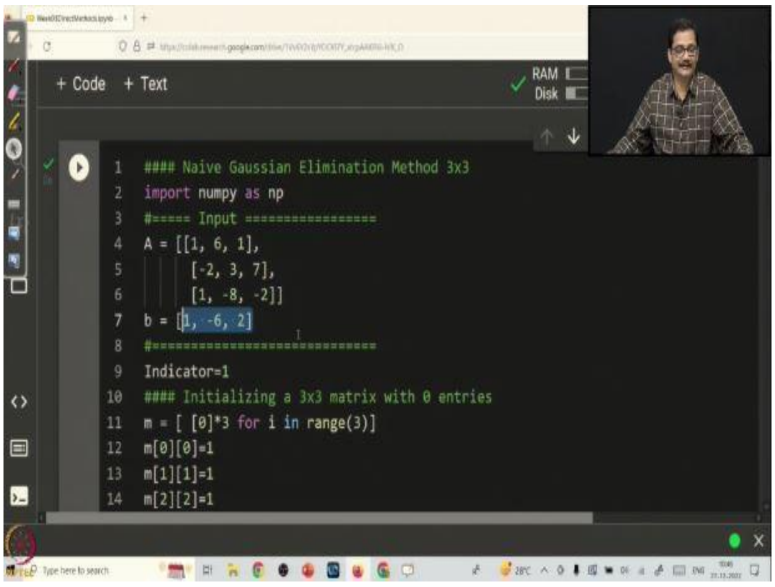
It is easy to verify that $LU = A$.

S. Baskar (IIT-Bombay) NPTEL Course Linear Systems - Direct Methods

If you recall we have also observed that you can have a LU-factorization of our matrix a using Gaussian elimination method, where U is nothing but the upper triangular matrix that we got here that itself will be taken as *U*. And then the lower triangular matrix is formed by collecting all the *m* values and arranging them in this format with all the diagonal elements as 1**.**

So, this is the upper triangular matrix and this is the lower triangular matrix and you can check that *A* can be obtained by multiplying *L* into *U*. So, our interest is to develop a python code for the Naive Gaussian elimination method. And we also want to get the upper triangular matrix as well as the lower triangular matrix. Let us see how to develop the python code for this.

(Refer Slide Time: 07:12)



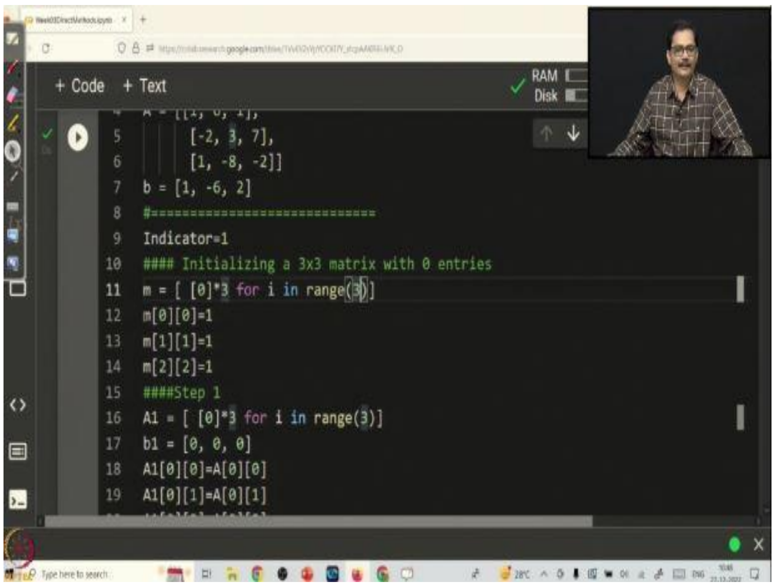
```
#### Naive Gaussian Elimination Method 3x3
import numpy as np
#===== Input ================
A = [[1, 6, 1],
     [-2, 3, 7],
     [1, -8, -2]]
b = [1, -6, 2]
#============================
Indicator=1
#### Initializing a 3x3 matrix with 0 entries
m = [ [0]*3 for i in range(3)]
m[0][0]=1
m[1][1]=1
m[2][2]=1
```

To save time, I have already developed the python code. I will just go to explain it here please note that our intention here is not to develop an efficient code, but just to implement the method into a python code. Therefore, often what we will do is we will implement the method in the way it is explained theoretically. In that process sometime we may even lose the efficiency of the coding part, but our intention is to learn coding as the first step.

Therefore we will often try to implement the methods just like how we have written mathematically and explained, the same way we will try to implement it. Efficiency is all together at different skill, which one has to develop later. Here, we will not concentrate on developing an efficient code. This is generally followed in all the codes that we develop. Sometimes, we may develop a code which is also efficient, but sometimes we may lose the efficiency, just because we want to put the coding part just like how we have explained it mathematically.

Having said that, let us come to explain the Naive Gaussian elimination code only for a 3 × 3 system. Just for an example, I have taken the system as A = [[1, 6, 1], [-2, 3, 7], [1, -8, -2]] this is just I have chosen randomly. And I am also taking the right-hand side vector b and that is given as [1, -6, 2]. If you want to change your matrix you just have to change this A parameter and also if you want to change the right hand side vector you have to just change here.

(Refer Slide Time: 09:18)

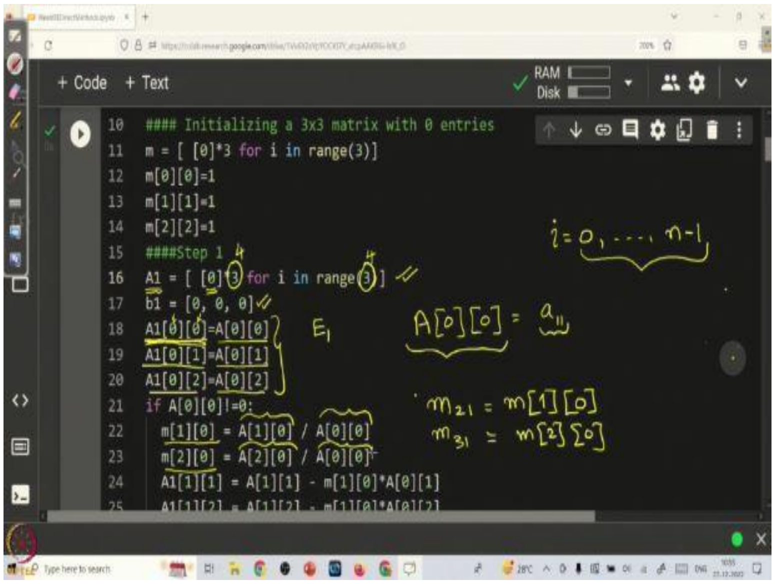


```
          [-2, 3, 7],
          [1, -8, -2]]
b = [1, -6, 2]
#=============================
Indicator=1
#### Initializing a 3x3 matrix with 0 entries
m = [ [0]*3 for i in range(3)]
m[0][0]=1
m[1][1]=1
m[2][2]=1
####Step 1
A1 = [ [0]*3 for i in range(3)]
b1 = [0, 0, 0]
A1[0][0]=A[0][0]
A1[0][1]=A[0][1]
```

And now, let us go to do the step 1. So, this is the first elimination process. Since, our interest is to formulate the matrix L, which is the lower triangular matrix. I am just defining my all my m parameters in the form of a matrix. Remember, this is just an idiom in python, which will initialize a 2-dimensional array with all the entries as 0. This is very useful. You can also do it with a pair of nested for loops, but this is often handy for us to do. You just have to remember this format and everywhere you can copy and paste wherever you want to initialize a matrix.

Here my interest is to initialize the 3 × 3 matrix. Therefore, the parameter 3 is sitting in these two places and then what I am doing is all the diagonal elements are initialized as 1. If you remember in the LU factorization we want all the diagonal elements as 1. Therefore, the matrix m, that I am defining here is basically the matrix L later I will take it as L finally. I could have just defined it as L, but in the theory we are doing all these parameters with the notation m, therefore I have initialized it as m and then all the diagonal elements are taken as 1.

(Refer Slide Time: 11:08)



```
#### Initializing a 3x3 matrix with 0 entries
m = [ [0]*3 for i in range(3)]
m[0][0]=1
m[1][1]=1
m[2][2]=1
####Step 1
A1 = [ [0]*3 for i in range(3)]
b1 = [0, 0, 0]
A1[0][0]=A[0][0]
A1[0][1]=A[0][1]
A1[0][2]=A[0][2]
if A[0][0]!=0:
  m[1][0] = A[1][0] / A[0][0]
  m[2][0] = A[2][0] / A[0][0]
  A1[1][1] = A[1][1] - m[1][0]*A[0][1]
  A1[1][2] = A[1][2] - m[1][0]*A[0][2]
```

Now for the first step, the eliminated matrix, that is the coefficient matrix of this system is what I am calling as A1 in my code, I am just initializing my A1 here. Again it is a 3 × 3 matrix therefore I have these parameters 3 here. If you put 4 here and 4 here you will get a 4 × 4 matrixes and so on. And all the elements of the matrix will be initialized as 0. Similarly, I am initializing the corresponding right hand side vector b as b1.

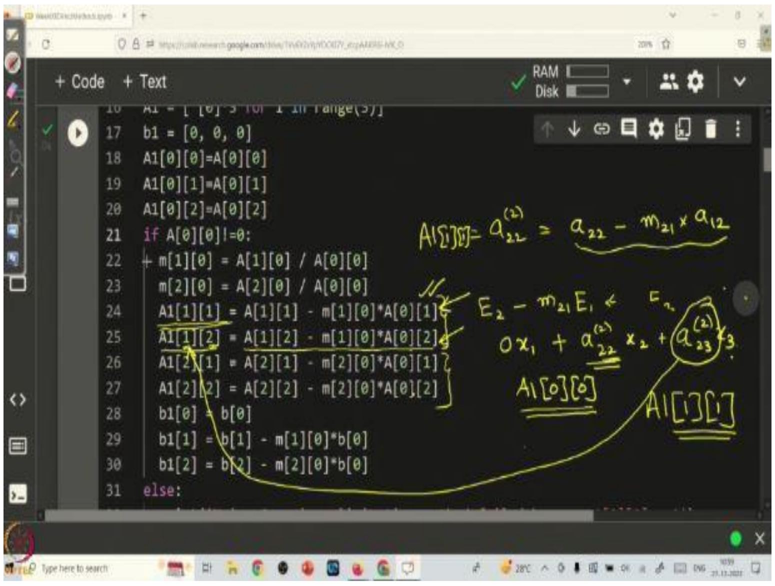
Mathematically, we have this notation and this is what I am initializing as b1 in my code. Now, if you recall the first equation is retained as it is**.** So, E_1 is retained as it is we are not going to disturb it, that is why I have taken A1[0][0] it means, what A1[0][0] is nothing but a_{11}, if you recall I have told in the last class that python always starts it is index from 0 and then it goes up to n – 1.

If you have n elements in your sequence it always goes from 0 to n - 1. Therefore if you are trying to access a_{11} mathematically then you should actually do A[0][0]. Perhaps we can write it as A here. So, this is what is so suppose mathematically you have a_{ij} then in the code it is A[i-1][j-1] is what we have to use in the code as the notation**.** So, that is what I am doing a_{11} is stored in the first element of this matrix.

And similarly this is a_{12} is taken as it is in the first step, a_{13} is taken as it is in the first step. So, this is what here we have done. After the first elimination step, the first equation should not be changed a_{11} should be as it is, a_{12} should be as it is and a_{13} should be as it is. That is what we are doing in the code here and that finishes the first equation. Now, we have to come for the second and third equation elimination process.

If you recall, we have to define m_{21} and m_{31}. In the python notation this has to be written as m, it is 2 therefore in python it should be 1 and the second index is 1, therefore we have to have it as 0. Similarly, this is m[2][0]. So, that is what I am doing here m[1][0], m[2][0] and that is a_{21} divided by a_{11}, similarly this is a_{31} divided by a_{11} all the index should be reduced by 1, that you should always keep in mind, in order to adopt your mathematics to python notation, this one thing that you have to remember.

(Refer Slide Time: 15:37)



```
b1 = [0, 0, 0]
A1[0][0]=A[0][0]
A1[0][1]=A[0][1]
A1[0][2]=A[0][2]
if A[0][0]!=0:
    m[1][0] = A[1][0] / A[0][0]
    m[2][0] = A[2][0] / A[0][0]
    A1[1][1] = A[1][1] - m[1][0]*A[0][1]
    A1[1][2] = A[1][2] - m[1][0]*A[0][2]
    A1[2][1] = A[2][1] - m[2][0]*A[0][1]
    A1[2][2] = A[2][2] - m[2][0]*A[0][2]
    b1[0] = b[0]
    b1[1] = b[1] - m[1][0]*b[0]
    b1[2] = b[2] - m[2][0]*b[0]
else:
```

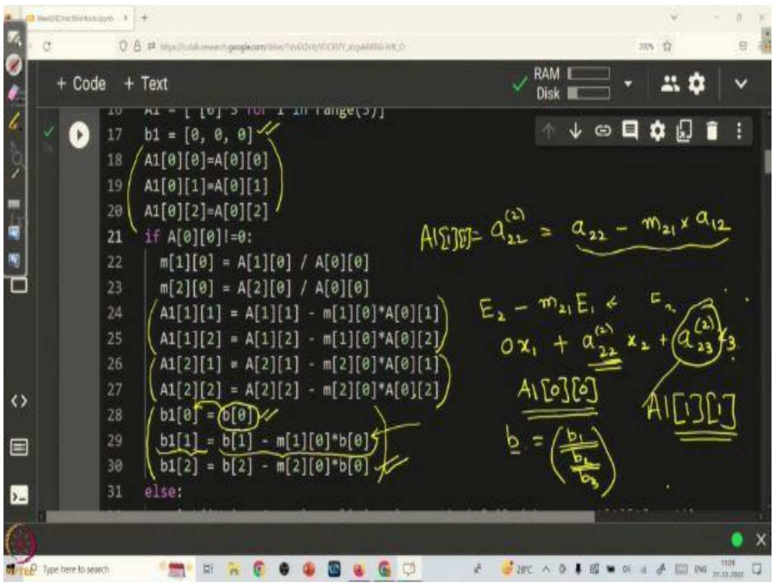
+ Code
+ Text
RAM
Disk
$A1[1][1] = a_{22}^{(2)} = a_{22} - m_{21} \times a_{12}$
$E_2 - m_{21}E_1 \leftarrow E_2$
$0x_1 + a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3$
A1[0][0]
A1[1][1]

Next is the second equation. If you remember the second equation is $E_2 - m_{21}E_1$ should be replaced by the equation E_2**.** Therefore in the matrix A1 remember here the first element is 0 into x_1. Therefore we should not touch A1[0][0], because we have already initialized the entire A1 matrix as 0 matrix. Therefore we will not touch this element at all. We want it to be 0 we only want to take this element mathematically we had this notation.

Now instead of this we are using the notation A1 therefore it should be A1[1][1] because this is 2, 2, therefore in python it should be 1, 1 that is what I am writing here that is A1[1][1] = A[1][1] - m[1][0]. Remember this is nothing but $a_{22}^{(2)}$ that is what we have taken as A1[1][1]. That is nothing but a_{22} minus that is the second equation minus $m_{21}a_{12}$**.** If you go back you can see this is the expression, we have defined in the method and that is implemented in this step.

And similarly you can now put $a_{23}^{(2)}x_3$, that is what we are writing here A1, instead of 2 we have 1 here, instead of 3 we have 2 here. Therefore, this element is what we are actually trying to compute here and that is given by this expression. You can compare it with the expression that we have given in the method theoretically and see how it comes. Similarly, you can go to do the third equation elimination and you can see that that is given by this expression in the code.

(Refer Slide Time: 18:31)



```
b1 = [0, 0, 0]
A1[0][0]=A[0][0]
A1[0][1]=A[0][1]
A1[0][2]=A[0][2]
if A[0][0]!=0:
  m[1][0] = A[1][0] / A[0][0]
  m[2][0] = A[2][0] / A[0][0]
  A1[1][1] = A[1][1] - m[1][0]*A[0][1]
  A1[1][2] = A[1][2] - m[1][0]*A[0][2]
  A1[2][1] = A[2][1] - m[2][0]*A[0][1]
  A1[2][2] = A[2][2] - m[2][0]*A[0][2]
  b1[0] = b[0]
  b1[1] = b[1] - m[1][0]*b[0]
  b1[2] = b[2] - m[2][0]*b[0]
else:
```

$A1[1][1] = a_{22}^{(2)} = \underbrace{a_{22} - m_{21} \times a_{12}}$
$E_2 - m_{21}E_1 \leftarrow E_2$
$0x_1 + \underline{a_{22}^{(2)}} x_2 + \boxed{a_{23}^{(2)}} x_3$
$\underline{A1[0][0]}$
$\underline{A1[1][1]}$
$\underline{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$

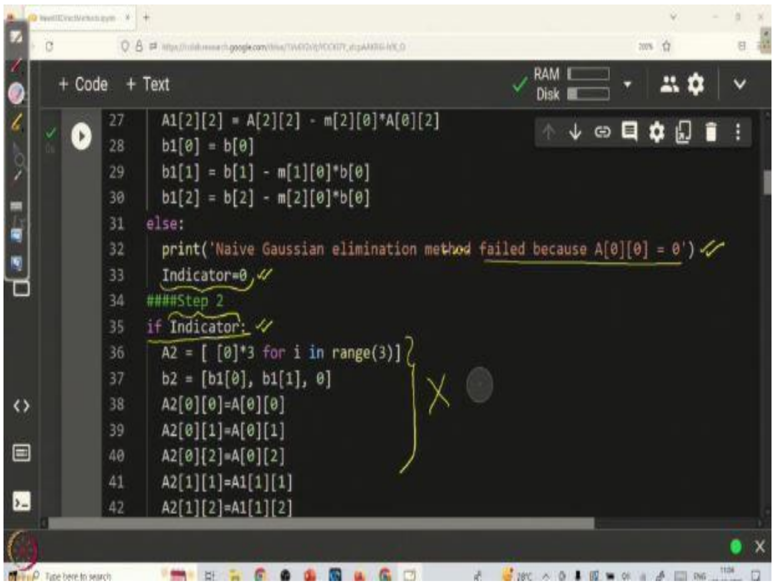
So, that defines your left-hand side coefficient matrix, that is**,** this is the first equation then this comes the second equation, where the first coefficient is 0 therefore we are not doing any computation with that. Similarly, the third equation is defined here again with the first coefficient. That is the coefficient of x_1 is set as 0 automatically when we are initializing the matrix A1.

And later we are not putting any effort to compute it, because we already know that its value should be 0. Therefore, we just put that and we never compute that. Now comes the right-hand side calculation that is the vector b is now changed to a new vector and the new vector is already initialized as b1 with 0 as values. Now, we have to put the actual values in b1. Remember the first equation is not changed.

Therefore the value in b[0] should be kept in b1 also**.** This is the right-hand side first coordinate of the vector b. And now b1[1] this is nothing but the second coordinate of the right hand side vector, after the first elimination process. You can go back theoretically and see what is the expression for that and come back and try to understand its corresponding notation in the python code, you will clearly see that this is the notation that needs to be written.

Similarly, the third equations hand side is given like this. So, with this we have completely calculated the reduced system at the step 1**.** Now let us go to do the step 2. For step 2, if you recall what we have to do, we have to first check whether this $a_{22}^{(2)}$ that should not be equal to 0 if that is so, then we will go to calculate m_{32}. Let us see how to do that. Well here also in the step 1 we have checked whether a_{11} is not equal to 0, only when a_{11} is not equal to 0 we will do this calculation.

(Refer Slide Time: 21:07)

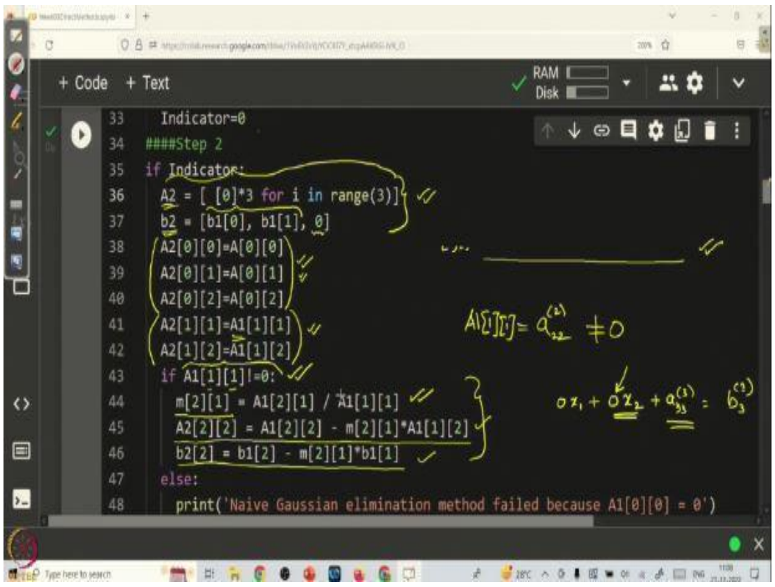


```
  A1[2][2] = A[2][2] - m[2][0]*A[0][2]
  b1[0] = b[0]
  b1[1] = b[1] - m[1][0]*b[0]
  b1[2] = b[2] - m[2][0]*b[0]
else:
  print('Naive Gaussian elimination method failed because A[0][0] = 0')
  Indicator=0
####Step 2
if Indicator:
  A2 = [ [0]*3 for i in range(3)]
  b2 = [b1[0], b1[1], 0]
  A2[0][0]=A[0][0]
  A2[0][1]=A[0][1]
  A2[0][2]=A[0][2]
  A2[1][1]=A1[1][1]
  A2[1][2]=A1[1][2]
```

If not what we will do we will simply put a warning command that Naive Gaussian elimination method failed, because of A[0][0] is 0**.** And then I have defined a variable indicator at the beginning of this code, if you go back, you can see I have initialized it as 1, that is, just as an indicator I will see whether I have encountered this 0 element anywhere in my steps**.** Anywhere it encounters then it will just put indicator as 0, it means the Gaussian elimination method is already failed**.**

And then it comes to the second step if the first step was successful then my initialized value of indicator will be there, which is 1. Now it will take as 1 here, this if condition will allow the control to go into it when this indicator carries any non-zero value**.** If it carries zero value then it will not get into this if condition. So, in the first step if the Naive Gaussian elimination method failed, then indicator is 0 therefore this will not be executed, otherwise it will carry the value 1 from the initialized level and therefore this if condition will be executed.

(Refer Slide Time: 22:43)



```
  Indicator=0
####Step 2
if Indicator:
  A2 = [ [0]*3 for i in range(3)]
  b2 = [b1[0], b1[1], 0]
  A2[0][0]=A[0][0]
  A2[0][1]=A[0][1]
  A2[0][2]=A[0][2]
  A2[1][1]=A1[1][1]
  A2[1][2]=A1[1][2]
  if A1[1][1]!=0:
    m[2][1] = A1[2][1] / A1[1][1]
    A2[2][2] = A1[2][2] - m[2][1]*A1[1][2]
    b2[2] = b1[2] - m[2][1]*b1[1]
  else:
    print('Naive Gaussian elimination method failed because A1[0][0] = 0')
```

+ Code
+ Text
RAM
Disk
$A1[1][1] = a_{22}^{(2)} \neq 0$
$0x_1 + 0x_2 + a_{33}^{(3)} = b_3^{(3)}$

If this if condition is executed let us see what happens. Again, you will initialize the matrix A2 which is our final tri-diagonal matrix, because we are working with 3 × 3 system. Therefore, we will only go up to two steps**.** So, we will be finishing our Gaussian elimination method with the second step. Therefore, this A2 will be our final upper triangular matrix. At the beginning of this step, I am initializing it as a 3 ×3 zero matrix and I am also initializing the right hand side vector b2, when I know already that the first two coefficients of b2 will not be changed.

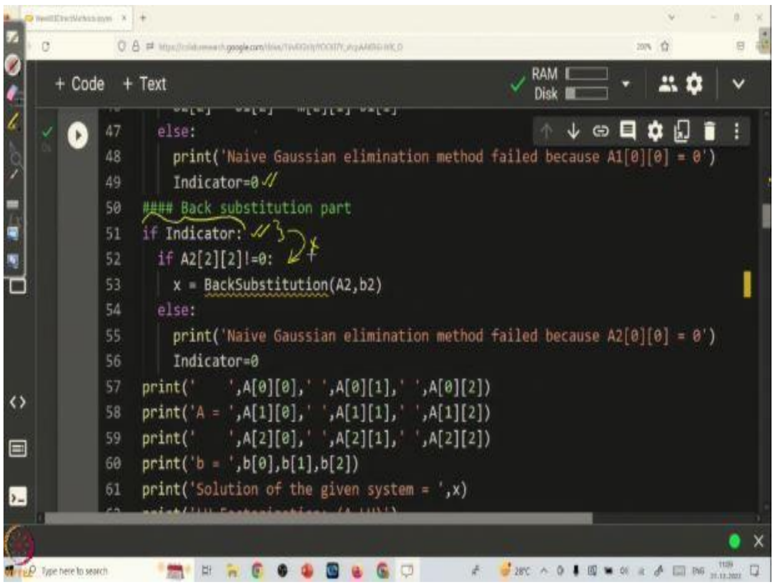
In fact, I can put it at the initialization level itself and then keep the third coordinate as 0 and later I will fill it up with the corresponding value. Now, the first two equations will not be changed at all when compared to the previous step. Therefore, the first equation is not changed right from the given equation. Therefore, I am taking the first equation same as the given first equation. The second equation is not changed from the step 1.

Therefore, the second equation is just taken from the A1 matrix, because A1 is formulated at step 1**.** And now for third equation we have to do the elimination process. For that I have to check whether A1[1][1] that is a_{22}, this is $a_{22}^{(2)}$. That is what we have called as A1[1][1]. That should not be equal to 0. This is what we are checking theoretically and I am doing it here. If it is not equal to 0 then I will go to find m_{32}**.**

In python it should be m[2][1], 1 less always and that is given by this. And then in the third equation remember the first one will be $0x_1$, which is eliminated in the first step. Now in the present step we are also eliminating the coefficient of x_2. So, when we know that this is 0, as usual we will not put any effort to compute it. It is already initialized in this step itself as 0. So, we will retain it as it is plus $a_{33}^{(3)}$. This is what we will be now computing here and similarly equal to $b_3^{(3)}$ and that is computed here.

You can go back to the expressions given theoretically and see how we have written this**.** And once you have it then finally we are having our upper triangular matrix. Now, you have to go for the back substitution process.

(Refer Slide Time: 25:54)

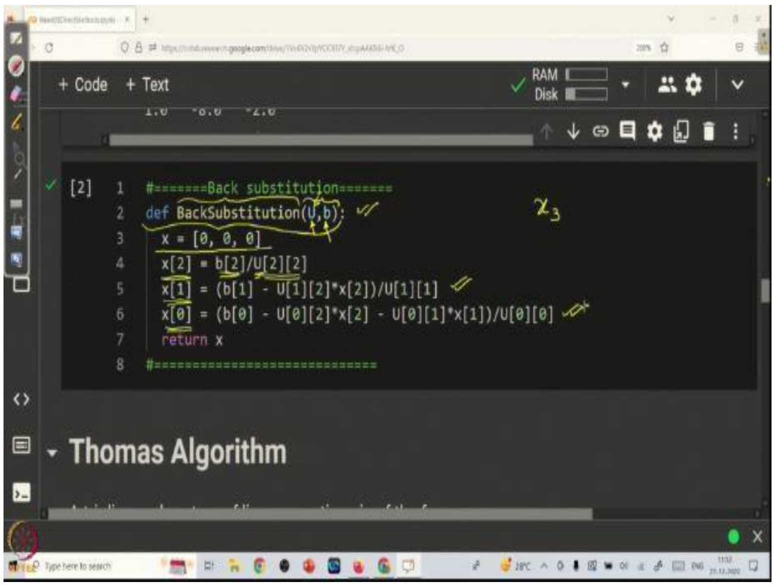


```
    else:
      print('Naive Gaussian elimination method failed because A1[0][0] = 0')
      Indicator=0
#### Back substitution part
if Indicator:
  if A2[2][2]!=0:
    x = BackSubstitution(A2,b2)
  else:
    print('Naive Gaussian elimination method failed because A2[0][0] = 0')
    Indicator=0
print('     ',A[0][0],' ',A[0][1],' ',A[0][2])
print('A = ',A[1][0],' ',A[1][1],' ',A[1][2])
print('     ',A[2][0],' ',A[2][1],' ',A[2][2])
print('b = ',b[0],b[1],b[2])
print('Solution of the given system = ',x)
```

Well if you recall again the back substitution should come only when you have successfully computed both the steps**.** Therefore, you will again check whether the indicator is still holding the value 1, because if any of the steps failed then your control would output indicator as 0 and therefore it would have switched off the process. And that is taken care here if the indicator is 0 it will never get into this if condition.

If indicator is 1 then only it will get into this or any non-zero for that matter, in our case it is 1. Once it comes in then you have to again check whether $a_{33}^{(3)}$ is non-zero that is what we have to check**.** That is what I am doing in the code I am checking whether A2[2][2] is not equal to 0. If it is not equal to 0, then I will do the back substitution process. Here, instead of typing the back substitution code what I did is I have separately defined it as a function. In python, if you know that you are going to use a set of lines repeatedly in many codes, then it is better to put it as a separate function and call that function into your program wherever it is needed.

(Refer Slide Time: 27:31)



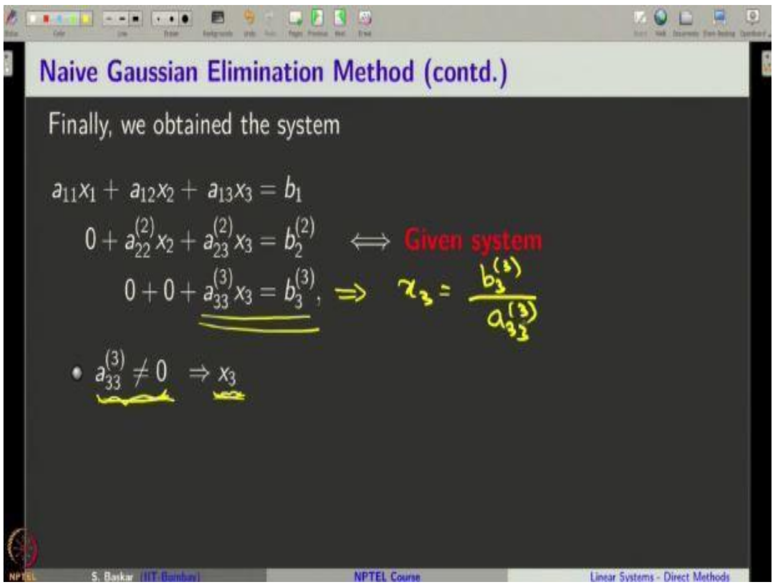
```
#=======Back substitution=======
def BackSubstitution(U,b):
    x = [0, 0, 0]
    x[2] = b[2]/U[2][2]
    x[1] = (b[1] - U[1][2]*x[2])/U[1][1]
    x[0] = (b[0] - U[0][2]*x[2] - U[0][1]*x[1])/U[0][0]
    return x
#============================
```

x_3

Thomas Algorithm

What I did here is? I have made this back substitution as a separate code. I have written it in the form of a function. How to do that? Well, that can be done with this command define a function, the function name is back substitution and what are all the parameters that I have to supply into this function. These are the upper triangular matrix and the right hand side vector. So, I am passing these two information into this function.

(Refer Slide Time: 28:27)



Naive Gaussian Elimination Method (contd.)

Finally, we obtained the system

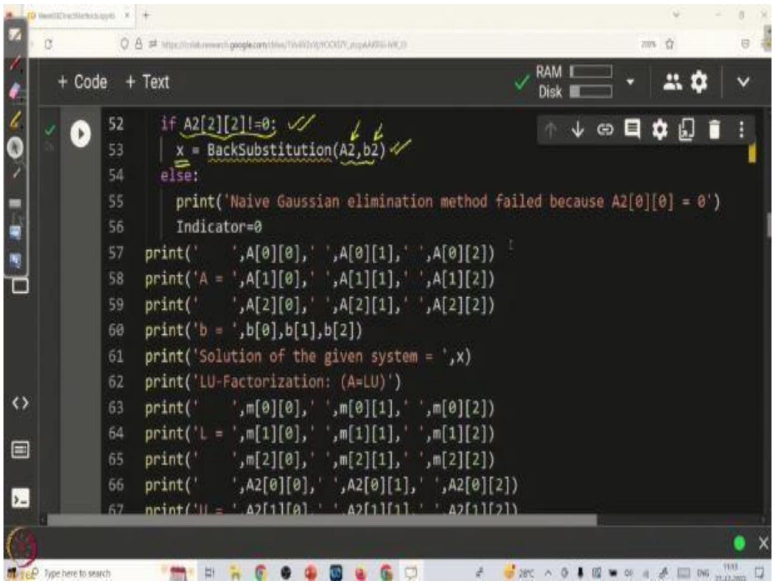
$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$
$$0 + a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 = b_2^{(2)} \iff \text{Given system}$$
$$0 + 0 + a_{33}^{(3)}x_3 = b_3^{(3)}, \Rightarrow x_3 = \frac{b_3^{(3)}}{a_{33}^{(3)}}$$

- $a_{33}^{(3)} \neq 0 \Rightarrow x_3$

And what that function does is it initializes a vector x as [0, 0, 0] and then x_3 is what is denoted as x[2] here. And what is x_3 theoretically if you see x_3 is obtained as $\frac{b_3}{a_{33}}$. That is what we are doing here, x_3 is $\frac{b_3}{u_{33}}$. Now, I have captured this matrix as U, therefore I have to use the notation U here. Similarly, x_2 you can compare the theoretical expression and see how it comes and x_1

theoretically here it is x[0] and its expression is given like this. Once it computes x_1, x_2 and x_3 it returns the value x.

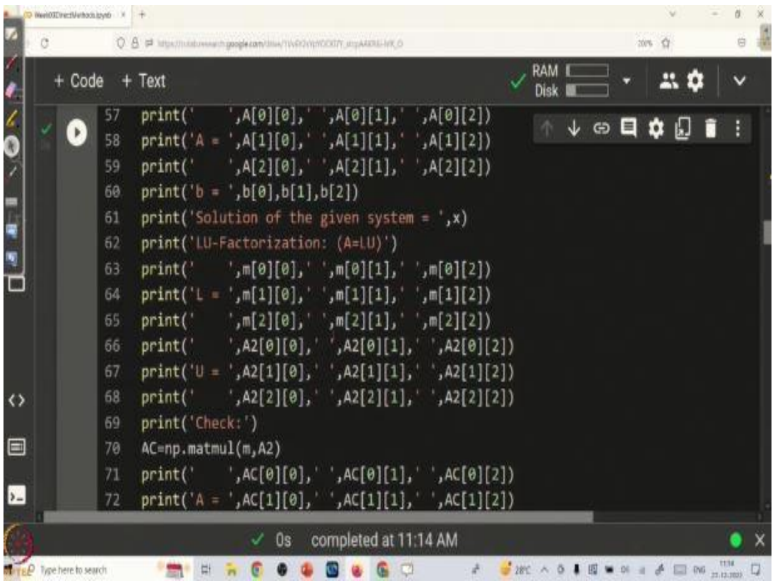
(Refer Slide Time: 29:11)



```
    if A2[2][2]!=0:
      x = BackSubstitution(A2,b2)
    else:
      print('Naive Gaussian elimination method failed because A2[0][0] = 0')
      Indicator=0
print('    ',A[0][0],' ',A[0][1],' ',A[0][2])
print('A = ',A[1][0],' ',A[1][1],' ',A[1][2])
print('    ',A[2][0],' ',A[2][1],' ',A[2][2])
print('b = ',b[0],b[1],b[2])
print('Solution of the given system = ',x)
print('LU-Factorization: (A=LU)')
print('    ',m[0][0],' ',m[0][1],' ',m[0][2])
print('L = ',m[1][0],' ',m[1][1],' ',m[1][2])
print('    ',m[2][0],' ',m[2][1],' ',m[2][2])
print('    ',A2[0][0],' ',A2[0][1],' ',A2[0][2])
```

And now we have to get back that value into our original program. Remember we have computed A2 and b2 from step 2. That I am pausing it into the function back substitution and got the value x back from the back substitution and that is going to be my solution. Again, at this level if the diagonal element of the third equation is 0 then I should not do the back substitution process. That is taken care of by this if condition.

(Refer Slide Time: 29:59)



```
print('    ',A[0][0],' ',A[0][1],' ',A[0][2])
print('A = ',A[1][0],' ',A[1][1],' ',A[1][2])
print('    ',A[2][0],' ',A[2][1],' ',A[2][2])
print('b = ',b[0],b[1],b[2])
print('Solution of the given system = ',x)
print('LU-Factorization: (A=LU)')
print('    ',m[0][0],' ',m[0][1],' ',m[0][2])
print('L = ',m[1][0],' ',m[1][1],' ',m[1][2])
print('    ',m[2][0],' ',m[2][1],' ',m[2][2])
print('    ',A2[0][0],' ',A2[0][1],' ',A2[0][2])
print('U = ',A2[1][0],' ',A2[1][1],' ',A2[1][2])
print('    ',A2[2][0],' ',A2[2][1],' ',A2[2][2])
print('Check:')
AC=np.matmul(m,A2)
print('    ',AC[0][0],' ',AC[0][1],' ',AC[0][2])
print('A = ',AC[1][0],' ',AC[1][1],' ',AC[1][2])
```

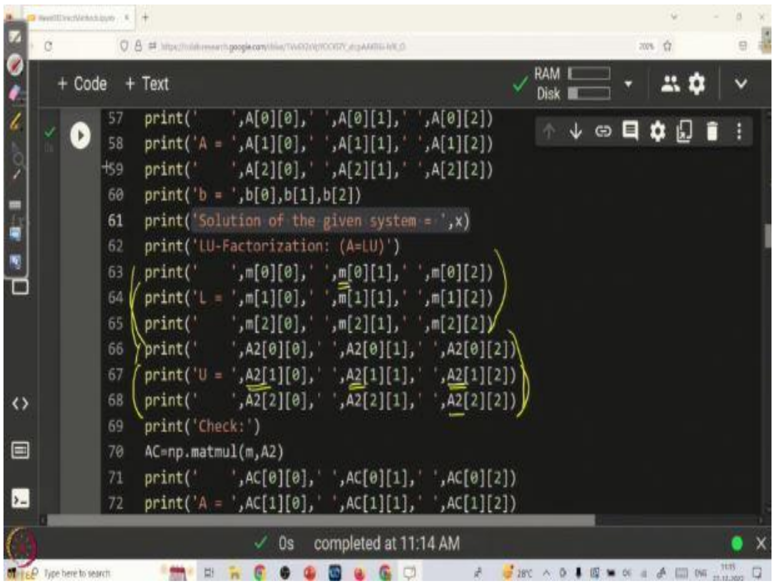
✓ 0s completed at 11:14 AM

Then the rest all are the print commands that is I am printing the output in all these commands. Let us try to see how this program runs for that I have to click this play button, it will take some time it has ah initialized some memory for us in the cloud machine and it has run the code.

(Video Starts: 30:19) You can see that the code has run successfully. This is the A matrix that I have given and this is the b matrix I have given.

And what is the solution? The solution is given like this. So, I am just printing the solution somewhere here, see I am I am printing the solution here and that is printed here the solution is given by this. **(Video Ends: 30:44)**

(Refer Slide Time: 30:45)

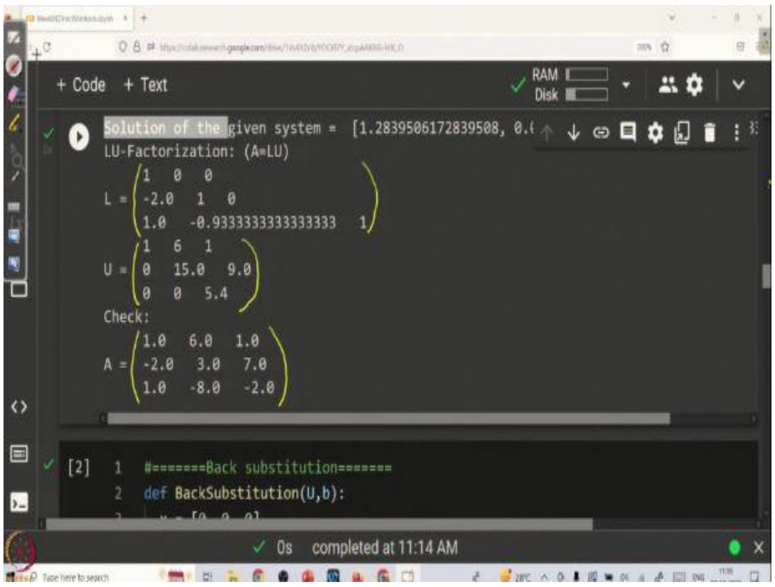


```
print('    ',A[0][0],' ',A[0][1],' ',A[0][2])
print('A = ',A[1][0],' ',A[1][1],' ',A[1][2])
print('    ',A[2][0],' ',A[2][1],' ',A[2][2])
print('b = ',b[0],b[1],b[2])
print('Solution of the given system = ',x)
print('LU-Factorization: (A=LU)')
print('    ',m[0][0],' ',m[0][1],' ',m[0][2])
print('L = ',m[1][0],' ',m[1][1],' ',m[1][2])
print('    ',m[2][0],' ',m[2][1],' ',m[2][2])
print('    ',A2[0][0],' ',A2[0][1],' ',A2[0][2])
print('U = ',A2[1][0],' ',A2[1][1],' ',A2[1][2])
print('    ',A2[2][0],' ',A2[2][1],' ',A2[2][2])
print('Check:')
AC=np.matmul(m,A2)
print('    ',AC[0][0],' ',AC[0][1],' ',AC[0][2])
print('A = ',AC[1][0],' ',AC[1][1],' ',AC[1][2])
```

Os completed at 11:14 AM

Now, I am also saving the matrix U, what is my matrix U? Here I am printing the matrix U, the matrix U is nothing but A2 in my program**.** So, I am printing it here, similarly the matrix L is printed here and that is nothing but m in my program notation. So, that is what you can see in the output.

(Refer Slide Time: 31:12)



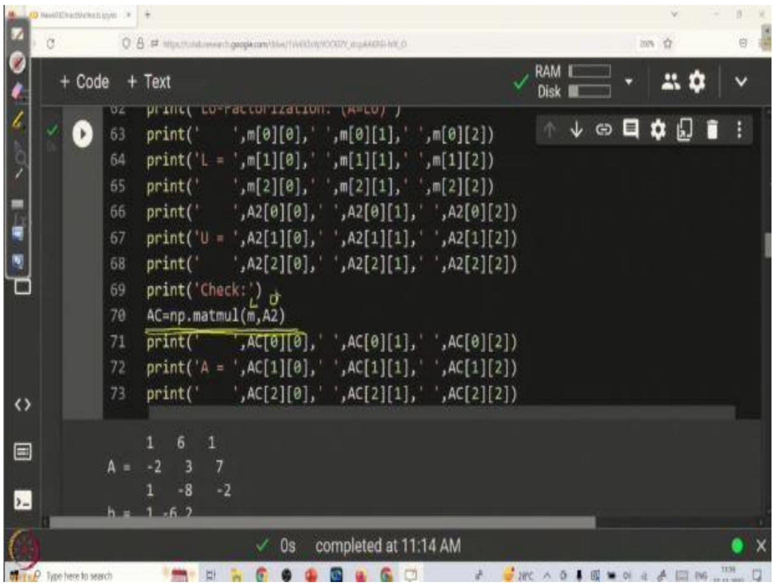
```
Solution of the given system =  [1.2839506172839508, 0.(
LU-Factorization: (A=LU)
     1    0    0
L =  -2.0   1    0
     1.0   -0.9333333333333333   1
     1    6    1
U =  0    15.0   9.0
     0    0    5.4
Check:
     1.0   6.0   1.0
A =  -2.0   3.0   7.0
     1.0   -8.0   -2.0
```

```
[2]
#=======Back substitution=======
def BackSubstitution(U,b):
```

Os completed at 11:14 AM

So, L is given like this and U is given like this. I am also checking whether L into U is giving back my A, you can also check that L into U is precisely the A that we have taken.

(Refer Slide Time: 31:37)



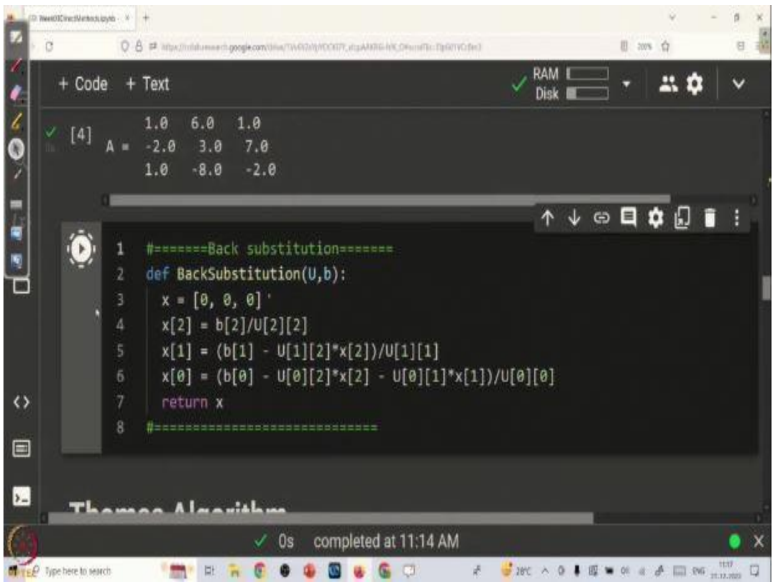
```
print('    ',m[0][0],' ',m[0][1],' ',m[0][2])
print('L = ',m[1][0],' ',m[1][1],' ',m[1][2])
print('    ',m[2][0],' ',m[2][1],' ',m[2][2])
print('    ',A2[0][0],' ',A2[0][1],' ',A2[0][2])
print('U = ',A2[1][0],' ',A2[1][1],' ',A2[1][2])
print('    ',A2[2][0],' ',A2[2][1],' ',A2[2][2])
print('Check:')
AC=np.matmul(m,A2)
print('    ',AC[0][0],' ',AC[0][1],' ',AC[0][2])
print('A = ',AC[1][0],' ',AC[1][1],' ',AC[1][2])
print('    ',AC[2][0],' ',AC[2][1],' ',AC[2][2])
```

```
     1   6   1
A = -2   3   7
     1  -8  -2
```

✓ 0s completed at 11:14 AM

You can see that I am just doing this at this level. This is nothing but the matrix multiplication of L into U**.** So, that exactly as what we want is given here and that completes the code for the Gaussian elimination method. I hope you have understood.

(Refer Slide Time: 31:48)



```
[4]      1.0   6.0   1.0
    A = -2.0   3.0   7.0
         1.0  -8.0  -2.0
```

```
#=======Back substitution=======
def BackSubstitution(U,b):
  x = [0, 0, 0]
  x[2] = b[2]/U[2][2]
  x[1] = (b[1] - U[1][2]*x[2])/U[1][1]
  x[0] = (b[0] - U[0][2]*x[2] - U[0][1]*x[1])/U[0][0]
  return x
#============================
```

✓ 0s completed at 11:14 AM

One thing is when you run the program, before running the main Gaussian elimination program you have to once run this function back substitution, otherwise the main program will give you an error message. With this we will end this class. Thank you for your attention.