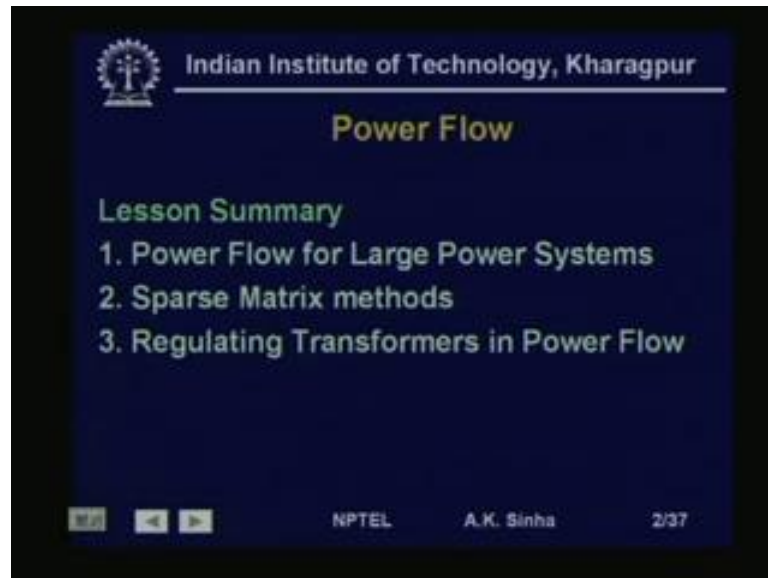


Power System Analysis
Prof. A. K. Sinha
Department of Electrical Engineering
Indian Institute of Technology, Kharagpur

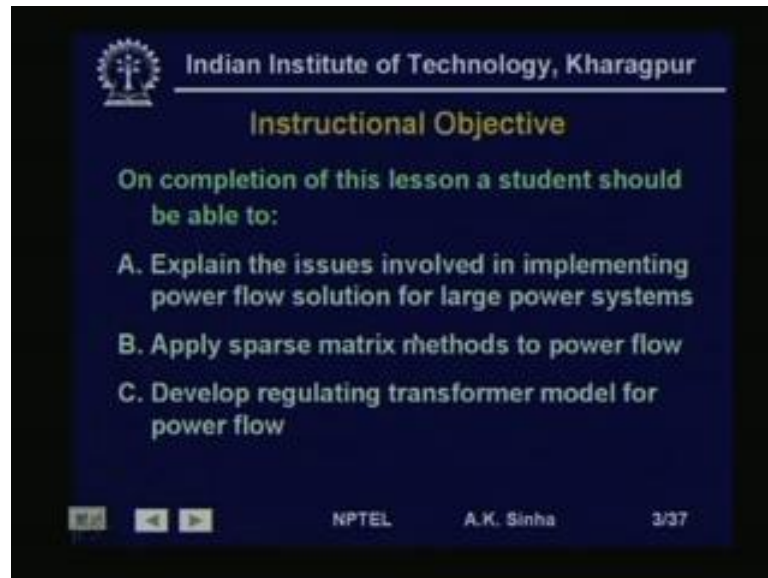
Lecture - 21
Power Flow VI

(Refer Slide Time: 00:57)



Welcome to lesson 21. In this lesson we will continue with the Power Flow problem. In this lesson we will first take up power flow problem for large power systems. How do we implement it? What are the implementations aspects. One of the aspects which we will look at will be sparse matrix methods and its application to power flow problem. And then we will also look into the modeling of regulating transformers and power flow problem.

(Refer Slide Time: 01:33)



Indian Institute of Technology, Kharagpur

Instructional Objective

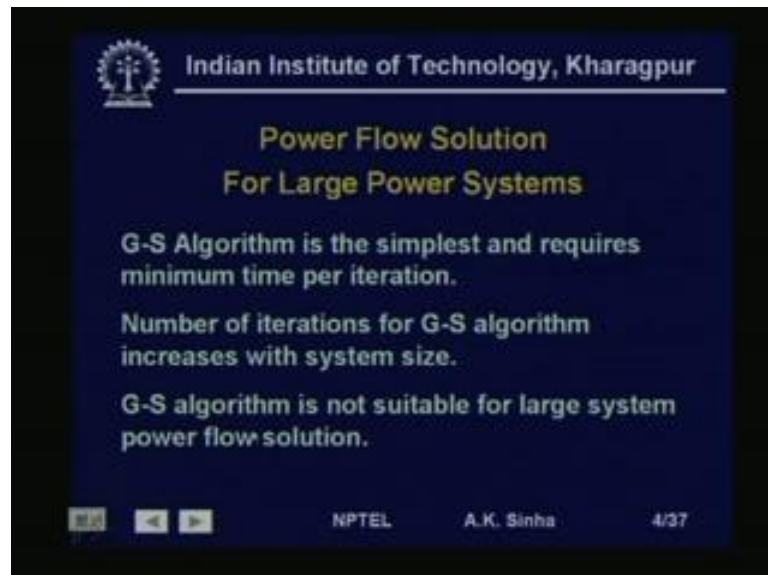
On completion of this lesson a student should be able to:

- A. Explain the issues involved in implementing power flow solution for large power systems
- B. Apply sparse matrix methods to power flow
- C. Develop regulating transformer model for power flow

NPTEL A.K. Sinha 3/37

On the completion of this lesson you should be able to explain the issues involved in implementing power flow solution for large powder systems. Apply sparse matrix methods to power flow and develop regulating transformer model for power flow solution.

(Refer Slide Time: 01:59)



Indian Institute of Technology, Kharagpur

Power Flow Solution For Large Power Systems

G-S Algorithm is the simplest and requires minimum time per iteration.

Number of iterations for G-S algorithm increases with system size.

G-S algorithm is not suitable for large system power flow solution.

NPTEL A.K. Sinha 4/37

Now, what are the types of power flow methods that we have learnt. Earlier we have discussed about the Gauss Seidel method for solution of power flow problem, then we looked into the Newton-Raphson method. Then we tried to take advantage of the

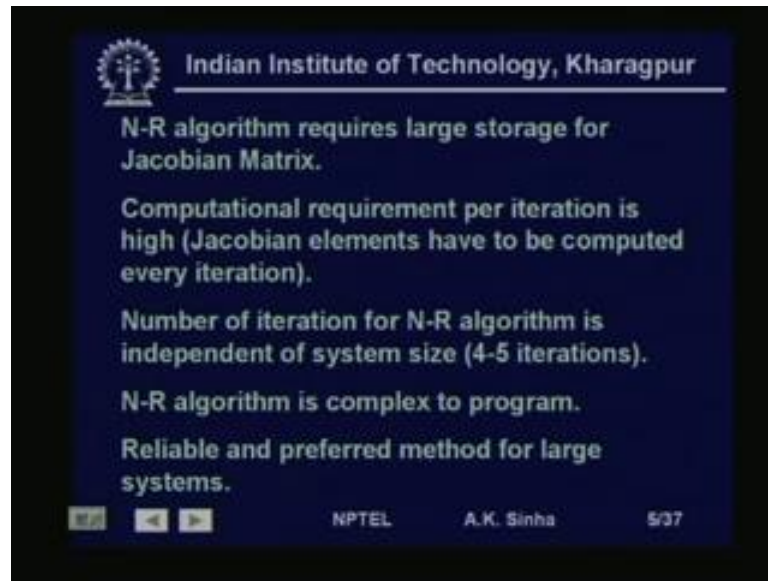
physical property of the power system, where the real power and voltage phase angle close association. And volt reactive power and the voltage magnitude close association was taken into account.

And the power flow equations were decoupled into p δ and q v two sub problems and then we saw that by taking certain physical characteristics of the power system. We can modify the problem into what we call the Fast-decoupled load flow problem, where the Jacobian matrices used become constant. That is we have the b dash and b double dash of matrices which are basically similar to the susceptance elements of the Y bus matrix. And therefore, they remain constant.

So, we would like to see out of these four different methods, which method would be suitable for large power system problems. We had seen that the Gauss Seidel algorithm is the simplest algorithm and requires minimum time for iterations because the computations involve is much less. Though we have to do this computation in the complex frame that is all quantities power voltage and admittances are used as complex quantities.

However, one problem encountered with the Gauss Seidel method is that the number of iterations for Gauss Seidel algorithm increases with the system size. That is a few are working for a smaller system. The number of iterations required for convergence is going to be small. But, if you are working with large system then this number of iterations for constraint accuracy of convergence is going to be very large. And this is one of the main reasons why Gauss Seidel method is not suitable for large power system, power flow applications.

(Refer Slide Time: 05:48)



In fact, Newton-Raphson method was known earlier. But, this method could not be used for power flow problem, mainly because for large system it required large Jacobian matrix solutions. That is solution of large system of equations which was not possible earlier.

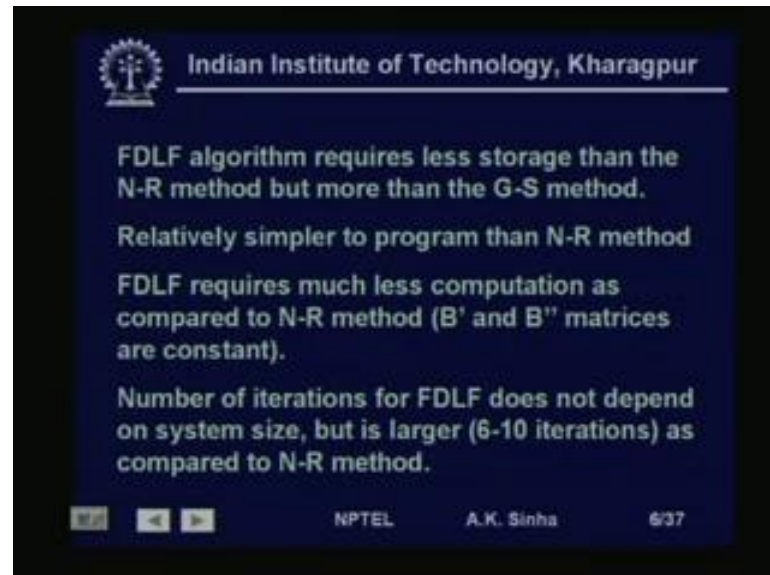
That is Newton-Raphson algorithm requires large storage for Jacobian matrix. Also the computational requirement per iteration is high, because Jacobian elements have to be computed at every iteration. Though this is somewhat complex, but an a major advantage of a Newton-Raphson method is that the number of iterations required for convergence is independent of the system size. That is normally that will take around three to five iterations for convergence even for very large systems.

N-R algorithm is somewhat complex to program. However, it uses the real and reactive power equations separately that deals with real numbers it does not deal with complex numbers. That is all quantities used in Newton-Raphson method for power flow are real numbers we do not work with a complex number in case of Newton-Raphson power flow.

N-R method is a very reliable and accurate method. And therefore, is the preferred method for large system power flow applications. That is most of the commercial programs for power flow use Newton-Raphson algorithm for the solution. And people

have solved problems of more than 10 to 20,000 buses using Newton-Raphson method of power flow.

(Refer Slide Time: 08:30)



The other method which we talked about was decoupled method, but decoupled method has some convergence problems. And therefore, this is not very much in use, but the variation of the decoupled method, which we call the fast decoupled method is very much in use, because it requires much less storage than the Newton-Raphson method.

Actually, since we have decoupled the real power and the reactive power equations. That is real power p δ equations and q v equations. Therefore, we have two sub system equations. One dealing with real power and voltage angle, another dealing with reactive power and voltage magnitude. And the size of the matrices that is B' and B'' are much smaller. That is they are almost half this size of that of the Newton-Raphson method.

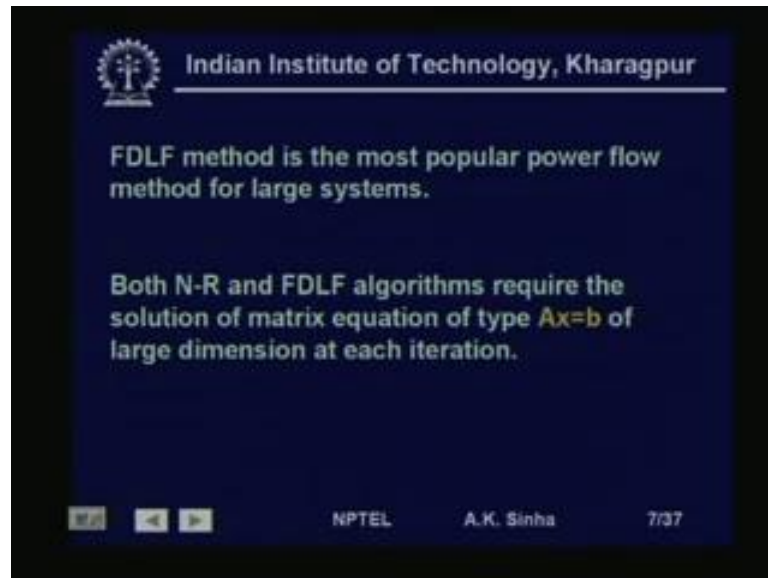
That is if Newton-Raphson method for N bus system requires $2n \times 2n$ size in Jacobian matrix B' and B'' matrix are going to be smaller than $n \times n$ matrix. The algorithm is simpler to program than the Newton-Raphson method again this is mainly because B'' are basically the susceptance is of the Y bus matrix, the susceptance elements of the Y bus matrix, and therefore much simpler to compute.

Fast-decoupled load flow method requires much less computation as compared to Newton-Raphson method because B' and B'' matrices are constant. And therefore, they need to be factored only once, whereas in case of Newton-Raphson method you have to compute the Jacobian matrix in each iteration. And then factorize it, in every iteration for finding out the updates for voltage magnitude and phase angle. Whereas in case of Fast-decoupled we do not have such a situation since B' and B'' matrices are constant, they can be factored and stored once for all the iterations. Number of iterations for Fast-decoupled load flow also is not very much dependent on the system size. But, is somewhat larger that is about 6 to 10 iterations are required for solution of large power systems.

So, though the number of iterations required here is somewhat larger the computations involved for iteration is much less. Therefore, the time taken by Fast-decoupled load flow method is considerably less than that for the Newton-Raphson method. Similarly, though the computation for Gauss Seidel method is much simpler. And computation for iteration required is much less than that of Newton-Raphson. But, because the number of iterations required are very large for convergence.

Finally, the time or the computation time for the Newton-Raphson method for large systems is smaller than that of Gauss Seidel method. And this is the reason why Newton-Raphson method and Fast-decoupled method are very popular for power flow solution of large power systems.

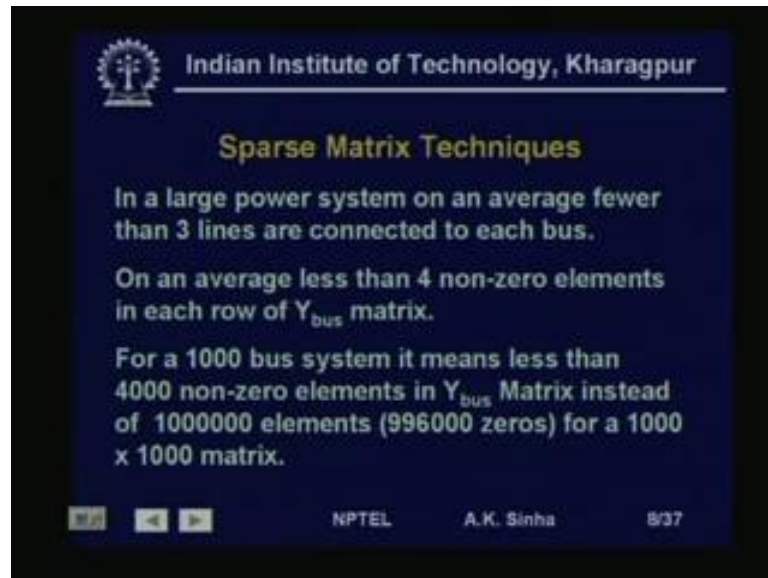
(Refer Slide Time: 12:58)



Now, both Newton-Raphson and Fast-decouple load flow algorithms require the solution of matrix equations of the type $Ax = b$ of large dimension at each iteration. This is the major problem that both these methods face that they need to solve large set of matrix equations of the type $Ax = b$. Where A is a Jacobian matrix or B dash matrix B double dash matrix in case of Fast-decoupled load flow. X is the vector of unknowns that is voltage magnitude and voltage phase angles and B is the mismatch that is Δp and Δq .

So, we need to solve this kind of an equation $Ax = b$ at each iteration. And for large size systems the size of A matrix is going to be very large. And this has been the major problem in implementing these methods that in Newton-Raphson and Fast-decoupled method for large power systems.

(Refer Slide Time: 14:31)



Indian Institute of Technology, Kharagpur

Sparse Matrix Techniques

In a large power system on an average fewer than 3 lines are connected to each bus.

On an average less than 4 non-zero elements in each row of Y_{bus} matrix.

For a 1000 bus system it means less than 4000 non-zero elements in Y_{bus} Matrix instead of 1000000 elements (996000 zeros) for a 1000 x 1000 matrix.

NPTEL A.K. Sinha 8/37

But one advantage that we take from the physical structure of the system is that. In a large power system on an average fewer than 3 lines are connected to each bus. That is each bus will be connected to maybe 2 or 3 other buses. In the system which simply means that on an average less than four non zero elements are there in each row of Y bus matrix. That is Y bus matrix contains only 3 to 4 non zero elements in each row.

That is if we take a 1000 bus system then we have only about less than 4000 non zero elements in the Y bus matrix. Instead of 1000 into 1000 elements in a 1000 bus system Y bus. That is instead of 1 million elements we have only 4000 elements that is the Y bus has almost 996000 0's present in it and only 4000 non zeros for a 1000 by 1000 Y bus matrix. Such kind of matrices which have very large number of zeros in them are called sparse matrix. And knowing this property of the Y bus of the system we try to take advantage this sparcity of the Y bus matrix in solution of the power flow problem. Let us see how we do this.

(Refer Slide Time: 16:42)

Indian Institute of Technology, Kharagpur

Matrix with very few non-zero (large number of zeros) elements are called **Sparse Matrix**.

Instead of storing them as 2-dimensional arrays they are stored as a linked list of elements to save computer storage requirement.

Similar is the case for Jacobian matrix, wherever $Y_{ij}=0, H_{ij}=M_{ij}=N_{ij}=L_{ij}=0$.

Compact storage as a linked list for a 1000 bus system with 100 PV buses (considering average of 3 lines connected to each bus) will require

NPTEL A.K. Sinha 9/37

So, what we have the matrix Y bus matrix with very few non zero elements is present and it is a sparse matrix. Now, if we store this matrix as a two dimensional array that is having elements A_{ij} . In the matrix of 1000 rows and 1000 column for a 1000 bus system, we store these non zero elements only the zero elements are not stored, but if we need to store only non zero elements we need to know which elements of the matrix are non zeros.

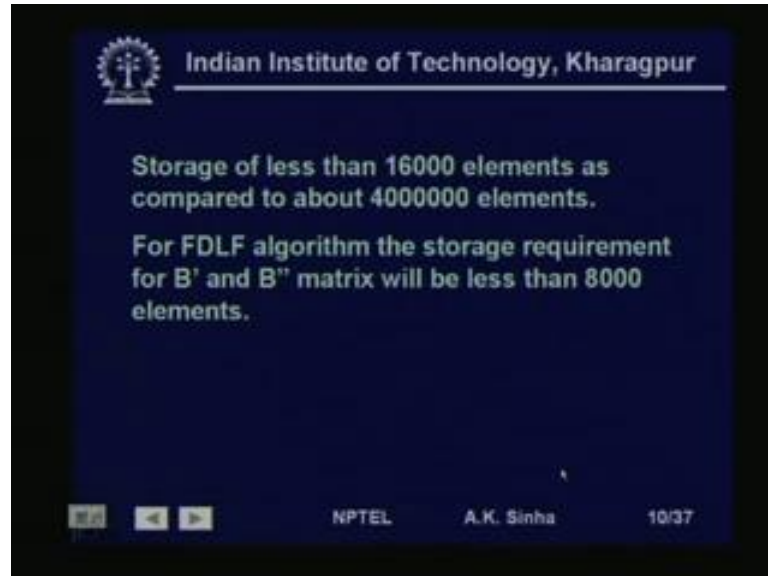
That is we need to identify these non zero elements with their row and column numbers. And therefore, we need to prepare a kind of a linked list. That is instead of storing the Y bus elements as a two dimensional array, they are stored as a linked list of elements to save computer storage requirement.

So, this way if we store them as a linked list then we will save large amount of computer storage. Similar is the case for the Jacobian matrix, because if we recall the elements of the Jacobian matrix ((Refer Time: 18:25)) they contain Y_{ij} element. And if Y_{ij} is zero for any i and j then H_{ij} sub matrix M_{ij} N_{ij} and L_{ij} sub matrix will also those elements. H_{ij} element M_{ij} element N_{ij} element and L_{ij} element will also be zero for this is matrices.

Therefore compact storage using linked list can be used for storing the Jacobian matrix elements as well. Now, let us see compact storage as a linked list for a 1000 bus system

with 100 PV buses. What will be the number of elements in a Jacobian matrix for such a case. Now, considering average of three lines connected to each bus.

(Refer Slide Time: 19:33)



So, this will require storage of less than 16000 elements as compared to about 4 million elements in the Jacobian matrix for a 1000 buses, because here each of these matrixes H, M, N and L will be having only about 4000 or less elements. Therefore, the total number of elements non zero elements and this system will be only of the order of 16000 as compared to 4 million.

For Fast-decouple algorithm the storage requirement is even less, because we need only B dash and B double dash matrixes and the size both these matrixes are going to be 1000 by 1000 or less for the PV bus B double dash matrices will not have any equations. Therefore, they will be having a size of only 900 by 900 for a 100 PV bus systems. Therefore the elements there are also going to be less compared to B dash elements. And the total size or total number of elements in B dash and B double dash will be less than 8000 elements only.

(Refer Slide Time: 21:19)

Indian Institute of Technology, Kharagpur

Linked List Storage

Let us take a 4 x 4 matrix A as

$$A = \begin{bmatrix} 1.0 & 5.0 & 6.0 & 7.0 \\ 8.0 & 2.0 & 0 & 9.0 \\ 10.0 & 0 & 3.0 & 0 \\ 11.0 & 0 & 0 & 4.0 \end{bmatrix}$$

NPTEL A.K. Sinha 11/37

So, how do we store these elements as a linked list, that for example take a very small 4 by 4 matrix A. So, A is given here as shown the diagonal elements have 1 2 3 and 4 of diagonal elements are 5 6 7 that is 8 took A 1 2 is 5, A 1 3 is 6, A 1 4 is 7 and A 2 1 is 8, A 2 3 is 0, A 2 4 is 9, A 3 1 is 10, A 3 2 is A 0, A 3 is 3, A 3 4 is 0. And similarly A 4 1 is 11, A 4 2 is 0, A 4 3 is 0 and A 4 4 is 4. Now, how do we store this matrix as a linked list.

(Refer Slide Time: 22:26)

Indian Institute of Technology, Kharagpur

CE = [1.0 5.0 6.0 7.0 8.0 2.0 9.0 10.0 3.0 11.0 4.0]
ICL = [1 2 3 4 1 2 3 1 3 1 4]
NXT = [2 3 4 0 6 7 0 9 0 11 0]
IRW = [1 5 8 10]

NPTEL A.K. Sinha 12/37

Let us see what we do. We create four arrays, one we call as C E there are of the compact storage elements. This array is single dimension array and contains all the non zero elements of the matrix as stored in the serial order as they are in the rows. So, we have 1 5 6 7 then 8 2 9. So, this is 1 5 6 7 8 2 9. And after this we have 10 3 then 11 4 then we have 10 3 11 4.

Now, this array ICL is an array of integer numbers. It identifies the column number for these elements. And the size of this array is same as the number of non zero elements in the matrix. So, ICL corresponding to the first element is 1 indicating this is in column number 1 the element number 2 which is five is in column number 2. So, ICL is two

Similarly, the third elements is 6 and this is in column number 3, the fourth element is 7 this is in column number 4, the fifth element is 8 and this is in column number 1. So, we have fifth element which is eight in column number 1. Then sixth element is 2 which is in column number 2, seventh element is 9 which is in column number 3 this should be 4 in column number 4. Then eighth element is 10, which is in column number 1, eighth element is 10 column number 1. Ninth element is 3 which is in column number 3. Tenth element is eleven this is in column number 1 tenth element is 11. And this is in column number 1 where as eleventh element is 4 and this is in column number 4. So, eleventh element is 4 and this is in column number 4.

We have another array again and an array of integer numbers, which is in of the size same as the number of non zero elements. And this is a very interesting array. What we do in this array is indicate the column number or the serial number of the next element in the same row. That is if you see CE is 1 and the next element is in position 2 and for this same row. So, if we see here we have 1 and then we have 5 as a next element, then we have 6 as the next element and then we have seven as the next element after this the row ends.

So, if we see here what we have, for the first element we have 1 and the next element is the serial number 2. Element is found in serial number 2 and again for this we have indicated 3 here because the next element after this is found in serial number 3 position. And for this element the next element is found in position number 4. So, position number 4 has 7 which is again an element in the same row.

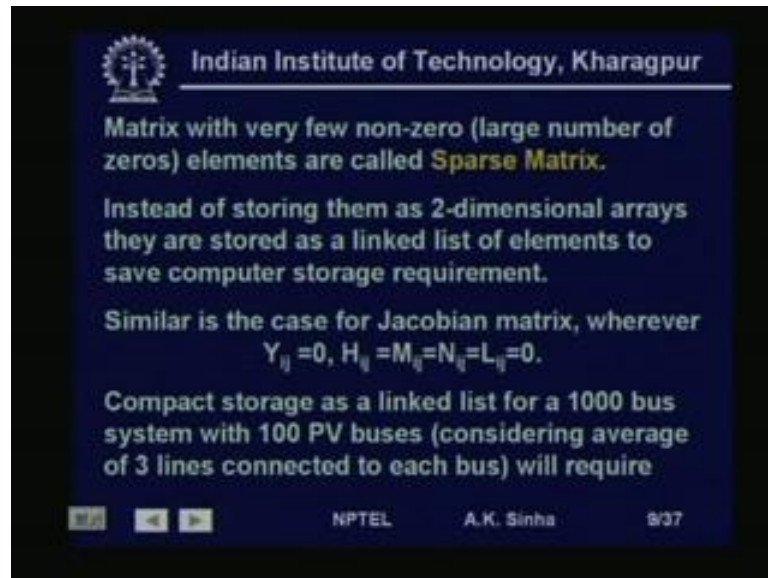
Now, after this seven there is no further element in the same row. So, the next element is no longer there for the same row and this is indicated by a null pointer that is by zero. So, this zero indicates that there are no more elements in this row. So, this row ends. Now, again if you look at second row starting we have 8, in the column number 1 and what is the next element to this, the next element to this is in position number 6.

So, we are putting 6 here and the next element in position number 6 is 2. That is 2, that is 2 2 for the same row here. And again for this, what is the next element the next element is in the same row is 9 and this is at position number 7. So, we have for this next number as 9. So, seventh position number 7, we have this element again the row ends. So, there is no next element and this is indicated by a 0. And then again we take the start with the third row and we have the next element at position number 9 for this and then after this 3 3 we do not have any element 3 4.

So, the next here is indicated by as 0. And so the next rows starts this is 4 1 element and the next element here is in position 11 as indicated by the NXT for this. And here this is the eleventh element and again after this we do not have any element. So, that is indicated by null pointer zero.

So, this now one question comes why do we use this array next. We will see later when we talk about the elimination process that while doing this elimination, we will get some feeling that is some of the elements, which were zero. In the original matrix will now become non zero, and since we cannot store them again at their requisite position in the matrix that is if we see here.

(Refer Slide Time: 30:03)



If we see here, if this element becomes one zero then what happens is we need to store that. Now, here we do not have any space for storing this. So, this has to be stored only at the end. So, when we store this at the end, what will happen is that we will have to indicate that number. That is this element is coming after this two. So, below this two we will have to indicate this next instead of 7 as 12. And this element can be stored here and then below that for the next will be 7, which is stored here.

So, this next is able to make us store non zero elements, which come as fillings during the factorization process. We have another row here are row vector which stores this, this is an array of integer numbers again. And this array is having a size which is equal to the size of the matrix that is it is a 4 by 4 matrixes. So, there are going to be four elements in this and what does this say. This array tells us the row number 1 starts at position number 1 row number 2 starts at as position number 5. Row number 3 starts at position number 8 and row number 4 starts at position number 10.

So, if we want to work on any particular row. We can directly using this I row we can directly go to that starting point of that row. And work with that row instead of going through this complete sequence of all the rows. So, this again is very useful for the factorization process. So, this is how we store the elements.

Now, if you look at this 4 by 4 matrix you will say that we have not saved any memory, because what we have done is instead of 16 elements that we have in that matrix. We are

storing eleven real numbers and 22 plus 4 26 integral numbers, which itself will be a large storage, which is more than the storage required for 4 by 4 matrix. But, if you look at a 1000 by 1000 matrix. Then instead of storing 1 million elements you will be storing say 4000 here and 4000 for these two as integers required half the number of bytes as the real numbers.

So, 4000 plus 4000 is 8000 plus 1000 that is again half the bytes for the real numbers. So, around 500. So, 8500 real number bytes are required only. So, instead of 1 million into 8 bytes, if we store the real numbers as double precision numbers. Then we require 8 byte for each real number. So, 8 into 1 million that is 8 million for 8 mega bytes will be required for a 1000 by 1000 matrix. Whereas, for this sparse matrix you will require 8 into 4000 that is 32000 plus 4 into 1000, this will be 4000. So, 32000 it will be again 32000, 64000 and this will be 4000. So, 68 kilo bytes only as compared to how much 8 into 8 million, 8 mega bytes.

So, this is the advantage that we get in case of large matrixes. Now, let us see if we do the factorization for this kind of a matrix what happens. Now, if we take a matrix like this, that we have used earlier, the same matrix and if you tried to use the Gauss elimination method for factorizing this matrix.

(Refer Slide Time: 36:03)

Indian Institute of Technology, Kharagpur

After 1st step of Gauss elimination matrix becomes:

$$A = \begin{bmatrix} 1 & 5 & 6 & 7 \\ 0 & -38 & -48 & -47 \\ 0 & -50 & -57 & -70 \\ 0 & -55 & -66 & -73 \end{bmatrix}$$

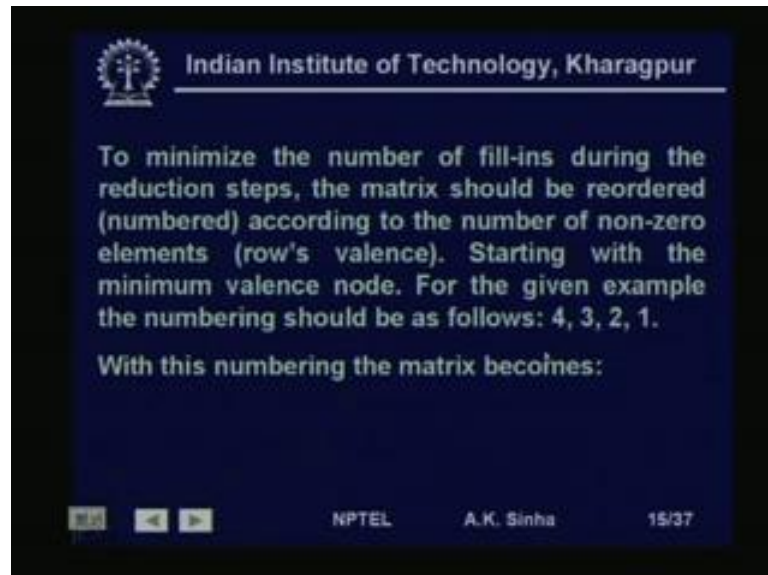
Sparsity of A is lost completely in this case

NPTEL A.K. Sinha 14/37

Then after the first step of Gauss elimination that is reducing all the elements in the first column as zeros, what we will have is, all these elements which were zero are now

getting filled that is instead of the sparse matrix. Now, we have a completely full matrix; that means, sparsity is lost, because we get large number of fill-ins in this case.

(Refer Slide Time: 36:38)



Now, let us see what we can do about it. So, if we trying to take a advantage of sparse matrix and we want to use this sparse matrix method. If we go through the factorization just like that for the given matrix. Then we find that the sparsity is no longer maintained because large number of fill-ins will come.

So, what do we do to minimize these fill-ins. So, to minimize these fill-ins during the reduction steps. The matrix is to be reordered or renumbered according to the number of non zero elements, which in each row we call the rows valence. So, if we take any row and see how many non zero elements are there in that row. And we call that as the rows valence. So, if we pick up the rows, which have minimum number of non zero elements or the rows with minimum valence are ordered as row number 1 and so on. We go till we reach the last row which will have the largest number of non zero elements. So, if we try to use that starting with the minimum valence node for the given example the numbering should be.

(Refer Slide Time: 38:19)

Indian Institute of Technology, Kharagpur

Ordering Schemes

Let us take the same matrix A

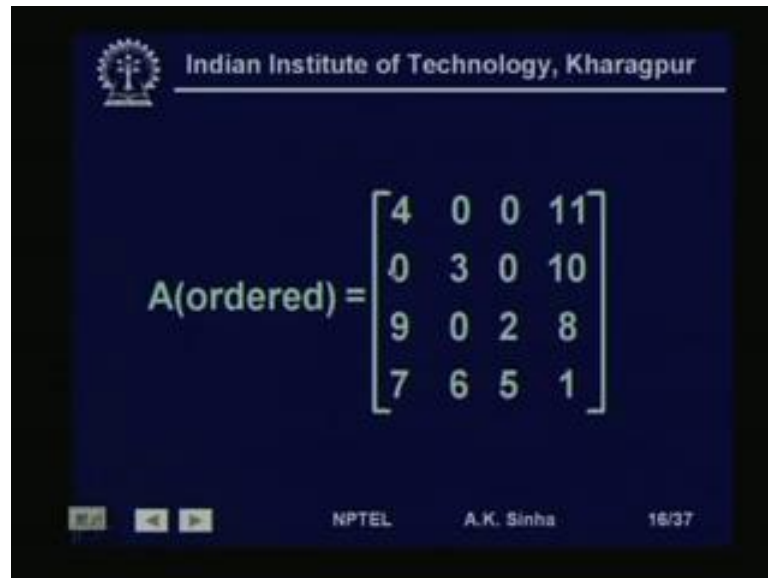
$$A = \begin{bmatrix} 1.0 & 5.0 & 6.0 & 7.0 \\ 8.0 & 2.0 & 0 & 9.0 \\ 10.0 & 0 & 3.0 & 0 \\ 11.0 & 0 & 0 & 4.0 \end{bmatrix}$$

NPTEL A.K. Sinha 13/37

Now, if you look at this matrix we find that we have only two elements in row 4, 2 elements in row 3. We can choose any one of them because here this row has 3 and this row has 4. So, if you look at the ranking this row has minimum these two rows have minimum number of non zero elements we can choose any one of them. Let us choose this row as row 1.

So, now we have ranked row 4 as 1 and row 3 as 2 that is the fourth row and column and now ranked as 1 1. So, this is 1 and this is 1. Similarly, this is 2 and this is 2 this is 2 and this row 3 is having three non zero elements and. So, this is numbered as 3. So, this is 3 and this is 3. And the first row has 4 non zero elements or the largest number of non zero elements. So, this is 4 and this is 4. So, this ranked 4. So, if we do this then what do we get?

(Refer Slide Time: 39:43)


$$A(\text{ordered}) = \begin{bmatrix} 4 & 0 & 0 & 11 \\ 0 & 3 & 0 & 10 \\ 9 & 0 & 2 & 8 \\ 7 & 6 & 5 & 1 \end{bmatrix}$$

We get the reordered matrix like this that is if we see this is 1 1 and this is 1 4 because this is 1, this is 1, this is 2, this is 2, this is 3, this is 3 and this is 4, this is 4 ((Refer Time: 40:15)). So, this is fourth column and this is the first row. So, we have here 1 1 and then 1 4.

Similarly, the other elements can be put like this, this is 2 and this is 2. So, 2 2 this is 2 1 is 0, 2 2 is 3, 2 3 is 0 and 2 4 is 10. So, 2 1 is 0, 2 2 is 3, 2 3 is 0, 2 4 is 10. Similarly for other rows that is row number 3 and row number 3 in ordered sequence. If we do then we get the reordered or the ordered matrix like this, that is renumbered matrix like this; this we call the ordered matrix. Now, for this if we do the Gauss elimination process as we did earlier the matrix is now still the same matrix, except that the rows and columns have been ordered in a particular sequence.

(Refer Slide Time: 41:34)

Indian Institute of Technology, Kharagpur

After 1st step the eliminated matrix becomes:

$$A^{(1)} = \begin{bmatrix} 4 & 0 & 0 & 11 \\ 0 & 3 & 0 & 10 \\ 0 & 0 & 2 & -16.75 \\ 0 & 6 & 5 & -18.25 \end{bmatrix}$$

No Fill-ins in this case.

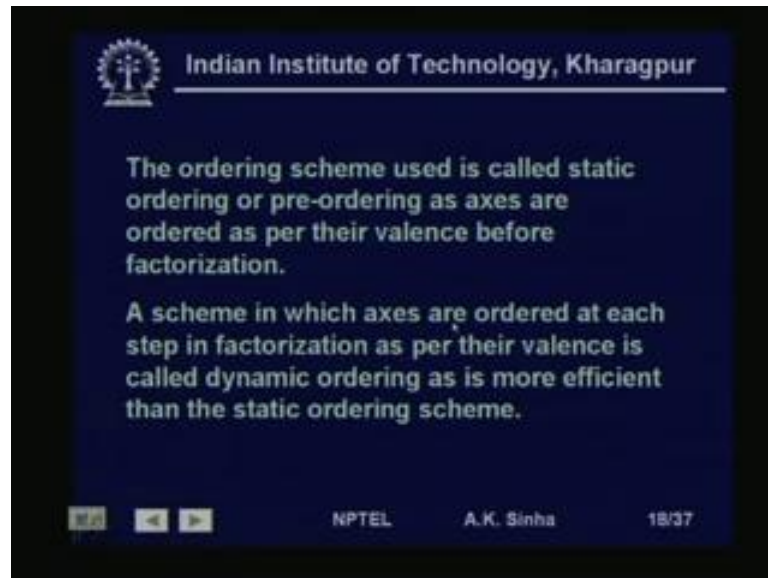
NPTEL A.K. Sinha 17/37

If we do this now then what we find. We find that all the fill-ins which were, all the zero elements which were there are still zero in case of this matrix. That is these elements if you see these are all zeros, which were there. And these are still remaining zero. While the factorization process goes on. That is there is no fill-in which occurs in this case.

Of course this is not always true that there will be no fill-ins, when we factorize the matrix. This is a special case, but most of the matrices we will get fill-ins. But, the number of fill-ins are going to be somewhat less as compared to, if we do not order the matrix proper.

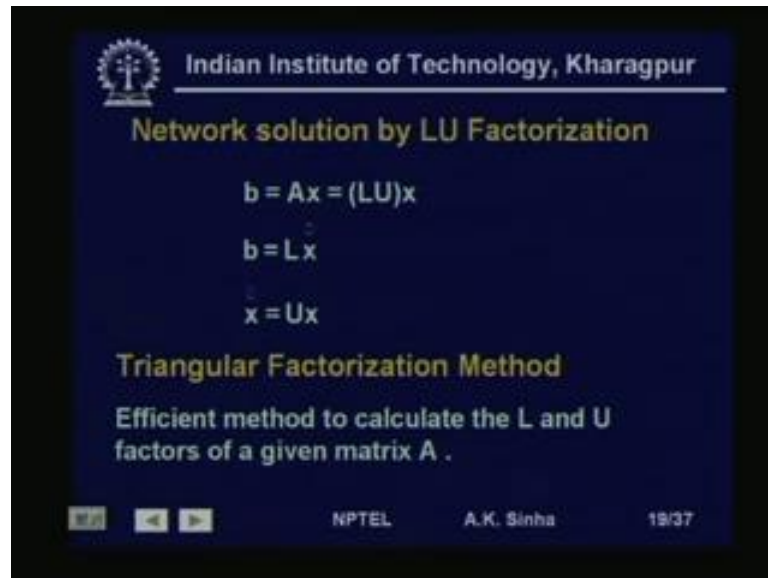
Even this kind of an ordering where we are ordering the matrix based on the valence of that row is called a static or pre ordering method. Mainly because, this ordering is done before the factorization process starts. That is you see the matrix you find out it is valence for each row of the original matrix. And then you order them accordingly, without going or looking into the factorization process itself. The if we try to order these matrixes at each step of the factorization itself as the rows are being factored. Then we have what we call a dynamic ordering.

(Refer Slide Time: 43:43)



That is a scheme in which axes are ordered at each step in factorization as per their valence is called a dynamic ordering, and is generally more efficient than the static or pre ordering algorithm, when this is very intuitive, because each time as we go through the factorization process, we are checking the valence. And since we have we are checking the valence in after each step of factorization. We are taking care of the fill-ins as well. And therefore, we expect that the number of fill-ins, which will be caused by the fill-ins cross by the previous operations are going to be less. Because we are taking care of the valence or the ordering at each step in the ordering process itself. So, dynamic ordering is generally a preferred ordering method for sparse matrix problems.

(Refer Slide Time: 44:57)



Next we will talk about network solution by LU factorization. Till now we talked about the Gauss elimination method. In fact, Gauss elimination method is quite efficient. But, one of the problem with Gauss elimination method is that each time I need to solve Ax is equal to b I have to do the complete factorization process.

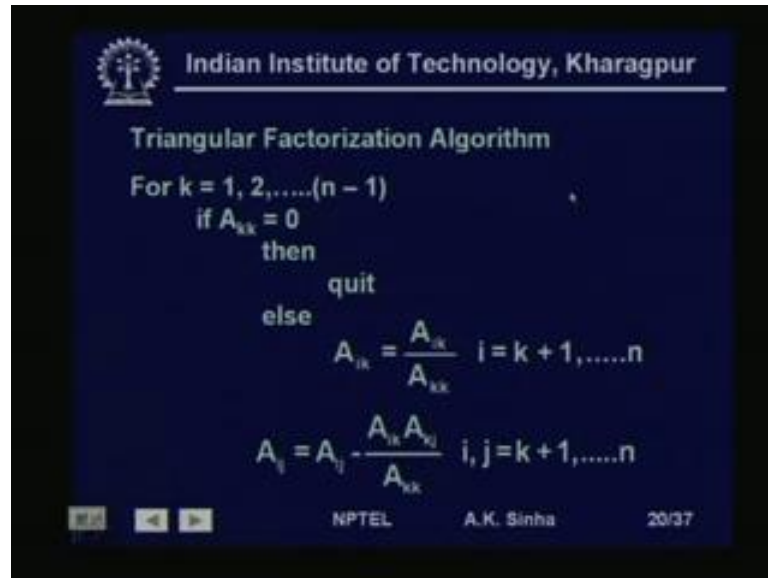
In many cases especially the case where the matrix A remains constant this is going to be an unnecessary overhead. That is processing the A again and again every time. Just like in Fast-decouple load flow we have B dash and B double dash matrices, which are costly. And therefore, in each iteration if we have to go through the factorization process for the B dash and B double dash matrices. Then we are having an undue overhead of computation. This can be reduced if we can store the matrix as factored matrices.

The process is very similar to the Gauss elimination method except that the process during elimination. The lower triangular part of the matrix which becomes 0 is used to store the numbers for processing of the column vector b for the operation. And this same thing can be looked as using the A matrix as multiplication of two matrices. The L matrix which is a lower triangular matrix with one's in the diagonal, and U as the upper triangular matrix very similar to the matrix, which we get after the Gauss elimination process.

So, let us say we have a set of equations b is equal to Ax . Now, this A matrix is an n by n matrix it can be B dash it can be B double dash, it can be the Jacobian matrix. Whatever

it is can be written as two a product of two matrices L and U, where L is the low triangular matrix and U is the upper triangular matrix. So, we have Ax is equal to LU into x. This we can write as b is equal to L into x hat.

(Refer Slide Time: 48:07)

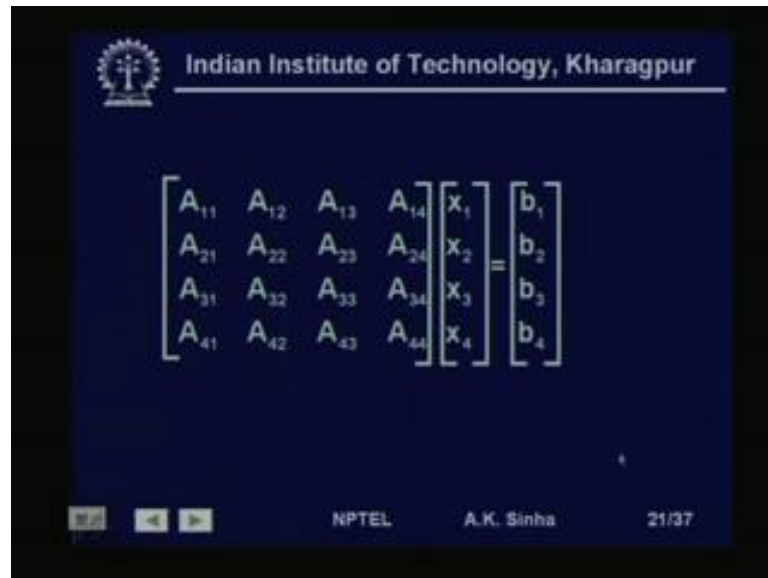


Actually this hat has not come here properly, this is x L into x hat and what is x hat? X hat is equal to U into x. So, what we do is we write this as L into U x and U x we are writing as x hat. Because, while we are working on a upper triangular matrix with x. And this will give us again a column vector x hat. And if we multiplied with L, then we will get as x is equal to where x hat will be equal to U x.

So, this is what we need to do. So, what we will have do is. First we will find out this in terms of x hat and then b is equal to L x hat needs to be solved. So, what we do is b is equal to L x hat. We will solve to get x hat and then we will solve x hat is equal to U x to get x. Now, how do we factorize this matrix an efficient method to calculate the L and U factors for a given matrix is triangular factorization algorithm.

Now, in this algorithm for k is equal to 1 2 up to n minus one if A k k is equal to 0, then just go out that is if the diagonal element is 0. Then go out otherwise you find out A i k is equal to A i k by A k k and for i is equal to k plus 1 to n. That is this is the factored element. Then similarly we can find out A i j as equal to A i j minus A i k into A k j by A k k, where i and j are from k is plus 1 to n.

(Refer Slide Time: 50:31)




Indian Institute of Technology, Kharagpur

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

NPTEL A.K. Sinha 21/37

So, this way we can try and get the both the lower and upper triangular part of the matrix. Let us take this example of a 4 by 4 matrix Ax is equal to b .

(Refer Slide Time: 50:51)



Indian Institute of Technology, Kharagpur

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21}^{(1)} & A_{22}^{(1)} & A_{23}^{(1)} & A_{24}^{(1)} \\ A_{31}^{(1)} & A_{32}^{(1)} & A_{33}^{(1)} & A_{34}^{(1)} \\ A_{41}^{(1)} & A_{42}^{(1)} & A_{43}^{(1)} & A_{44}^{(1)} \end{bmatrix}$$
$$A_{i1}^{(1)} = \frac{A_{i1}}{A_{11}} \text{ for } i = 2, 3, 4$$
$$A_{ij}^{(1)} = A_{ij} - \frac{A_{i1}A_{1j}}{A_{11}} \text{ for } i, j = 2, 3, 4$$

NPTEL A.K. Sinha 22/37

Now, here as we are doing the Gauss elimination process, what we do is, we multiply this row with whatever value we had at this position. And then that is multiplied by this divide by this and then subtract from this. So, this element will become 0 and we will get these elements modified. Same way we can make this 0 right and for doing that we again multiplied by A_{31} divide by A_{11} and so on.

So, this way we can again try to make this zero in Gauss elimination. But, what we are doing we are saying A_{i1} that is these elements is equal to A_{i1} divided by A_{11} , this what we are doing we are dividing it by this and then subtracting.

So, here we are storing this A_{21} divided by A_{11} . So, A_{21} after the first processing is equal to A_{21} divided by A_{11} for i is equal to 2 3 4. So, this and A_{ij} these elements will become A_{ij} what was this minus $A_{i1} A_{1j}$ into A_{1j} divided by A_{11} .

That is we have this element which is i is equal to 2 in this case. So, A_{21} into A_{11} A_{12} A_{22} into A_{12} divide by A_{11} this will be this element and so on. This element and this element all this processing can be done. So, what we are doing in Gauss Seidel elimination this used to be 0. But, now what we are doing is we are storing these elements there.

Similarly, when we process this column then we will have these elements stored again in the same fashion like this. And these elements will be what we will get during the factorization process. So, these elements are given by this, again when we process this element. So, the third column when we are doing we can process this. So, we will get these elements this and this elements this elements and this elements here, using the same relationship.

(Refer Slide Time: 54:06)

Indian Institute of Technology, Kharagpur

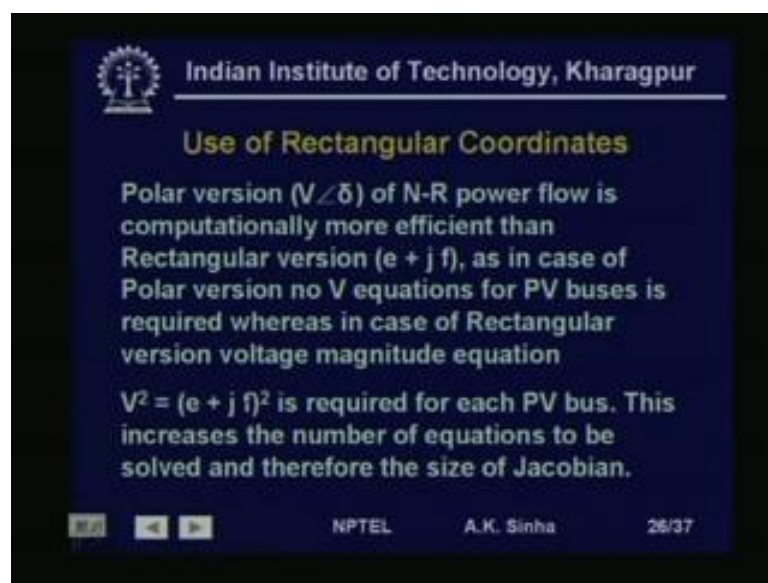
$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ A_{21}^{(1)} & 1 & 0 & 0 \\ A_{31}^{(1)} & A_{32}^{(2)} & 1 & 0 \\ A_{41}^{(1)} & A_{42}^{(2)} & A_{43}^{(3)} & 1 \end{bmatrix}$$

$$U = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ 0 & A_{22}^{(1)} & A_{23}^{(1)} & A_{24}^{(1)} \\ 0 & 0 & A_{33}^{(2)} & A_{34}^{(2)} \\ 0 & 0 & 0 & A_{44}^{(3)} \end{bmatrix}$$

NPTEL A.K. Sinha 25/37

And then if we do this then what we have is, we have a factored matrix here, where these elements. The upper triangular elements are same as what we get in case of a Gauss elimination. And the elements below the diagonal are the factors that we need for processing of the b vector or the vector of the mismatches. And therefore, this matrix now can be written as a factored matrix LU , where we have these elements of the lower triangular part of the matrix. And the diagonal elements have 1. Whereas, the upper diagonal element of the matrix are denoted as U , then lower diagonals of course, as 0 here upper diagonals are 0. So, this is the LU factors that we will get.

(Refer Slide Time: 55:19)



So, we can use this LU factors for storing the factors of A matrix and use this for solution using this then what will do is. We will work with b is equal to Lx . So, we will calculate L is known b is known. So, we will calculate x and once we know x we will solve for this x is equal to Ux and we will get x .

So, this is how we do the solution. So, let the matrix remain constant this b may keep changing and we need to perform only these two operations. Because this is already been done, this saves considerable amount of time. In solving the large set of equations, and this is what normally is done for power flow solutions.

So, with this we will stop today in the next lesson. We will talk about how we can improve the computational efficiency further by using rectangular coordinate version for

the voltage and admittance. That is what we would try to do this avoid calculation of trigonometric functions.

Because when we are dealing with the polar coordinate version of the power flow, then we have sin theta, cos theta, terms involve. And computation of trigonometric function takes considerable time on computer. So, in order to avoid these trigonometric functions, we will see how we can use the rectangular coordinate formulations. to help in reducing the computation time. Then we will talk about the regulating transformers and how we can model them and the power flow solution. And then we will see some comparison of the Newton-Raphson load flow and Fast-decoupled load flow algorithms. So, with this is, we will stop today. And we will take up in the next lesson the other things as I discuss just now.

Thank you very much.