**Advanced Numerical Analysis**
**Prof. Sachin Patwardhan**
**Department of Chemical Engineering**
**Indian Institute of Technology - Bombay**

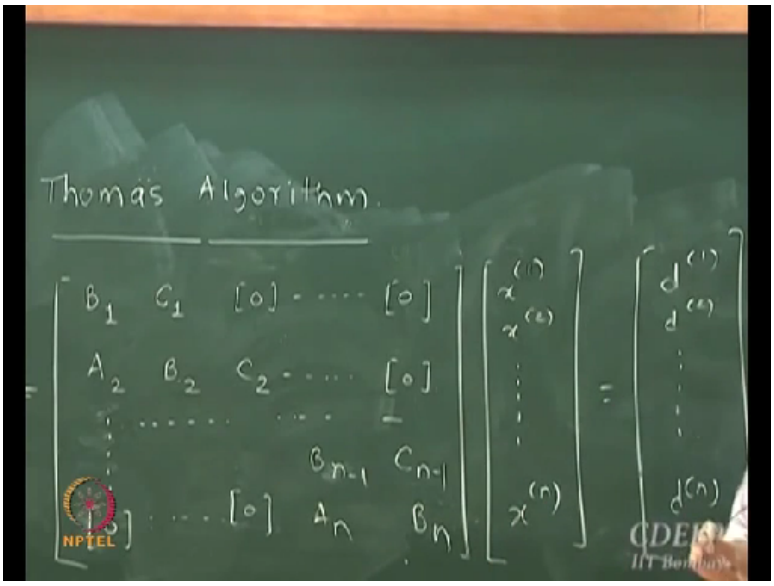
**Lecture – 26**
**Methods of Sparse Linear Systems (Contd.) and Iterative Methods for Solving Linear Algebraic Equations**

So in the last lecture, we were looking at direct methods for solving sparse linear systems. Sparse linear systems are ones in which the matrix is filled with lot of 0's, very few non-0 elements and towards the end of the lecture, we looked at the Thomas algorithm. The Thomas algorithm is for tridiagonal systems. Before that we also looked at block diagonal systems. Block diagonal systems, you can exploit the structure to come up with a computation method which is significantly, computationally efficient as compared to the conventional Gaussian elimination.

So now to continue with this, we looked at Thomas algorithm yesterday and I said that the number of multiplications and divisions is linearly proportional to n which is significant reduction from n cube/3 particularly for large dimensional matrices. If you have n to be 1000, n cube/3 would be a very large number as compared to 5n-8. Just compare the number of multiplications and divisions that you have to do for a modest n of 1000, okay.

So just continuing on this, I am going to talk today about Thomas algorithm for block triangular matrices or block tridiagonal matrices.

**(Refer Slide Time: 02:14)**




So for sparse systems, we looked at tridiagonal systems and now we will look at this spatial class which is, so this is a block, this is a matrix. So I am trying to solve an equation Ax, where A is given by this matrix, okay. I am going to split again x into subvectors. So I have this vector x1 x2 up to xn. So this is a block tridiagonal system, okay. In earlier case, we had scalars here. In earlier case, B1 C1 A2 B2 C2, all these were scalars and then what we had derived earlier was Thomas algorithm when the diagonal matrices are scalar.

If you look carefully, this matrix which you get here is not really tridiagonal because each one of them could be a dense matrix. So each one of them is a dense matrix, the raw matrix is not block, is not tridiagonal. It may have multiple diagonals, okay. But we are able to identify this B1 C1 A2 B2 C2 and so on such that this is a block tridiagonal matrix. What I mean by block tridiagonal matrix? The so-called elements of this matrix are themselves matrices.

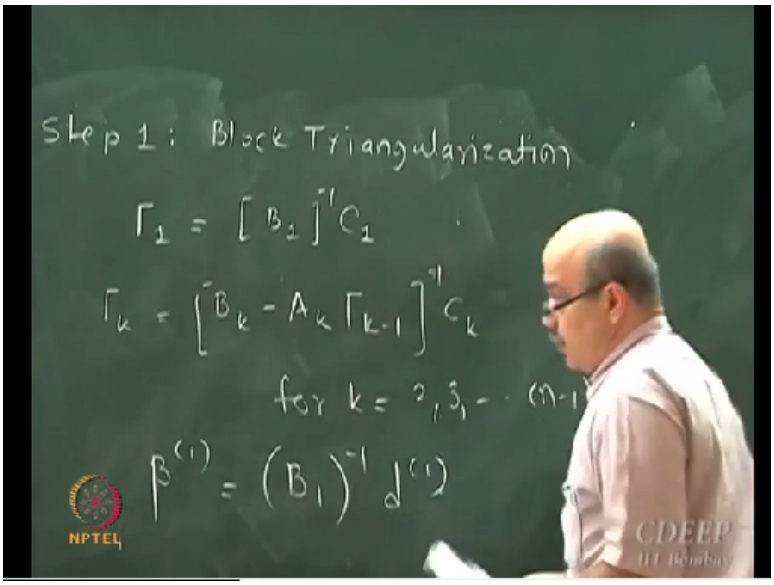
B1 is a matrix, C1 is a matrix. A2 is a matrix, B2 is a matrix, C2 is a matrix of appropriate dimensions. Is everyone with me on this, is this clear. See these are submatrices within this huge matrix, these are submatrices, okay. So this could be the 3*3, this could be the 3*3, this would be then, let us say this is 4*4, then this will be 4*3 and so on, okay. You will have to have appropriate dimensions here. They need not be all of the same dimension.

They can be of different dimensions, okay. So now I want to exploit the spatial structure this has,

a lot of 0's or 0 matrices or these are 0 matrices, okay. That is why I am writing them in square brackets. So this is a huge matrix whose elements are submatrices, okay. These are subvectors x superscript 1, next x superscript 2 are subvectors of appropriate dimension, okay. If this is 3*3, this x1 will be 3*1 and so on, okay.

If this matrix is 4*4, this x2 will be 4*1 and so on. So this matrix has appropriate dimensions. So these are submatrices which are and these are subvectors. The right-hand side also has been divided into subvectors and now I am just going to apply the Thomas algorithm that we came up with.

**(Refer Slide Time: 06:34)**



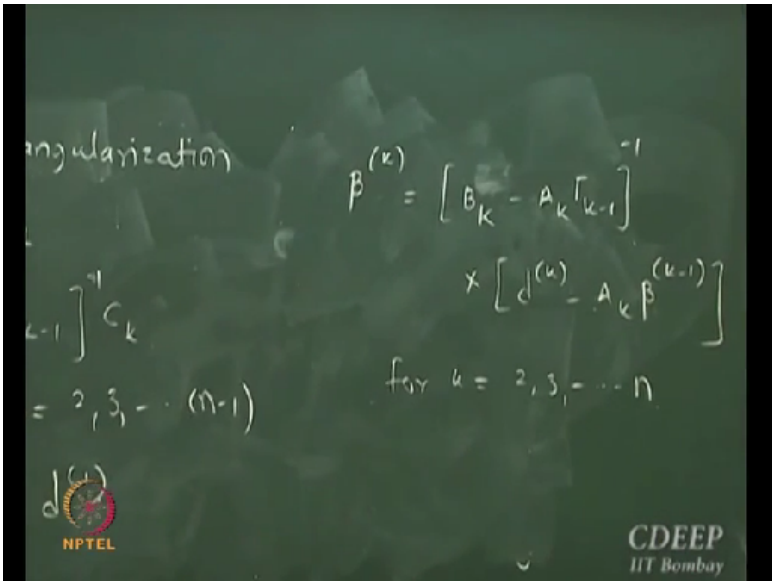

The first step, the step 1 is called block triangularization. What you do in Gaussian elimination? What do we do in Gaussian elimination? In Gaussian elimination, we make the diagonal below the main diagonal 0, okay. Here we are not going to make the diagonals. We are going to make the block diagonal below main block diagonal 0, okay. So we are going to retain matrices along the main diagonal. I am going to make matrices along the main diagonal.

I am going to make everything that is below that is 0, okay. So this block triangularization would be, first matrix is gamma 1 is B1 inverse C1 and gamma k is… if you look at the Thomas algorithm which are developed for the scalar case, you would notice that what I am doing here, it just a matrix analogue of what I have done for the scalar case. It is not different, okay. So earlier

I had written scalar gamma, I am writing here capital gamma, okay.

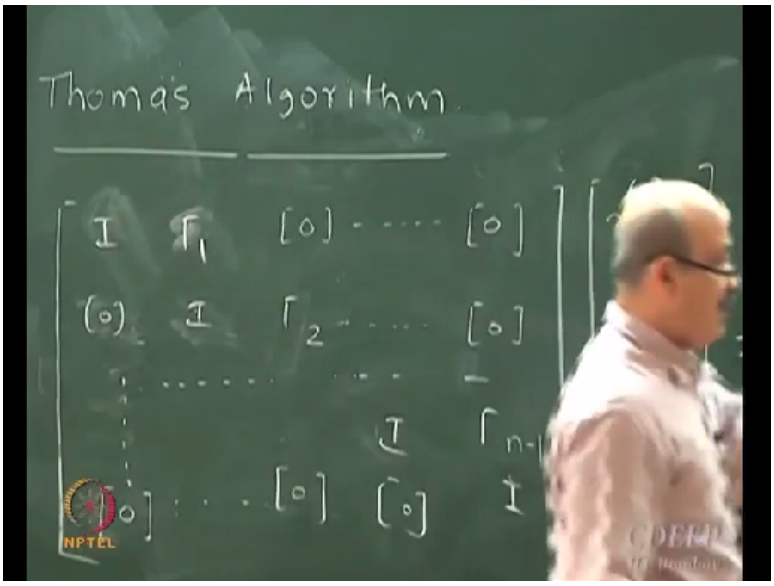
You have to be careful now because you have matrices. You have to talk about pre-multiplication, post-multiplication very very carefully, okay. You cannot interchange the order and whatever was a division earlier, would come out to be a matrix inverse here and so on. Then you have this beta 1 vector will be B1 inverse, d1. So you have to worry about these elements when we do the transformation to block triangularization.

**(Refer Slide Time: 09:04)**



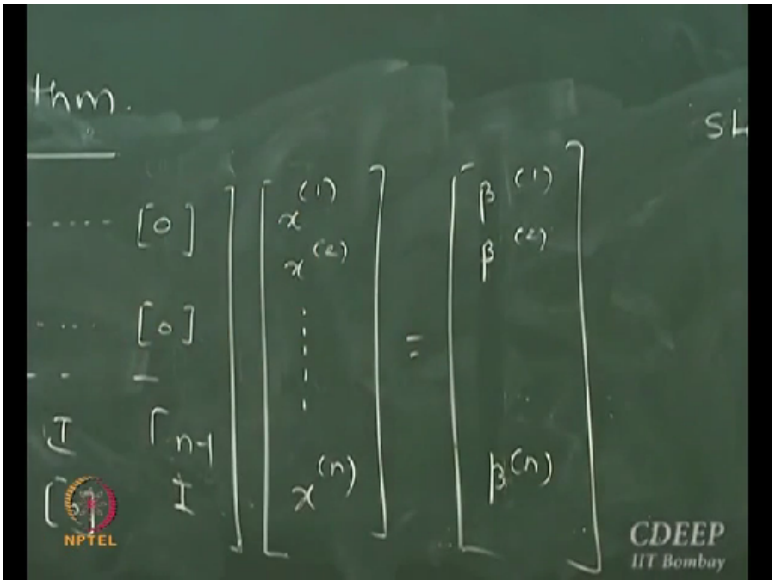

And beta k will be Bk, yes, we will say Bk… So my first step is block triangularization, okay. So in the block triangularization step, what I am going to do is, if I go back here, I am going to replace this by I 0 I.

**(Refer Slide Time: 09:47)**




So all these will be replaced by identity matrices of appropriate dimension. Everything below the main diagonal will be null matrices and these upper one will be replaced by gamma 1, this will be gamma 2 and this will be gamma n.

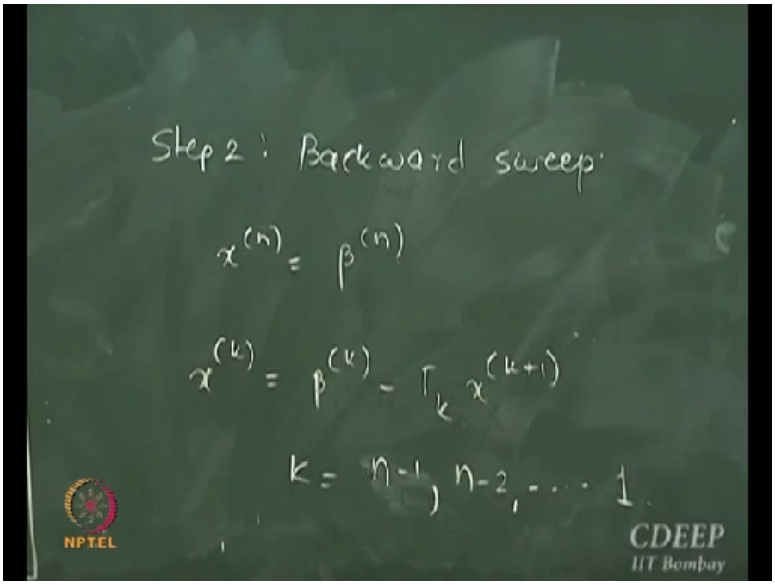
**(Refer Slide Time: 10:26)**




These vectors will be replaced by beta 1, beta 2, beta n, okay. So I have done block triangularization. I have done block triangularization, okay. This is block diagonal identity, this is gamma 1, gamma 2 which we are calculating here. Now notice one thing, even though you are doing matrix inversions here, even though you are doing matrix inversions here, these are small dimension matrices. Each one of them is 3*3, 4*4 whatever. Even if it is 10*10, these are small dimension matrices as compared to the big matrix.

So the individual inverses which are involved here are small dimension inverses. So these you could compute by some standard method, Gaussian, Gauss-Jordan method or Gaussian elimination, Gauss-Jordan method basically. So this you can do quick computations because the number of computations involved is relatively small as compared to the big matrix, okay. If I were to Gaussian elimination into the big matrix, the computations will be much much more than during these small inverses, okay. So that is why. The next step is of course backward sweep.

**(Refer Slide Time: 11:50)**




So in backward sweep, we start from this end, okay. Look at this I times xn=beta n. So this is my first, so itself is a solution for xn component, okay. Using xn component, in the second equation, I can recover xn-1, xn-2, I go back and then I recover, I recover the entire state vector. So this algorithm, it exploits large number of 0's. So actually, if you go back and look at this matrices, this matrix in its original form, right now I have written it in a block triangular or block tridiagonal matrix form.

In original form, this will be a banded matrix, okay. So there are few diagonals which are non-0, okay. Rest all above and below are 0. We are able to express those diagonals in terms of these matrices, in terms of these matrices, submatrices, okay and that is why we are able to come up with a computationally efficient solution for solving this particular problem, okay. There are many other such forms and as I told you that if you go to Matlab toolbox, you will find.

There is a Matlab toolbox for sparse systems and there are many more forms which one, another simple form is to solve our triangular matrices. Triangular matrices are either lower triangular matrices or upper triangular matrices. Now where do you get upper triangular, lower triangular matrices? In Gaussian elimination, you get lower triangular, upper triangular matrices, right. In Gaussian elimination, you will get an upper triangular matrix and then you do backward sweep, right which is, so if you have lower triangular matrices, you can do computations very fast and then likewise if you have a block lower triangular matrices, okay.

You can come up with algorithm which is again a very fast and exploit the structural, not worry about the 0's which of there and then try to come up with the solution. Just look at your notes, I am not going to do on the board. You have to move on to something else. There is one more concept I want to introduce here. Now block triangular or lower triangular, upper triangular, these are spatial structures.

And likewise, you can go on exploiting these structures to come up with efficient Gaussian elimination algorithms or efficient modifications of Gaussian elimination algorithms which are suited for a specific structure and you can do fast computations, that is the idea. So as I said, my motivation behind talking about sparse matrices was to sensitise you that there exists something called sparse matrix computations, okay.

Now what is the origin of this problems, is problem discretization that is discretization of boundary value problems, discretization of partial differential equations, okay on finite element or finite difference. All these methods, orthogonal collocations on finite elements, all these will give rise to certain nice structures which has lot of 0's and then you can exploit that to come up with solutions which are efficient.

This is particularly useful when you have iterative procedures. In iterative procedures, you may have to solve Ax=b kind of equations, Newton-Raphson, okay. You are actually solving multiple times Ax=b, okay. You never actually, when you actually do the Newton-Raphson step, in a large-scale problem, you never do A inverse. It is fine to do it for a beginner problem which has 3

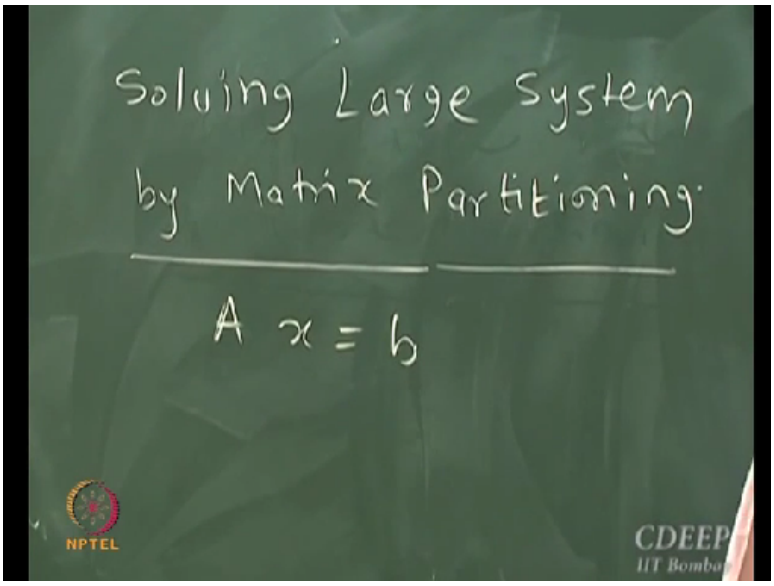
variables or 4 variables but when you have large number of equations and when you are doing Newton-Raphson, you actually solve A delta x=-f at xk, solve the linear problem and then substitute delta x and get a new x, okay.

So you have to solve multiple times linear algebraic equations and that is where exploiting the structure. Now if your solution scheme gives rise to a specific structure to A matrix, that will happen in every iteration. It is not going to be different, only the numbers will change, iteration to iteration the structure of the matrix will remain same, okay. So if you are calling a specific subroutine, sparse subroutine within your Newton-Raphson, that subroutine will remain same.

It is not going to change, okay. So if I am doing the problem this TRAM problem, okay. In TRAM problem, suppose I decide to do say orthogonal collocations on finite elements, so 3 finite elements, every time I will get a matrix in the iterations whose numbers might change but the structure will be same, okay. Whenever 0's appear, the 0's will appear every time. That is because of your structure of discretization, okay or if you decide to do it using finite difference, okay.

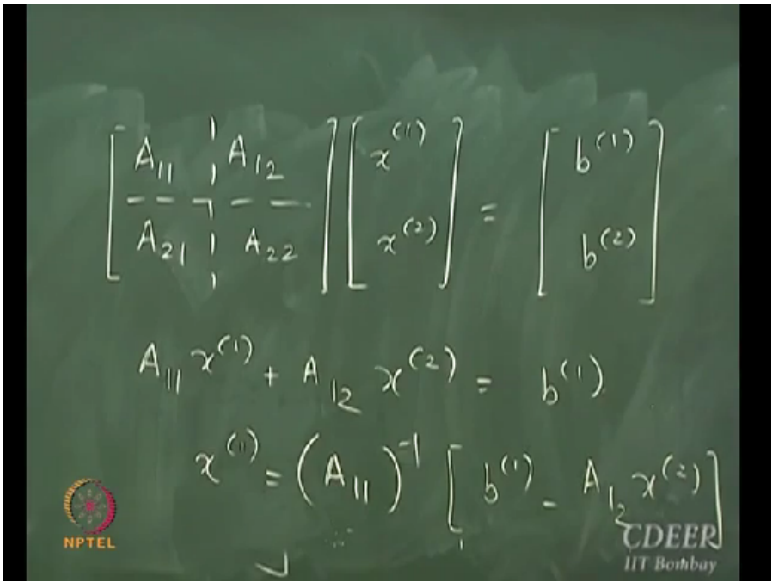
Within the iterations, that matrix is always going to be tridiagonal. It is not going to change, okay. So those features will remain same, those features are not going to change, just remember that. So you can actually make use of those features and come up with. There is one more trick for reducing the computations. This is, if you have a large system, now I am not talking about, not necessarily talking about sparse matrices. The thing what I am going to talk about need not be a sparse matrix but it is a trick to make computations fast.

**(Refer Slide Time: 18:39)**




So solving a matrix by matrix partitioning, okay. I am going to explain the basic concept. It can be extended to more complex partitioning. I want to solve Ax=b of course, where A is large matrix. I do not want to invert the large matrix or I do not want to use Gaussian elimination on the entire big matrix. One possibility is that I transform this equation into 2 subequations, okay.

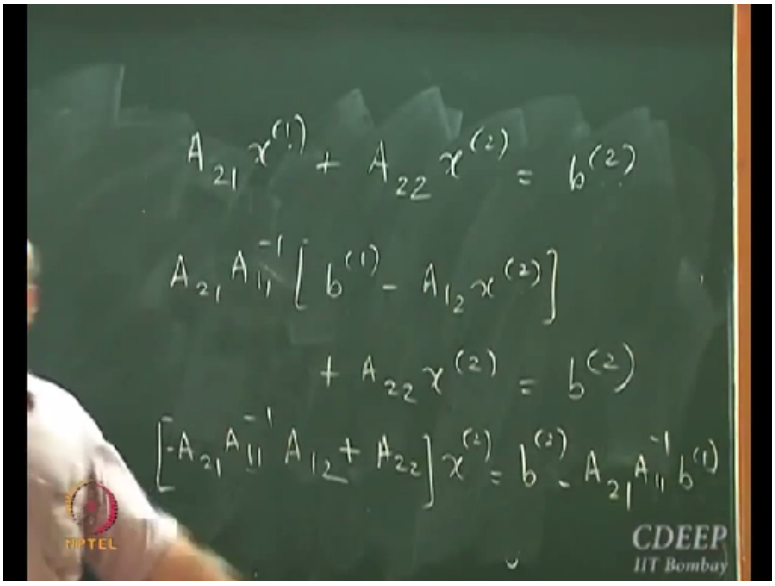
**(Refer Slide Time: 19:24)**




So this A matrix is written as a partition A11 A12 A21 A22. So this big matrix A, I am going to partition into 4 submatrices, okay. Well if some of them, let us say this one is almost a null matrix or this is a null matrix grid. You have a simplified solution x1 x2 and b1 b2. So I am going to partition this equation, okay. How will I proceed now? The way I am going to proceed is, well I will say that we have 2 equations A11x1+A12x2=b1.

Can you try and solve this? Can you make an attempt? How will I proceed now? Let us say A11 is invertible. Then can you eliminate and write. Can I eliminate x1 using the first equation? How do you solve? Suppose I decide to write x1 as A11 inverse b1-A12x2, what next? How do I solve the next part? I just take the second equation, okay.

**(Refer Slide Time: 21:29)**




A21x1+A22x2=b2, okay and then I substitute this x1 here, okay. So it is A21A11 inverse b1-A12x2+A22x2=B2 and now I can rearrange. I can put all the terms of x together, okay. I can put all the terms of x together. So I will get an equation which is A21A11 inverse A12+A22x2=B2-… What is the advantage of doing this? See n cube, you just remember that Gaussian elimination would require computations to the proportion of n cube/3, okay.

Suppose this is 1000*1000 matrix, I decide to divide it into 500*500 500*500, okay. So this will be the first problem, first problem. I have to do a Gauss-Jordan for a 500*500 matrix, okay and next time, so I will get an A inverse, A11 inverse once, I will store it, I use it in the next step, again I have to do a Gaussian elimination of 500*500, 2 Gaussian eliminations of 500*500 requires less computations than one 1000*1000, okay. Two 500*500 is less than 1000*1000.

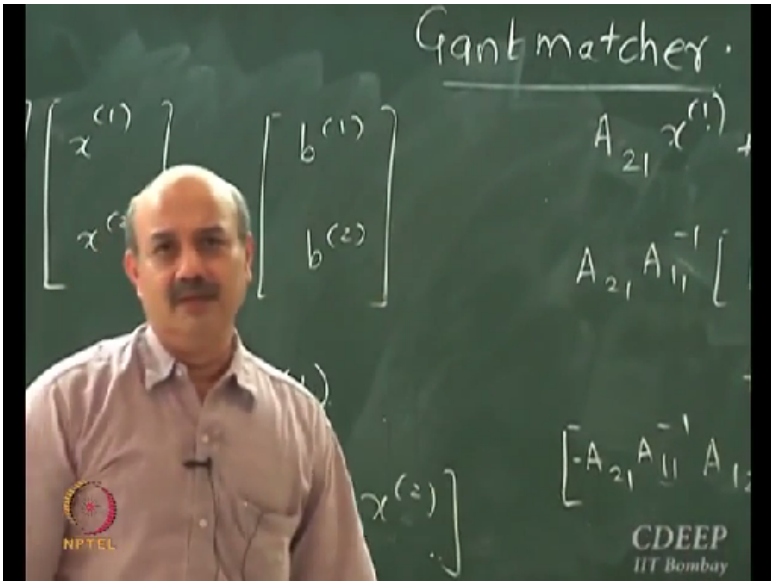
So likewise, I have just done a simple partitioning here. I could think of A11 A12 A13 A21 A22 A23, I can make multiple partitions. Actually matrix theory is very very interesting and it has

history of almost more than 100 years, 150 years. Cayley was one of the founders of matrix theory. You probably know Cayley-Hamilton theorem. Cayley and who was the second mathematician? I forgot his name, well I will try to remember and tell you. They were friends and incidentally Cayley's friend, so he did not get.

Though he was a mathematician, that time in England, if you have to be graduate, you have to take a oath to be devout Christian and he was a Jew and he refused to take that oath. So he was not given the degree in mathematics. So he took to law, okay and during the recess, during the cold recess between 2 sessions, these 2 guys used to work on matrix theory, that is what troubles us now but they came up with voluminous matrix theory which is now, well they are known as twins of… I will tell you the name tomorrow.

So they lot to matrix theory and lot of things have been developed. The bible of matrix theory would be, there is a book called Gantmatcher, okay.

**(Refer Slide Time: 26:19)**

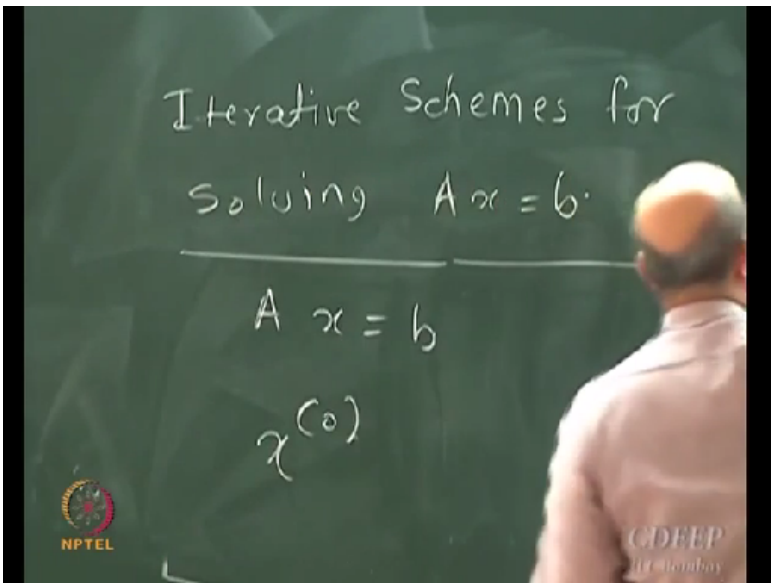



This book was published in I think 1945. We have this book in the library. It is like, it is 2 times Perry's handbook. It is huge and this is in 1945, so you can imagine what must be matrix theory by now. I do not think you can have it in this… So what actually we look at as a matrix theory is just tip of the iceberg and linear equations solving is something that you, that is just bread and butter of numerical competence.

So you need it everywhere, you cannot live without, okay. So this is about sparse matrices or matrix partitioning and you have all kinds of tricks to make your computations faster. Now I am going to move from but this method was I talked about matrix partitioning, was still belonging to the direct methods. I had not gone to, I had not gone to iterative methods yet. Now I am going to move on to iterative methods.

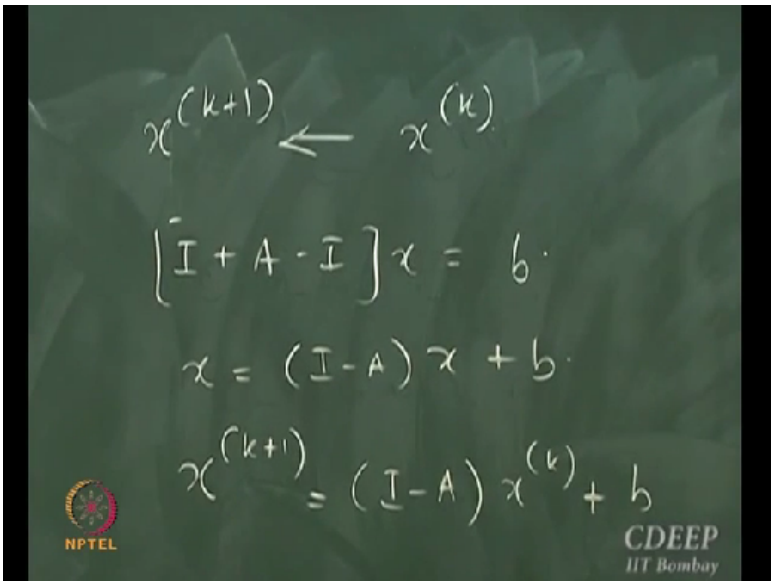
So iterative methods, the idea is that you start with the guess solution, okay. So I want to solve again Ax=b and then many times, well it is difficult to come up with a general number of multiplications and divisions for iterative methods but in general, the number of modifications and divisions for iterative methods can be much smaller than the Gaussian elimination-based methods.

**(Refer Slide Time: 28:16)**




So iterative schemes and next few lecture, I am just going to spend on how to solve this problem iteratively. So what is iterative schemes. I start with the guess solution. I want to solve this. I start with some guess solutions. So let us say x0 is my guess solution, okay.

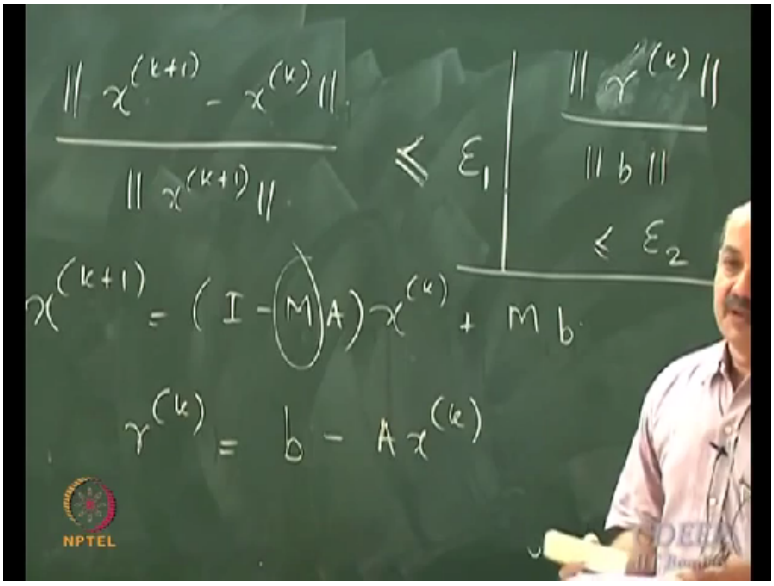
**(Refer Slide Time: 28:46)**




And somehow I form a sequence so, I want to form a sequence such that xk+1 is generated from xk, okay. I start with x0, so x0 will give me x1, x1 will give me x2, x2 will give me x3 and so on, okay. A very very simple crude way of doing this is, I will say I rewrite this equation as I+A-I*x=b. So I will just write this as x=I-Ax+b. I have just rewritten the same equation, Ax=b, I have written as, okay.

And then I can form an iteration scheme from this as xk+1=I-Axk+b. So I start with a guess solution x0, I start with a guess solution x0, I will get x1, I put back x1, I get x2, I get a sequence of vectors, okay. Get a sequence of vectors and then well I need to talk about convergence. So I will have to say when to terminate this.

**(Refer Slide Time: 30:31)**




So I going to terminate this when xk+1-xk, this becomes, this ratio becomes very very small. This is less than or equal to Epsilon. So Epsilon is typically very small number 10 to the power -10, 10 to the power -8 or something. So I am going to go on doing this iterations, well I will write a more generic form of the iterations. I am going to put a generic iteration here. Xk+1=I-MA… where M is the matrix which I have to choose such that this sequence will converge to the solution, okay.

The sequence will converge to the solution, that is what I want, okay. What will happen if M is exactly equal to A inverse? If M is exactly equal to A inverse, then you will get the solution right because it is A inverse b, okay. Here M is called as approximate inverse of A. How to do these iterations? How to construct the iterations, we will see now. But basic idea is this that I am starting with a guess. I construct a new guess and I go on iterating till it converges to a solution, okay.

Another way of looking at the convergence is through a residue. So I look at this residue vector which is… I look at this residue vector, b-… When you arrive at the solution, what should be this difference? It should be exactly equal to 0 but well in numerical computations, we do not look for 0's, we look at a small number. So another way, another criteria for convergence could be, this rk, r vector, it is good to normalise this with b, is less than or equal to, let us say this is Epsilon 1 and this is Epsilon 2.

So either I look for this criteria to be satisfied or I look for this criteria to be satisfied for terminating my iterations, okay and I am going to iterate till I reach a solution. This is a philosophy of iterative schemes. Now next few lectures, we will see how to form this iterations? Under what conditions you are guaranteed to converge to a solution. See the problem you might face is that, well I am trying to solve a large system of equations, okay and I have to guess a solution.

So tough problem because if you have to guess a vector which is 1000*1 or 10,000*1, how do you guess. Well that is where you have to use your knowledge from physics, engineering, chemical engineering but fortunately here, even if you give a wrong guess, what we see is that, if you take care of correctly choosing this M matrix, okay, you are guaranteed convergence to the true solution, okay.

So which means even if you give a wrong guess, completely arbitrary guess, okay. You are guaranteed convergence, if you understand what makes the convergence work and that is what we are going to see. First you are going to look at this methods, algorithmic part of it. After we have done with algorithmic part, we will move to analysis under what conditions these methods converge to the solution, okay.

Can you do some tricks to make the solution converge to those solutions. So those things we will look at, okay. So let us start developing these methods one by one. I am going to talk about 3 methods which are very prominently used. One is called the Jacobi method. Some of you might have done this in your undergraduate curriculum. Now these iterative schemes are very often used while solving partial differential equations because you just want to go very quickly to the solution, converge to the solution, okay.

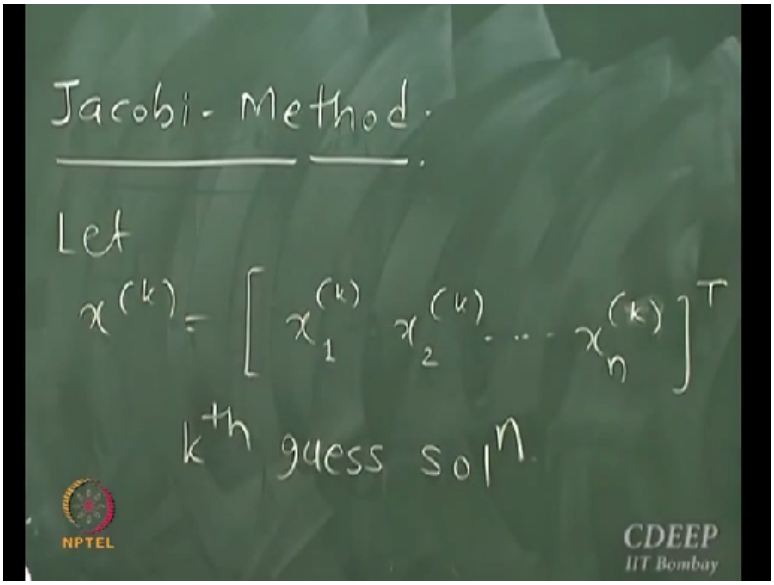
Well, you might wonder by these iterative methods am I going to the true solution or am I going close to the true solution. Well, you are getting an approximate solution, no doubt but even when you do Gaussian elimination, then too you are getting an approximate solution because you are doing truncation, there are all kinds of errors. Even when you want to solve a problem Ax=b,

okay, you normally end up solving A tilda x=b tilda, okay.

That is because for example if you have an element pi coming in your matrix. You cannot represent pi exactly, so you… when you do multiplications, okay, the computer has a finite precision. So you truncate overflow errors. So even when you solve using Gaussian elimination, you cannot construct the true solution. There also there are approximations, here also there are approximations, so nothing to feel bad about approximate solution, okay.

So let us look at this Jacobi method which is the simplest one.

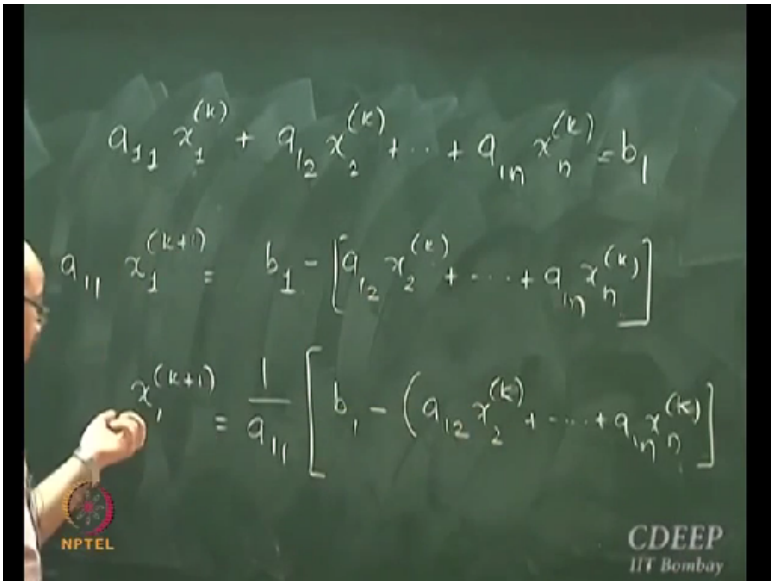
**(Refer Slide Time: 36:36)**




Let us say this is my guess solution, okay, it is guess solution, okay where the k starts of course from 0. So this is my kth guess solution. Now starting from this guess, so this I is my guess solution. Starting from this guess, I want to create a new guess, okay. Now the way I am going to do is, I am going to look at each equation in this set of equations, line by line, okay. I have how many, how many equations I have. N equations and n unknowns, right.

I have n equations and n unknowns. I am going to look at each equation line by line. Let us look at the first equation.

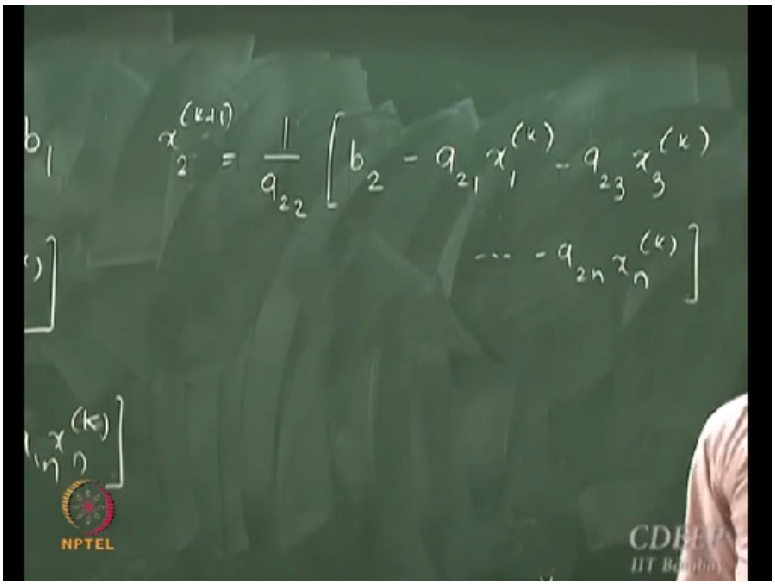
**(Refer Slide Time: 37:36)**




So a11x1k a12x2k up to a1nxnk=b1, this is my first equation. Everyone with me on this, this is my first equation. Well actually since it is a guess, this is not exactly equal, okay but for the time being, just understand how the method is developed, okay. I have this guess solution, I want to construct a new guess, what I am going to do is, I am going to rearrange this equation and say that x1k+1=b1, okay, -, okay I will put this a11 here, b1-a12x2k a1nxnk, is this fine, okay. So now my new guess is going to be x1k+1=1/a11*b1-…

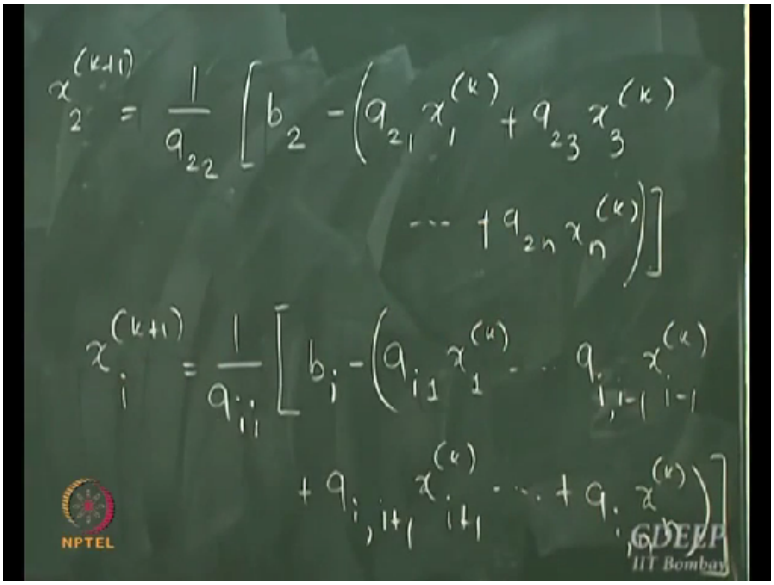
See I am starting from the previous guess, I am starting from the previous guess, okay and constructing the new guess for x1. Is this clear what I have done. Just looked at one equation, okay. Well of course my assumption is that a11 is not 0, a11 is not 0. How will I use the second equation? I can do the same trick, okay. So my second equation if I just skip in between steps.

**(Refer Slide Time: 39:49)**



So I can write my second equation x2k+1=1/a22*b2-a21x1k-a23x3k-a2nxnk. Use the second line, okay, take x2 on the left-hand side, use the second row in the equation, second equation, take x2 on the left-hand side, okay and you will get this equation. I have just done the same thing which is here, okay.

**(Refer Slide Time: 40:43)**



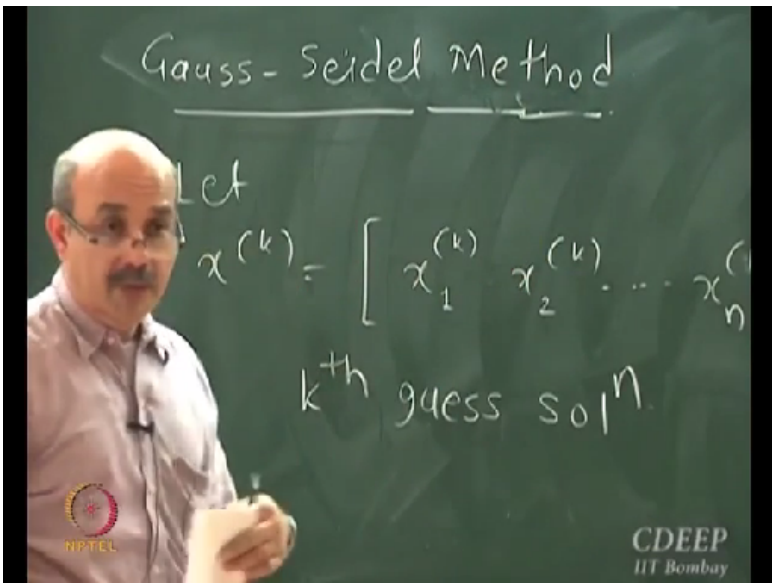
If you are confused with, I can take this inside, if plus helps better to compare with the previous expression, then there is a close brace here, okay. This is the first element. This is the third element. Second element has been taken on the left-hand side. Is everyone with me on this, is this clear, okay. Likewise, I can go and in general, I can write that xik+1/1/aiibi-, well equation 59, if you have the notes, just correct them. This should be not b2, it should be bi. It is bi-ai1…,

okay.

So I have written the expression in general for the ith row. I have written expression for the ith row. In ith row, what is the representing assumption here? The represented assumption is that all the diagonal elements are non-0. So if it is not like that, you have to do rearrangement of your equation such that all the diagonal elements are non-0, that is the critical thing here. You cannot implement this algorithm unless that is done.

But you can see here, it is very very simple to do these calculations. I am just generating from the old guess, I am generating a new guess, okay, yes. **"Professor - student conversation starts**" To use (()) (42:49). That is another modification. So she has rightly guessed the next modification which is called as Gauss-Seidel method, okay.

**(Refer Slide Time: 43:06)**




Smitha, right. So Smitha like… (()) (43:23) yes, yes but this is the next, the obvious thing to do. **"Professor - student conversation ends"** Next is that if you are generating a new guess here, in the next expression when you go here, in x2, why continue with the old guess, right. See I am generating here in this step, I am going to start from 1, okay. So I will first go to generate x1k+1. Having generated x1k+1, here I need not use old guess, I could replace this by k+1, okay.

I could just replace this by k+1. Likewise, when I go to x3, I will use x1 x2 new and x4 to xn old

and so on, okay. So in general here ith expression, I will use x1k+1 up to i-1 and I+1 onwards I will use the old values, okay. I-1 I will use the old values. This is obvious modification. This is called as Gauss-Seidel method and what you can show is that convergence of this method is much better than the Jacobi method, okay.

This method will converge much faster than the Jacobi method and this is very very simple to implement. In fact, even in terms of computer storage, this is easier to do because here in the earlier case, you have to maintain 2 separate vectors. One is the old guess, other is the new guess. Here you can just keep using the new value which is created. Those of you who know programming well will appreciate that you do not have to maintain 2 vectors.

You can just have 1 vector and just write these equations. The new value will be automatically used in the next equation, okay. So implementation wise, this is much much efficient. This is also convergence properties are better. We will see why convergence properties are better a little at. So this is one modification that we do to come up with iterations schemes. Now there is one more modification called as overrelaxation method and I will talk about it in my next class or relaxation method.

So we say that well from Jacobi to Gauss-Seidel, actually you end up making the convergence faster. So why not even further accelerate by putting some acceleration parameter, okay. So this acceleration parameter businesswomen, it is actually called relaxation parameter. This relaxation parameter business we will see later but these iterative methods tend to work much faster than the conventional methods, okay. So for large-scale systems, many times these iterative methods are preferred as against and the solutions that you get are pretty much close, okay.

If you do a or Matlab experiment in which you solve a problem using iterative methods and probably you can also check how many number of multiplications and divisions are required. So you will get an idea which particular method is faster.

So the next class onwards we will start on how to analyse this convergence. How do you make sure that convergence will occur? Are there any tricks to make? enhance the convergence. So all

those things we will see from our next class. So it would help if you bring these notes because there are too many summations I J K business and instead of writing it, I want to explain it more than spending time on writing on the board.

So just get copy of the notes and then we can do it. So other thing which I am going to do is, this is what I have written is the algorithmic part of it, okay. I am going to rearrange these equations in the matrix form, okay, because once I write these equations in the matrix form, it is easier for me to analyse this equation, the convergence behaviour. It is difficult to analyse in this raw form.

This raw form is useful for implementation, we can write algorithm in this way, okay but analysis, when will it converge and all, will require rearranging these calculations into some nice matrix forms and we will look at the matrix properties and say when will convergence occur, okay. So that is what we will do next.