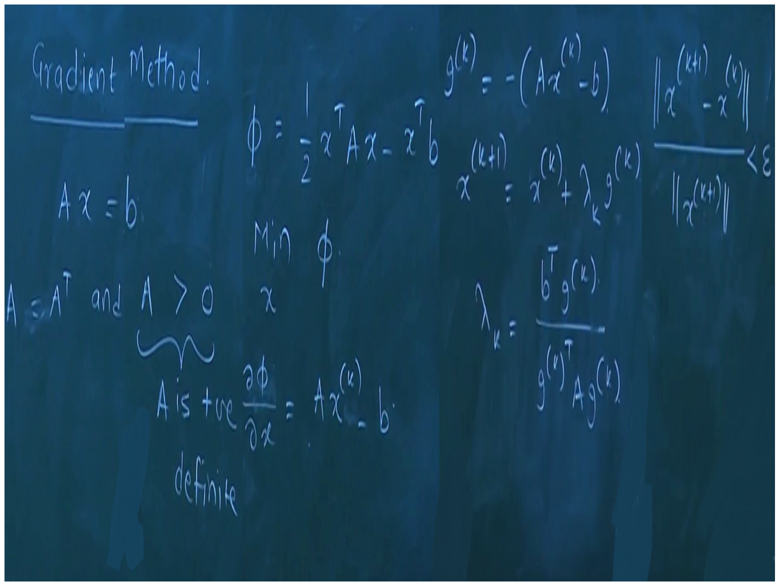
Advanced Numerical Analysis

Prof. Sachin Patwardhan
Department of Chemical Engineering
Indian Institute of Technology - Bombay

Lecture - 33
Conjugate Gradient Method, Matrix Conditioning and Solutions of Linear Algebraic Equations

So we have been looking at this method of optimization numerical optimization to solve linear algebraic equations, and what we derived in the last class was the gradient method or the Cauchy method. The gradient method does a search iterative search in the negative of the gradient direction.

(Refer Slide Time: 00:52)



Gradient Method

$Ax = b$

$A = A^T$ and $\underbrace{A > 0}_{\text{A is +ve definite}}$

$\phi = \frac{1}{2}x^T A x - x^T b$

$\min_x \phi$

$\frac{\partial \phi}{\partial x} = Ax^{(k)} - b$

$g^{(k)} = -(Ax^{(k)} - b)$

$x^{(k+1)} = x^{(k)} + \lambda_k g^{(k)}$

$\lambda_k = \frac{b^T g^{(k)}}{g^{(k)T} A g^{(k)}}$

$\frac{\|x^{(k+1)} - x^{(k)}\|}{\|x^{(k+1)}\|} < \varepsilon$

And if I summarize this algorithm this was so we are trying to solve A x=b, and we have this we are assuming that A is symmetric or A=A transpose, and A is well in Applied mathematics or in linear algebra you often write that A is positive definite by saying A>0 okay that means A is positive definite that does not mean all elements of A are >0, when you write A>0 it means A is positive definite A is asymmetric.

And then we formulated an objective function, well we know what to do if A is not symmetric positive definitely, we pre-multiply both sides by A transpose and then we get we work with the modified problem. So phi is 1/2 x transpose A x-x transpose b we want to minimize this with respect to x phi, and then we use this gradient method to come up with the search direction. So what was this gradient method? The gradient method was we find the gradient direction.

So dou phi/dou=A x-b this is the gradient and we want to move in the negative of the gradient, so we take g k=-A x k-b this is my negative of the gradient direction, and then we move how much do we move this time, we saw a one-dimensional optimization problem. And then we move actually x k+1=x k+ lambda k g k, and we found that lambda k is nothing but b transpose g k okay, so we make an optimal move every time in the negative of the gradient direction.

This is negative of the gradient direction, lambda k times g k that is the move that we make every time, and this is the classical gradient method or the Cauchy method. And then these iterations are stopped, when you reach a point or gradient is not changing either way of them is fine, x k is not changing the new iterations are not adding too much information we specified an epsilon, epsilon could be something like 10 to the power-8 10 to the power-10 a very small number okay.

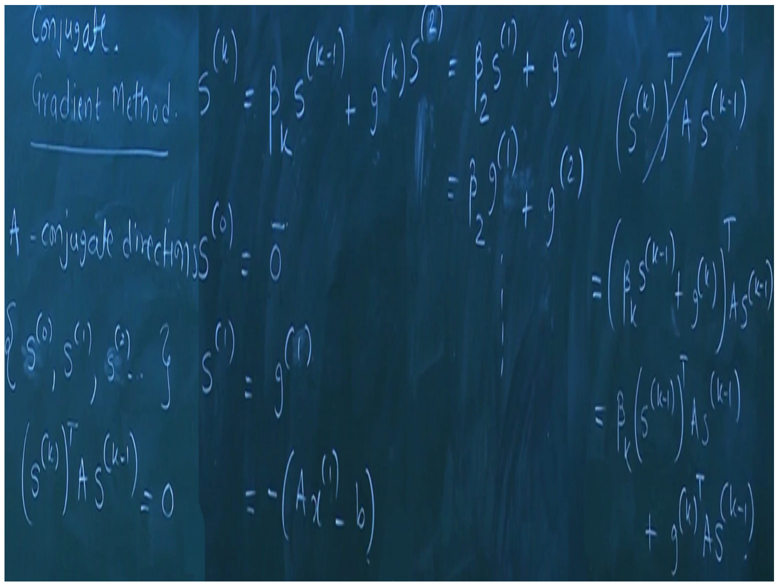
And we say that the relative change in x as the new iterates are generated this is insignificant that is when we stop okay. So this is iterative scheme, and if you remember if you see here every time you need to compute A x k-b k you need to compute A x k-b this is the residual, actually when you solve this this g k will be 0 gradient direction is 0, which is the necessary condition for optimality for this particular objective function okay.

So finding the optimum is equivalent to reaching the or meeting the necessary conditions for optimality, and we satisfy or we terminate based on even you can terminate based on g k, if g k is very, very small okay, the gradient vector is very, very small. Now this slight modification of this because this the problem with this method it works very well when you are far away from the optimum, when you come close to the optimum this tends to become very, very slow okay.

The moves that you make every time when you come close to the optimum are quite small, so that trouble is this we need to do something to accelerate the convergence, when you come to closer to the optimum okay. Now what I am saying is not just true for this particular problem, it is in general to that gradient method is one of the very basic methods for doing numerical search or numerical optimization, but the problem with gradient method is that it works very well when you are away from the optimum.

And when you come very close to the optimum it becomes very, very slow, so convergence to the optimum is slow okay, and we need to do something about it okay.

(Refer Slide Time: 06:32)



Conjugate. Gradient Method.

A - conjugate directions $S^{(0)} = \bar{0}$

$\{S^{(0)}, S^{(1)}, S^{(2)} .. \}$ $\quad S^{(1)} = g^{(1)}$

$(S^{(k)})^T A S^{(k-1)} = 0 \quad = -(Ax^{(1)} - b)$

$S^{(k)} = \beta_k S^{(k-1)} + g^{(k)}$

$S^{(2)} = \beta_2 S^{(1)} + g^{(2)}$

$= \beta_2 g^{(1)} + g^{(2)}$

$(S^{(k)})^T A S^{(k-1)} \rightarrow 0$

$= (\beta_k S^{(k-1)} + g^{(k)})^T A S^{(k-1)}$

$= \beta_k (S^{(k-1)})^T A S^{(k-1)}$

$+ g^{(k)T} A S^{(k-1)}$

So there is a variant which is called as conjugate gradient method okay, so what we do is we move in the conjugate directions A conjugate directions okay. So given a matrix A okay A is our matrix, we call see a set of directions S 0, S 1 let us say that S 1, S 2, S 3 are directions in which we want to move okay, these directions are called as A conjugate, if this condition is satisfied if you take 2 directions k and k-1 not any 2 k and k-1 okay say new direction and previous direction okay.

And if this in a product is 0 okay or if this whatever you want to call it dot product is this 0, which is weighted by A matrix okay then these directions are called as A conjugate directions okay. And what you can show is that instead of moving along the gradient direction if you move along the A conjugate directions, then you reach the optimum very, very fast okay. If you have the notes there is a small correction, it is on equation 129.

So how do you generate this A conjugate directions okay, we are going to use this formula S k=beta k S k-1+, so we are going to use the gradient direction it is not that I am not going to use gradient direction okay, it is the conjugate gradient algorithm. So I am going to use the gradient vector current gradient vector, but to that I am going to add some correction which is based on the previous, this is based on the previous you know previous direction okay.

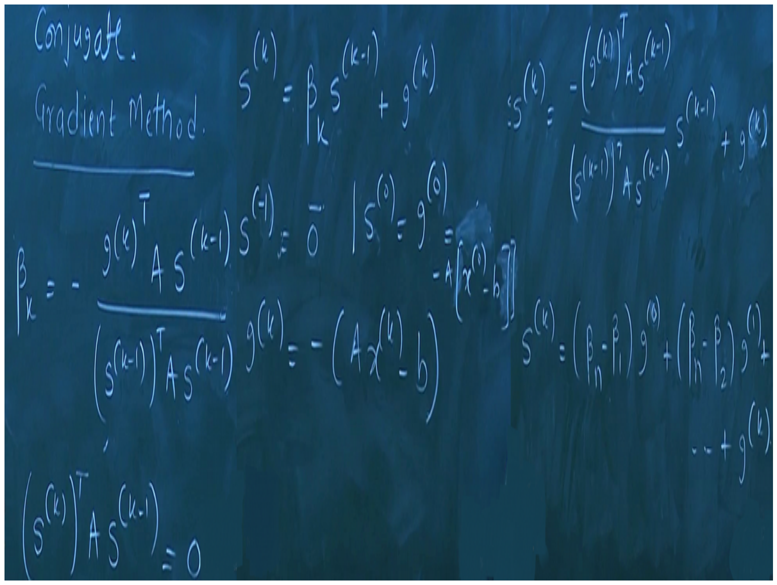
And I kick off this algorithm with S 0=, so what means sorry S 0=0 which means S 1 the first move that I am going to make okay, the first move that I am going to make is g k g 1 okay the first gradient. Let us say I am starting my iterations at 1, I am taking S 0 to be 0, so the first direction is the gradient direction, what will be the subsequent direction, well how do I choose this beta k and all that will become clear soon.

What is S 2? S 2 will be beta 2 S 1+ g 2 that means gradient at point 2, negative of the gradient at the point 2 sorry, g here is negative of the gradient we are going to move in the negative of the gradient we are minimizing not the gradient direction, g is the negative of the gradient direction okay, So g 1 is nothing but -of A x 1-b okay and so on okay, I cannot just go on doing this, so you can see, see what is this? This is beta 2 g 1+g 2 right.

So I am taking the linear combination of the 2 gradient directions not just negative of the current gradient direction, but I am taking linear combination and so on, so every iteration I am going to do this okay. Now I am going to invoke this A conjugacy, so what I am going to do now is to take S k transpose and pre-multiply it by A S k-1 okay, so what is this beta k S k -1+g k transport A S k-1 right, now these are A conjugate directions.

What should be this? This should be 0, so since these are A conjugate this left hand side is 0 okay, so I get what do I get from here 0=beta k S k-1 A S k S k -1 transpose A S k see S k-1, let us put it in the bracket become easier to understand, this is S k -1 transpose A S k-1 okay +g k transpose A S k -1, this see what is g k is the negative of the current gradient direction g k is known to me okay, S k -1 is the previous direction okay so using this I can find out beta k okay.

(Refer Slide Time: 13:18)


Conjugate Gradient Method.
$S^{(k)} = \beta_k S^{(k-1)} + g^{(k)}$
$S^{(k)} = -\frac{(g^{(k)})^T A S^{(k-1)}}{(S^{(k-1)})^T A S^{(k-1)}} S^{(k-1)} + g^{(k)}$
$\beta_k = -\frac{g^{(k)T} A S^{(k-1)}}{(S^{(k-1)})^T A S^{(k-1)}}$
$S^{(-1)} = \bar{0}$
$S^{(0)} = g^{(0)} = -A[x^{(0)} - b]$
$g^{(k)} = -(Ax^{(k)} - b)$
$S^{(k)} = (\beta_n \cdot \beta_1) g^{(0)} + (\beta_n \cdot \beta_2) g^{(1)} + \dots + g^{(k)}$
$(S^{(k)})^T A S^{(k-1)} = 0$

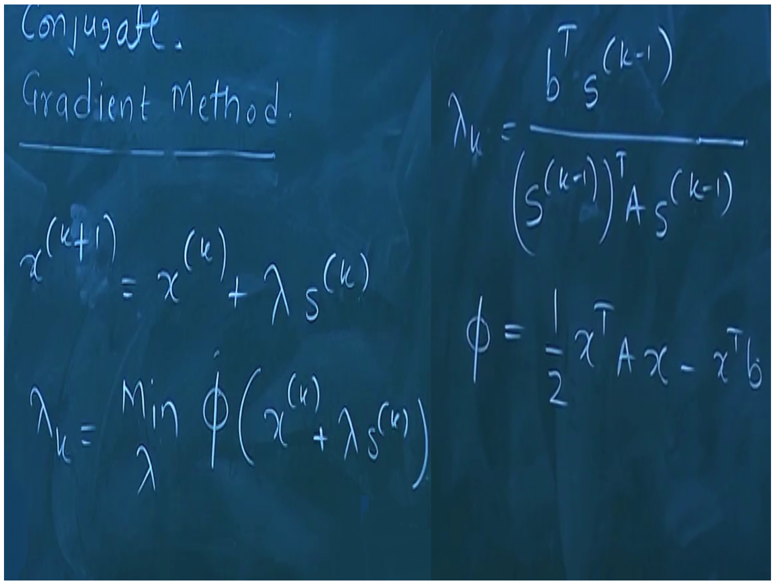
So that gives me a formula for beta k, how much to move, what is the linear combination that should be made okay for A conjugate directions, so now and what is this if I write g k to be -of A x k-b that completes the formula, so I have this g k which is given by this, this is how you kick off your iterations, and this is how you compute beta k okay, this is how you combine okay. So actually I can combine this and okay I can combine the 2 and I can find out this as my iteration scheme.

This is a scalar remember, this is a scalar ratio of 2 scalars, this is a vector direction and this is current negative of the gradient direction. So this is how you combine, and see how beta k changes with little bit of algebraic jugglery, what you can show is that I am skipping this I have given the derivation here you can just go over it. It is that S k with some algebraic manipulation can be shown to be beta and -beta 1, well if you do A little bit of index jugglery.

If I just come back here I will do a little bit of index jugglery, I will call this -1 S -1=0 okay, so if S -1=0 then S 0 becomes g 0 which is-same as -A x 0-b okay, if you do a little bit of index jugglery you called 0 as the starting point -1 previous to the starting point okay, because if you put k=0 whether it will start from k=1 or k=0 it is a matter of just writing the algorithm, to be consistent with other notation here I will just so g 0 is the negative of the initial gradient direction.

So what you can show with the little bit of jugglery is that, so in this case you find the current direction in which to move using a linear combination of all the previous gradient directions, you do not use only one current gradient direction, you use a linear combination of all previous gradient directions okay. Now this is the direction in which you want to move, how much do you want to move? You have to use this lambda business okay.

(Refer Slide Time: 16:46)


Conjugate.
Gradient Method.
$x^{(k+1)} = x^{(k)} + \lambda s^{(k)}$
$\lambda_k = \min_{\lambda} \phi(x^{(k)} + \lambda s^{(k)})$
$\lambda_k = \frac{b^T s^{(k-1)}}{(s^{(k-1)})^T A s^{(k-1)}}$
$\phi = \frac{1}{2} x^T A x - x^T b$

So after we have decided this S k, what we do is okay, so my new point that is x k+1=x k+ lambda, S k is only the direction just number okay, and then I find out so lambda k=min with respect to lambda phi of x k+ lambda S k, this is =min with respect to lambda okay. And then this can be solved very, very easily, lambda k turns out to be this scalar, how much to move in this direction is given by this particular scalar okay.

Now what I can show is that theoretically I can reach the optimum of phi only in n steps okay, I have started with this objective function where A is symmetric positive definite, so theoretically you can show that 1/2 x transpose A x-x transpose b okay, solution of this can be reached starting from arbitrary initial condition only in n steps okay. But of course what happens is there are numerical errors when you compute in a computer and you cannot practically reach in n steps.

On paper you should converge to the solution only in n steps very, very powerful method okay, now just look at a matrix which is say 1000 cross 1000 okay, your gradient to reach very close to the solution in 1000 steps okay, each step only requires one matrix multiplication right, it requires some matrix multiplications and you know you converge to the solution very fast okay. So this is actually one of the popular methods for solving linear algebraic equations for large-scale systems and very quickly you can reach the solution so that is one of the.

So you can get very close to the reasonably accurate solution, you can get very quickly. **"Professor - student conversation starts"** yeah, (()) (19:48) you the lambda k can have some problem, but you are multiplying by A right g k transpose A g k, no but I do not think that will happen in conjugate gradient, gradient method you can end up in the problem because of that you can get not 0/0, you get a small number by small number.

So that can sometimes cause a problem, but more problem with in gradient method is not because of that the difficulty is because it becomes very slow as you come, and this does not become slow, this homes down to the solution in the n steps that least theoretically, so it is a much better method okay. **"Professor - student conversation ends."** So we move on to the next thing now, well we have talked about different methods of solving A x=b, let me just take a brief review of what we have done till now.

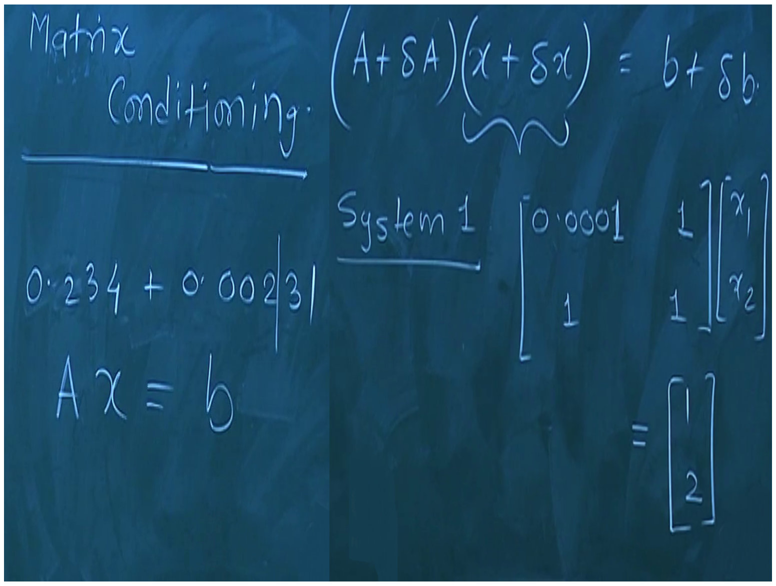
We started looking at direct methods we had a at least taste of what are called as sparse matrix methods, now we try to exploit number of 0s that are present in the matrix, and then we come up with a way of solving very quickly in less number of multiplication and division. Then we moved on to iterative methods for solving A x=b which was completely new to you, because you have been looking at Gaussian elimination, Gauss Jordan everything that is using.

But iterative methods can particularly in let us say in Newton-Raphson, when you are solving multiple A x=b problems it is not that important that you exactly get the true solution at the iterates, you can very quickly get the solution approximate solution very close to the approximate solution using iterative methods. So Jacobi method, Gauss Seidel method we had a look at those methods. We also looked at why and how they converge, how to make them converge and so on.

So we relate to the eigenvalues, we said that inside unit circle knows if you have I eigenvalues of S inverse, you can guarantee convergence to the solution necessary and sufficient conditions, we also derive some sufficient conditions some necessary and sufficient conditions, or how do you transform the problem to guarantee solution and so on. And we looked at very quickly gradient based methods, optimization based methods okay gradient method and conjugate gradient method.

If you do a course on optimization, you will see that gradient method and conjugate gradient method they form you know basis of many, many variants, and there are more powerful variants of, but these are 2 basic methods which are used in numerical optimization. The next thing that I want to move is what is called as a condition number, well what I want to talk about now is very, very important aspect of numerical computing, this is how well-conditioned or ill-conditioned system of linear algebraic equation is okay.

(Refer Slide Time: 23:31)



Matrix Conditioning.

$0.234 + 0.002|31$

$Ax = b$

$$(A+\delta A)\underbrace{(x+\delta x)} = b + \delta b$$

System 1

$$\begin{bmatrix} 0.0001 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

So now the problems that occur in numerical computing, one of the reasons for these problems is fixed point arithmetic okay, what is fixed point arithmetic? Fixed point arithmetic is that we have finite number of bits to represent a number okay, and then we truncate okay. So suppose a number goes beyond this representation or if there is a number which cannot be represented exactly using finite number of bits, we just take the nearest approximation and work with it okay.

Just to give you an idea that if you have a computer which has 3-bit precision okay, and if we are to add point you know suppose I want to do this addition okay, a 3 bit precision computer or 3 digit precision let us not say 3 bit, 3 digit precision computer cannot actually look at this part of the number, it will ignore this part of the number it will only take this+0.002 okay, and that 3 digit computer will commit an error of 3.1 10 to the power-4 in this addition okay.

And just imagine this is now of course we have numbers floating point which is 16-digit precision and so on, but there is something that is being neglected okay, and even though every time it is very, very small okay each time when you do it, it is very, very small number that is neglected, but you know you are doing millions of operations right, you are doing millions of operations large matrix, you are doing so many 1000s of multiplications, division.

Each time you are neglecting a small number okay, each time you neglect a small number that can cascade and grow and build up and create a problem, now this happens only for those cases where the matrix is problematic, when matrix is not problematic okay the errors have natural tendency to die down and things work out, you get reasonable solutions. So actually what you have to remember is that when you are solving A x=b using a computer.

You actually never end of solving this exact equation, you end up solving something like this you end up solving A+ delta A x+ delta x=b+ delta b, I want to solve this A x=b I end up solving this problem, that is because I cannot represent A accurately inside my computer, I cannot represent b accurately inside my computer. If A and b are not accurately represented, I cannot get exact solution to my problem okay.

Now just to give you, so what I want to do is I want to have some idea when this solution will be far away from what I really want to get okay. What are the factors that influence how this solution behaves, first of all the right hand side b is something which is specified by the person who is using this set of equations by the user let us say, so b is not in our hands okay, what is fixed about this system of equations is matrix A? okay.

So how the solutions behave, it actually depends upon how you know A matrix is or how well-conditioned or how ill-conditioned A matrix is, so our main you know culprit or when things go wrong is A matrix, how to recognize whether A matrix is good or A matrix is bad. If A matrix is bad whatever you do okay, to some extent well good programs like mat lab can recover and give you a reasonable solution, but beyond the point even mat lab fails.

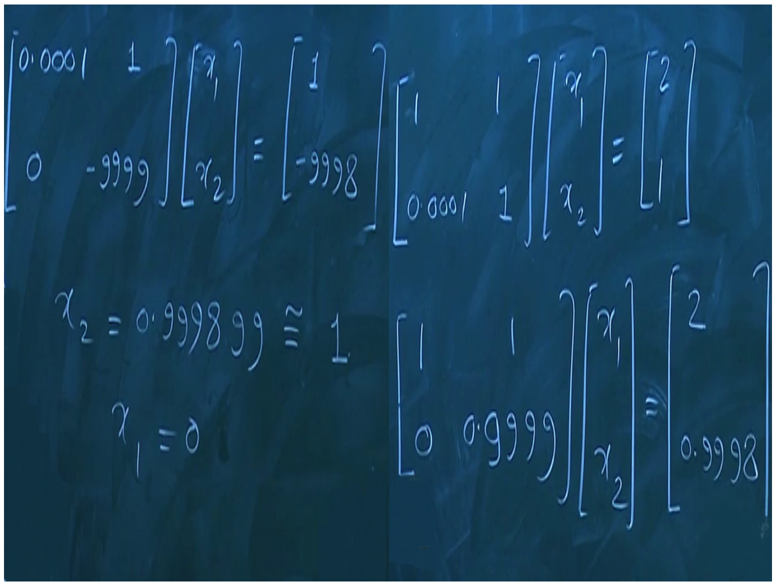
And it is not because there is something wrong with mat lab, because there is A problem with the matrix, the matrix is ill-conditioned. In iterations sometimes you may have received a message like this matrix is ill-conditioned, the results will not be reliable, so mat lab does its job it tells you that there is something going on wrong here you should know what it means, you should know what the warning means, you should understand and analyze okay.

So let me give you a motivation for what is going to happen next that is condition numbering, I will need one more lecture to cover this business of condition numbers, but I am going to give a motivation for this today, why do I want to look at condition number, then we will look at the mathematical details okay. So this example I have taken from Strang, it is a very nice simple example which illustrates, he using a very simple example 2 cross 2 example.

Strang has illustrated the point that sometimes the calculations go wrong because you are not done proper ordering of the calculations, if you reorder the calculations things can be okay. But sometimes the matrix is bad whatever you do there will be a problem okay. So this system 1 he considers is the simple system, see this is the very, very simple system 0.0001, 1, 1 and 1 okay. So now how will you do Gaussian elimination?

If you do it without thinking about the problem you will just take this is a non 0 number okay, you will take this as a pivot and then proceed with making you know the matrix to be lower triangular.

(Refer Slide Time: 31:03)



$$\begin{bmatrix} 0.0001 & 1 \\ 0 & -9999 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -9998 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 \\ 0.0001 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$x_2 = 0.999899 \cong 1$$

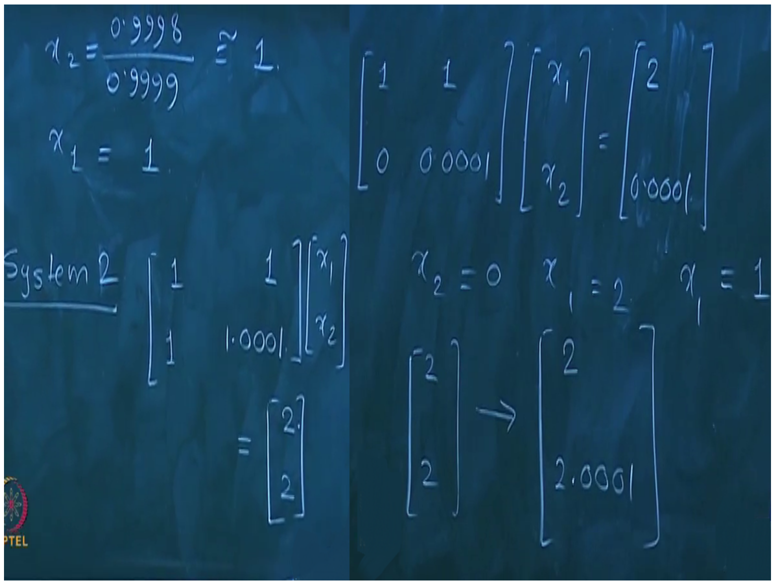
$$x_1 = 0$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 0.9999 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0.9998 \end{bmatrix}$$

So your next step would be 0.0001 and 1, 0, -9999, x1 x2, so this will give you x2=this, and then you will back substitute and get x1 right. So we round this off to 1, let us say you have a computer which has finite digit position so you round this off to 1 okay, 0.999 will get rounded off to 1, if this is rounded off to 1 what is the solution for x1. If this is rounded off to 1, what will happen x1 will be 0 okay, this is because of the rounding off error.

Now what is the exact what is the correct way of doing calculations here, well I will do maximal pivoting, you know what is maximal pivoting? In Gaussian elimination what you do you rearrange the rows such that the diagonal elements are maximum, so if I do maximal pivoting this will be 1, 1, 0.0001, 1, x1 x2=2, 1 my modified problem same system okay, I have done maximal pivoting I have moved this pivot here 1, now do Gaussian elimination okay. So now if I do Gaussian elimination here I will get 1, 1, 0, 0.9999 okay.

(Refer Slide Time: 33:32)



$x_2 = \frac{0.9998}{0.9999} \cong 1.$
$x_1 = 1.$
System 2
$\begin{bmatrix} 1 & 1 \\ 1 & 1.0001 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$
$\begin{bmatrix} 1 & 1 \\ 0 & 0.0001 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0.0001 \end{bmatrix}$
$x_2 = 0 \quad x_1 = 2 \quad x_1 = 1$
$\begin{bmatrix} 2 \\ 2 \end{bmatrix} \rightarrow \begin{bmatrix} 2 \\ 2.0001 \end{bmatrix}$

What is x2 right, what is x1 now? If x2=1 x1=1, you see the problem here same system of equations I re-order the calculation, once I got solution 0 and 1, here I got solution 1 and 1 rounding off okay, what mattered here this is the correct solution 1, 1 is the correct solution, what mattered was way I ordered my calculations. If I do wrong ordering my calculations, this particular matrix is not ill-conditioned matrix it is well-conditioned matrix okay.

Why this well-conditioned matrix will come to that little later, but right now just take it for granted, this is a well-conditioned matrix what went wrong was we did ordering of calculations, and okay we got a solution which is not acceptable but we did not recognize it of course that type, we thought everything is logical, we did rounding off there and then we got a solution which is. But here when you correct the order of calculations you recover the solution 1 1 and then okay.

Now let me take another system, now second system my system 2 is this 1, 1, 1, 1.0001, x1 x2=2, 2 okay. If I apply you know first step in the Gaussian elimination this will be 0, this will be 0.0001 right, I just did Gaussian elimination 2 times right, so I just subtracted this row from this row, this from this I got 0 on this side right. And then what is the solution x2=0 and x1=2 okay, is this okay? Okay.

What I am going to do now, I am going to say that well this was a slight error on representing the right hand side okay, now you would say well how does it matter if this slight 10 to the power-4 okay, I make an error of 10 to the power-4 on the right hand side, so instead of presenting 2 and 2 you know I ended up representing 2 and 2.0001 okay, instead of typing 2 and 2 let us say I typed 2 and 0.0001, you would expect what is there if these are these are 2 very close vectors, error is of the order of just see what I will happen.

If this was instead by mistake you know represented as 2.0001 this will become, what next if this happens my x1 will be=1 because 0.0001/0.0001 okay x1 will become 1, what will be x2? Or the other way around, suppose this was you know 1.9999, it will be x1 will be-1 x 2 will be 3. So a slight error on the right hand side slight 10 to the power-4 which you think you know looking at these numbers 10 to the power-4 we are working with 1, 1, 1, 2 and all that 10 to the power-4 error.

But such a small error is causing a drastic change in the solution, such a small error this will not happen for system 1 system, once you reorder the calculations if you perturb the right hand side slightly it is not going to change the solution too much okay. Here, if you have a solution which is if you have the right hand side which is slightly perturbed okay you have a problem okay, the solution changes drastically okay, so from 0 to 2 it changes to 1 to 1, it is not a small change right not small change.

So this problem what we will show a little later is that this is because whatever you try to do here, you rearrange the calculations you do by pivoting, well here there is no scope for pivoting both pivots are 1, 1, so there is no question of maximal pivoting. But here nothing is going to change this problem, there is a slight error in the representation on the right hand side you will get drastically different solution okay. So this problem is ill-conditioned no reordering is going to help you.

The first problem is not ill-conditioned; it is a problem of you know reordering the calculations to get the correct solution okay. Now I want measure that will separate these 2 situations, where I can say well this A matrix is ill-conditioned, whatever I do I will get into trouble. This A matrix is well-conditioned if my solutions are absurd I have made an error okay, which is not the case for the second case, second case solutions are absurd it is because there is a limitation. We will continue in the next lecture.