

Matrix Solvers
Prof. Somnath Roy
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture - 51
Numerical Issues in BICG and Polynomial Based Formulation

Hello. We have been discussing about Biconjugate Gradient Method for solving linear systems of equation. This is an iterative method and this method has emerged from the inspiration we got from conjugate gradient method that is a Krylov subspace method can be devised for symmetric matrices in case of conjugate gradient method which is a very first method and also very simple implementation null method.

Now, the problem with conjugate gradient method was that it is restricted only for symmetric matrices. So, we thought of developing a method which is for asymmetric matrices or for any general matrix and for that Lanczos's Biorthogonalization method has been developed which we which we have discussed in last few classes and in Lanczos's by orthogonal method, we start with the Krylov subspace method, a subspace of the matrix A as well as of the left matrix which is A^T .

And we show we have seen that how this method has been developed into Biconjugate gradient method which with suppose to be to give a convergence rate which is kind of similar as the conjugate gradient method.

Now, we have discussed up to the development of Biconjugate gradient method in our last class. As I said Biconjugate gradient method is a iterative solver, which is inspired from conjugate gradient method. However, this is applicable for any general matrix. It is not only strictly restricted to symmetric matrices as conjugate in methods where that.

So, if we look into the Biconjugate Gradient Algorithm which we are discussing in the last session that this is directly copied from a absurd's book.


(Refer Slide Time: 02:06)

Biconjugate Gradient (BCG) Algorithm – Computational Issues


ALGORITHM 7.3: Biconjugate Gradient (BCG)

1. Compute $r_0 := b - Ax_0$. Choose r_0^* such that $(r_0, r_0^*) \neq 0$.
2. Set, $p_0 := r_0, p_0^* := r_0^*$
3. For $j = 0, 1, \dots$, until convergence Do:
4. $\alpha_j := (r_j, r_j^*) / (Ap_j, p_j^*)$
5. $x_{j+1} := x_j + \alpha_j p_j$
6. $r_{j+1} := r_j - \alpha_j Ap_j$
7. $r_{j+1}^* := r_j^* - \alpha_j A^T p_j^*$
8. $\beta_j := (r_{j+1}, r_{j+1}^*) / (r_j, r_j^*)$
9. $p_{j+1} := r_{j+1} + \beta_j p_j$
10. $p_{j+1}^* := r_{j+1}^* + \beta_j p_j^*$
11. EndDo

Further, irregular convergence is observed due to rounding errors with increased computational operations



IIT KHARAGPUR



NPTEL ONLINE
CERTIFICATION COURSES

14

And we can see that this algorithm has a number of matrix vector multiplications. As we have discussed earlier, Matrix vector multiplications are the most costly operations in matrix solvers and if even if we can bring down the number of iterations; but if number of matrix vector multiplication in each iteration is a quite high, it will take a lot of time to give the results. So, we usually try to reduce the matrix vector multiplication.

However, if we look here that the first matrix vector multiplication is for getting the initial guess which is not required at the later iteration stage throughout. But the later matrix vector multiplication like this finding this Ap and finding this $A^T p$, they are done again and again whenever we go to one new level of iteration, we have to do this two matrix vector multiplications.

So, two matrix vector multiplications are associated here, which is if we think compared with SOR or Gauss Seidel type of method, it has only 1 matrix vector multiplication each in each iteration. So, number of mathematical operations in each iteration is high in one sense.

However, there are few other issues one important issue is that there is an $A^T p$ multiplication and we will discuss later. This multiplication is not only a matrix simple matrix vector multiplication, rather a matrix transpose vector multiplication.

So, given a matrix, I have to rewrite the code to do a matrix transpose vector multiplication because when we have seen the programming implementations or the code implementations, we have seen that there is a subroutine which does the matrix vector multiplication and this subroutine is called again and again when we need this multiplication.

However, that that particular subroutine will not work rather we have to write a different sub routine which is also not of that much problem. But the problem may be that when we try to access the matrix A, we first access row wise and then we are increasing column wise.

But when we will think of $A^T p$, the axis of different memory elements of a will be completely changed and in some of the computational framework, it creates significant amount of problems. There is another very serious problem here that if I look into the term if I look into the term α ; α is obtained through $r^T (A^T p - r)$.

r^T is the residue in the left space or a transpose Krylov subspace and r is the residual of x is equal to p divided by a dot product between $A^T p$ and p^T ; p^T is again auxiliary vector in the left Krylov subspace or Krylov subspace of A^T r_0 . Now, $A^T p$ this is one matrix vector multiplication I am doing in $A^T p$ which is used to calculate α ; when we have calculated calculating r_{j+1} , we are doing r_{j+1}^T which is the new residual in the A^T Krylov subspace.

We are you doing at this multiplication $A^T p^T$ which as I said its difficult multiplication matrix transpose vector multiplication and multiplying it with α ; what is the implication? Implication is that we have earlier discussed about round off error. I do some numerical calculation; I multiply the 3.3333; that means, say 10 by 3 into 1 by 7. I really do not get it as 10 by 21 rather I get a numerical value and this numerical value is not something like an infinitely long decimal; after certain decimals places this value is truncated and we accrue some round off error.

So, one round-off error is accrued here because matrix vector multiplications has lot of multiplications and additions and so, the round off errors will be build up to some substantial value. Otherwise round off errors for one single operation, it is very small; but when we do thousands of operation, they add up and there is substrate some those small some value added up here.

Now, we are getting some we are encountering some round off error in this step. Alpha is carrying this round off error and when we are coming into the r_{j+1} step, this round off error is again being multiplied with the round off error or again being subtracted to the round off error through our multiplication which is obtained by multiplication of A^T into p_j .

So, the error is actually building up and then, we are dividing r_j by doing this calculation again taking dot product dividing it all these divisions multiplications or adding up round off error. So, there is some conjugacy between the round off errors which is building up and that is a serious issue in Biconjugate gradient algorithm.

As a matter of fact this is from my personal experience if I try to write a code on this and run it this will not run in a single precision machine or with single precision variables. I have to define all the variables to be double precision so that I can give I can reduce the round off error to a much smaller value and then, only this particular algorithm will work.

However, there can be cases that even with double precision this algorithm is failing for certain type of matrices and that is the main concern especially that we should will devote this particular lecture on seeing that what are the issues of the round off error, issues that the round off error is creating and also what are the remedies for it. So, the issues that irregular convergence is observed due to round off rounding error with increased computational operations. Another important thing is that convergence is not very simply defined for Biconjugate gradient algorithm.

We can understand that this algorithm comes from approximating the basis vectors in two Krylov subspaces; one is Krylov subspaces of A^T , r_0 ; r_0 is initial residual; another is the Krylov subspace of A r_0 and we try to approximate the solution in both the subspaces and generate more basis vectors and finally, converge to the right result.

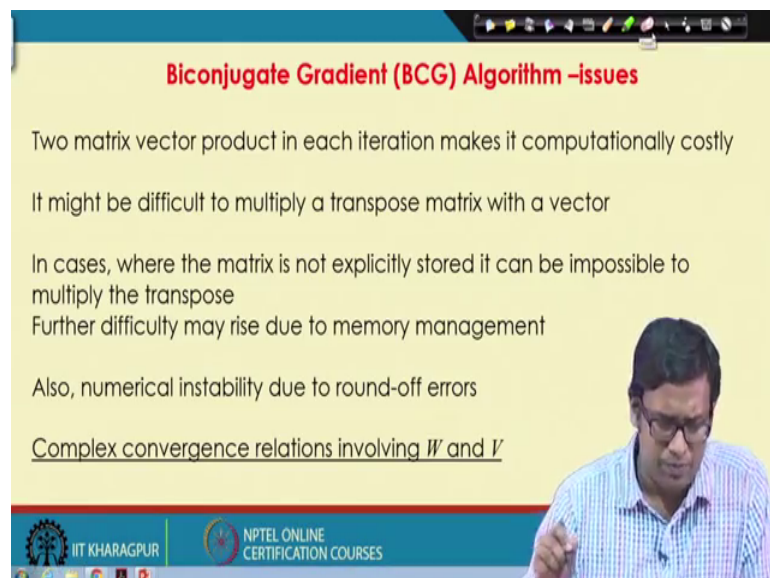
What happens in this stages that that error is some way a function of how the eigen vectors of both the subspaces are well approximated and we see that, this the residual in one subspace depends on the eigenvectors of the or the basis functions we obtained in the in the other subspace.

So, we can see that that none of the all the parameters like all the residual parameters are dependent on the other subspace in the algorithm. So, what happens that we cannot write a strict convergence criteria, that error is always less than the error of or residual is always less than the residual of the previous iteration for this case. Then, we will see that the error some oscillation of the iterations and when the residual value is quite high, the convergence is good and when the residual value reduces to a small number, the convergence slows down.

This things make this methods are little more complex, at least the results of this method to be represented in which looks little more complex and we see that there is an oscillation in the residual when it is converging to a small value and if there is a round off error, if there are high round off error, this oscillation some time does not converge to a right result and sometimes this round off error builds up and those the entire calculation, it does not converge or it converges to a wrong result.

So, this is this is typically identified as numerical instability. We call this to be a numerical instability of this particular method Biconjugates Gradient method. So, we need to find out the remedies how to take care of that.

(Refer Slide Time: 11:27)



Biconjugate Gradient (BCG) Algorithm -issues

- Two matrix vector product in each iteration makes it computationally costly
- It might be difficult to multiply a transpose matrix with a vector
- In cases, where the matrix is not explicitly stored it can be impossible to multiply the transpose
- Further difficulty may rise due to memory management
- Also, numerical instability due to round-off errors
- Complex convergence relations involving W and V

The slide also features a video inset of a man speaking in the bottom right corner and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

So, the issues are that two matrix vector product in each iteration makes it computationally costly. Also it might be difficult to multiply a transpose matrix with a

vector. We are deliberating on this point in the last slide that when we do the multiplication say we are writing say A into b .

So, each element in a row of A is multiplied with corresponding element in the column vector b . This is multiplied with this; this is multiplied with this and they are added. So, if we think of a computer, how is it doing that? It is going into the memories in these memory space computer stores everything in its ram and it starts with a pointer which moves around the stored values.

So, it goes into a particular memory location and takes this value and goes into the memory location of b and takes this value and then, the control goes to the next memory location also both in the row of a as well as in the b vector. So, memory access is kind of a in a contiguous manner a contiguous system of memory is being accessed (Refer Time: 12:54).

Specially, when you think of using high performance computing architectures like GPU cards, this is important that how this memory is being accessed because memory to computing code bandwidth is very slow there. Now, if I have to do A transpose b instead of multiplying b with the row elements of A , I have to go column wise.

However, the storage algorithm that I write for A or the way I try to I have my computer access the memory of A is based on doing $A b$ multiplication because $A b$ matrix vector multiplication what do we do, matrix-matrix addition what do we do or matrix vector multiplication what we do regularly.

So, there is a different nature of memory access which will be required, which can be difficult as well as much complex from computational aspect. It can kill down the performance. Another thing is possible that sometimes, we do not form the A matrix explicitly, there are cases when you are dealing with very large system of equation.

We do not store the matrix, we go to each row and see what are the neighboring values if we can recall how the matrix is formed from finite difference or finite volume or finite element method, we go to one element and see all that neighboring values and then do the required multiplication with the variables given. In that case A is not formed explicitly and getting A transpose is also extremely difficult.

Sometimes it is impossible because we are not forming A explicitly. We are somewhere getting the products $A b$, but that is through the equations through the through each linear equation with not forming the matrix a transpose. So, this gives certain over rate and sometimes it becomes impossible to handle the systems. In cases when the matrix is not explicitly stored, it can be impossible to multiply the transpose.

Further, difficulty may rise due to memory management and with is discussing this and the important thing is that when we discuss about GPU computing or doing high performance computing and using good memory coalescence, writing efficient kernels for GPU computing which needs lot of algorithmic to like considering how the memory is being accessed and how data in terms of memory; how data is shared by different computing course and things are going in parallel.

There this transpose vector multiplication can really be a limiting step. Also the numerical instability that is due to accrued round off errors, numerical instability can kill the final algorithm oh and furthermore, there is one issue which is complex convergence relations involving W and V involving both the vectors Krylov subspaces.

Because it is not restricted to only one Krylov subspaces, there is another Krylov subspaces which is important here and therefore, the rate of fall of residue is in a complex manner and we see an oscillatory pattern here. I will show you at the end of this and the next lecture, I will show you some of the results using Biconjugate Gradient method and we will see that the residue does not monocratically drop down to 0; rather called oscillator due to the complex convergence vector.

So, considering all this that this is an oscillatory method on top of that there is some numerical oscillation. The entire method might give us some wrong result. Also there can be implementational issues due to matrix transpose multiplication. So, two things have to be seen; one is that that how we can avoid the matrix transpose matrix into vector multiplication which is very important in terms of the actual performance of the solver because we are only discussing till now, we have only discussed about how many iterations it will need and what are the computational costs etcetera.

But we have not discussed which is little more computer science aspect, importance is there in computer science aspect that is the performance of the solver in terms of memory handling; in terms of the fact that how the processors are getting data from ram

and how efficiently it is using the cache. And also the high performance computing or GPU computing aspect, memory handling is important and when you doing a transpose matrix and vector multiplication, this memory handling is very complex and it might stop the entire process.

Also as the algorithm is developed, sometime we can very easily see that Gauss in a Gauss Seidel Iteration method, we may not write it in the matrix form, we can take each iteration and try to find out the updated value from guess value. Right, you think of a Gauss Seidel Iteration, you may not write the entire matrix form; you just write the equations and use the equations for the iteration for getting the updated value based on guess value. So, there can be cases when you are not explicitly forming the matrix rather we have sets of equations.

Because explicitly formation explicit formation of the matrix also need a data structure how the neighboring; what are the relations of neighboring nodes in terms of i, j, k rectangular element which is not present. In certain case, there finding a transpose is impossible. So, there are certain cases for which Biconjugate gradient, though it has tremendous potential in terms of the due to the facts that it is applicable for any general matrix and it is a Krylov subspace based method which can very first give us the solutions. Though this potentials are already there for Biconjugate gradient method, it can have this few problems. One is that implementation wise there can be a problem due to $A^T b$.

There can be a problem in terms of the code performance due to memory management of $A^T b$ and also as the round off errors are increasing as well as the method has a complex convergence pattern or oscillatory convergence pattern, there can be another problem. And why the round off error is increasing? Because matrix vector products are giving us round off error as large number of multiplications and additions are involved there and one matrix vector product is further being utilized to get and being multiplied with another matrix vector product to get that r_j plus 1 star.

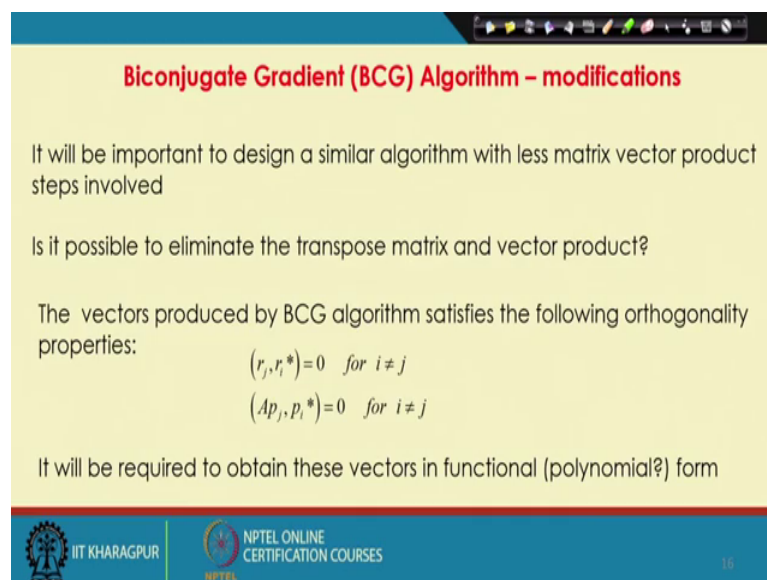
So, matrix vector products are in a way calculate. So, that they it round off errors can further achieve up and we can have lot of numerical errors, numerical oscillations along with the, it inherent oscillation of the scheme when the values when the residual value is

coming down. So, where residual value is very small of 10 to the power of minus 8 and I am getting round off error of the same order. So, 10 to the power minus 9 again.

So, it is it will be difficult to get the method converged because when the round off error is when the when the residual is falling down, the cutoff value or the epsilon the round off error is substantial. So, it does not allow residual to find out and it might converge to a different error also.

So, all these issues are there. So, to see that how we can get ahead with, we can get away with the transpose matrix and vector multiplication part. As well as how can we reduce the round off errors so that one matrix vector product is decoupled from another matrix vector product, they are never multiplied together (Refer Time: 21:05).

(Refer Slide Time: 21:09)



Biconjugate Gradient (BCG) Algorithm - modifications

It will be important to design a similar algorithm with less matrix vector product steps involved

Is it possible to eliminate the transpose matrix and vector product?

The vectors produced by BCG algorithm satisfies the following orthogonality properties:

$$(r_i, r_j^*) = 0 \quad \text{for } i \neq j$$
$$(Ap_i, p_j^*) = 0 \quad \text{for } i \neq j$$

It will be required to obtain these vectors in functional (polynomial?) form

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, it will be important to design similar algorithm similar to Biconjugate Gradient or BCG algorithm with less matrix vector product steps involved. It is also is it possible the question is it possible to eliminate the transpose matrix and vector product? This is one of the limiting step implementation wise; some cases some cases memory wise or the performance wise, it can we eliminate this step.

The whatever we do the finally, the vectors produced by Biconjugate gradient method must satisfy the following 2 orthogonality properties that is $r_j \cdot r_i^*$ is the

residual of Ax is equal to b and r_i^* is the residual of A^T . x is equal to b^* something is if they are orthogonal to each other if i is not equal to j , they are 0..

Similarly, the auxiliary vectors p of a , a and A^T and a are orthogonal; that means, $A^T p \cdot p^*$ is equal to 0 for i is not equal to j . If worked on these properties, explicitly you have shown how to derive these properties and this is the basis of Biconjugate gradient method that these two conjugate properties is to orthogonal probability must hold.

So, can we get these two properties even without going into so many matrix vector products as well as avoiding the transpose matrix vector product. So, will it will be required to obtain these vectors r_i and r_j in functional form, where instead of doing A^T we can probably have a have some functional of form of r and p is a polynomial form of r and p so that the dot product doesn't mean necessarily inverse transpose of a matrix.

So, the idea will be that instead of forming the vectors as simple column vector will try to express these vectors in a polynomial form and we will try to take the dot product and other operations in the polynomials only. So, that this polynomials are never explicitly formed, but only the polynomials are used as certain basis to create these vectors so that, we can avoid from the computational complexities of Biconjugate gradient method.

(Refer Slide Time: 23:47)

Transpose free variations-- formulation

Designed to avoid matrix transposes in BCG!

This method starts with assuming a polynomial form of the residual vector:

$$r_j = \phi_j(A)r_0$$

i.e., the j -th residual vector is obtained by multiplying a j -th order polynomial of matrix A with initial residual, obtained from guess value.

This polynomial is defined as: $\phi_j(0) = 1$ Or, $\phi_j(\{0\}) = I$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

17

So, what we look for is A transpose p variations of this method. Transpose p variation means as it says clearly that this is designed to avoid matrix transposes in which are in the Biconjugate gradient, usual Biconjugate. So, this method starts with assuming a polynomial form of the residual vector and is I start with r is equal to the initial residual vector r_0 based on the guess value and that any iteration level the new residual vector r_j is r_0 multiplied with a polynomial of the width of the matrix A ϕ_j into $A r_0$.

So, it might look a little difficult to appreciate how a polynomial of a matrix will look like? A polynomial of a matrix, a polynomial a scalar will give us a scalar a polynomial of a matrix will give us a matrix. So, finally, it is a matrix multiplied with r_0 that is the j th residual vector is obtained by multiplying a j th order polynomial of matrix A ; ϕ_j is a j th order polynomial with initial ratio obtained from the guess value. This polynomial is defined as ϕ_j , if I instead of A like this polynomial can this is an operator this can operate on a scalar as well as it can operate on a vector or a or a matrix.

So, if I write say ϕ^2 is equal to ϕ^2 of t is a scalar; $\phi^2 t$ is equal to $a t^2$ plus $b t$ plus c . So, ϕ^2 of the matrix A will be A into A square which is A into A plus $b A$ plus c instead of. So, there is a there is some matrix C some matrix C . So, here it is a scalar c and it will be a matrix C and that there is a exactly that ϕ of 0 is a scalar 1 . But if I put a 0 matrix instead of ϕ , the polynomial will give us a matrix which is an identity matrix.

So, $\phi_0 r$ is r is A is equal to 0 ; that means, all the elements of $A r_0$, then r_j is equal to $\phi_j r_0$ and this is identity matrix i . So, r_j is equal to r_0 . So, this is the same operations which we are supposed to do over a scalar, we are now doing it over a matrix.

(Refer Slide Time: 26:54)

Transpose Free Variations of BCG

Similarly, polynomial form can be assumed for auxiliary vector

$$p_j = \pi_j(A)r_0$$

Also, in the W -space the vectors can be assumed of similar form:

$$r_j^* = \phi_j(A^T)r_0 \quad p_j^* = \pi_j(A^T)r_0$$

Now, in a BCG algorithm, parameters α_j and β_j are defined as:

$$\alpha_j := (r_j, r_j^*) / (Ap_j, p_j^*) \quad \beta_j := (r_{j+1}, r_{j+1}^*) / (r_j, r_j^*)$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, now, what is done is that that similarly polynomial can be also assumed earlier we are assuming a polynomial form for the residue vector. Similarly, we can assume a polynomial form for the auxiliary vector also. Also in the W space or the A transpose space, vectors can be assumed of similar form that r_j^* is $\phi_j(A^T)r_0$ and p_j^* is $\pi_j(A^T)r_0$ and ϕ_j and π_j are two polynomials two different polynomial functions and when they are operated over A , they give us $\phi_j(A)r_0$ gives us r_j and $\pi_j(A^T)r_0$ gives us the residual in the left space or the A transpose Krylov subspace which is r_j^* . Similarly, for the auxiliary vector we get the polynomials.

Now, in a typical BCG algorithm α_j and β_j are defined as α_j is $(r_j, r_j^*) / (Ap_j, p_j^*)$ and β_j is $(r_{j+1}, r_{j+1}^*) / (r_j, r_j^*)$. Similarly, β_j is $(r_{j+1}, r_{j+1}^*) / (r_j, r_j^*)$ and what we will do is that we will try to replace these by the polynomials expression of the residual and the auxiliary vectors and we will see that if in some way we can eliminate some of the matrix vector multiplications mostly we will try to eliminate the matrix transpose vector multiplication through this. We will see it in the next class.

Thank you.