Matrix Solvers

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

Lecture - 42
Developing Computer Programs for Projection Based Methods

Welcome. We are continuing our discussion on Computer Programming for the iterative solvers. In the last session we have looked into Jacobi solvers and we will see how this can be modified into Gauss-Seidel solvers. The examples will be given on similar problem which you are considering in during Jacobi solvers, there is square block in which we have to solve Laplacian 2-D Laplacian equation and all the boundary conditions are specified as the essential boundary conditions there.

(Refer Slide Time: 00:49)

So, we look into the Gauss-Seidel iterative method. The matrix equation is very similar as the Jacobi method except look into at x, we have to solve matrix Ax is equal to B. A is diagonally dominant.

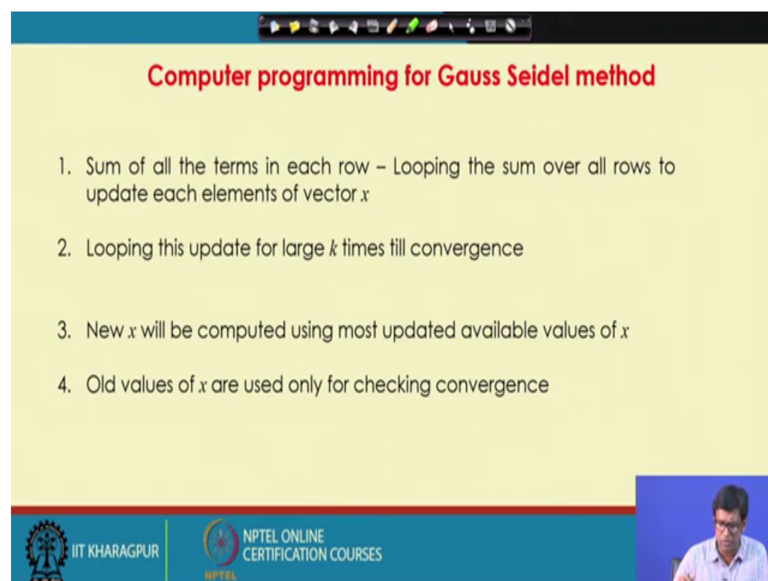
You have to do the exactly same thing that start with the gauss value x is equal to x 0 and then, update Ax; but this update instead of what we have done in Jacobi, what will presently do in Gauss-Seidel is that for the upper row terms, for the terms for that which you already have an updated value of x available, we use the updated values. And for the terms in the lower rows, where the x is till the guess value that has not been updated, we

use the guess values or what we can see say that for getting updated value of one particular element of the x matrix, we use the most recent values of all other x's.

So, in a sense programming will be actually easy because we do not have to use the old value at during the iterations. It is will update the one variable and that will be used by in subsequent iterations. So, we will look into it. I will show you couple of examples (Refer Time: 02:14) c code which will discuss right now and then, we will see a Fortran Code, where much simpler implementation has been done.

Well. So, x is updated like that and then, till convergence we have to see that the maximum value of xk minus xk plus 1, the difference of any element of x in 2 iteration levels and the maximum of that difference, maximum the absolute value of the difference is less than the small number epsilon. In a sense, it is very same as very much similar to Jacobi method.

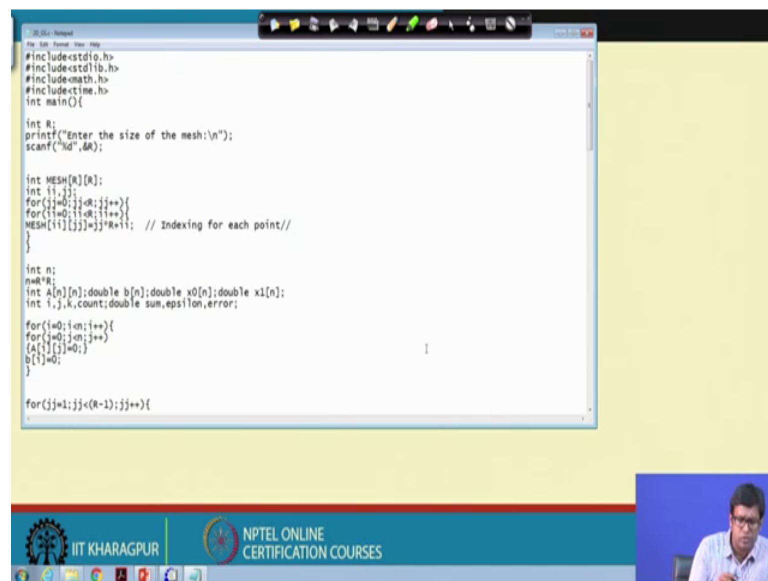
(Refer Slide Time: 02:53)



And for programming also we have to sum of find sum of all terms in each row and loop over sum of all rows to update each element of vector x. So, there is a lot of summation operation, summation in each row for finding that particular expression; summation on each row for finding this particular expression, this summation. As well as there is this is summed over one particular i and similarly for all i's for all the rows we have to do this summation.

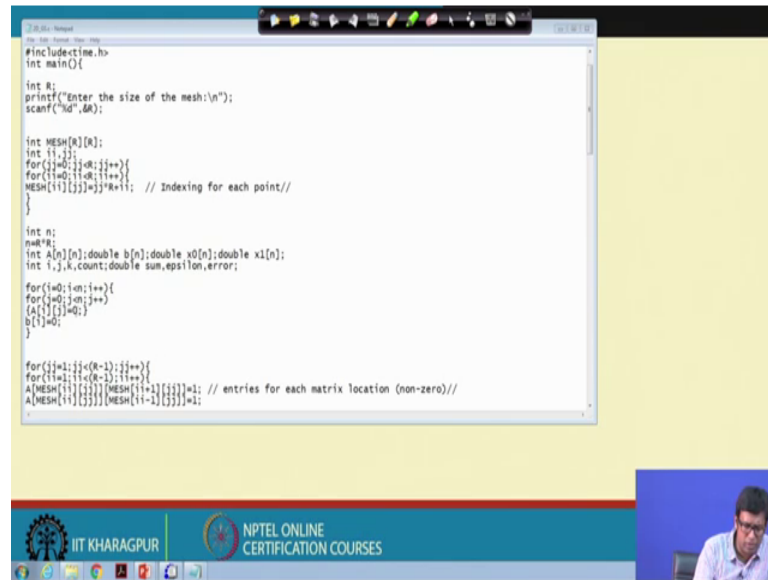
So, there are basically 2 rows of summations here and then, this has to be looked for a large value of k till we get convergence and the new x which is computed will be will use for computing of new in this new x, we will use the updated available values of x. However, the older x has to be stored because we have to check the difference from the older value of x and then, the new value of x and what is the maximum of that absolute difference for checking the convergence.

(Refer Slide Time: 04:01)



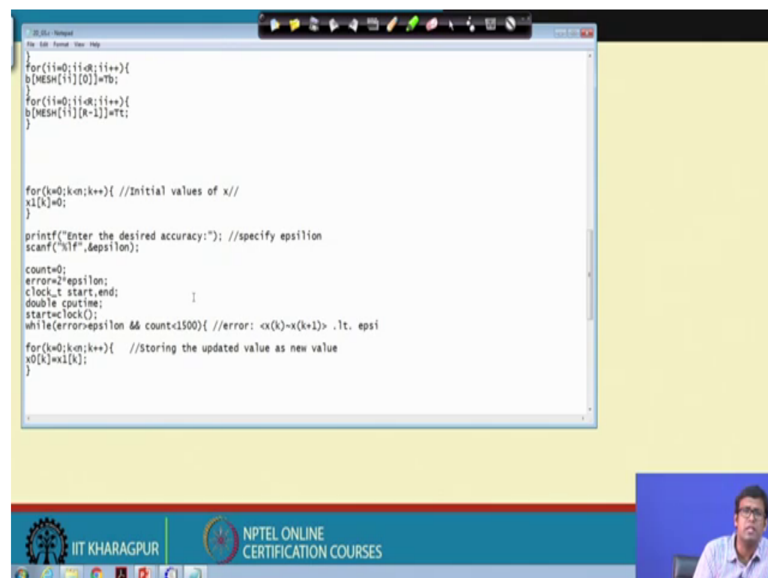
So, we will quickly look into a Gauss-Seidel code. So, it starts with all including all the header and time files and then, there is a mesh which is which basically gives pointer of ij value in a Cartesian mesh to one particular row of the matrix equation. And now, what we are doing here?

(Refer Slide Time: 04:26)



We are, so, what we are we have to here is that write down the initialize everything with the 0 and then, put the right boundary condition define the right coefficient of matrix A, it is inter diagonal matrix with the diagonal term minus 4 and all of diagonal 0 except few terms which are 1 and substitute the values of A for the boundary conditions and then, put the boundary conditions in the B matrix and so we get A and B fixed here.

(Refer Slide Time: 05:07)

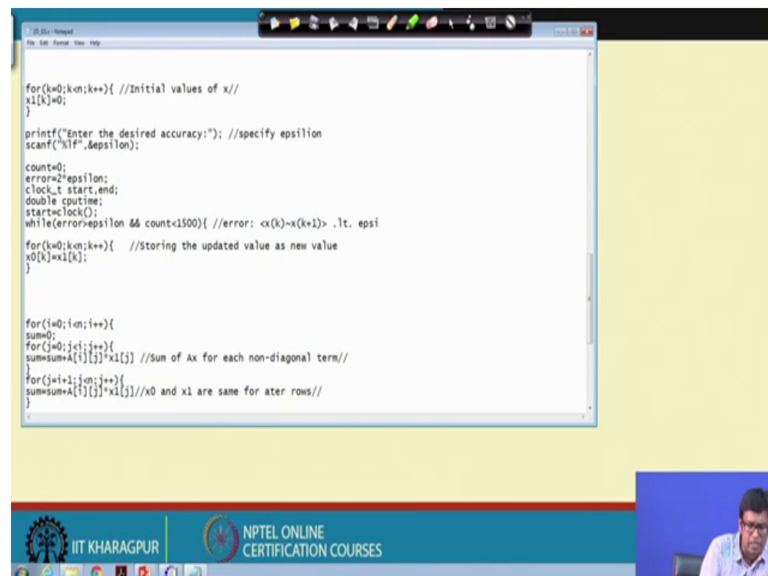


Then, we start with we have to give the desired value of accuracy which is epsilon that the user gives a gives it as a input and the program reads this value. The initial value is x

is equal to 0 and with x is equal to all the x’s are 0, x 0. So, x 0 which is given as x 1, the guess value is defined as x 1 and sorry the updated value is defined as x 1 and guess value is x 0 but we are initializing updated value x 1 is equal to 0.

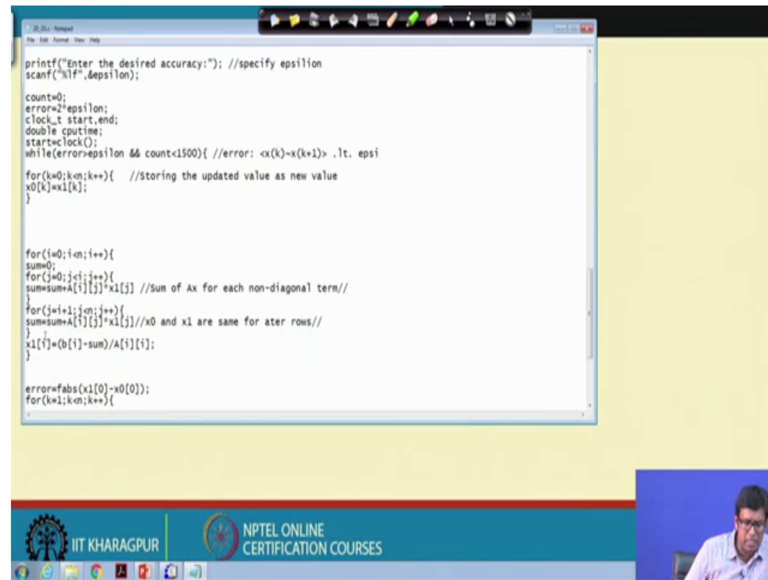
Later, we will change updated we assign the updated value to the guess value. So, the initial gauss value become 0 and till we have an error which is greater than epsilon, initially, we have given error defined error to be twice of epsilon and during calculation we will find out a error, till error is greater than epsilon and till we have less than 1500 iterations. We follow these loops.

(Refer Slide Time: 06:08)



So and there is there are some comments for checking the computational time required for that. So, what you are doing here is that we are doing a sum for each sum of Ax for each non diagonal term until j is less than i. So, for all the non diagonal terms above the particular row is summed and this is total sum this is for one particular value of i. So, there is a loop over j; j is equal to i; j is equal to 0; j is less than i and there is another loop j is equal to i to j is less than n.

(Refer Slide Time: 06:53)



So, this is sum in a particular sum for a particular row and this sum is done for all the rows and we can check that when doing this sum reducing x 1; x 1 is the updated most updated value. For the rows above this one particular row x 1 is the updated value during this iteration. For the rows below this particular row x 1 is the value which has been updated in the last iterations which has gone as the guess value now.

So, when you calculate updating one particular element of x as x 1, we are using all the recent, last recent available values or last updated values as the guess values. We are not using the guess values, which is the updated values in the last iterations. Whatever has been updated during this iteration is all being used as guess value and we are we calculate B is equal to x 1 i is equal to b minus sum by A i i.

(Refer Slide Time: 07:47)

```
clock_t start,end;
double cputime;
start=clock();
while(error>epsilon && count<1500){ //error: <x(k)-x(k+1)> .lt. epsi

for(k=0;k<n;k++){   //Storing the updated value as new value
x0[k]=x1[k];
}



for(i=0;i<n;i++){
sum=0;
for(j=0;j<i;j++){
sum=sum+A[i][j]*x1[j] //Sum of Ax for each non-diagonal term//
}
for(j=i+1;j<n;j++){
sum=sum+A[i][j]*x1[j]//x0 and x1 are same for ater rows//
}
x1[i]=(b[i]-sum)/A[i][i];
}


error=fabs(x1[0]-x0[0]);
for(k=1;k<n;k++){
if(error<fabs(x1[k]-x0[k])){error=fabs(x1[k]-x0[k]);}// Calculation of error
}
count++;

}
end=clock();
```

IIT KHARAGPUR   NPTEL ONLINE CERTIFICATION COURSES

So, this is done for one particular x 1 and similar and then, we calculate what is error what is the maximum value of x 1 minus x 0 for all the elements and check that check we will check what is the value of error and well like error is greater than epsilon, this loop will continue that means, count will be added by plus 1 and the control will be shifted here.

So, for the other values of k, for the other values like for the higher values of iteration count is less than 1500 this will continuing iterating till it gives epsilon error is less than epsilon. Error is less than epsilon is defined as something say 10 to the power minus 8, we get error is less than 10 to the power minus 8.

The only difference is if we look into Jacobi, there we have used x 0, but in Gauss-Seidel we are using x 1 which is the most updated value. So, it will also be apparent that as we using updated value using the iterations, the convergence will be faster and we have looked into convergence rate, convergence factor of the matrices and observed that Gauss-Seidel has the faster convergence rate then Jacobi will demonstrate it via using these codes also.

(Refer Slide Time: 09:17)

Successive Over Relaxation

Matrix equation: $Ax=b$

A is diagonally dominant

i-th row of the matrix represents the equation: $\sum_{j=1}^{n} a_{ij}x_j = b_i$

Now, let us assume a guess solution : $x=x^{(0)}$

For k=0,1,2.... Update x as $x_i^{(k+1)*} = \frac{b_i - \sum_{j<i}^{n} a_{ij}x_j^{(k+1)} - \sum_{j>i}^{n} a_{ij}x_j^{(k)}}{a_{ii}}$; $x_i^{(k+1)} = x_i^{(k)} + w\left(x_i^{(k+1)*} - x_i^{(k)}\right)$

>1 for SOR

Till convergence: $\max_i \left|x_i^{(k)} - x_i^{(k+1)}\right| < \varepsilon$

sor code

So, the next method is what is called the successive over relaxation method. In successive over relaxation method, everything is same except the x variable is not updated during the iteration directly as a Gauss-Seidel; rather we get a semi iterate semi guess semi level guess value at using Gauss-Seidel which is x k plus 1 star which is not the exact iteration value, but it is a semi iterated value and will see that the difference between x k plus 1 minus the old x k. So, this is difference between x k plus 1; this is x k plus 1 star.

This x k plus 1 star minus x k; what is this difference and we multiply this difference by omega which is greater than 1 for SOR. For successive over relaxation, the value of omega is greater than 1. So, increase the if x k plus 1 star is the updated value, we increase the iteration value to x k plus 1 star, from x k plus 1 star to x k plus 1 by multiplying it with the value greater than 1.

(Refer Slide Time: 10:43)

So, the marching towards convergence is faster and the code is also very similar exactly similar steps are followed until we do the updates step.

(Refer Slide Time: 10:45)

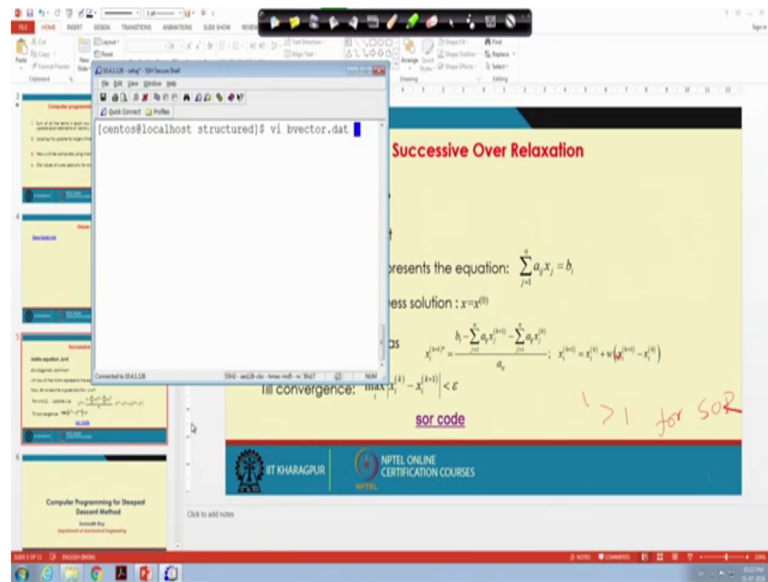
(Refer Slide Time: 10:47)

(Refer Slide Time: 10:49)

In the update step, we use the fact that the updated x is 1 minus omega into old x plus omega into the changes into b i minus sum by A i i.

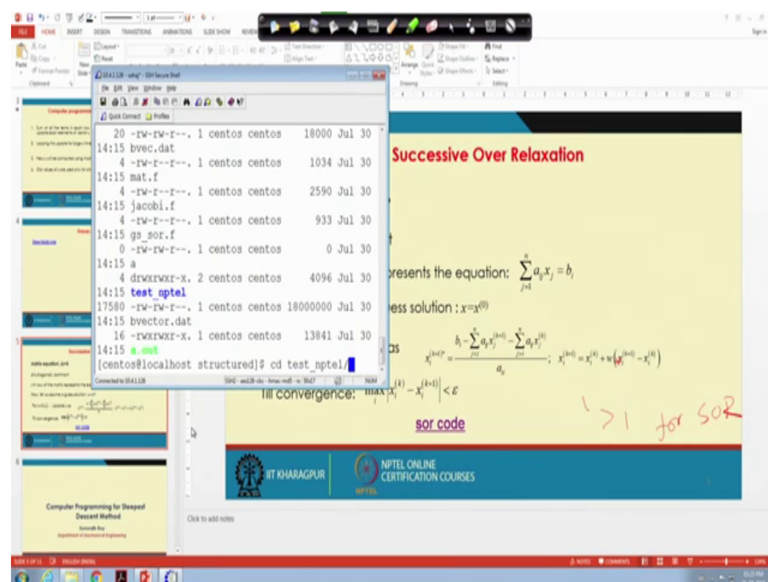
So, this update statement is different apart from that the entire code is similar. So, what we can do? We can connect to a server and see how these codes are behaving for one at least for one particular problem and then, will move to the Steepest Descent type of methods.

(Refer Slide Time: 11:33)



So, here I am connected to a server which has C and FORTRAN compilers, I will show some examples where FORTRAN code and this is my b vector. I am solving x is equal to b. This is my b vector; sorry.

(Refer Slide Time: 11:51)

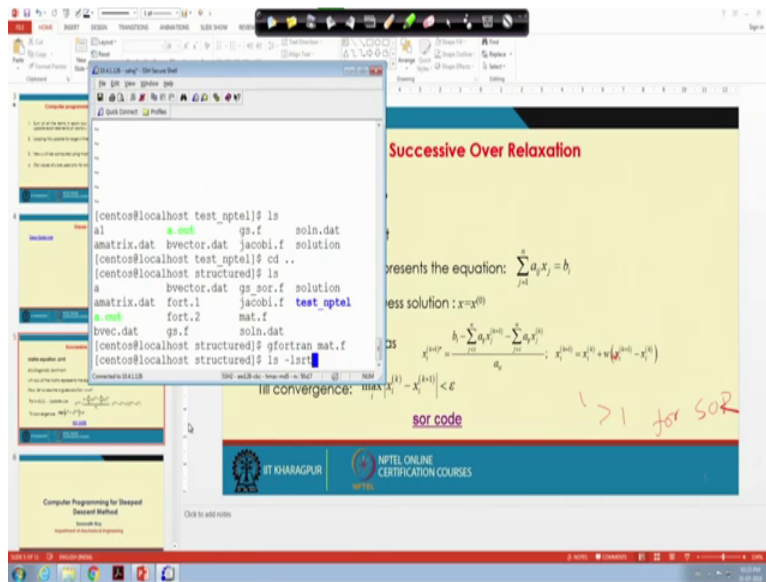


(Refer Slide Time: 12:05)

(Refer Slide Time: 12:11)

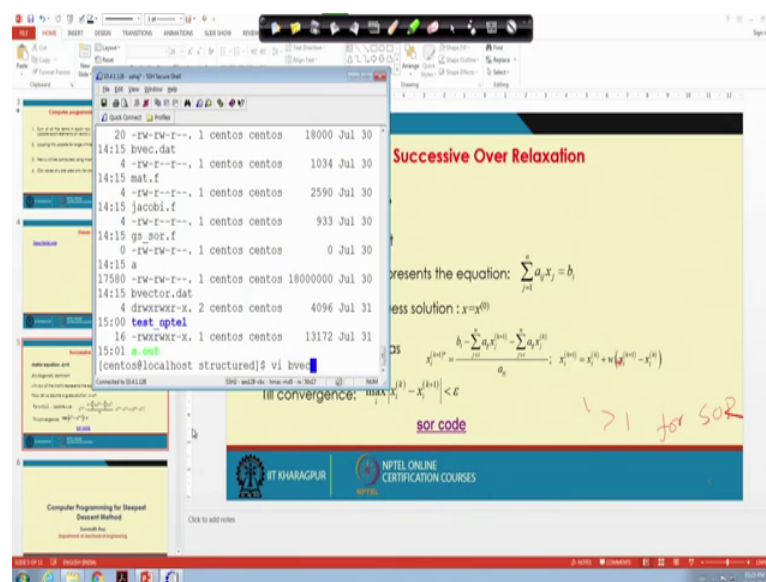
So, b i this is my what happened? (Refer Time: 12:18).

(Refer Slide Time: 12:15)



So, I have a small matrix generation code which I will run to generate the matrix and this code generates both a, a for x is equal to b; both a and b for a this is for a 3-D Laplacian equation.

(Refer Slide Time: 12:45)

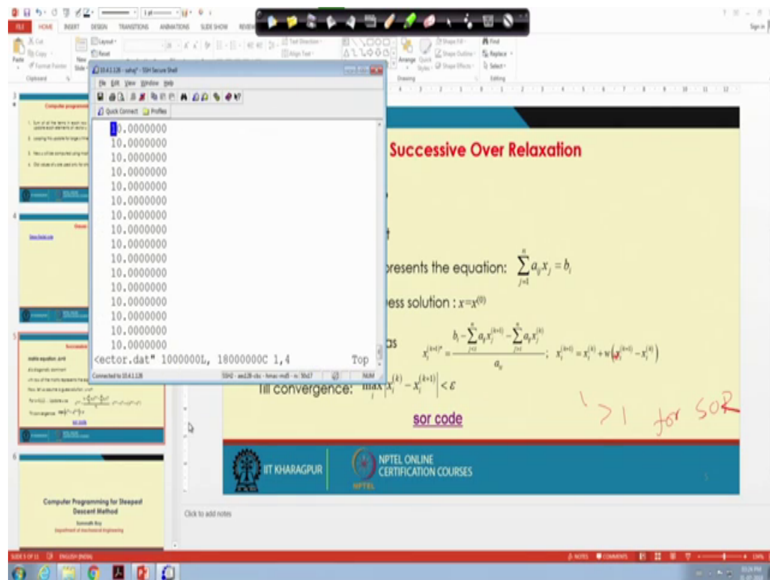


So, if I look into my b vec; I will look into the b vector.

(Refer Slide Time: 12:51)

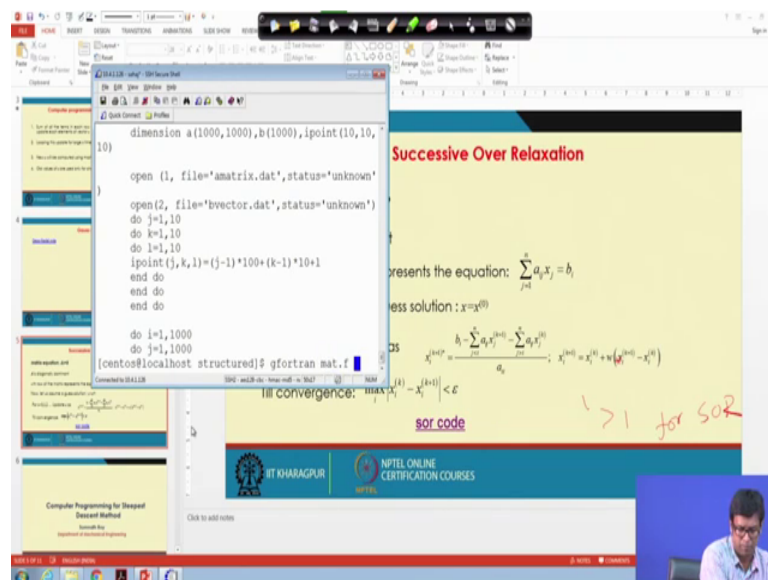
(Refer Slide Time: 12:59)

(Refer Slide Time: 13:07)



This has basically 100 1000 lines. Why should it be vi a matrix just one second.

(Refer Slide Time: 13:43)



(Refer Slide Time: 13:51)

So, b vector should have 100 points or 1000 points only gfortran.

(Refer Slide Time: 14:07)

Wait will have to go to some.

(Refer Slide Time: 14:31)

So, I will take up b vector as well as a matrix which has g. So, b vector has 1353 columns; that means, the matrix is 1353 into 1353 matrix and the b is the 1353 into 1 vector.

(Refer Slide Time: 15:05)

So, if I compile Jacobi and run it, the dimension of the matrix is 1353, it is a square matrix. So, I will giving 1 as the dimensional input.

(Refer Slide Time: 15:23)

And we can see that the iteration number 333 334, the error is reducing as a iteration number is increasing and this will go for a large number of iterations we can find it out more than few 1000 iterations we will go for.

So, here I am not given that the iteration count cannot be greater than 15000; 1500 because it is a large program and it will take lot more iteration to convert, right. Then, takes the large number of iterations as evident and the value reduces and finally, it will reduce below a small number.

(Refer Slide Time: 16:39)

So, it came to convergence when the error is less than 10 to the power minus 9 here and it took 3287 iterations.

(Refer Slide Time: 16:49)

So, if I look into the Jacobi code, I have specified epsilon. I am not(Refer Time: 16:55) the epsilon from the user, but it is specified as 1 e 10 to the power the epsilon is specified as 1 e 10 to the power minus 8. So, if the value is less than 10 to the power minus 8, the iterations will be done.

(Refer Slide Time: 17:15)

And now we do the same thing using a Gauss-Seidel and the iteration level is same. So, if I run the same code and this was 3000, more than 3000 steps.

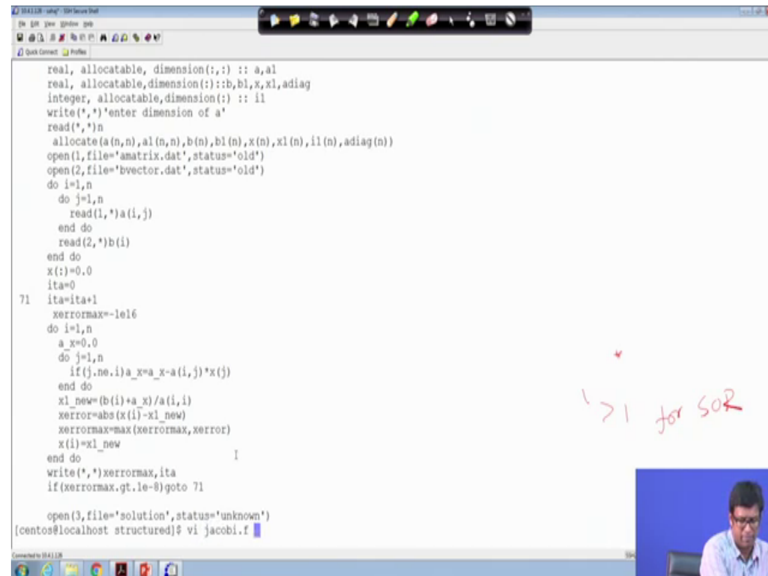
(Refer Slide Time: 17:29)

If I run the small problem using Gauss-Seidel, again 1353 at the number of mesh points.

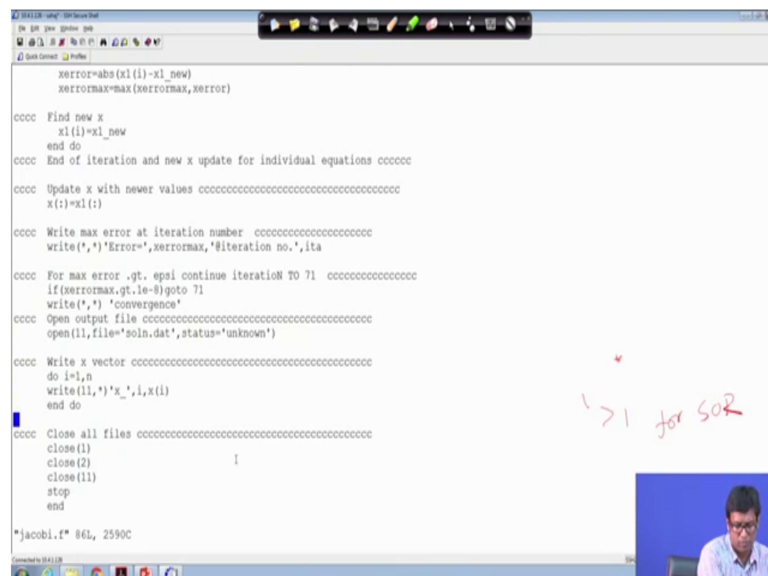
(Refer Slide Time: 17:33).

And let us see how many iterations does it takes. So, it took 1640 iteration have the number of iterations. The only change in Gauss-Seidel code is that when we are doing this calculations of multiplying a with x, I am not using the older value of x, I am using the most updated value of x. This is the most updated value of x here.

(Refer Slide Time: 18:35)



(Refer Slide Time: 18:39)



But if I look into the Jacobi, when I am using 2 x's; one is the old x and other is the new x.

(Refer Slide Time: 18:47)

And when I am updating x, I am finding out the new x based on the older value of x. I am not using the most updated values of x ok.

(Refer Slide Time: 19:03)

The Jacobi the Gauss-Seidel this is the update line in Gauss-Seidel and this is how it is being updated x, finding out an x 1 new and b minus x by A i i.

(Refer Slide Time: 19:29)

So, the same thing can be looked into an SOR, remember that it took around 1640 iterations for Jacobi; the same thing can be looked into an SOR.

(Refer Slide Time: 19:41)

And where there is a parameter alpha, x is equal to x old x plus new x minus old x into alpha. So, alpha represents omega here and alpha we read from the, we has the SOR value we read from the user. So, if we write the, if we run this code alpha may be we give 1.3 as alpha.

(Refer Slide Time: 20:13)

And see how does it convergence. So, it converges at 927 iterations where the j Gauss-Seidel took 1640 iterations and Jacobi took 3400 iterations much larger number of iterations. Now, if we know that there should be an optimum omega on which or optimum SOR factor on which we should get the maximum convergence rate. So, at 1.1, we are getting 9; 1.3 we got 9.27.

So, let us give alpha is equal to 1.6 and see if it still improves.

(Refer Slide Time: 21:09)

Now, we can see the radius run much more than 97 iterations and the values are actually if I close it, the value is actually not reducing it. It has become a constant. So, the solution is actually not converging with 1.6. So, you have run beyond the optimum SOR value.

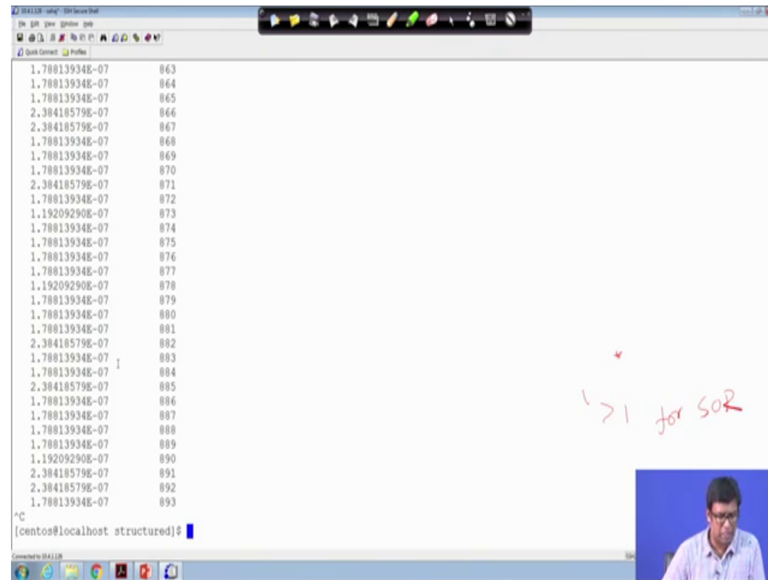
(Refer Slide Time: 22:01)

Let us run this with 1.4. Remember at 1.3, it was 927. See it is 760.

(Refer Slide Time: 22:23)

It is improved than 1.3 and if you run it at 1.5 SOR 1.5, its 720 plus something at 1.4. It goes to it is again does not converge.

(Refer Slide Time: 22:57)



So, and we can see that this is actually oscillating alpha is 2.38; then 1.7; then 1.19 and then again goes to 2.38. If the equation system at all converges the not alpha the error should always reduce, error should monotonically reduce, but it is not reducing. So, it is not converging.

So, somewhere between 1.4 and 1.5 is the optimum SOR after which the equation the system starts diverting. So, there is a value of omega which is greater than 1, for which we get the minimum number of iterations. This things can be verified by numerical experiments like this you run with different values of omega and c where you are getting the best performance or you have to look into the iteration matrix for SOR, the g matrix and spectral radius and from there, you can or the spectral radius of the Jacobi matrix SOR Gauss-Seidel matrix. Using that spectral radius, you can specify what is the optimum value of omega, there is a very nice formula for that, we have discussed it earlier.

So, now will also look on the other programming of few other methods and quickly go on Computer Programming for Steepest Descent Method.

(Refer Slide Time: 24:29)

So, what is the Steepest Descent Algorithm? That is you start with the guess value x 0 then, k is for different iteration level; k is equal to 0, 1, 2. Compute rk is equal to b minus Ax k. Then, compute a parameter alpha which is rk transpose rk dot product between the r and rk transpose Ar k and update x as xk plus 1 is xk plus alpha k rk and if rk is less than epsilon set, k is equal to sorry; if rk is greater than epsilon, if rk is it is not less than if rk is greater than epsilon, if the value is greater than something, then go to 2 and repeat this loop else you will say that the iteration are converged.

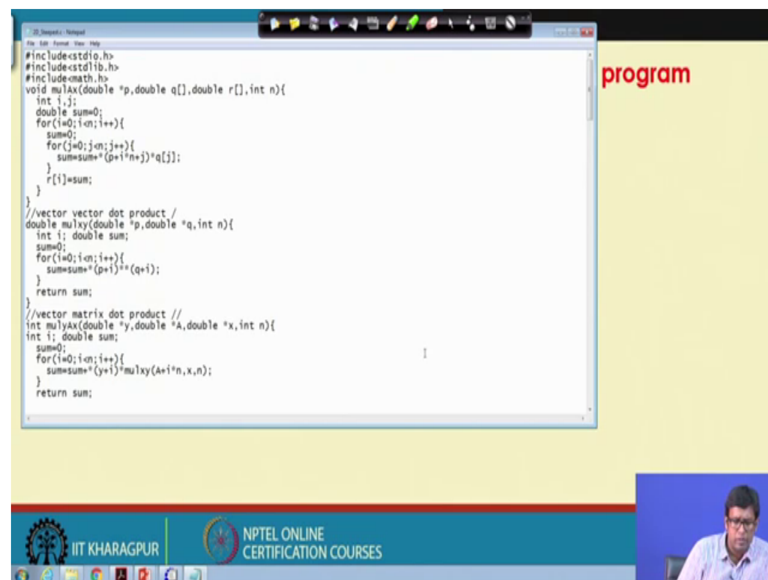
So, now we will see how we can write a computer program for this. So, what are the steps involved in major computation operational steps? There is a matrix vector multiplication here. A has to be multiplied with x. There is a vector-vector multiplication here, r has to be dotted there is a dot product of r and r and there is another matrix x vector multiplication which is A r and then a dot product of the resulted vector with the vector r.

So, there are 3 vector-vector; 2 vector-vector multiplication and 2 matrix vector multiplications and major operational steps will be carrying out this multiplication because A into r means I have to multiply when I am finding A r I have to find each row of A, I have to multiply each element with each of the r vectors. So, there has to be 2

loops; one loop is for multiplication of for one particular row and other loop is for multiplication of doing these for over number of rows.

And vector-vector multiplication is also I have to I have one loop which multiplying each element with corresponding element of another vector and then summing them up. So, now, if we look into the C program.

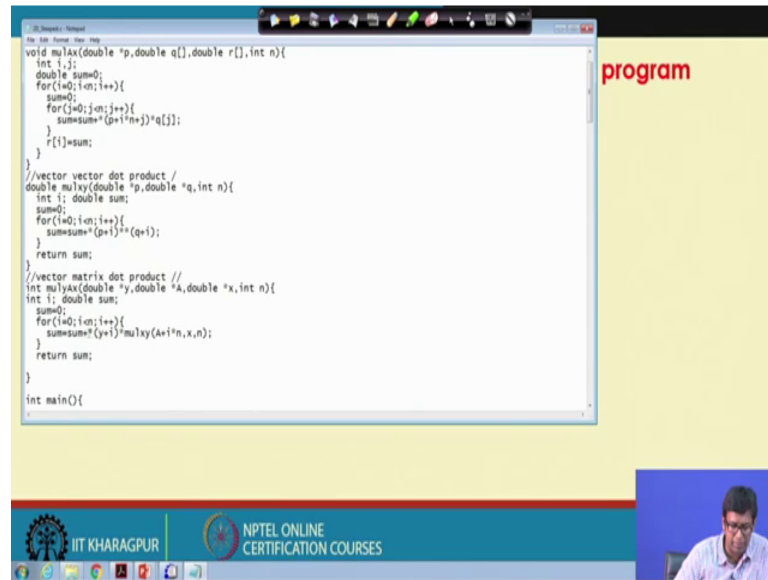
(Refer Slide Time: 26:57)



So, before this, before we go to the actual program we know that there are 2 vector-vector multiplication which is a multiplication of A with x. So, we write a small program for that and we can see that two loops; one loop is for multiplying within a row multiplication, within a row finding a results for a row and then, doing it for all the other loop rows all rows. So, for one particular row, this is for one particular row, where each element of that row is being multiplied by corresponding element of that vector and then, doing this loop programming this loop over all the rows of the particular matrix.

And then, there is a vector-vector product which is only one loop, there is another matrix vector product which is there is vector matrix product right which is again sorry.

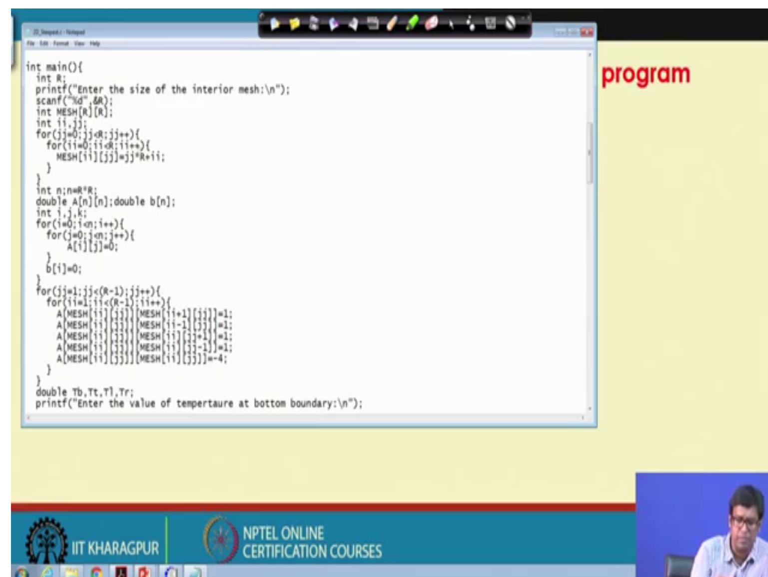
(Refer Slide Time: 28:01)



So, which will interestingly have a matrix vector product and then, there is another loop which is doing a vector-vector multiplication. So, this is matrix vector product which is matrix into vector product. This particular matrix in vector is again coming as the multiplication of 2 vectors.

So, which you have seen that there are 2 vector-vector products, 2 matrix-matrix product and 1 matrix 2 matrix vector product and for 1 matrix vector product, the product is further multiplied by a vector. So, with these 3 subroutines are written previously.

(Refer Slide Time: 28:37)



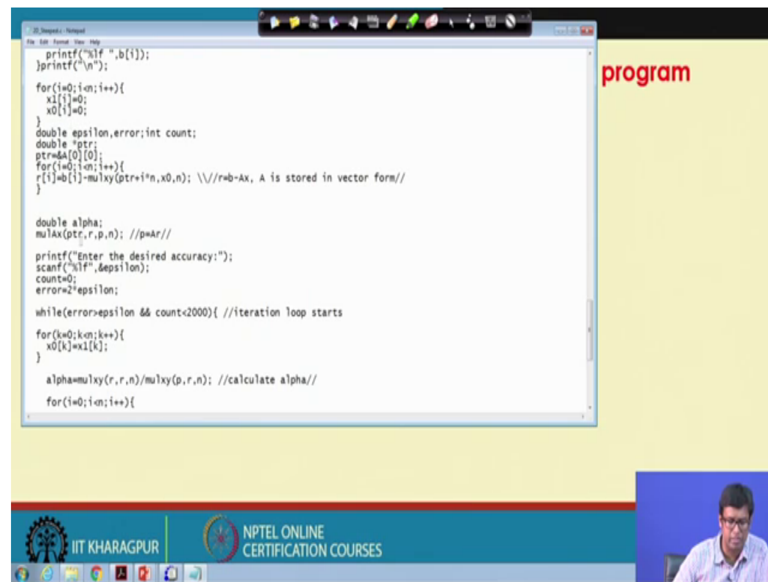
And similarly, the mesh file is defined for the pointers and we get the coefficient matrix A; give the boundary conditions; change the matrix accordingly as per boundary conditions so that it looks like a symmetric matrix because steepest descent is only applicable for symmetric matrix.

(Refer Slide Time: 28:47)

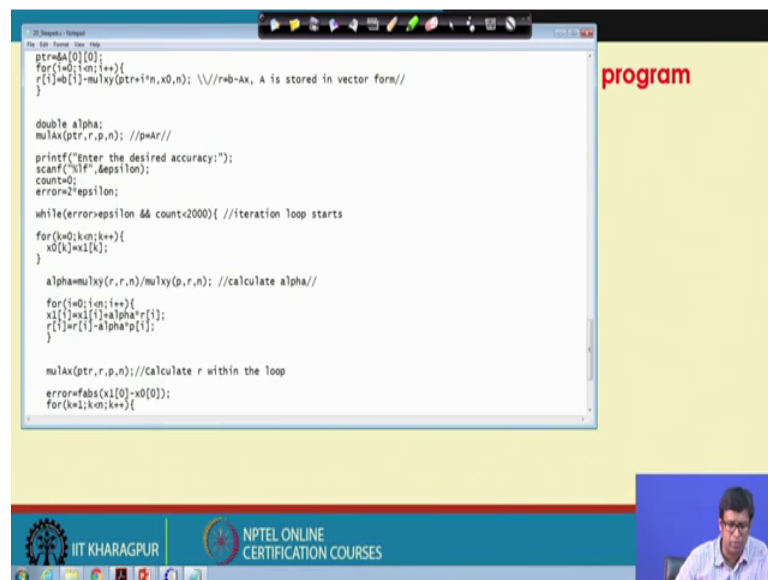
(Refer Slide Time: 28:55)

And then, r is equal to b minus multiplication of Ax; r is equal to b minus Ax and then we say it an accuracy epsilon. See this is for we start with the x is equal to 0 guess value and get r is equal to b minus Ax.

(Refer Slide Time: 29:07)



(Refer Slide Time: 29:13)



And then, we update x calculate alpha by r dot r divided by r dot A r; p is the product of A r; p is calculated as multiplication of the between the vector r and the matrix A.

(Refer Slide Time: 29:33)

```
printf("Enter the desired accuracy:");
scanf("%lf",&epsilon);
count=0;
error=2*epsilon;

while(error>epsilon && count<2000){ //iteration loop starts

for(k=0;k<n;k++){
  x0[k]=x1[k];
}

  alpha=mulxy(r,r,n)/mulxy(p,r,n); //calculate alpha//

  for(i=0;i<n;i++){
  x1[i]=x1[i]+alpha*r[i];
  r[i]=r[i]-alpha*p[i];
  }


  mulAx(ptr,r,p,n);//Calculate r within the loop

  error=fabs(x1[0]-x0[0]);
  for(k=1;k<n;k++){
  if(error<fabs(x1[k]-x0[k])){error=fabs(x1[k]-x0[k]);}
  }
  count++;
  printf("Iteration %d Error=%lf\n",count,error);
}

for(k=0;k<n;k++){
printf("Final x[%d]=%lf\n",k,x1[k]);}
```

program

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, then we update x and r and we check that whether error the difference between old x and new x is lesser value epsilon, if it is so, the control goes here and the loops goes on. So, it is this is how a steepest descent type of program is written and now we can look into 2 other projection methods.

(Refer Slide Time: 29:51)

Minimum Residual Method- Algorithm

Start with guess values of $x=x^0$

1. Compute $r=b-Ax$ and $p=Ar$
2. Until convergence, DO
3. Compute $\alpha=\frac{p^T r}{p^T p}$
4. Update $x \rightarrow x+\alpha r$
5. Update $r \rightarrow r-\alpha p$
6. Compute $p=Ar$
7. End do

Similar as steepest descent

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

One is minimum residual method, where very similar to the algorithm also looks similar to steepest descent except alpha is called calculated as p transpose r by p transpose p and p is equal to A r. So, here I am sorry here, we need to have 1 matrix vector product rather start with 2 matrix vector product, and then, this there is a vector-vector multiplication, there is a vector-vector multiplication.

Again we have to do one matrix vector product. So, there are 2 matrix vector product. This can be found before the iteration for only x is equal to 0 because later r is updated as that and then, there are 2 vector-vector products. So, you have to write similar subroutines for that.

(Refer Slide Time: 30:39)

Residue norm steepest descent- Algorithm

Start with guess values of $x = x^0$

1. Compute $r = b - Ax$
2. Until convergence, DO
3. $V = A^T r$
4. Compute $\alpha = \frac{\|V\|_2^2}{\|AV\|_2^2}$
5. Update $x \rightarrow x + \alpha V$
6. Update $r \rightarrow r - \alpha AV$
7. End do

2 matrix-vector product

2 vector-vector product

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

And the next method is the steepest descent is the residue norm steepest descent method, where there is a again a matrix vector product, but now we take transpose. So, they have to be stored in transpose form and found the product is since little more involved in terms of computational processes also because when we stored a matrix, we store the data row wise its using right pointers we can use a contiguous chunk of memory which is which will be very easy for the processors to access.

But when you look into A transpose, the memory ordering is changed and the processor has to access different locations of the memory, it might be a slower process, it will be consider a large matrix in distributed systems or in graphics (Refer Time: 31:30) systems.

However, so, there is a one matrix vector product. This is I l2 norm is basically what is the vector-vector product and this is the matrix vector product 2 and then, a vector-vector product. So, there are 2 operations; one vector-vector product V 2 square AVAV is also vector-vector product, but finding AV is the matrix vector product and AV is already found out. So, you not have to do anything there. So, you have 2 matrix vector, operation

product and 1 and also 2 vector-vector product and the program has to be modified accordingly.

You can try if you start with the steepest descent program, you can modify it accordingly to get residue norm and minimum residual program, the basic structure of the program remains same. The main part of the program are the matrix vector and vector-vector products.

So, this is how we will we showed in little detail how programming of the iterative solvers can be done and in similar way, we you can explore more to spend some time and look into the codes, also many number of codes are available online as open source software's. We can write the computer programs for even for a very large size matrix using these algorithms.

Thank you.