

## Numerical Methods And Programming

P. B. Sunil Kumar

Department of Physics

Indian Institute of Technology, Madras

### Lecture No. # 16

#### Matrix Elimination And Solution To Linear Equations

Today, we will continue to you are looking at the last class that is **is** a general non linear models and its implementation.

(Refer Slide Time: 01:22)

**General non linear Models**

To find the parameter values of nonlinear function that best fit the data we use the following algorithm.

$$\chi^2(a) = \sum_{i=1}^N \left( \frac{y_i - f(x_i, a)}{\sigma_i} \right)^2$$

We represent by  $a$  the set of parameter values  $(a_1, \dots, a_n)$ .  
Assign a guess value  $a = a_0$  that minimizes  $\chi^2(a)$ .  
Assuming this parameter set to be close to the true value  
Taylor expand  $\chi^2(a) = \chi^2(a_0) - d \cdot \Delta a + \frac{1}{2} \Delta a D \Delta a + \dots$

$$d_k = \frac{\partial \chi^2}{\partial a_k} \quad , \quad D_{ij} = \frac{\partial^2 \chi^2}{\partial a_i \partial a_j}$$

So, we saw that, a non linear function if you want to fit, then non linear function to a certain data points given to you. Then, we would declare as we have done for the case of a linear function, a chi square which is the difference between function values with the data points with the data given to you, a difference square divided by **divided by** sigma i square.

There is a function, the non-linear function we mean non linear in the parameters  $a$ . So, we saw and also in the parameters  $x$ , we saw that earlier when this function is non-linear in parameters in the variable  $x$ ; but not in the parameter  $a$  then we could still use a linear

effect. And our idea here was to find out this non-linear parameters  $a$  by minimizing the chi square that is where we stop in the last lecture.

So, we said that we have a set of parameters  $a_1, a_2, \dots, a_n$ , here is a  $I, a_1$  to  $a_n$ . And we which you would represent by a column vector  $a$ , and then we have a chi square  $\chi^2(a)$ . And so, instead of straight a way minimizing thus and trying to find the solution to that set of non-linear equations which we can do; if you write down a chi square, if  $\chi^2(a)$  full functional form, we could differentiate chi square with each of this  $a_i$ 's  $\chi^2(a)$  equations non-linear equations, because  $f$  is non-linear now. And you could find the solution to the set of  $\chi^2(a)$  non-linear equations that is what to do.

So, one method of doing that in way is to actually do a iterative procedure instead of doing a straight forward solution, we do a iterative procedure. There is we would assume a value of  $a$  or set of values for all the  $a$ 's, that is call vector  $a_0$  now; that is  $a_{01}, a_{02}, \dots, a_{0n}$  where  $n$  is the number of parameters. And then, we would expand the chi square around that value, the chi square of  $a_0$  minus Taylor expansion  $\chi^2(a_0)$  because, this  $a$  is a column vector; so, you have  $d$  is now vector containing all the derivatives of chi square with respect to  $a_k$ . So, that is with respect to  $a$ 's, that is the vector  $d$ .

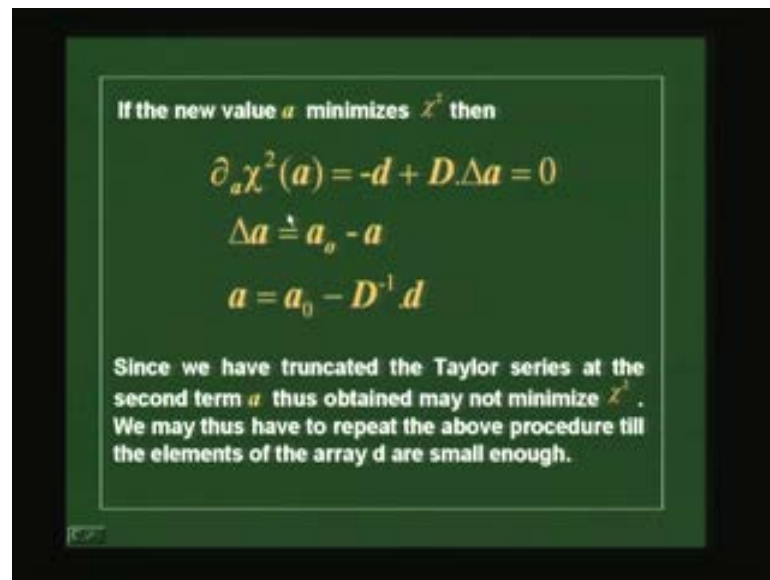
And then you have matrix  $D$ , which is the second derivative, which is second derivative of chi square with respect to two values,  $a_k$  and  $a_j$ . So, that is a Taylor expansion for chi square. So, what is a idea here is that, if  $a_0$  is a good assumption for the solution of this minimization; and then, if I expand chi square around that  $a_0$ , this derivative should be very small, because if it is a  $a_0$  actual minima then, all the derivatives should vanish.

And if it is not an actual minima,  $\chi^2(a_0)$  close to the minima then, this will be a small value. So, I can actually expand it around that point and also by expanding this way, I can go to the actual minima. So, that was the idea right. So, we go to the by expanding chi squared around values which we assume to start with for  $a_0$ , we can go to the actual minima.

So, as mean this chi square  $a$  we assume here what we get the chi square  $a$  is actual minima; that is the derivative of chi square  $a$  with respect to  $a$  is should be 0. So, to summarize, we are not writing down the chi squared here, we are not minimizing this chi square straight away with respect to  $a$  and finding the solution; instead of that, we assume a set of values  $a$  naught values for  $a$ , call  $a$  naught expand chi square around that

a naught; and then say that, this new chi square that is the value del a delta a,  $\chi^2(a)$  from a naught is the actual minima. So, then I minimize, I find the derivative of this chi square with respect to a, and equate that to 0 and solve for delta a and thus go to correct minima. So, that is what we were doing.

(Refer Slide Time: 06:10)



And delta a thus obtained and from which we can get the a, delta a is obtained may not be the correct one again. Because, we are going to say that, we are going to first of all ignore higher terms here in the Taylor expansion, we kept only up to a second derivative. So, that is going to be an assumption. So, what the delta a we have to get from this solution, will still not be a correct solution.

So, what we have done in the process is to minimize it produces non-linear problem that is a set of non-linear equation, which we would get if I differentiate this with respect to a to a set of linear equations n delta a to solve. But, this delta a set of linear equation which you obtained also because when approximation in the Taylor expansion we made. So, the a naught the new a value we get we know may not be the actual minima of chi square, it will not this new set a is may not minimize chi square.

(Refer Slide Time: 07:25)

It is conventional to remove the factors of 2 by defining

$$\beta_k \equiv -\frac{1}{2} \frac{\partial \chi^2}{\partial a_k}, \quad \alpha_{kl} \equiv \frac{1}{2} \frac{\partial^2 \chi^2}{\partial a_k \partial a_l}$$

$$a_{new} = a_0 - D^{-1} d \quad \text{becomes}$$

$$\Delta a = a_0 - a = D^{-1} d$$

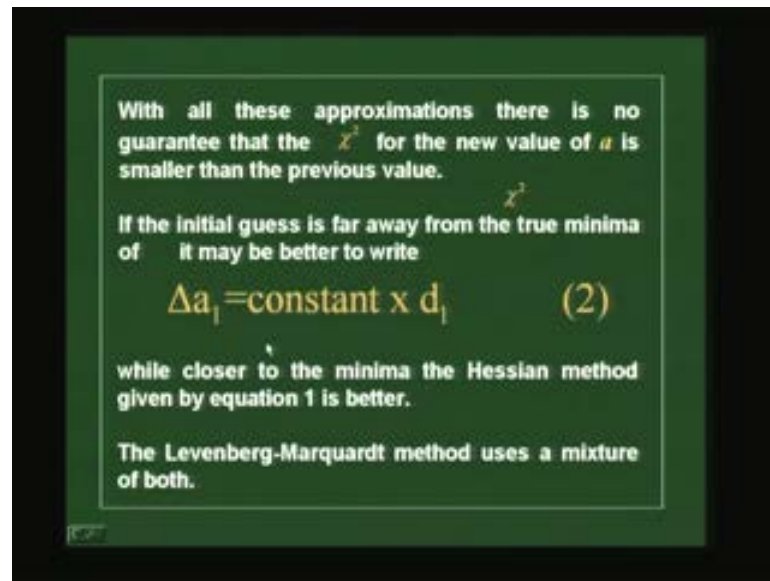
$$\sum_{l=1}^M D_{kl} \Delta a_l = -d_k \quad (1)$$

So, what we do now? We take that as a new a naught and go back and then, continue this loop that is what we saw. And we saw a sample program which implements this too. And we said that, this D's elements of D's which I call alpha k l here; and elements of this d which I call beta k, beta here and thus the first derivatives and second derivatives. And these elements again have to be determined as a second derivative of the **of the** chi square and the first derivative the chi square.

So, we will again assume the second derivative of the chi square, which as actually we saw yesterday were two terms, that is the second derivative of the function f with respect to **(( ))** and **(( ))** a k and a l. And the product of the first derivative's of chi square with respect to **(( ))** and with respect to a l. So, here again make an assumption that we will not actually do a second derivative of the function. We just use the product of the two first derivatives as elements of this.

This is something which we saw yesterday. And with that, we can actually solve for we make that assumption, because any way we have making another assumption that, the Taylor expansion can be **(( ))** beyond the second derivative; and we are doing a iterative scheme. And after all the correct solution should be something which should make the first derivative **(( ))**. So, we could make the effort to make this assumption.

(Refer Slide Time: 08:44)



And then solve for this thing. So, again the problem was that, if you not make the correct initial assumption, then we may not get go to the correct solution. There is again there is  $a$ , we are making an assumption here that, we make an approximation for  $a$ , and as a naught. And we do this a iterative scheme, and we get a hope that we know we go to the correct minimum, that is the chi square will get minimized; or the this iterative scheme will lead to  $a$ , which would give me the first derivative as 0; or minimum in the chi square, the chi square will reduced and go to a minimum **that is** that is our assumption.

But, that may not true if we choose a delta a **a a** value, which is far away from the minima of the chi square. If you are, if you have a choose a new value, the value  $a$  was far away from the values of chi square; then the new value we obtain, may not be guarantee to be smaller than the previous one. So, that is something which can occur in that kind of situation, instead of solving an equation using the second derivatives in this form.

One would like to actually go to a either a new assumption for a naught or multiply this a naught with the constants  $M$  go to new value. So, that is what we **(( ))** one possibility of going to new set of  $a$ 's or we could multiply the set of  $a$ 's with a constant value and then, go to new **new** value, that is the two possibilities.

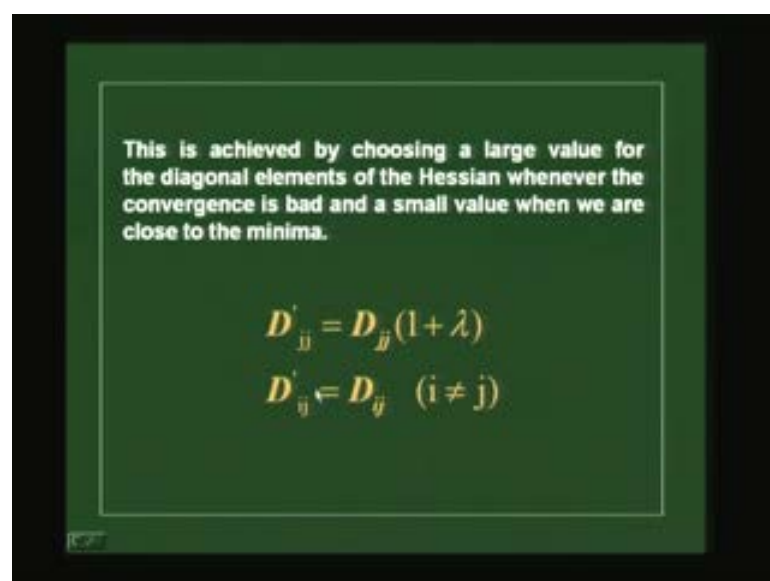
And second one, where we take the delta  $a_1$  we will take the constant multiplied by  $d_1$  that is a you go to a new set of values which are constant times multiplied by the  $d_1$  this

is the first derivative So, if you change in delta a 1 instead of solving that matrix times d 1 we would simply multiplied by constant, the first derivative and go to a delta a 1. So, if you look at this, it is saying that basically saying in our Taylor expansion we would be even ignore the second term and we will just keep this **this** term (Refer Slide Time: 10:40), that is what we are trying to say. So, we will just go to a new delta a value by simply multiplying the present delta a value by as a present d value the derivative by a constant. So, that is what we are going to do.

So then, the question arises that what is the value of a which you would choose, what was the constant value which you would choose. So, then if **(( ))** the a is in the **in the** function could have difference scale or different **different** dimension. So, we cannot take the same constant for all the d values. So, the correct thing to do would be take the second derivative in the second derivative **(( ))** only the diagonal element, because they are if you look at the diagonal elements of **of** this matrix delta k l; so, which is now will be del f by del a k in del f by del k l. So, that is what this D k l would be this matrix.

So, if the diagonal elements of that is del f by del k into del f by del k. So, that would have a correct scale for each of the a's. So, we could use the instead of giving using the full matrix which obtained from the second derivative, we could modify that matrix and give the diagonal elements a little more weightage. So, that is the idea behind this Levenberg- Marquardt method.

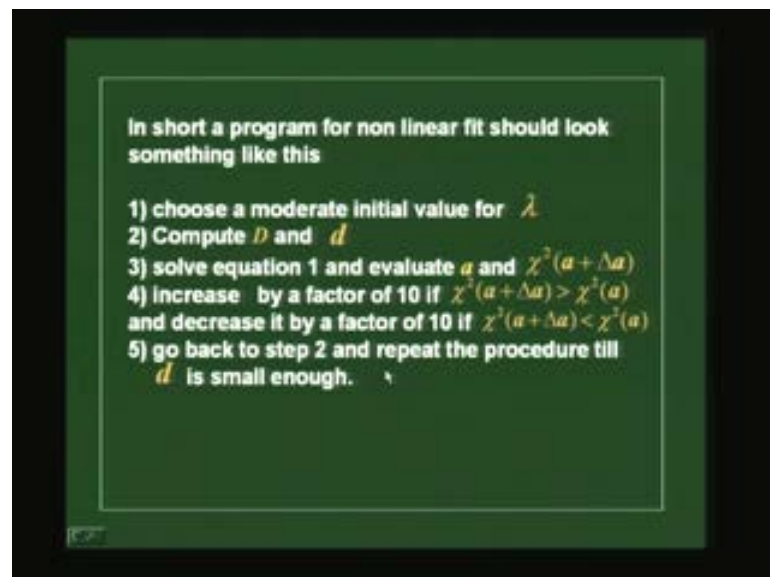
(Refer Slide Time: 12:19)



So, that is what we would implement here. So, that is to say that we will take the matrix elements and we will take the diagonal elements and multiply it by quantity called 1 plus lambda and then, get a new diagonal element values. So, we will modify only the **only** **the the** diagonal elements, the half diagonal elements when  $i$  naught equal to  $j$  is left as it is in this matrix.

Remember, this matrix is actually the second derivative of the functions of the chi square, there is approximated as product of the first derivative of the function  $f$  which we want to fit in.

(Refer Slide Time: 13:37)



So, now then what we would say is that, if we make an assumption  $a$  naught and in the first step if the chi squares the new chi square we obtain from the new  $a$  values; by obtained by solving that linear equation is less than the chi square in a previous step, then we will continue with this matrix. Or if it is greater than that is going in the opposite direction to the minima that we are not going down the value in chi square, but we are going up; and in that case, what we would do is we would increase the value of lambda and thus give more **(( ))** to the diagonal elements. That is what we would like to implement.

That is in short a program for non-linear fit will now look like this. We will choose apart from the  $a$  assumption a moderate value for lambda, which is the weightage which are going to give for the diagonal elements. And then we will compute the matrix matrices  $D$

and  $d$  that is the first derivative and second derivative the Hessian matrix. And we solve the equation, equation 1 which is the equation which says that  $d$  times,  $\Delta a$  is the  $(( ))$   $d$  and evaluate the matrix  $a$ . We solve a set of linear equations and solve evaluate a new value of  $a$  that is and we would evaluate chi square at the new value at  $a$  naught plus  $\Delta a$ .

And then will look compare this chi square with the whole chi square which we had at chi square  $a$  naught or chi square. And then we would see, **if the** if this chi square is increasing, then the **(( ))** if chi square a new chi square values greater than the whole chi square, then we would increase  $\lambda$  by a factor of 10; that is this  $\lambda$  is now increased by a factor of 10. But, if it is less than, the new chi square is less than whole chi square, then we will decrease it by a factor of 10.

And eventually, the  $\lambda$  could go to 0 and so, diagonal elements would be just since it is multiplied by  $1 + \lambda$  it will remain as the Hessian the true Hessian. So, that is **that is a** idea here that we will choose a moderate value of  $\lambda$  and we will increase the  $\lambda$  if it is going away from the minima, and we will decrease the  $\lambda$  if it is going towards the minima. And then we go back to the step 2 and repeat the procedure till  $d$  is small enough; that we will go back sure and then again continue this thing and till the first derivative is small enough, the tolerance value specified by us.

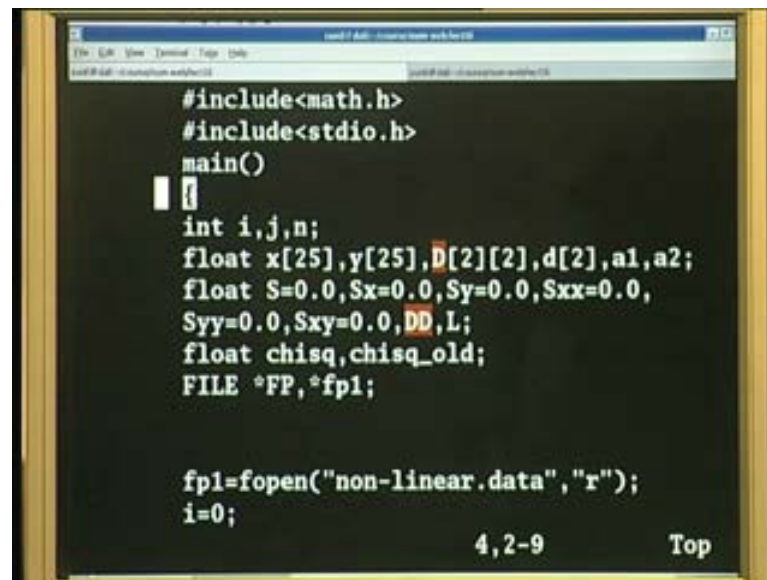
Remember, this is an important procedure in numerical method, because it is only just a fit a non-linear function, we could also use this to for any non-linear function minimization. So, this is a general technique for minimizing a non-linear function, a function of non-linear function of many variables. You could use this particular method, this Levenberg method to minimize that **that** functions with respect to this parameters, that is what we would be looking at.

So, we will now see an implementation of this scheme this  $\lambda$  scheme which is very important and before we go in to the more general idea for solving a linear set of linear equations. Remember that, we have to actually solve this equation if you want to get the  $a$ 's, we have to set this we have to solve a set of linear equations, there is this particular equation for we have to solve basically **(( ))** get the  $a$  new we have to do this quantity; that is what the idea is.



And this we can do very simply if our case, because we are find the D inverse, we can simply do this in the case of 2 by 2 matrix **matrix**; that is, if the number of parameters are only 2, then we can very easily do this, that is what we would be we will check as a program **(C)**.

(Refer Slide Time: 17:11)

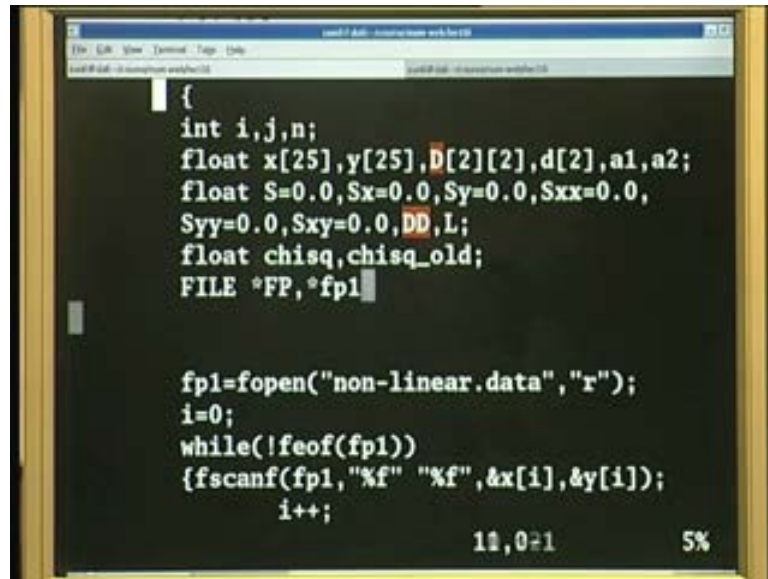


```
#include<math.h>
#include<stdio.h>
main()
{
int i,j,n;
float x[25],y[25],D[2][2],d[2],a1,a2;
float S=0.0,Sx=0.0,Sy=0.0,Sxx=0.0,
Syy=0.0,Sxy=0.0,DD,L;
float chisq,chisq_old;
FILE *FP,*fp1;

fp1=fopen("non-linear.data","r");
i=0;
```

So, here is a program for that **which we** part of it which we saw in the last class, that is we have the data points stored in the x and y; and we had a matrix a 2 by 2 matrix and we have the first derivative and second derivative the Hessian here and first **first** derivative in this matrix. And we have only two parameters to determine a 1 and a 2. So, all of them are floating points and I have declared them as the floating points.

(Refer Slide Time: 17:49)

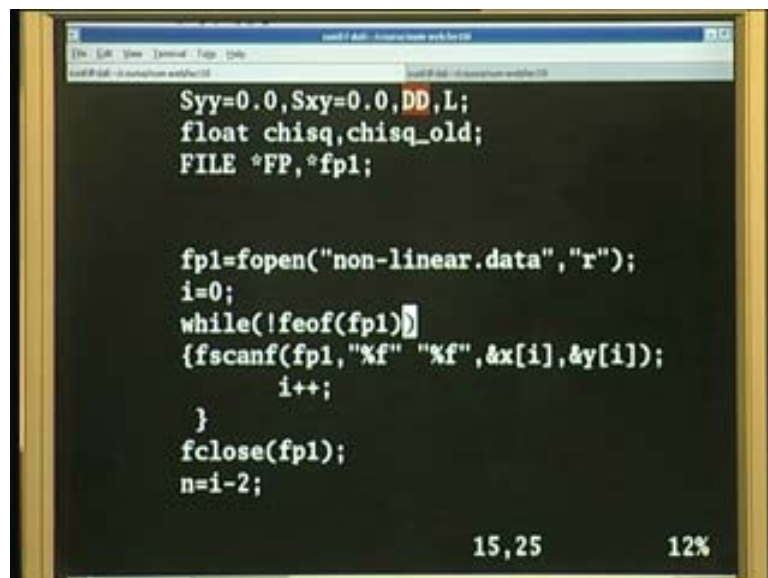


```
    {
    int i,j,n;
    float x[25],y[25],D[2][2],d[2],a1,a2;
    float S=0.0,Sx=0.0,Sy=0.0,Sxx=0.0,
    Syy=0.0,Sxy=0.0,DD,L;
    float chisq,chisq_old;
    FILE *FP,*fp1;

    fp1=fopen("non-linear.data","r");
    i=0;
    while(!feof(fp1))
    {fscanf(fp1,"%f" "%f",&x[i],&y[i]);
      i++;
    }
    11,0+1      5%
```

So, this is the array for x and y, which  $x_i$  and  $y_i$ , here is matrix of the second derivative matrix and here is the first derivative and here the two parameters,  $a_1$  and  $a_2$ . And then we need to state this chi square and chi square whole, because every time which change the  $a$  we go through 1 step in the iteration, we need to store whole old chi square value, because you want to compare to change the lambda. So, we will also have to have a lambda, which I call L here; and this would be the determinant of this D matrix which we would need to find its inverse.

(Refer Slide Time: 18:39)



```
    Syy=0.0,Sxy=0.0,DD,L;
    float chisq,chisq_old;
    FILE *FP,*fp1;

    fp1=fopen("non-linear.data","r");
    i=0;
    while(!feof(fp1))
    {fscanf(fp1,"%f" "%f",&x[i],&y[i]);
      i++;
    }
    fclose(fp1);
    n=i-2;
    15,25      12%
```

So, we will open a file and then using this file pointer which I have declared as here file pointer `fp1` and I will use this to you should be now very familiar with this that is `(( ))` use the open the file using the `fopen` command and then it is for reading that file so `r`. And I will read using this while statement, while till the end of file statement to read the all the elements in that file. And we will just compute the number of points in that number of lines in that file, which is now non-linear dot data.

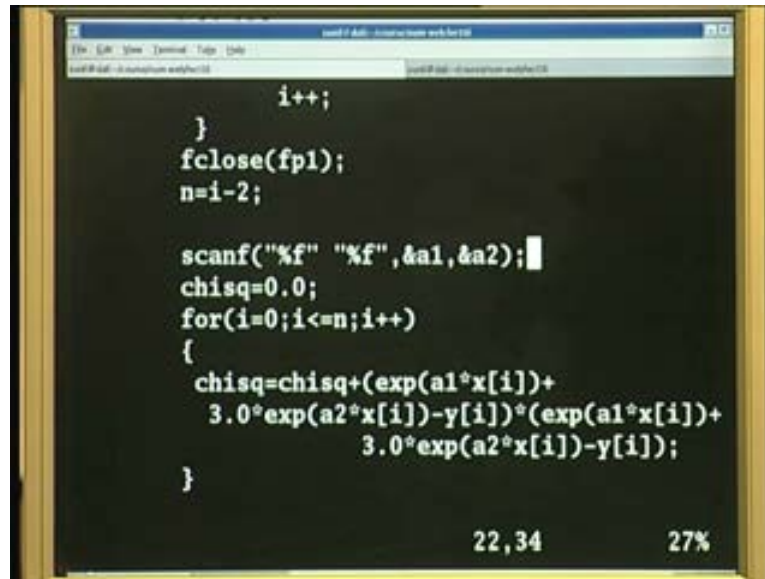
(Refer Slide Time: 18:56)

A screenshot of a terminal window showing C code. The code reads a file line by line, storing x and y values in arrays. It then calculates a chi-squared value for a non-linear fit. The code is: 

```
i=0; while(!feof(fp1)) { fscanf(fp1, "%f" "%f", &x[i], &y[i]); i++; } fclose(fp1); n=i-2; scanf("%f" "%f", &a1, &a2); chisq=0.0; for(i=0; i<=n; i++) { chisq=chisq+(exp(a1*x[i])+ 3.0*exp(a2*x[i])-y[i])*(exp(a1*x[i])+ 26,25 22%
```

So, having obtained that data in to  $x_i$  and  $y_i$ , remember the way to read as use the ampersand sign, it is pointing to that particular memory location; and we read of that  $x_i$  and  $y_i$ . Once you have that now, once you have data points when read off, then you now need the initial guess for  $a_1$  and  $a_2$  and way this is simple program, which is have only two functions that two values.

(Refer Slide Time: 19:20)



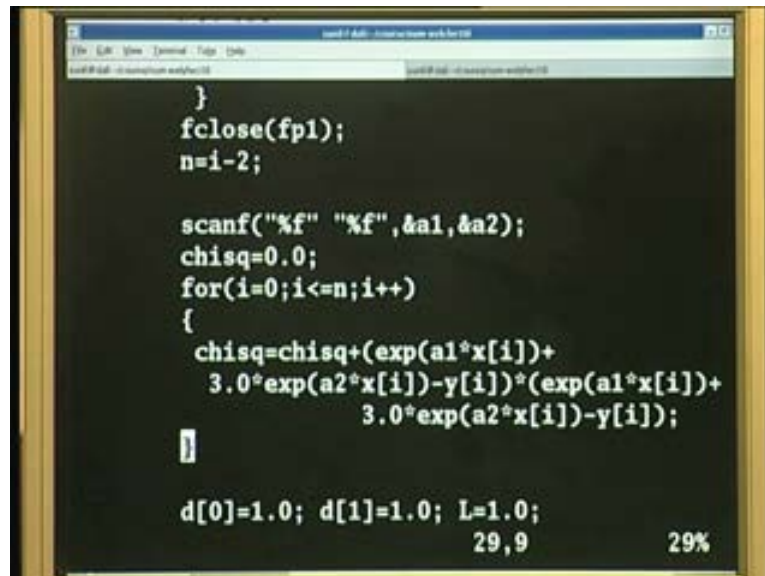
```
        i++;
    }
    fclose(fp1);
    n=i-2;

    scanf("%f" "%f", &a1, &a2);
    chisq=0.0;
    for(i=0; i<=n; i++)
    {
        chisq=chisq+(exp(a1*x[i])+
            3.0*exp(a2*x[i])-y[i])*(exp(a1*x[i])+
            3.0*exp(a2*x[i])-y[i]);
    }

    22,34      27%
```

And the functional form we are going to fit is this one that is exponential  $a_1 x$  of  $i$  plus 3 times exponential  $a_2 x$  of  $i$ . So, that is my  $f$  of  $x$   $i$ , exponential  $a_1 x$  of  $i$  plus 3 times  $a_1 x$  of  $i$  that is my functional form. So, it is very simple function, only two parameters  $a_1$  and  $a_2$  (()). So, that we can do this we can find the inverse very easy.

(Refer Slide Time: 19:55)



```
    }
    fclose(fp1);
    n=i-2;

    scanf("%f" "%f", &a1, &a2);
    chisq=0.0;
    for(i=0; i<=n; i++)
    {
        chisq=chisq+(exp(a1*x[i])+
            3.0*exp(a2*x[i])-y[i])*(exp(a1*x[i])+
            3.0*exp(a2*x[i])-y[i]);
    }

    d[0]=1.0; d[1]=1.0; L=1.0;

    29,9      29%
```

So now, we have to we need a initial guess for  $a_1$  and  $a_2$  which you have to type in, so they will read off using this scanf function. And now you just remember this is scanf function, while this was reading from the file using fscanf, so this is reading from the

terminal that is (( )) screen. So, that is it will read both a1 and a2, both of the floating point. And I will set chi square equal to 0 and then go through all the points (( )) from 0 to n to compute 0 to n is a number of data points which we have we will compute that and obtain the chi square value. So, we are computing that basically f minus y that is f of x i it is exponential a1 x i plus 3 times exponential a2 x i minus y of i; and the product of square of that, so I multiplied by twice it is a product of x 2 again. So, it is basically exponential a of x i plus 3 times exponential a2 of x i minus y of i whole square y i y i whole square, that is what my chi square is.

(Refer Slide Time: 21:14)

```

chisq=chisq+(exp(a1*x[i])+
3.0*exp(a2*x[i])-y[i])*(exp(a1*x[i])+
3.0*exp(a2*x[i])-y[i]);

d[0]=1.0; d[1]=1.0; L=1.0;
while(fabs(d[0])+fabs(d[1])>0.01)
{
D[0][0]=0.0;
D[0][1]=0.0;
D[1][0]=0.0;
D[1][1]=0.0;
d[0]=0.0;
d[1]=0.0;
}

```

29,9 43%

Sigma here is divided by sigma i square which we put it equal to 1. So, sigma i square is 1, because all data points are equally reliable that is what the assumption here. And then I now need to compute the matrix elements, there is a next element, the next part of the program; and I need to do this till my first derivative which has stored in the (( )) the (( )) 0 and (( )) 1 goes to 0.

(Refer Slide Time: 21:20)

```
d[0]=1.0; d[1]=1.0; L=1.0;
while(fabs(d[0])+fabs(d[1])>0.01)
{
  D[0][0]=0.0;
  D[0][1]=0.0;
  D[1][0]=0.0;
  D[1][1]=0.0;
  d[0]=0.0;
  d[1]=0.0;
  for(i=0;i<=n;i++)
  { D[0][0]=D[0][0]+2*x[i]*x[i]*
    exp(2*a1*x[i])*(1.0+L);
    D[0][1]=D[0][1]+6*x[i]*x[i]*
```

So, until the first derivative are 0 that means I have put a tolerance of 0.01 here, till this elements of the d vector that is d of 0 and d of 1 the absolute value of that, that is mod of that value d of 0 and d of 1 sum **sum** of these two; so, that is define as the error or that should be **that should be** less than my tolerance, which is 0.01. So, if it is a less than this, then it will stop program. So, till that it will continue the iteration.

So, initialize my d of 0 and d of 1 to 1, and lambda to 1 and then I go back here. So, I can **I can** do that here. So, I do the D of 0 0 **all the all the initialization is done here sorry** all the initialization of the vectors have done here. And then, I actually compute I compute the second derivative; you can see the second derivative is computed as the product of the 2 first derivative of the function.

So, the first derivative with respect to a 1, a's x of i exponential a 1 x i, and the first derivative with respect to a 1 again, because this is D 0 0; so, this is the product of the first derivative with respect to a 1. Now, this is the diagonal element of the matrix, it is D 0 0 is it diagonal element of my second derivative matrix; so there as we multiplied by 1 plus lambda which is 1 plus L.

So, there is doing exactly what we are saying here. So, we will use this idea that all the diagonal elements will be multiplied by 1 plus lambda (Refer Slide Time: 23:09). So, multiplied by 1 plus lambda here, and this is the half diagonal elements, so that is not been multiplied by lambda; so, just the same as the product of the 2 first derivatives

exponential  $a_1 \times i$  and exponential  $a_2 \times i$  3 times  $x$  of  $i$  into 2 times  $x$  of  $i$ . So, that is **that is** half diagonal element.

(Refer Slide Time: 23:33)

```

D[1][1]=0.0;
d[0]=0.0;
d[1]=0.0;
for(i=0;i<=n;i++)
{ D[0][0]=D[0][0]+2*x[i]*x[i]*
  exp(2*a1*x[i])*(1.0+L);
  D[0][1]=D[0][1]+6*x[i]*x[i]*
  exp(a1*x[i])*exp(a2*x[i]);
  D[1][0]=D[1][0]+6*x[i]*x[i]*
  exp(a1*x[i])*exp(a2*x[i]);
  D[1][1]=D[1][1]+2*x[i]*x[i]*9
  *exp(2*a2*x[i])*(1.0+L);
  d[0]=d[0]-(y[i]-(exp(a1*x[i])+3.0*exp
  (a2*x[i]))) *x[i]*exp(a1*x[i]);
  }
49, 28-35      61%

```

Everything has 2, because we have absorbed one 2 into this derivative second derivative here.

(Refer Slide Time: 23:43)

```

for(i=0;i<=n;i++)
{ D[0][0]=D[0][0]+2*x[i]*x[i]*
  exp(2*a1*x[i])*(1.0+L);
  D[0][1]=D[0][1]+6*x[i]*x[i]*
  exp(a1*x[i])*exp(a2*x[i]);
  D[1][0]=D[1][0]+6*x[i]*x[i]*
  exp(a1*x[i])*exp(a2*x[i]);
  D[1][1]=D[1][1]+2*x[i]*x[i]*9
  *exp(2*a2*x[i])*(1.0+L);
  d[0]=d[0]-(y[i]-(exp(a1*x[i])+3.0*exp
  (a2*x[i]))) *x[i]*exp(a1*x[i]);
  d[1]=d[1]-(y[i]-(exp(a1*x[i])+3.0*exp
  (a2*x[i]))) *3*x[i]*exp(a2*x[i]);
  }
49, 13-20      65%

```

So now, then we have the  $d$  matrices which are  $y$  of  $i$  minus that is the function the  $y$  of  $i$  minus the function times the derivative of the functions. So, that is what is been given here, so that is been put in here. And then we have the, so that  $d$  of 0 and  $d$  of 1 is being

computed which we saw (( )); no change in this from the previous simple iteration scheme, change comes here only the diagonal elements, where you put 1 plus lambda.

(Refer Slide Time: 24:14)

```

d[0]=d[0]-(y[i]-(exp(a1*x[i])+3.0*exp
(a2*x[i])))^x[i]^exp(a1*x[i]);
d[1]=d[1]-(y[i]-(exp(a1*x[i])+3.0*exp
(a2*x[i])))^3*x[i]^exp(a2*x[i]);
}
DD=D[1][1]*D[0][0]-D[0][1]*D[1][0];
printf("%f %f %f %f\n",D[0][0],D[1][0],
D[0][1],D[1][1]);
printf("%f %f\n",d[0],d[1]);
a1=a1-(D[1][1]*d[0]-D[0][1]*d[1])/DD;
a2=a2-(D[0][0]*d[1]-D[1][0]*d[0])/DD;
printf("n = %d      a1 = %f   a2 = %
f      L = %f\n",n,a1,a2,L);
chisq_old=chisq;

```

55, 25-32      77%

(Refer Slide Time: 24:23)

```

printf("%f %f %f %f\n",D[0][0],D[1][0],
D[0][1],D[1][1]);
printf("%f %f\n",d[0],d[1]);
a1=a1-(D[1][1]*d[0]-D[0][1]*d[1])/DD;
a2=a2-(D[0][0]*d[1]-D[1][0]*d[0])/DD;
printf("n = %d      a1 = %f   a2 = %
f      L = %f\n",n,a1,a2,L);
chisq_old=chisq;
chisq=0.0;
for(i=0;i<=n;i++)
{
chisq=chisq+(exp(a1*x[i])+3.0*exp(a2*
x[i])-y[i])*(exp(a1*x[i])+3.0*exp(a2*x[i])-y[i]
));
/x

```

62, 25      83%

And then, now I just find this step is actually to find the (Refer Slide Time: 24:19), if this step is just to find the the determinant, so that is the determinant of this function (( )) the matrix D. And then I am just printing out the values of all the elements here matrix elements here. Now, this is the solution of the equation that is the this is the this is the inverse of the matrix, inverse of matrix times d that is what the particular expression is.

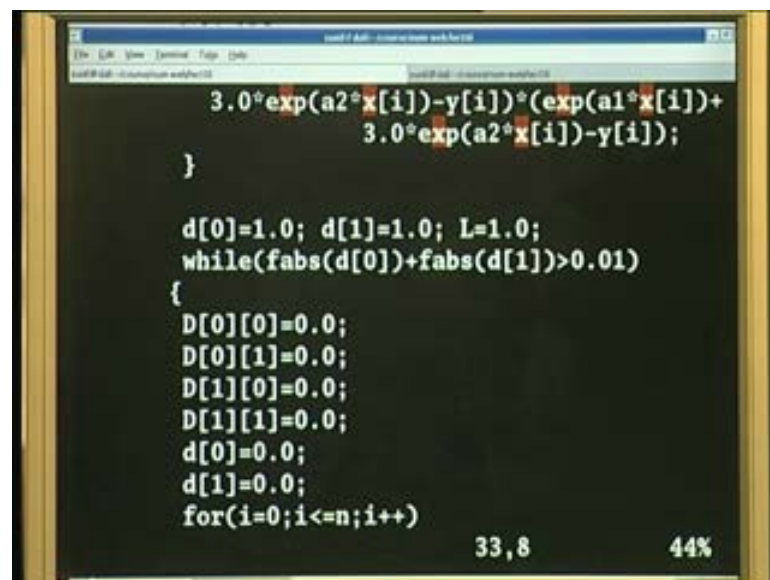


So, **so** inverse of the d matrix multiplied by D minus a 1 is my new a 1 value. So, new a 1 and a 2 is thus obtained, I can compute now the new chi square value, now I have the new a 1 and a 2; so, I store the old chi square which I had into a think called chi square old. And I set chi square equal to 0 again, and compute the new chi square here. So, this simple and same as what we been doing.

So, I compute the chi square again here and now I say that, if chi square new that is chi square is greater than or equal to old chi square, and then I will increase the L by a factor of 10. On the other hand, if it is less then I will reduce it by a factor of 10 that is multiplied by 0.1. So, that is the idea.

I print out the chi squared, the standard error and the chi square and the standard error we print out that that is what **(( ))** in this program. Because, we can run this program and then we can see what we actually get. So, here again I am printing out in this point I am printing out a 1 once I get the new a 1 a 2 values, I will print out the a 1 a 2 values and also the **the** new L value. So, a 1 and a 2 and L I am printing out. And at the end, once the loop is over, so this is where the **this is where the** while loop ends **(( ))**.

(Refer Slide Time: 26:13)



```
3.0*exp(a2*x[i])-y[i]*(exp(a1*x[i])+
3.0*exp(a2*x[i])-y[i]);
}

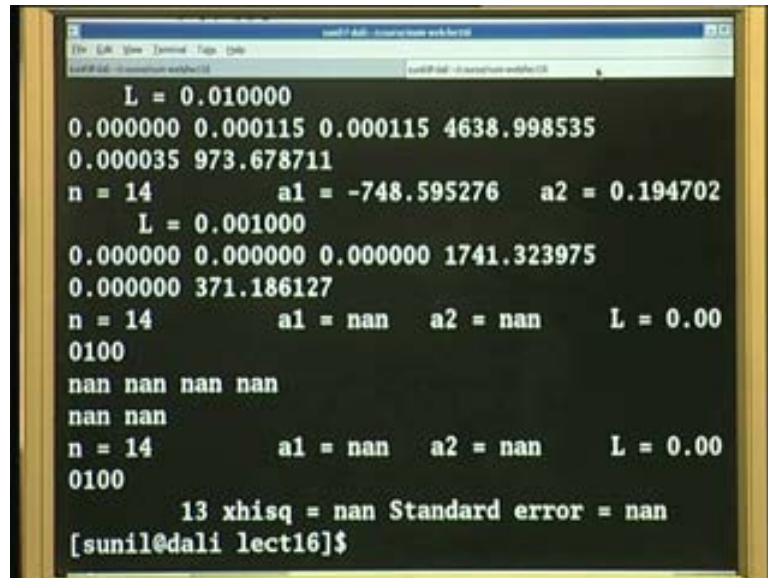
d[0]=1.0; d[1]=1.0; L=1.0;
while(fabs(d[0])+fabs(d[1])>0.01)
{
D[0][0]=0.0;
D[0][1]=0.0;
D[1][0]=0.0;
D[1][1]=0.0;
d[0]=0.0;
d[1]=0.0;
for(i=0;i<=n;i++)
```

33,8 44%

So, this loop will continue from here to calculation of the new chi square and changing the L up to that, till I have achieved this particular accuracy (Refer Slide Time: 26:26), which I have demanding **(( ))** a demanding here. There is the d the sum of the d elements has to go is to be less than 0.01. So, once I have done that I will print out this value and

the chi square and the standard error, `(( ))`. So, that is what we **we** will be seeing here. So, I will compile this program here (Refer Slide Time: 26:46), and then I run this.

(Refer Slide Time: 26:58)



```
L = 0.010000
0.000000 0.000115 0.000115 4638.998535
0.000035 973.678711
n = 14      a1 = -748.595276   a2 = 0.194702
L = 0.001000
0.000000 0.000000 0.000000 1741.323975
0.000000 371.186127
n = 14      a1 = nan      a2 = nan      L = 0.00
0100
nan nan nan nan
nan nan
n = 14      a1 = nan      a2 = nan      L = 0.00
0100

13 xhisq = nan Standard error = nan
[sunil@dali lect16]$
```

So, it is `(( ))` very initial value I gives some initial value let me give as minus 0.5 and 0.8 that **that** see, this particular case that is seem to work, because `(( ))` going to any it did not `(( ))` it all it is just `(( ))` whole program. So, we will have to give some initial another initial value minus 0.5 minus 0.2. So, this seems to converge, so again this program does not seem to converge for all values that is something which we should we discussing in the next section. So, why does not the program converge; so we will discuss that in next section.

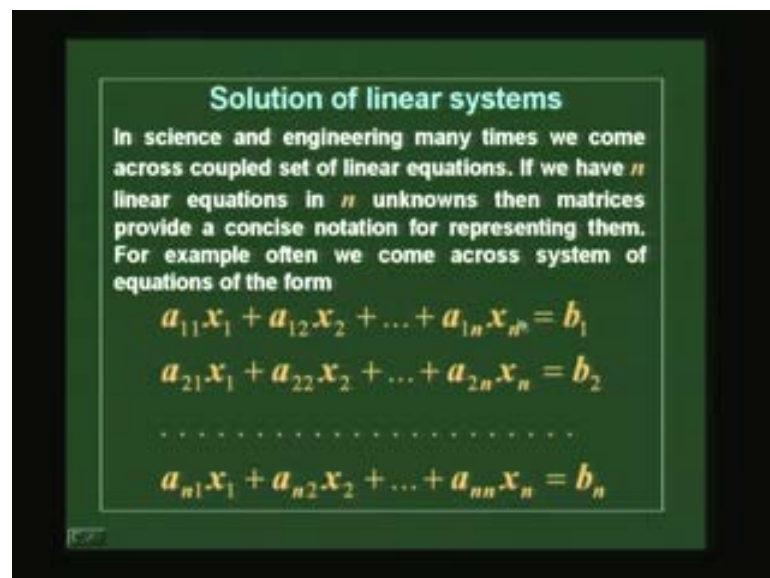
So, now here this particular value it did converge and then, we went to values of minus 1.86 and minus 1.132 these are a 1 a 2 values we have obtained. And I got a reasonable chi square and reasonable standard error. And you can see, if you look at the L value, the L values are `(( ))` to 0, because it was converging; and if it was not, then it would `(( ))` to some last number. And then, we would program would eventually x L, because number is `(( ))` large.

So now, we could do this program here, because we could do this program here because, we had used only two parameters. So, two parameters were easy to implement, because we could actually compute the inverse by hand. So, but now let us say we have a function with larger number of parameters, n parameters; and then how do we **how do**

we solve this. So, we we need to have a method to solve. So, now in this in this program I need to go I need to put in something here, I have computed here the inverse of the matrix by hand; but here, I need now a program a function, which would which would tell me what the inverse of the matrix or what is solution of this equation is; that is my set of linear equation, which I have obtained by finding the derivatives of the Taylor expansion with respect to a.

So, what the solution of that equation is, I need (( )) function to do that and then, I would call this function from here. So, that is what I am going to do. So, that is what we will see the next section and then, we will see what what case we can get the solution of that and where we will that kind of such an equation could fail that is what we should look at.

(Refer Slide Time: 29:14)



So, in in short you could have a set of linear equations like this. So, we would have some many parameters let us say now this  $x_1, x_2, x_3$  are unknown, so the  $x_n$  unknown; and  $a_{11}, a_{12}, a_{1n}$  here are coefficients. And now, I need to solve the set of linear equation. So, now in 1 or 2 class will discussed detailed, how do we go about solving such an equation a numerically. And we have  $n$  equations and we have  $n$  unknowns linear and so, and now we want to solve this linear equation.

We can see that, if I write it in this form which in the previous case problem of minimizing the the chi square or fitting a non-linear function to a set of data, we would get by finding the derivative of the Taylor expansion (( )) square. Or in general, we could

have a set of linear equation and you want to solve them, there are many situations, where this kind of problem arises. One example being which are what we just now solved. So, our idea would be that to develop an algorithm for solving a set of linear equation like this, and then put that back into our equation for finding the parameters **parameters** of the non-linear function.

So, more than two parameters you should put this incorporate this program into that into that function and I get a general program for either minimizing a set of non-linear equations or fitting a set of non-linear function of many parameters into **(( ))** set of data points. So, here is the matrix for that then we have this linear set of linear equations. And we **we** can write this linear equations in a **in a** matrix form **right**. So, we write this in a matrix form.

(Refer Slide Time: 31:11)

This can be written in the matrix form  $[A][x] = [b]$  that is

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

So, that is the first step to identify it is simple to see. That we have a set of linear equations like this (Refer Slide Time: 31:22) and **we can** I can simplify write this as a 1 1, a matrix which contains a 1 1 and a 1 2, a 1 n, a 2 1, a 2 2, a 2 n etcetera to a n 1, a n 2, a n n multiplied by the column vector x 1, x 2, x 3, x n, x 1, x 2, x 3 up to x n which gives you b 1, b 2, b 3, b n as a column vector.

(Refer Slide Time: 31:42)

This can be written in the matrix form  $[A][x] = [b]$  that is

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{bmatrix}$$

We will see how to solve such an equation for the unknown column vector  $[x]$

So, that is what we have here. So now, we have to invert this matrix and then, multiplied by both side of equation by the inverse of this matrix a, then we could get the solution A x that is what (()). So, basically idea would be either to find the inverse of this matrix or some way solves this equation. So, that is what, so we will do.

(Refer Slide Time: 32:03)

### The elimination method

A simple way to solve this equation is, which may not be always the best, is to extend the method we always use to solve two linear equations in two unknowns, the elimination method.

Elimination method is based on the fact that the equations do not change if,

- 1) if we multiply or divide any row by a number ,
- 2) if we exchange the rows on either side of equation or
- 3) replace one of the equations by a linear combination of equations,

So, the first way to do that it is the simplest way is to use a very simple elimination method, we actually saw that already in our function minimization sorry in our in our data fitting the data the function, which we actually solved. When we are doing just for a

set of linear equations in the case of fitting a data fitting a function through a set of data points, we (( )) this elimination method in in practice. So, here we will do a in general case what I mean by that is, if you look at this matrix equation, and if I can make this into an upper triangular matrix all the diagonal elements, all the elements below the diagonal element diagonal matrix.

If I make if I could make this diagonal all the elements below this diagonal matrix to 0 without changing the actual value without changing this equations or altering this equations without changing its result; if I could do that and then, I can see that if I have only these elements non-zero only upper triangular elements upper triangular elements non-zero; and then, the last equation would simply read  $a_{nn}x_n = b_n$ , because everything to the left of that will be 0 right.

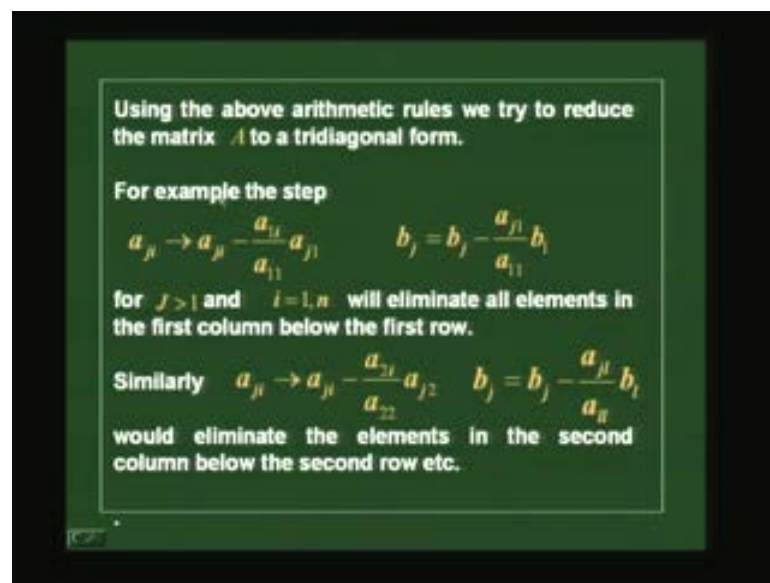
So, last equation is simply  $a_{nn}x_n = b_n$ . And then, I could solve  $x_n$  as  $b_n/a_{nn}$ . So, if I have a matrix here which is only the upper triangular elements and (( )) below the diagonal element, then the last equation be simply  $a_{nn}x_n = b_n$ . And that means  $x_n = b_n/a_{nn}$ , once I obtained  $x_n$  then I can go back to the equation above that which will have  $a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1}$ ; and that will be contain that equation will contain  $x_{n-1}$  and  $x_n$ . But, I know  $x_n$  already, so I can solve that for that. So, that is called back substitution.

Once we could if we could make this matrix such that all the element below its diagonal element are 0, then I could solve it from the bottom that is by back substitution, I can solve for all the  $x$  values. So, that is the idea of a simple elimination scheme; and that relies on the fact that, if we multiply or divide any row by a number, equation does not change, the solution of the equation does not change. That is if I could divide any row by a number multiply or divide any row by a number with the solution of the equation do not change.

Or if we exchange the rows on either side of the equation; so there are many things with the three things we can do, we can either multiply or divide each row by a number. Or we could exchange the rows on either side, the right and left hand side of the equation, where is a simply making the 1st equation to the  $n$ th equation or  $n-1$ th equation to the  $n-2$  equations etcetera, we are just shifting the equation. So, there will not change the solution we know (()).

We also know that, we have an equation  $f$  of  $x$  equal to  $a$ , then if I multiply either side by a number it is not going to change the solutions. So, that is  $(( ))$  as the first thing. Or we could replace one of the equations by a linear combination of other equations that is next method. What we will do? Take one of the equations and replace it by a linear combination of the equations that will also not change either the solution. So, that is the idea here.

(Refer Slide Time: 35:43)



So, that is been summarized here in terms of what happens to each of the elements. So,  $so$  these operations that is multiplied or divided by number or exchange each rows by an either side of the equation or replace one of the equation by linear combination of equation is, that is what we going to implement in this thing. So, what I mean is that, what I am going to say is that  $(( ))$  equation again.

See, if I look at equation again (Refer Slide Time: 36:13), now if I want to make this element 0 this particular element 0, what I could do is? I could take this I could subtract from this row; I could subtract form this whole row of this row, this particular row divided by the first element and multiplied by this element; that is what I am trying to says.

If I take this row and divide this row by a 1 1; so every element in this row get divide by a 1 1,  $b_1$  also divided by a 1 1 that does not change my equation, and does not change

the solution of my equation. So, I could take this first row, first row of the matrix and divide it by the first element of that matrix and that will not change my equation.

And then I multiplied that by this a 2 1, now I divide by this row by a 1 1 and I multiplied by multiplied by a 2 1, so what what will happen is this will become this a 2 1; and then I subtract from this element are from this row this row, if I subtract if I do a linear combination of rows, the solution does not change we know that.

So, the linear combination of this row this this row 1 and row 2, so I replace row 2 by a linear combination of row 1 and row 2 (( )) what I do as, I take the row 2 and replace it by a row 2 plus a constant multiplied by row 1. And what is that constant? That constant is a 2 1 by a 1 1. So, I take a 2 1 the whole row a 2 a 2 1 up to a 2 n, and subtract from that this row a 1 1 up to a 1 n and multiplied by a 2 1 and divided by a 1 1. So, all elements of this row multiplied by a 2 1 divided by a 1 1 subtracted from this will give me a 0 here and all this element will change. So, I replace this row by a linear combination of these two.

And I continue for the next row next row again I will do that. So, the factor which I have to multiplied will now become a 3 1 by a 1 1 and then I subtract from this; so, then this will become 0 and by this procedure I could make all the all the elements below this a 1 1 go to 0. And then, I could repeat this process for the next element a 2 2. So, I could make them again go to 0 that is idea of elimination method. So, that is what we have done here.

So, j here is the is the row index and i is the column index. So, the first element would be j equal to 1 and i going from 1 to n, (( )) 0 to n minus 1. So, let us take to the index as 1 to n as we mark the columns in a program it would actually go from 0 to n minus 1. So a j i, so a 1 1 is the first element and a 1 2 is a second element of the first row. So, j is the row index and i is the column index. Remember this, j is the row index and i is the column index.

So, what I am trying to what I am doing here is now, I take a j i the first in the first step I will take all j's which are below which below the first row all columns and I subtract from the I replace that row by each element of the row, there is i going from 1 to n i going from 1 to n for all j greater than 1 I do this operation. That is I take a j i and subtract from that a j 1 that is the element of the row above that in the corresponding



column and then, multiplied by a  $1/i$  and subtract by a  $1/i$ . So, a  $1/i$  is the element in the first row in the column  $i$  and you divided by a  $1/i$ , there is the first element and multiplied it by a  $1/j$  that is the first element of  $(i)$  column  $j$ . By this procedure, you can see that, if for  $i$  equal to  $1$  for any  $j$ ,  $i$  equal to  $1$  this would mean by a  $1/j$  will go to  $0$ . So, that will happen for all **all** rows below  $1$  that is what I just show you before.

And the similar thing has to be done on the right hand side too, because  $(i)$  equation would change  $(i)$  prevent the equation from changing. So, we would the solution from changing we have also do the on the right hand side. That is the exact same operation is done on the right hand side. That is we subtract from  $b_j$  for all  $j$  values greater than  $1$ , a  $1/j$  by a  $1/i$  into  $b_1$ . Hope this process is clear that is what we doing is a taking the first in the first step we are taking making the first column below the first row to be  $0$ ; all the elements of the first column below the first row goes to  $0$ . That is what this step would do.

I guess this is clear for  $j$  greater than  $1$ ,  $j$  is the row index for all  $j$  greater than  $1$  all elements  $i$ , we subtract from that the first row is the first row that is a  $1/j$   $i$ . The elements of the first row multiplied by a  $1/j$  by a  $1/i$  in this process I could make all the row elements same.

Now, I could do this for the next element in this way that is instead of now  $1$   $(i)$  to  $2$ . And then, now I will start from the second element **right**, because the first element of the **first** second row is already  $0$ . So now, I do the second element and I go down all the elements; and that process I will eliminate all the  $(i)$  second row into  $0$ .

So, in this if I continue this phases, I can make all the elements below the diagonal to  $0$ , a simple elimination scheme. So, that is what we are doing. So, we take the first row divide the elements by its first element that is the diagonal elements; and second term second step will do the second diagonal element divided by second diagonal element. The second row we divided by this diagonal element and multiply it by the **the** element below that is  $j$  greater than  $2$  this is be done for  $j$  greater than  $2$  the element below and then subtract it from the column.

So, let me summarize that here. So, what we will be doing is. So, first once we make all of them  $0$  and then, we come to this one and then we will take this column and divided by a  $2/2$  multiplied by a  $3/1$  and subtract from this, now this will goes to  $0$ , this will goes

to 0 etcetera. So, once we have done that (Refer Slide Time: 43:11), we have the diagonal matrix and then, so in general that is what we will do as this **this** subtraction for all elements.

And then we have an upper diagonal matrix. As we can see, this procedure will not work if any of these diagonal elements go to 0. So, that is one example over it is bound to fail. In this process of elimination, if any of the diagonal elements go to 0 then we will not get a solution to this, because this will **(( ))** and we will not be able to work with this. So, already there is a **there is an** if there we have to make sure that, diagonal elements are non-zero.

(Refer Slide Time: 43:48)

We will then have

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$

$$a_{22}x_2 + \dots + a_{2n}x_n = b_2$$

$$\vdots$$

$$a_{nn}^{(n-1)}x_n = b_n^{(n-1)}$$

So, if you succeed in doing this without any diagonal element going to 0 and then we have a matrix like this in set of equation of like this, is the matrix would be you can see the corresponding matrix for this would be 0 below the diagonal and non-zero above the diagonal **(( ))** you have this equation. Now, you can see as I said before, we can solve this equation here  $x_n$  as  $b_n$  minus 1 divided by  $a_{nn}$ ,  $b_n$  divided by  $a_{nn}$ ; now this **this** upper index here to show that this has being changed  $n$  minus 1 times.

Each time we do a elimination, all the elements below that particular row gets changed. So, the last row gets changed  $n$  minus 1 times by this elimination process, that is what meant by this index **(( ))**. **(( ))** changed once and this will be changed  $n$  minus 1 times. So, similarly on the right hand side too.

(Refer Slide Time: 44:42)

**Back Substitution**

The above equations can be solved easily to obtain solutions of the form,

$$x_n = \frac{b_n^{(n-1)}}{a_{nn}^{(n-1)}}$$

$$x_{n-1} = \frac{b_{n-1}^{(n-2)} - a_{n-1,n}^{(n-2)} x_n}{a_{n-1,n-1}^{(n-2)}}$$

So, now we can solve this equation by back substitution. As I said, we can see written the way previous one that  $x_n$  as  $b_n$  by  $a_{nn}$ . Similarly,  $x_{n-1}$  is now  $b_{n-1}$  minus  $a_{n-1,n} x_n$  by  $a_{n-1,n-1}$  (()). But, now we know  $x_n$  from the previous solution, so we can substitute that back here.

(Refer Slide Time: 45:05)

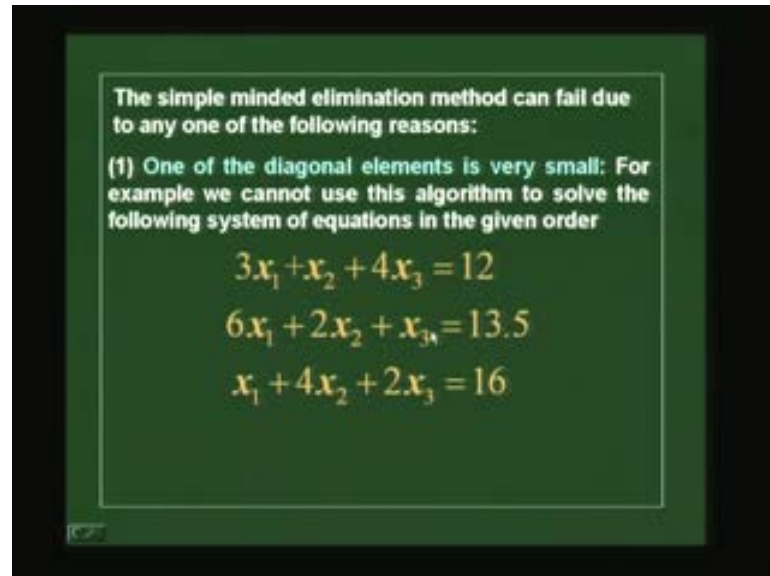
$$x_k = \frac{b_k^{(k-1)} - \sum_{j=0}^n a_{kj}^{(k-1)} x_j}{a_{kk}^{(k-1)}}$$

Will this always work? Obviously not when  $a_{kk} = 0$

So, we can write general formula for this and as  $x_k$  is equal to  $b_k$  minus 1 minus sum over  $j$  going to be 0 to  $n$   $a_{kj}$   $x_j$  by  $a_{kk}$ . So, we can do this. We have to start from  $k$  equal to  $n$  we have to we can do this solution starting from the (()). So, again will

it always work; obviously not, when you have the diagonal elements going to 0. So, we will see and implementation of this thing.

(Refer Slide Time: 45:37)



So, one of the let us take the matrix a set of equations like this, and then we could solve this equation. And see, how do we what is the solution we get, this is also various special equation that is why we shown here. That this equation this whole process also not work, if one of the diagonal elements are goes to very small value. So, there is also a (()).

In the elimination process, we may not make any elements 0 diagonal element, but the diagonal element could be go to very small value compared to other elements in the row. So, we could have other elements in the row much very large, but one of the diagonal element it could be very small; in that case, also we will have a lot of error (()) we get a solution, where solution could be very (()).

So, we could have lot of errors in the problem and (()) matrix called ill condition matrices. And if we cannot eliminate that, there are ways of eliminating as such such situations that is diagonal element going to very small value. But, we will discuss that little later. First let us let us see the implementation of what we are discussed so far, that is simple elimination scheme and to solve this equation. So, that is what we would just look at now.

(Refer Slide Time: 46:59)

```

d[0]=d[0]-(y[i]-(exp(a1*x[i])+3.0*exp
(a2*x[i])))*x[i]*exp(a1*x[i]);
d[1]=d[1]-(y[i]-(exp(a1*x[i])+3.0*exp
(a2*x[i])))*3*x[i]*exp(a2*x[i]);
}
DD=D[1][1]*D[0][0]-D[0][1]*D[1][0];
printf("%f %f %f %f\n",D[0][0],D[1][0],
D[0][1],D[1][1]);
printf("%f %f\n",d[0],d[1]);
a1=a1-(D[1][1]*d[0]-D[0][1]*d[1])/DD;
a2=a2-(D[0][0]*d[1]-D[1][0]*d[0])/DD;
printf("n = %d      a1 = %f   a2 = %
f      L = %f\n",n,a1,a2,L);
chisq_old=chisq;
:
```

(Refer Slide Time: 47:00)

```

[sunil@dali lect16]$
[sunil@dali lect16]$
[sunil@dali lect16]$
[sunil@dali lect16]$ ls
a.out          non-linear.c
eliminate.c   non-linear.data
[sunil@dali lect16]$
[sunil@dali lect16]$ vi non-linear.c
[sunil@dali lect16]$
```

(Refer Slide Time: 47:04)

```
#include<math.h>
#include<stdio.h>
main()
{
int i,j,n,k,m;
float D[3][3],d[3],x,s[3];
FILE *FP,*fp1;
n=2;
D[0][0]=3.0;
D[0][1]=2.0;
D[0][2]=1.1;
D[1][0]=6.0;
D[1][1]=2.05;
D[1][2]=1.;
"eliminate.c" 57L, 1139C      13,2-9      Top
```

(Refer Slide Time: 47:08)

```
#include<math.h>
#include<stdio.h>
main()
{
int i,j,n,k,m;
float D[3][3],d[3],x,s[3];
FILE *FP,*fp1;
n=2;
D[0][0]=3.0;
D[0][1]=2.0;
D[0][2]=1.1;
D[1][0]=6.0;
D[1][1]=2.05;
D[1][2]=1.;
"eliminate.c" 57L, 1139C      6,9      Top
```

So, we have a program here which would do that job. So, we have a 3 by 3 matrix right. So, we have a 3 set 3 unknowns  $x_1$ ,  $x_2$  and  $x_3$  and we have 3 linear equations. So, we have 3 by 3 matrix; so, 3 1 4 6 2 1 1 4 2 is the elements of the matrix. That is what (( )) in here 3 2 1 6 2 6 2 1 and 1 4 2 are the elements of my matrix, it is a 3 by 3 matrix. And the right hand side is 7.65, 13.5 and 16. So, that is what this equation is (Refer Slide Time: 47:45), so that this equation this is what I am trying to solve.

So, we have that equations here and I am just printing out the matrix here, so that we can see the matrix, in the matrix form and then I do the **I do the** elimination scheme here. So, now here is what I am starting the elimination scheme. Remember, I go from index k to 0 to n, so that is n now in my cases 2, because this is program is goes from 0 to n minus 1, so n is 2 here.

So, there 3 by 3 goes from 0 to 2 and I take all the **all the** rows. So, I take the first row here that is 0, first **first** iteration in the elimination I will have 3 rounds to do, the first round it was 0 and then I take. So, whatever the k value is start to elimination I have to make all the rows which were below thus below this to 0, all the **the** elements of that column go to 0.

So, if it is 0 and then, all elements of this 0 th column which are below this k should go to 0, that is **(( ))** k minus k plus 1 onwards, I want to make them to 0. So, I defined this  $D_{jk}$  by  $D_{kk}$ ; so, I take the diagonal elements of that particular k value for the k row, the k th row. So, I take the k th row and I take all the rows below the k th row and my idea is to make all the columns start elements in the column the k the column go to 0; all the elements of the k th column below the k the row should go to 0. So, this is the elimination process.

So, I have **I have** defined a parameter x here have which would be  $D_{jk}$  divided by  $D_{kk}$ . So, I am doing exactly what is here, so my idea is to do this. So, I am doing that **that** process here, so I have the  $D_{jk}$  divided by  $D_{kk}$  **(( ))** call  $d_{jl}$  divided by  $a_{ll}$ ;  $a_{ll} a_{jl}$  by  $a_{ll}$  this is a is matrix here and d is the matrix in this program. So, that is what it is here.

And as I multiply  $D_{ki}$  by that is the whole elements of that row by this quantity and subtracted from **(( ))** the value the whole element all element of the j th row **right**. So, what we can see x into  $D_{ki}$  would simply mean that, I take the i th element i th column element of the k th row **right** multiply it by  $D_{jk}$  that is the element the j th row the **j th row** k th row column element that is what it is.

So and I would subtract it from this one. So, I am changing the j th row all the element of the j th row by this process. And **and** you can see that, when i equal to k this will go to 0, for i equal to k this will go to 0, because will  $D_{jk}$  minus  $D_{jk}$  divided by  $D_{kk}$  into  $D_{kk}$

k right. So, this will be x will be for the i equal to k for this x would be simply  $D_{jk}$  divided by  $D_{kk}$  into  $D_{kk}$  that is will go to 0 that is that is the idea.

And then I have to change the right hand side of the equations also. Then I will print out the (( )) for every step I will print out this matrix. Just to see that, how each row by column by column how do I change this. So, once I have all the matrix elements going to 0 below the column going to 0. So, we will see that first here.

(Refer Slide Time: 51:38)

```

3.000000  2.000000  1.100000  7.650000
0.000000 -1.950000 -1.200000 -32.400002
-0.000000  3.333333  1.633333  8.349999

3.000000  2.000000  1.100000  7.650000
0.000000 -1.950000 -1.200000 -32.400002
-0.000000  0.000000 -0.417949 -102.419228

3.000000  2.000000  1.100000  7.650000
0.000000 -1.950000 -1.200000 -32.400002
-0.000000  0.000000 -0.417949 -102.419228
The solution
2  245.052155
1  -134.185944
0  0.151027
  
```

Let us let us look at this column by column. So, this is the 3 iteration (( )) 3 by 3 matrix I have 3 iterations. So, first iteration we can see that elements below the first row as gone to 0. And the second iteration, the element below the second row as gone to 0. And the third iteration, I have the element below the third row gone to 0. So, that is the procedure that is idea I will do.

So, now this is the this is the procedure, so by this (( )) I kept (( )) this is only 3 by 3 matrix. So, I already have here a a diagonal an upper diagonal matrix I have upper diagonal matrix and all the elements below that as 0, so it only (( )) do here. So, upper diagonal matrix and I have only the elements of the diagonal and the upper diagonal elements being non-zero.

And similarly, I would make the changes on the right hand side also, right hand this is the this this what is printed here is the right hand side of my linear equations. And then I

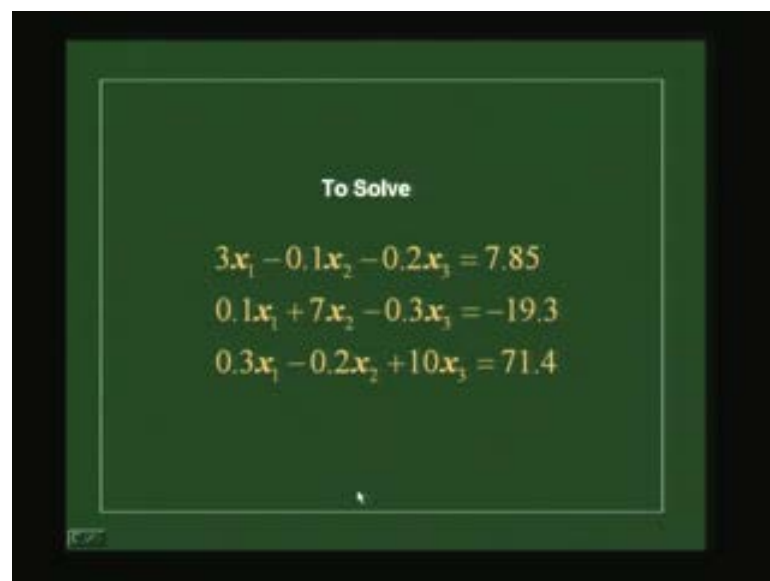


do a back substitution and I get the solutions of this form. And I am just showing there, if I substitute this back solution, back into the equation I am getting the same answer that is what I should be getting expected to get. This is actually the finding the solution of this equation.

So, back substitution is implemented here as I go from now starting from bottom. So, I start from  $n$  to  $0$ , so I go from  $n$  to  $0$  to find the solutions again steps of minus 1 that is the back substitution and I am using **I am using** this formula here which we have. Then I just simply use this formula **to get the** to get my equation in this is general formula written here (Refer Slide Time: 53:33). **(( ))** take the formula and I **I** will go back I will get the solutions here. That is the idea of the implementation of a simple elimination scheme.

So, let us try to summarize what we done before we go into what are all the different problems which can have. I will try to summarize what we have done, so in this courses.

(Refer Slide Time: 53:56)



The slide displays a system of three linear equations with three variables, presented in a yellow font on a dark green background. The equations are:

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

So, here it is, so you want to solve such an equation of this form, some equation of this form linear equation which has  $x_1$   $x_2$   $x_3$  are unknown; and we have 3 **3** linear equations I have to solve.

(Refer Slide Time: 54:12)

**Operation Performed**

$$a_{ji} \rightarrow a_{ji} - \frac{a_{ji}}{a_{i1}} a_{j1}$$

Equations	Matrix associated
$3x_1 - 0.1x_2 - 0.2x_3 = 7.85$	$\begin{bmatrix} 3 & -0.1 & -0.2 &   & 7.85 \\ 0.1 & 7 & -0.3 &   & -19.3 \\ 0.3 & -0.2 & 10 &   & 71.4 \end{bmatrix}$
$0.1x_1 + 7x_2 - 0.3x_3 = -19.3$	
$0.3x_1 - 0.2x_2 + 10x_3 = 71.4$	

And then, what we do? We write this equation in the form of a matrix equation. So, we have this equation can be written in a matrix form the associated matrix should be this, this multiplied by  $x$   $1 \times 2 \times 3$  would give me this solution.

(Refer Slide Time: 55:25)

**Operation Performed**

$$x_i = \frac{b_i^{(i-1)} - \sum_{j=i+1}^n a_{ij}^{(i-1)} x_j}{a_{ii}^{(i-1)}}$$

$7.00333x_2 - 0.293333x_3 = -19.5627$

**Results**

$x_3 = 7.00003$

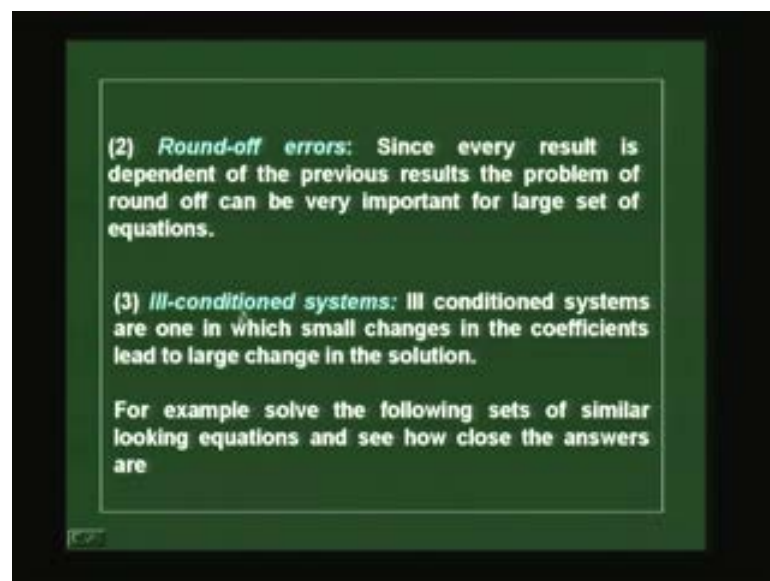
And then I will do this **this** operation that is I will subtract from this row, this row multiplied by **multiplied by** find **(( ))** divide by 3, so that will make you all of them 0. So, **that you** that you will make all of my elements in the first element row, the first column elements below the first row to 0. And then I repeat that, so that matrix would go in to

this, my equation is going in to this. And then I repeat that process again for the second row. So, that would make my equations change to that form and equation change to this form and my matrix change this form.

Now, I can solve for  $x_3$  from here we can see and then, I can substitute that  $x_3$  into this and solve for a  $x_2$  and once I have a  $x_2$  and  $x_3$  I can solve for  $x_1$ . So, that is my back substitution process, so you can you get an idea. Then, so from this I can solve for  $x_3$ . So, once I have that  $x_3$  value I could just now solve for  $x_2$  (( )) from this equation, but I have  $x_2$  equation. So, I know the  $x_3$  value I substitute that here and get the  $x_2$  equation.

And then once I have the  $x_1$   $x_2$   $x_3$  then I take the first equation which has  $x_1$   $x_2$   $x_3$  and I substitute for  $x_2$  and  $x_3$  and I get the  $x_1$  values. So, that will be the summary of the process, which we just follow. So, there are many many cases where it can actually fail. So, that is what been listed here. So, there are many cases, where it can actually fail this equation, this solution.

(Refer Slide Time: 56:09)



So, we could have round off errors coming on the way in our in our multiplication and division; or it could have ill conditioned systems, where one of the elements is going to be one of the diagonal element is going to very small value; whatever we do, we could get one of the elements going to be a small value. So, in this kind of situations we we have to do we have be little more clever and then change the change equations change the set of equations appropriately. And that is using some technique called (( )) we can

actually set the we can inter change the rows such that, the diagonal elements are always very large and that is something which you would discuss in the **in the** next class.