

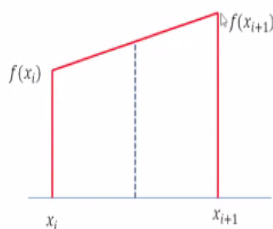
Computational Chemistry & Classical Molecular Dynamics
Prof. B. L. Tembe
Department of Chemistry
Indian Institute of Technology – Bombay

Lecture - 22
Numerical Integration and Differential Equations

Hello. We shall now continue our discussion on integration. So last time we considered Trapezoidal rule. What was Trapezoidal rule? It assumes that the function varies linearly between every two adjacent points.

(Refer Slide Time: 00:33)

Linear Interpolation/Trapezoidal rule



So look at this particular set of points. I have a function which is defined at several points. So between $f(x_i)$ and $f(x_{i+1})$ assume that is a constant. Similarly, between $f(x_{i+1})$ and $f(x_{i+2})$ there will be another line so I assume that to be constant I assume that to be linear between every two points.

(Refer Slide Time: 00:57)

- $P_1^i(x) = f(x_i) + \frac{\Delta f(x_i)}{h}(x - x_i) \quad \forall x_i \leq x \leq x_{i+1}$
- where $\Delta f(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h}$
- $I_i = \int_{x_i}^{x_{i+1}} P_1^i(x) dx = \frac{\Delta f(x_i)}{h} \int_{x_i}^{x_{i+1}} (x - x_i) dx + f(x_i)h$
- $f(x_i)h + \frac{\Delta f(x_i)}{h} \cdot \frac{h^2}{2} = \frac{h}{2} (f(x_{i+1}) + f(x_i))$
- $I = h \sum_{i=0}^{n-1} \frac{f(x_{i+1}) + f(x_i)}{2} = \frac{hf(x_0)}{2} + h \sum_{i=1}^{n-1} f(x_i) + \frac{hf(x_n)}{2}$



Once it is linear between two points my integral is nothing but between every trapezoidal the area is $\frac{f(x_{i+1}) + f(x_i)}{2} \cdot h$ between every two points it is half of the it is average of the two points multiplied by h, so between first point and the last point, so the final answer is $\frac{h}{2} f(x_0) + h \sum_{i=1}^{n-1} f(x_i) + \frac{h}{2} f(x_n)$ and all the intermediate points you have to multiply by h because both from the left side and the right side half contribution will come.

Therefore, each of this is multiplied by h. It is very easy to program this, so I will this as an exercise.

(Refer Slide Time: 01:42)

Derivation of Simpson's rule

- $I = \int_{x_{2i}}^{x_{2i+2}} P_2^i(x) dx$
- $I_i = \int_{x_{2i}}^{x_{2i+2}} \left[f(x_{2i}) + \frac{\Delta^1 f(x_{2i})}{h}(x - x_{2i}) + \frac{\Delta^2 f(x_i)}{2h^2}(x - x_{2i})(x - x_{2i+1}) \right] dx$
- Where $\Delta^1 f(x_i) = f(x_{i+1}) - f(x_i)$
- And $\Delta^2 f(x_i) = \Delta^1 f(x_{i+1}) - \Delta^1 f(x_i) = f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)$
- $I_i = h \left[2f(x_{2i}) + 2\Delta^1 f(x_{2i}) + \frac{1}{3}\Delta^2 f(x_{2i}) \right]$
- $= \frac{h}{3} [f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})]$



Next what we want to consider is what is the Simpson's rule? Simpson's rule what it does it fits a quadratic function between 3 adjacent points. Suppose I have point x_i, x_{i+1}, x_{i+2} there are these three points, these three points I will fit through a quadratic polynomial $P_2(x)$. What is $P_2(x)$? This is a Newton's forward interpolation polynomial which we have already considered. So I give here the entire expression for $P_2(x)$.

The expression for $P_2(x)$ is $f(x_i) + \frac{\Delta x}{h} f'(x_i) + \frac{\Delta x^2}{2h^2} f''(x_i)$. So this is my quadratic polynomial between x_i and x_{i+2} and that polynomial I integrate, so when I integrate these are all the steps let me define what is my first ordered difference is $f_{i+1} - f_i$. Second ordered difference is first ordered difference at x_{i+1} - the first ordered difference at x_i , when you expand everything it is $f_{i+2} - 2f_{i+1} + f_i$. This is my second ordered difference.

This is my first ordered difference. So you put all those differences here and integrate. So when you integrate my final result is $\frac{h}{3} [f(x_i) + 4f(x_{i+1}) + f(x_{i+2})]$. So that is the endpoints x_i and x_{i+2} contribute only one time whereas the middle point x_{i+1} contributes 4 time and whole thing you multiply by $h/3$. This called Simpson's 1/3 rule. So between every interval this is my integration.

(Refer Slide Time: 03:44)

$$I = \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + 4f(x_{n-1}) + f(x_n)]$$

1 2 3 4 n

$$\bullet (x_0, x_2), (x_2, x_4) \dots (x_{n-2}, x_n) \text{ or } (x_{2i}, x_{2i+2}) \text{ with } i = 0, 1, 2 \dots \left(\frac{n}{2} - 1\right)$$

• the value of n = even and the total number of data points $n + 1$ is odd.



So now let us that I split the function to many, many intervals, so let my function be x_0 to x_2 is not one interval. Why is it? x_0 to x_2 ; x_0, x_1, x_2 . I need three points to have a quadratic function

fitting there, okay. First three points one function between x_2 and x_4 that is x_2, x_3, x_4 . Another quadratic polynomial between x_{n-2} and x_n another quadratic polynomial, okay. So have these quadratic polynomial between every three points and so I want to integrate between every three points.

So since I started with three points 1,2,3 then 2,3,4 this is the next point, so you will see that you need an odd number of points you need an odd number of points to do the Simpson's rule, okay. So between 1 and 3 I will have $f(x_0)$ $4 f(x_1)$ between 3 and 5 I have $f(x_2)$ $4 f(x_3)$ and $2f(x_4)$ so the; it varies like this the first point contributes only weight of 1 the last point also has a weight of 1.

And the intermediate points will have 4 and 2, okay. So first point 0th point I have called weight of 1. Next point, this is weight of 4, third point weight of 2 and I go on doing that and multiply by $h/3$. This is my Simpson's $1/3$ rule. So the way I do it would be I will execute the program and show you how to do that. So the only problem will come, suppose n is even, if n is even you cannot implement this because there will last two points which will create trouble.

So what we do, if n is even between $n-1$ and n I use a trapezoidal rule and between 1 and $n-1$ again this is an odd number of points I shall use Simpson's rule. So whenever n is odd Simpson's rule is very straightforward. Whenever n is even the first $n-1$; now $n-1$ is odd I will use Simpson's rule and the last two points I will use Trapezoidal rule. I will illustrate with a program. So this how, I shall be using the Simpson's rule.

(Refer Slide Time: 06:05)

Simpson's Rule

- Program Execution in the next lecture



Before I conclude this session the execution will be in the next lecture, but before I conclude this let us consider differential equation briefly.

(Refer Slide Time: 06:10)

Differential equations:

- $y'(x) = \frac{dy}{dx} = f(x, y)$
- $y(x_{n+1}) = y(x_n) + hy^{(1)}(x_n) + \frac{h^2}{2!}y^{(2)}(x_n) + \dots + \frac{h^n}{n!}y^{(n)}(x) + T_{n+1}$
- Taylor expansion of a function y
- $T_{n+1} = \frac{h^{n+1}}{(n+1)!}y^{(n+1)}(\bar{x}_n), \quad x_n < \bar{x}_n < x_{n+1}$
- $y^{(k)}$ refers to the k^{th} derivative of y



So what are differential equations? In a differential equation, so this is the first ordered differential equation $y' = f(x, y)$ which is the derivative dy/dx that is given by $f(x, y)$. So I want to solve this. So how will I solve this? The strategy is again to expand y in a Taylor series. So when I expand y as a Taylor series so y at a new point x_{n+1} point is given in terms of y at the x_n point. h is the spacing between x_{n+1} and x_n .


So y_1 is the first derivative; $\frac{h^2}{2!} y_2$ is my second derivative. So I am expanding my function as a Taylor series which involves all the derivatives. And the error in truncating into the n th term I have taken up to n th derivative, the error is given by T_{n+1} this is the error in truncating the series up to n term that error is given by $\frac{h^{n+1}}{(n+1)!} y_{n+1}$ first derivative at some point in between at some point between x_n and x_{n+1} .

This is my Taylor series expansion. So y_k as I said refers to the k th derivative. We will consider the two ways of solving. One is by Euler method. What does the Euler method do? It approximates y at $x_{n+1} = y(x_n + h) = y(x_n) + h y_1$ this is the first derivative. So this first derivative is nothing but this function, so it truncates here. There is my Euler expansion.

(Refer Slide Time: 07:53)

Euler's Method

- Using the Euler's expansion up to the term for $y' = f(x, y)$
- $y(x_{n+1}) = y(x_n) + hf(x_n, y_n)$
- This is the Euler's method
- The errors accumulate to orders of h^2 and thus h has to be very small for using this algorithm



See this one now. The Euler's method you expand the Taylor series only up to the first derivative and you say at new point x_{n+1} I have $y(x_{n+1}) = y(x_n) + h f(x_n, y_n)$. So this is how I solve the differential equation. I know the derivative f is my derivative, using that derivative x_n, y_n I generate $y(x_{n+1})$. Now to generate y at x_{n+2} I already know the y at x_{n+1} and now take the derivative at x_{n+1} ; I apply the same thing to the next step. What is my next step?

y at $x_{n+2} = y$ at $x_{n+1} + h f(x_{n+1}, y_{n+1})$ extend to $n+2, n+3$ and so on I can do a number of steps, the only problem here if h is very, very large errors will accumulate severely but if h is very,

very small then you can use this algorithm, so errors are of the order of h square. So this one Euler's method.

(Refer Slide Time: 08:59)

The fourth order RUNGE KUTTA method, which is very popular is outlined below

- $y' = f(x, y)$ with
- $y_{n+1} = y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$
- Where $k_1 = h f(x_n, y_n)$
- $k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$
- $k_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$
- $k_4 = hf(x_n + h, y_n + k_3)$



There is one more very popular method which is called a RUNGE KUTTA method. I will briefly describe this and these are not very hard to program. In the Runge Kutta method, my y prime which is the derivative is given as a function of x and y . So I want to obtain an algorithm for the new y in terms of an old n ; from y_n I want to determine y_{n+1} , the formula is y_{n+1} is $y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}$, this is my Runge Kutta algorithm.

And what are these coefficient k_1, k_2, k_3, k_4 ? k_1 is $h \cdot f(x_n, y_n)$. f already is known. So k_1 is $h \cdot f$ of x_n, y_n . k_2 is $h \cdot f$ of $x_n + h/2, y_n + k_1/2$. I already know k_1 so I evaluate $k_1, k_1/2$ then k_3 is $h \cdot f$ of $x_n + h/2$ and $y_n + k_2/2$ and finally k_4 is $h \cdot f$ of $x_n + h, y_n + k_3$. So through this algorithm I can solve y_{n+1} in terms of y_n and once I get y_{n+1} I can go to y_{n+2}, y_{n+3} so I can solve it for all the remaining values that I want as many steps as I want. This is my Runge Kutta method.

So I will conclude lecture part here. Now we will start over computational part of the work. If you remember the last time, there are three things I want to do. One is fitting a function through a given set of data points. Remember, we fitted a cubic polynomial through a set of data points that is shown in my slide.

(Refer Slide Time: 10:48)

$$\begin{bmatrix} 11.0 & 5.5 & 3.85 & 3.025 \\ 5.5 & 3.85 & 3.025 & 2.5333 \\ 3.85 & 3.025 & 2.5333 & 2.20825 \\ 3.025 & 2.5333 & 2.20825 & 1.987405 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 6.023 \\ 4.28907 \\ 3.408407 \\ 2.8773135 \end{bmatrix}$$

$$a_1 = -0.000129, a_2 = 1.004383, a_3 = -0.019651, a_4 = 0.190405$$

$$P_3(x) = -0.000129 + 1.004383x - 0.019651x^2 + 0.190405x^3$$

$$f(x) = \sinh(x) = x + \frac{x^3}{3!} + \dots$$

This shown here. You will see that I had y1, y2, y3, y4 okay and I needed to determine these coefficients a0, a1, a2, a3. So determine these coefficients these were all given in terms of xi's, okay. So on 11 set of data points for the 11 set of data points first was number of data points then sum of xi sum of xi square these are all determined from our polynomial fit. This is a fitting a cubic polynomial to a set of 11 data points.

Once we fitted those data points this I need a0, a1, a2, a3. So how do I get these things? I have to invert this matrix. So if I inverse this matrix and multiply on the right by the inverse I get a0, a1, a2, a3. This is what was shown to be my set of coefficients and my polynomial will be this my a0, a1x, a2 x square and a3 x cubed. Actually this was nothing but a function sine hyperbolic function and the data points were all for the sine hyperbolic function.

Now I shall show the program how to calculate this particular set of coefficient a0, a1, a2 and a3.
(Refer Slide Time: 12:15)


```

bexec      expte.f      inputw.dat      ouputw.dat      rest.f
bexecd     fitting     integral        ouputx.dat      three.f
bt.f       frt.f        interp         outnativcorr.dat two.f
cost.f     input1.dat   klv.f         output1.dat      uj.f
cv.f       inputfit.dat knn.f          outputa.dat      output.dat
dataInt    inputold.dat matrices         output.dat

bltembe@bltembe-Inspiron-3521:~/prog$ cd fitting
bltembe@bltembe-Inspiron-3521:~/prog/fitting$ ls
a.out  fitpoly2.f  fitpoly.f  fitpoly-ok.f  input1.dat  matinsub.f  output.dat
bltembe@bltembe-Inspiron-3521:~/prog/fitting$ vi fitpoly.f
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$ +++
No command '+++' found, did you mean:
Command 'c++' from package 'clang-3.3' (universe)
Command 'c++' from package 'libc++-helpers' (universe)
Command 'c++' from package 'clang-3.4' (universe)
Command 'c++' from package 'g++' (main)
Command 'c++' from package 'clang-3.5' (universe)
Command 'g++' from package 'g++' (main)
+++: command not found
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$ =====
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$

```

So I am in my directory here. So let me do a pwd, pwd tells me the present working directory. You will see that the present working directory is I will raise it further okay. So my present working directory is home/bltembe/prog/fitting. So I want to go in the fitting function. I want to fit a polynomial through my data points. So I shall see what are the files in that; these are all the files, okay or let me just do ls.

(Refer Slide Time: 12:56)

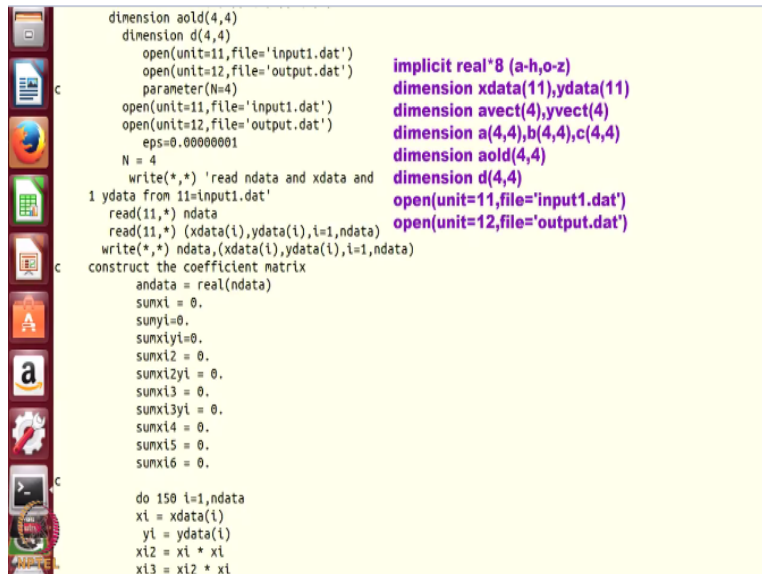
```

bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$ pwd
/home/bltembe/prog/fitting
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$ ls -l
total 60
-rwxrwxr-x 1 bltembe bltembe 20467 Feb 21 13:13 a.out
-rw-rw-r-- 1 bltembe bltembe 7377 Feb 21 13:13 fitpoly2.f
-rw-rw-r-- 1 bltembe bltembe 6840 Feb 17 20:17 fitpoly.f
-rw-rw-r-- 1 bltembe bltembe 6603 Feb 17 19:56 fitpoly-ok.f
-rw-rw-r-- 1 bltembe bltembe 7227 Jan 23 21:41 input1.dat
-rw-rw-r-- 1 bltembe bltembe 4091 Jan 23 20:40 matinsub.f
-rw-rw-r-- 1 bltembe bltembe 2581 Feb 21 13:13 output.dat
bltembe@bltembe-Inspiron-3521:~/prog/fitting$ ls
a.out  fitpoly2.f  fitpoly.f  fitpoly-ok.f  input1.dat  matinsub.f  output.dat
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$
bltembe@bltembe-Inspiron-3521:~/prog/fitting$

```

So these are all my files in that fitting directory. fitpoly.f fitpoly.f fitpoly-ok.f input1.dat this is a matrix inversion and output, okay. So I will edit that fitpoly and tell you what all that fitpoly is doing, okay. So I do I do that?

(Refer Slide Time: 13:23)



```

dimension aold(4,4)
dimension d(4,4)
open(unit=11,file='input1.dat')
open(unit=12,file='output.dat')
parameter(N=4)
open(unit=11,file='input1.dat')
open(unit=12,file='output.dat')
eps=0.00000001
N = 4
write(*,*) 'read ndata and xdata and
1 ydata from 11=input1.dat'
read(11,*) ndata
read(11,*) (xdata(i),ydata(i),i=1,ndata)
write(*,*) ndata,(xdata(i),ydata(i),i=1,ndata)
construct the coefficient matrix
andata = real(ndata)
sumx1 = 0.
sumy1=0.
sumx1y1=0.
sumx12 = 0.
sumx12y1 = 0.
sumx13 = 0.
sumx13y1 = 0.
sumx14 = 0.
sumx15 = 0.
sumx16 = 0.

do 150 i=1,ndata
xi = xdata(i)
yi = ydata(i)
x12 = xi * xi
x13 = x12 * xi

```

implicit real*8 (a-h,o-z)
dimension xdata(11),ydata(11)
dimension avect(4),yvect(4)
dimension a(4,4),b(4,4),c(4,4)
dimension aold(4,4)
dimension d(4,4)
open(unit=11,file='input1.dat')
open(unit=12,file='output.dat')

So this is my program to fit a polynomial function, okay. So what I am going to do, we already considered a matrix inversion program. So that matrix inversion program what did he do? Once you have given a matrix it will give you the inverse of that program. But here I need that matrix inversion as subroutine to this program so this program will call that matrix inversion subprogram and multiply all the y_i 's by this inverse to get a_0, a_1, a_2, a_3 that is my purpose.

So let us go through this step by step. So first line here says `implicit real*8 a-h, o-z` it means all the variables that I am going to use are in a double precision, `Real*8` means double precision, `Real*4` means single precision. Whenever you invert a matrix if all your operations are in double precision you get much more accurate results so that is why I have this `Real*8`. And I have x data 11, y data 11. I had 11 values of x and 11 values of y. So this is what is my data.

Then I give a vector a_0, a_1, a_2, a_3 four dimensions and y vector also have four values, okay. So then I also define several matrices $a(4,4); d(4,4); c(4,4)$ it is always good to define a large number of matrices which could be useful, okay. So what is the purpose? I have x data; y data I want to fit the best polynomial through my x y data. So then my input is given in file 11 so I say `open unit=11, file='input1.dat'` then `open unit=12, file='output.dat'`.

This is the parameter so sometimes you do not want to go on changing all the variables in the program, so it is good to define parameters, okay. So here there is a slight problem. I have two

open statements, one of them I should cancel, okay. So it is opening them twice, so let me just, so I have commented I do not want to open them twice, okay. So now I have defined an epsilon. Remember in the matrix inversion program you do not want to divide by a very small number.

So since 0 will rarely occur anything $<$ that 0.0000000 I think there are 1, 2, 3, 4, 6, 7 any number $< 10^{-8}$ I take it as 0 okay. So I have done this. So read that ndata and xdata, okay. So first what I want to do is to read that ndata. How many data points are there, I already know there are 11, so from file 11 it will read ndata so then once I read ndata it will read xdata i, ydata i, i going from 1 to ndata. It will read the pairs for the ndata number of points.

And to make sure whatever is read is written I write on the screen so that everything is visible. Okay. So now I construct the coefficient matrix. Remember I had a coefficient matrix which had sum of xi, sum of yi, sum of xi yi, sum of xi square, sum of xi square * yi, sum of xi cube all these sums I have so I am determining. So in Linux if you do anything wrong it will create problem so I have to now delete all these things.

(Refer Slide Time: 17:33)

```

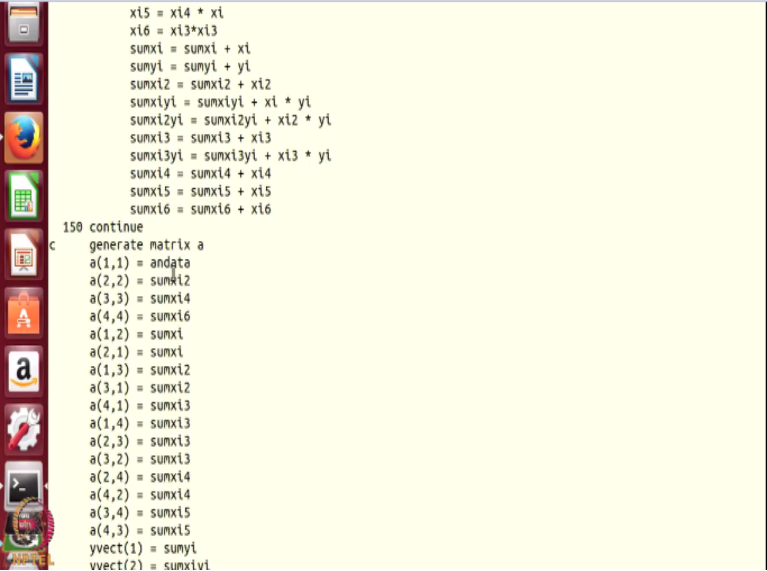
sumxi1=0.
sumxi2 = 0.
sumxi2yi = 0.
sumxi3 = 0.
sumxi3yi = 0.
sumxi4 = 0.
sumxi5 = 0.
sumxi6 = 0.
:
do 150 i=1,ndata
  xl = xdata(i)
  yl = ydata(i)
  x12 = xl * xl
  x13 = x12 * xl
  x14 = x13*xl
  x15 = x14 * xl
  x16 = x13*x13
  sumxi = sumxi + xl
  sumyi = sumyi + yl
  sumxi2 = sumxi2 + x12
  sumxiyi = sumxiyi + xl * yl
  sumxi2yi = sumxi2yi + x12 * yl
  sumxi3 = sumxi3 + x13
  sumxi3yi = sumxi3yi + x13 * yl
  sumxi4 = sumxi4 + x14
  sumxi5 = sumxi5 + x15
  sumxi6 = sumxi6 + x16
150 continue
generate matrix a
a(1,1) = andata
a(2,2) = sumxi2
a(3,3) = sumxi4

```

So I in an attempt to show something I use the mouse wrongly, okay. So I initialized all the sums I have to calculate. Now in this loop 150 i going from 1 to ndata. I will generate all these sums, okay. I will generate all these sums, okay. So for example what is xi5, xi5 is okay, let us start from the beginning xi=data I; yi is ydata i; xi2 is xi*xi; xi3*xi2*xi that is xi cube.

xi4th, xi5th xi to the 6th power then sumxi is sumxi*xi; sumyi is sumyi*yi; sumxi2 this is a sum of xi square sumxi2+xi2 so exactly; sumxi6 is sumxi6+xi6 so this sums all the 6th power of that xi, okay.

(Refer Slide Time: 18:48)



```

x15 = x14 * x1
x16 = x13*x13
sumxi = sumxi + x1
sumyi = sumyi + y1
sumxi2 = sumxi2 + x12
sumxiyl = sumxiyl + x1 * y1
sumxi2yl = sumxi2yl + x12 * y1
sumxi3 = sumxi3 + x13
sumxi3yl = sumxi3yl + x13 * y1
sumxi4 = sumxi4 + x14
sumxi5 = sumxi5 + x15
sumxi6 = sumxi6 + x16

150 continue
c
generate matrix a
a(1,1) = andata
a(2,2) = sumxi2
a(3,3) = sumxi4
a(4,4) = sumxi6
a(1,2) = sumxi
a(2,1) = sumxi
a(1,3) = sumxi2
a(3,1) = sumxi2
a(4,1) = sumxi3
a(1,4) = sumxi3
a(2,3) = sumxi3
a(3,2) = sumxi3
a(2,4) = sumxi4
a(4,2) = sumxi4
a(3,4) = sumxi5
a(4,3) = sumxi5
yvect(1) = sumyi
vvect(2) = sumxiyl

```

So once you generate all the sums I want to generate that matrix. So my matrix I have called it a, a 1,1 was 11 that is andata; a 2,2 is sumxi2 that is a 2,2 is a sum of xi square. a 3,3 is the sum of xi to the 4th this is all the matrix. Now only thing I want to see carefully, I have said I did not say a 1,1 = ndata I called it andata because whenever I have a real variable all real variables will start from a, b, c up to i, j, k, l, m, n are all integers.

So I want to start all real variables with those particular characters other than i to h. So the other than i, j, k, l, m, n. So these are all my, this my matrix, so once I have this matrix. Now I have to generate that y vector also because you know a; that y vector on the right side sumyi is the y vector 1; sumxiyi is y vector 2; sumxi square yi is y vector 3; sumxi3yi is y vector 4. So now; so I will write whatever I read I want to write on the screen so that is what it will do;

(Refer Slide Time: 20:15)

```

a(1,3) = sumx12
a(3,1) = sumx12
a(4,1) = sumx13
a(1,4) = sumx13
a(2,3) = sumx13
a(3,2) = sumx13
a(2,4) = sumx14
a(4,2) = sumx14
a(3,4) = sumx15
a(4,3) = sumx15
yvect(1) = sumyi
yvect(2) = sumxiyi
yvect(3) = sumx12yi
yvect(4) = sumx13yi
write(*,*) 'a matrix and a vector'
write(*,*)
write(*,900) a
write(*,*)
write(*,900) yvect
do 440 ii = 1,N
  do 440 jj = 1,N
440  aold(ii,jj) = a(ii,jj)
  call gausse(a,b,c,N)
write(*,*) 'the matrix a after gausse'
write(*,900) ((a(i,j),j=1,N),i=1,N)
write(*,*) 'the inverse matrix'
write(*,900) ((b(i,j),j=1,N),i=1,N)
450  continue
900  format(2x, 4E12.5)
  write inverse matrix and determinant
  write(12,*) 'determinant=',det
  write(12,*) 'the elements of the inverse matrix are'

```


Write, I have write that matrix a, and write that vector on the screen this is what I will do. So now whenever I call that program to invert the matrix it will change all the values of the matrix element therefore, the original value of a that I had I am saving in a old array, aold array just saves me; whatever is a it is saved in the matrix aold. aold just saves the values of a.

So once I have done it now I will call this subroutine. call gausse a, b, c, n. So this particular call statement it calls the subroutine to invert the matrix by gauss elimination. So I have called it gausse, a was my original matrix, b was a blank matrix, b c are not known and n is the dimension, n is the dimension, okay. So it calls that matrix program to invert my matrix and a will be an identity matrix and b will be the inverse.

Remember I use Gaussian elimination that has matrix a and matrix b. At the end of the Gaussian elimination what we got was a become an identity matrix and b is my inverse. So that is how I will call this Gauss elimination. Then these are all some write statement, I want to write all the values of a so that now a has become an identity matrix I want to make sure it is an identity matrix, b has become the inverse so I want to write it.

So at this point I write a, I write b, I also write the determinant okay. I also, all these are write statements. They are not so important but they will help you to verify what you have done.

(Refer Slide Time: 22:07)



```

write(*,*) 'the inverse matrix'
write(*,900) ((b(i,j),j=1,N),i=1,N)
450 continue
900 format(2x, 4E12.5)
write inverse matrix and determinant
write(12,*) 'determinant=',det
write(12,*) 'the elements of the inverse matrix are'
write(12,200)((b(i,j),j=1,N),i=1,N)
write(12,*)
write(12,*) 'the modified original matrix is'
write(12,200)((a(i,j),j=1,N),i=1,N)
write(12,*)
200 format(1x,4E15.6)
do 600 i=1,N
xx = 0.0
do 590 k=1,N
xx = xx + b(i,k1) * yvect(k1)
590 continue
avect(i) = xx
600 continue
do 800 ii=1,N
do 800 jj=1,N
ww = 0.0
do 790 kk = 1,N
ww = ww + b(ii,kk) * aold(kk,jj)
790 continue
d(ii,jj) = ww
800 continue


write(*,*) 'The product of inverse and original'
write(*,900) ((d(i,j),j=1,N),i=1,N)
960 write(12,*) 'the resulting avect is'

```

So at this point my x vector okay my x; so that final a0, a1, a2, a3 I get by multiplying my y vector by this b, b is the inverse of a. So when I multiply the y vector by inverse matrix I will get all my xx, xx is my a0, a1, a2, a3. So what I am doing through this loop, 590 loop I am just multiplying a column vector y by inverse of that matrix, so b is a inverse so inverse * y is my x, x is the a matrix. Okay. So that is my x.

Then once I do that I will write that xx on the screen. So I want to know what are a vector, a vector is what I am looking for. Once I have that a vector then that is my fitted polynomial, okay. So it determines to that.

(Refer Slide Time: 23:12)



```

do 590 k=1,N
xx = xx + b(i,k1) * yvect(k1)
590 continue
avect(i) = xx
600 continue
do 800 ii=1,N
do 800 jj=1,N
ww = 0.0
do 790 kk = 1,N
ww = ww + b(ii,kk) * aold(kk,jj)
790 continue
d(ii,jj) = ww
800 continue

write(*,*) 'The product of inverse and original'
write(*,900) ((d(i,j),j=1,N),i=1,N)
960 write(12,*) 'the resulting avect is'
write(*,*) (avect(i),i=1,4)
write(12,*) (avect(i),i=1,4)
write(12,*) 'polynomial at different points and the sinh funct'
write(12,*) 'var,poly(pres prog),anal fun, poly1(book)'
do 1100 i=1,11
var = real(i-1) * 0.1
var2 = var * var
var3 = var * var2
poly = avect(1) + avect(2) * var + avect(3) * var2 + avect(4)*var3
analfn = sinh(var)
poly1 = -0.000129 + 1.004383*var -0.019651 *var2 + 0.190405* var3
write(12,*) 'var,poly(pres prog),anal fun, poly1(book)'
write(12,*) var,poly,analfn,poly1
1100 continue
end

```

So then what I want to do after I get my coefficients I want to compare my fitted polynomial with the original polynomial, okay. So this is that come to this loop, what I want to do here, the rest is also interesting, I will discuss the rest; what is important once I determine all the coefficients I want now to calculate the function and see whether it matches the sine hyperbolic function.

Remember I said that the actual function which was used here is a sine hyperbolic function. So the 11 points that I have now fitted, I want to calculate in this manner. So do 1100 i going from 1 to 11. My variable, okay I want; my variable is 0.1, 0.2, 0.3; my variable 2 is; this variable is the value variable I want to calculate my polynomial at that variable, okay at 11 points in this case, okay. Now variable 2 is x squared, variable 3 x cubed.

So my polynomial which I want to calculate is a vector $a_0 + a_1x + a_2x^2 + a_3x^3$. This is the fitted polynomial, okay. Poly is the fitted polynomial and analytic function is the actual function sine hyperbolic function. Okay. So this poly is the calculated value using our program, analytic function is one calculated using the analytic function and poly1 is calculated using the coefficient which we got this coefficient we got from some material in a textbook.

So I will calculate now three polynomials. One is our fitted polynomial, one analytic function and one polynomial using the textbook polynomial and write all that in file 12, okay.

(Refer Slide Time: 25:25)


```

poly = avect(1) + avect(2) * var + avect(3) * var2 + avect(4)*var3
analfn = sinh(var)
poly1 = -0.000129 + 1.004383*var -0.019651 *var2 + 0.190405* var3
write(12,*) 'var,poly(pres prog),anal fun, poly1(book)'
write(12,*) var,poly,analfn,poly1
1100 continue
end

subroutine for matrix inversion

matrix inversion program by Gauss elimination
subroutine gausse(a(N,N),b(N,N),aold(N,N),N)

subroutine gausse(a(4,4),b(4,4),aold(4,4),N)
subroutine gausse(a,b,aold,N)
implicit real*8 (a-h,o-z)
dimension a(4,4),b(4,4),aold(4,4)
dimension a(N,N),b(N,N),aold(N,N)
parameter(N=4)
eps=0.00000001

define the identity matrix B=I
do i=1,N
do j=1,N
aold(i,j)= 0(i,j)
b(i,j)=0.0
if (i.eq.j)then
b(i,j)=1.0
end if
end do
end do
write(*,*) 'the original matrix'

```

So this is the matrix, I will just comment on the matrix inversion program, okay. So this is a matrix; gausse matrix inversion program. Remember a, b in the main program it was a, b, c here I have called it a, b, aold, N. And remember this N is a variable. This is the most important thing in my subroutine. So what I have done dimension a N, N; b N, N; aold N, N. So this means in the subroutine I have used a variable dimension because in our case it was a 4/4 matrix.

Suppose next time you wanted to do at 10/10 matrix you do not want to create a new subroutine with a 10/10. So this subroutine has a variable dimension so that it can be called by any program of any dimension. So this is a variable dimension in the subroutine and this is the most important thing. And the rest of this is entirely your matrix elimination; matrix inversion by elimination that is exactly the same. Okay.

So what I will do now I will execute this program and let us see what it does, okay.

(Refer Slide Time: 26:44)


```

bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$ pwd
/home/bltenbe/prog/fitting
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$ ls -l
total 60
-rwxrwxr-x 1 bltenbe bltenbe 20467 Feb 21 13:13 a.out
-rw-rw-r-- 1 bltenbe bltenbe 7377 Feb 21 13:13 fitpoly2.f
-rw-rw-r-- 1 bltenbe bltenbe 6840 Feb 17 20:17 fitpoly.f
-rw-rw-r-- 1 bltenbe bltenbe 6603 Feb 17 19:56 fitpoly-ok.f
-rw-rw-r-- 1 bltenbe bltenbe 7227 Jan 23 21:41 input1.dat
-rw-rw-r-- 1 bltenbe bltenbe 4091 Jan 23 20:40 matinsub.f
-rw-rw-r-- 1 bltenbe bltenbe 2581 Feb 21 13:13 output.dat
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$ ls
a.out fitpoly2.f fitpoly.f fitpoly-ok.f input1.dat matinsub.f output.dat
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$
bltenbe@bltenbe-Inspiron-3521:~/prog/fitting$

```

So I will now g Fortran. So it has compiled without any problem. So let us do ls let us see what are all the files here, my input1 data let us see vi.

(Refer Slide Time: 27:11)

```

0.3 0.3045
0.4 0.4108
0.5 0.5211
0.6 0.6367
0.7 0.7586
0.8 0.8811
0.9 1.0265
1.0 1.1752

4. 8. 2. 1.
1. 5. 3. 8.
2. 7. 1. 4.
3. 8. 2. 1.

1.000000E+01 0.496705E-08 0.596046E-08 -0.100000E+01
-0.352273E+00 -0.795455E-01 0.147727E+00 0.397727E+00
-0.170455E+00 0.284091E+00 -0.670455E+00 0.579545E+00
0.159091E+00 0.681818E-01 0.159091E+00 -0.340909E+00

0.100000E+01 0.496705E-08 0.596046E-08 -0.100000E+01
-0.352273E+00 -0.795455E-01 0.147727E+00 0.397727E+00
-0.170455E+00 0.284091E+00 -0.670455E+00 0.579545E+00
0.159091E+00 0.681818E-01 0.159091E+00 -0.340909E+00

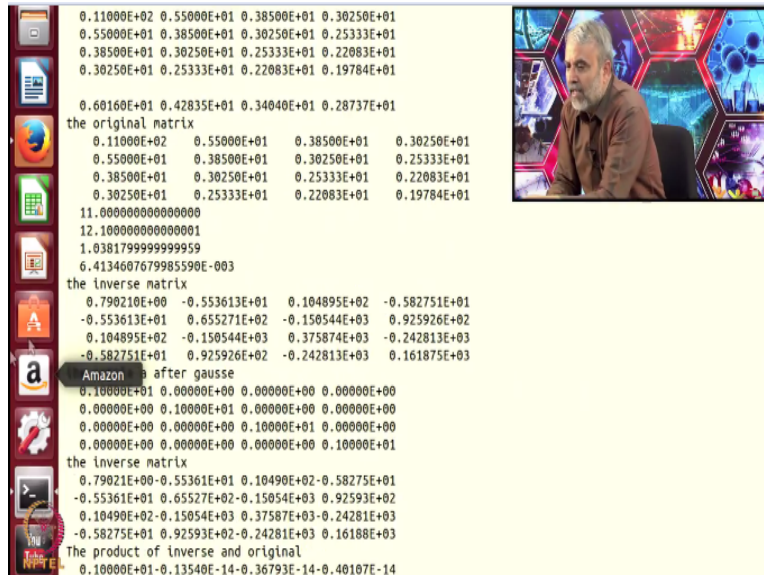
4. 8. 2. 1.
1. 5. 3. 8.
2. 7. 1. 4.

```



So remember these are all my 11 data points. First point is 0 0, second point is 0.1*0.1002 last point is 1.0 and 1.1752 so my input is intact, so I will;

(Refer Slide Time: 27:29)



```

0.11000E+02 0.55000E+01 0.38500E+01 0.30250E+01
0.55000E+01 0.38500E+01 0.30250E+01 0.25333E+01
0.38500E+01 0.30250E+01 0.25333E+01 0.22083E+01
0.30250E+01 0.25333E+01 0.22083E+01 0.19784E+01

0.60160E+01 0.42835E+01 0.34040E+01 0.28737E+01
the original matrix
0.11000E+02 0.55000E+01 0.38500E+01 0.30250E+01
0.55000E+01 0.38500E+01 0.30250E+01 0.25333E+01
0.38500E+01 0.30250E+01 0.25333E+01 0.22083E+01
0.30250E+01 0.25333E+01 0.22083E+01 0.19784E+01
11.000000000000000
12.100000000000001
1.038179999999999
6.4134607679985590E-003
the inverse matrix
0.790210E+00 -0.553613E+01 0.104895E+02 -0.582751E+01
-0.553613E+01 0.655271E+02 -0.150544E+03 0.925926E+02
0.104895E+02 -0.150544E+03 0.375874E+03 -0.242813E+03
-0.582751E+01 0.925926E+02 -0.242813E+03 0.16188E+03
Amazon a after gausse
0.10000E+01 0.00000E+00 0.00000E+00 0.00000E+00
0.00000E+00 0.10000E+01 0.00000E+00 0.00000E+00
0.00000E+00 0.00000E+00 0.10000E+01 0.00000E+00
0.00000E+00 0.00000E+00 0.00000E+00 0.10000E+01
the inverse matrix
0.79021E+00 -0.55361E+01 0.10490E+02 -0.58275E+01
-0.55361E+01 0.65527E+02 -0.15054E+03 0.92593E+02
0.10490E+02 -0.15054E+03 0.37587E+03 -0.24281E+03
-0.58275E+01 0.92593E+02 -0.24281E+03 0.16188E+03
The product of inverse and original
0.10000E+01 -0.13540E-14 -0.36793E-14 -0.40107E-14

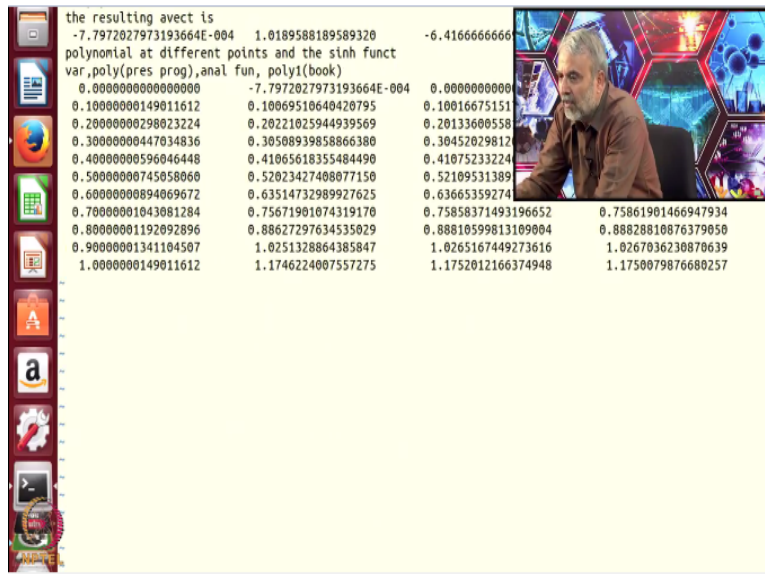
```

So now I will execute I have compiled so I will, so when I execute okay, so it has executed okay. So this is my original matrix which I which is my a matrix okay. So I have written this original matrix here, okay. So this is my determinant it is; this is my inverse matrix, okay inverse matrix.

And matrix a after gausse, see that a has to become identity matrix, b is my inverse. And what I do I also calculate; this is an important thing. I calculate the product of a and a inverse. The product should be identity matrix. So look at this last value. Last value says product of the inverse and the original should be an identity matrix. You will see that all the diagonal values are 1, all the diagonal values are; look at the lower most part in the slide.

All the diagonal values are 1. And off-diagonal values are let us say it is 0.40×10^{-14} . So in a computer 10^{-14} is as good as 0. And I get this because I have used the double precision. If I use the single precision my 0 will not be 10^{-14} but my 0 maybe 10^{-7} . So it also tells you what is the advantage of using double precision. So I will just find the output file now, okay that is output.dat vi.

(Refer Slide Time: 29:10)



So what we have done here, so this is the main result of today's work. On the left extreme I have the variables, I took 11 values of the variable from 0 to 1 and first one was the polynomial calculated using the present program. The second one is my analytic function. The third one by a program by the result a_0 , a_1 , a_2 , a_3 given in a textbook, so you will see that; this is my analytic function the third one; this is my present program results. This is my other confidants.

And they seem to agree fair well. Look at this. Analytic function says 1.1752, this is an analytic function. And our fitted function give 1.174, okay whereas the other one give 1.175. So the starting value there is a problem. The sine hyperbolic of 0 is 0. But our 0 is actually -0.0007. So the first point is not represented very well, the error is in the 4th digit, so on a computer 0.007 okay 0.007 is different from 0 by 0.007 so that is my error. So errors are in the fourth decimal places.

Look at this also 0.6366 whereas from my program I get 0.6351 whereas I get 0.6365 so there will be errors in the third and fourth decimal place, okay. But still, this is how we got a fitted function. In this case I knew that the function was an analytic function. When I do not know whether it is an analytic function I can fit using my programs and then I can get a fitted function. What is the advantage of this fitted function? I can use a fitted function to really now integrate.

Suppose I want to integrate the fitted function using a Simpson's rule. I can easily integrate using a Simpson's rule. I can calculate derivatives. I can do many, many things once I have a fitted function. So what you have shown today is that given a set of points x and y I can fit a polynomial through the point y through a ; today we discussed a third ordered polynomial. So in the next class we will do random numbers and integration.

The other two programs we will illustrate in the next class. So what I urge you, you have seen this program, compile it and execute it so that you are comfortable in this program. And this program also tells you how to use a subroutine with variable dimension. So I will conclude here. Thank you.