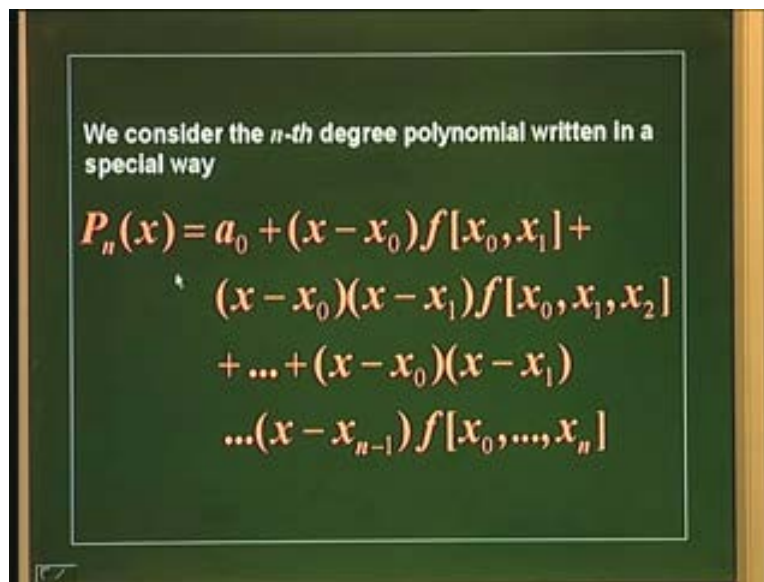


Numerical Methods and Programming
P. B. Sunil Kumar
Department of Physics
Indian Institute of Technology, Madras
Lecture - 11
Piecewise Polynomial Interpolation

Today's lecture we are going to again look at some examples of polynomial interpolation. Thus we would look at some of the implementation of newton's form which we saw yesterday. We will just start with that and then we look at the implementation of the newton's form for equally spaced data and then we look at the lagrange form of the polynomial and the implementation and then we would go on to looking at the error in this polynomials. So to start with, we revise what we did in the last classes that is, the nth order polynomial. We said that in the newton's form, we can write in this fashion that is this is the polynomial which would pass through x_1 x_2 up to x_n minus 1.

(Refer Slide Time: 02:00)



We consider the n -th degree polynomial written in a special way

$$P_n(x) = a_0 + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \dots + (x - x_0)(x - x_1) \dots (x - x_{n-1})f[x_0, \dots, x_n]$$

This is the n the order polynomial, this is the coefficient of the polynomial which this coefficients we saw can be obtained by looking at the divided difference method by the divided difference method where, we had tabulated the values at x_0, x_1, x_2, x_3, x_4 etcetera. As f_0, f_1, f_2, f_3 and f_4 and then we computed the divided differences and we stored this divided difference into the same function values, function array f_0, f_1, f_2, f_3, f_4 .

(Refer Slide Time: 02:42)

Here is the divided difference table,

x_i	f_i	$f[x_i, x_{i+1}]$	$f[x_i, x_{i+1}, x_{i+2}]$	$f[x_i, x_{i+1}, x_{i+2}, x_{i+3}]$
x_0	f_0	$f[x_0, x_1]$	$f[x_0, x_1, x_2]$	$f[x_0, x_1, x_2, x_3]$
x_1	f_1	$f[x_1, x_2]$	$f[x_1, x_2, x_3]$	$f[x_1, x_2, x_3, x_4]$
x_2	f_2	$f[x_2, x_3]$	$f[x_2, x_3, x_4]$	
x_3	f_3	$f[x_3, x_4]$		
x_4	f_4			

We saw that in the program that I could create an array called x and f these 2 arrays and then I could store this values of x and f in this value in this array and so that is 0 to 3, x_n values and f going from 0 to 3 for the y values that is, the “f” function values and then what we did was we computed the divided differences that is, f_i minus f_{i-1} for different levels.

(Refer Slide Time: 03:25)

```
f[3]=-1265.45;
n=3;
for(i=0;i<=n;i++)
{ fprintf(FP,"%f %f \n",x[i], f[i]);}
fclose(FP);
for(j=0;j<n;j++)
{
df=f[j];
for(i=1;i<=(n-j);i++)
{
df1=df;
df=(f[i+j]-f[i+j-1])/(x[i+j]-x[i+j-1]);
if(i>1)f[i+j-1]=df1;
}
}
```

22,12-19 45%

Here I, j was indicating whether it is the 1st divided difference or the 2nd divided difference etcetera. So we start with j equal to 0 that will be the 1st divided difference and then we had f_i minus f_{i-1} divided by x_i minus x_{i-1} . So that is basically

computing these quantities right, f_1 minus f_0 divided by x_1 minus x_0 , f_2 minus f_1 divided by x_2 minus x_1 etcetera. So that gives us these divided differences and then what we did yesterday was to say that, these values can be stored in the function itself.

So that is what we did here. So the function takes, this divided difference was stored into this function values the reason being that um in the polynomial construction, we need only f_1 , $f_{x_0 x_1}$, $f_{x_0 x_1 x_2}$, $f_{x_0 x_1 x_2 x_3}$. We need only these as the coefficients of the polynomial and so when I compute this divided differences here, these values I could start storing these set of values into these 4 array elements and the next set of values this 3 could be stored in these 3 elements and these 2 in these 2 elements and the last one which is not shown here could be stored in this number.

So that is exactly what we did yesterday. And then, we could print out that and once you have the coefficients in this form as f_0 , $f_{x_0 x_1}$, $f_{x_0 x_1 x_2}$ etcetera. We could compute the polynomial by using a nested form and that is what is shown here. So I start with f of 0 remember this is-, now you remember f of 0, f of 1, f of 2 and f of 3 are the coefficients of the polynomial which are the 3. The function at 0, the first divided difference at 1 and the 2nd divided difference and the 3rd divided difference and these are the coefficients and then I will write this in the nested form. We can see the nested form here.

(Refer Slide Time: 07:01)

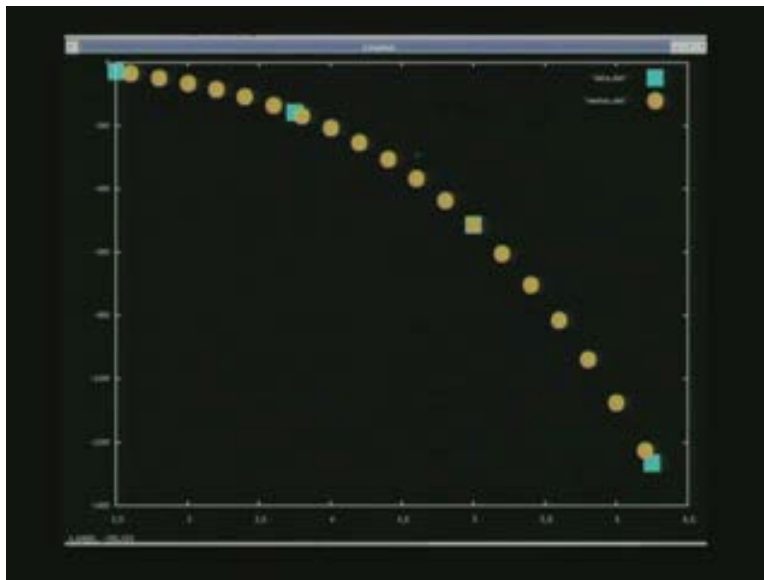
Problem1.
Find using Newtons divided different method, the polynomial of order four that passes through the points,

x	$F(x)$
2.5	-28.62
3.75	-159.36
5.0	-513.97
6.25	-1265.45

So you have x_1 minus x_0 and a bracket starting here and this bracket would be closing only at the end. You can see this matching brackets it will close only at the end and then this x_1 minus x_0 multiplies f of 1 and then, you have x_1 minus, this x minus x_1 here this x_1 is the value at which we are evaluating the polynomial. So x minus x_1 that multiplies, now another bracket starting which again closes only at the end into f_2 plus x minus x_2 into f_3 .

That is nested form which we discussed few classes ago and then we can evaluate these functions at various values for x_1 , that is what we had done. And then, we could plot this function which also we did. So we could plot this function and we could get these function values in various forms. We will just, what we wanted to do is we want to write down this functions and we are doing exactly for these set of values. So that “2.5”, “3.75”, “5.0”, “6.25” and these are the function values here and we are having a polynomial which interpolates through these points that is what we wanted to show. We can see this thing being plotted here these function values and what do we see here as this square blocks here are the values tabulated at “2.5”, “3.75”, “5” and “6.25” and this symbol, this round symbols here, these symbols are the interpolated values and you can see that it actually goes through this polynomial very well.

(Refer Slide Time: 07:33)



So it does not matter what the axis here is, what we have to see is that this interpolated polynomial actually goes through these set of points which we have mentioned here, the square being the function values given to us and round symbols being the interpolated values and they go through that, that is what we saw. So, we could also look at other problems which we look at a little while later in using other methods when we compare different polynomials, we would these functions also to compare them. we could for example, take a functional form and evaluate this function between 1 and 8 for example, and construct a set of discrete points and then construct a polynomial which goes through that, that tells us something about our error in interpolating polynomial. It will be a useful thing to learn.

(Refer Slide Time: 08:30)

Problem1.
Find using Newtons divided different method, the polynomial of order four that passes through the points,

x	$F(x)$
2.5	-28.62
3.75	-159.36
5.0	-513.97
6.25	-1265.45

Problem2.
Find a sixth order polynomial approximation to $\log(x)$ between $x=1$ and $x=8$ using newtons divided difference method. Choose different values of x_i , $i=1$ to 8 in the interval $1 < x < 2$ and compare the function values at $x=1.43$

(Refer Slide Time: 08:51)

```
Newton.c
#include<math.h>
#include<stdio.h>
main()
{
    int i,j,n;
    float x1,x[5],f[5],y,df,df1;
    FILE *FP;
    FP=fopen("data.dat","w");
    x[0]=2.5; x[1]=3.75; x[2]=5.0; x[3]=6.25;
    f[0]=-28.62; f[1]=-159.26; f[2]=-513.97;
    f[3]=-1265.45;
    n=3;
```

(Refer Slide Time: 09:01)

```
for(i=0;i<=n;i++)
{
    fprintf(FP,"%f %f \n",x[i], f[i]);
}
fclose(FP);
for(j=0;j<n;j++)
{
    df=f[j];
    for(i=1;i<=(n-j);i++)
    {
        df1=df;
        df=(f[i+j]-f[i+j-1])/(x[i+j]-x[i-1]);
        if(i>1)f[i+j-1]=df1;
    }
}
```

(Refer Slide Time: 09:11)

```
f[n]=df;
printf("%d %f \n", j, f[j+1]);
}
FP=fopen ("newton.dat", "w");
for(x1=2.6;x1<=6.25;x1=x1+.2)
{
    y=f[0]+(x1-x[0])*(f[1]+(x1-x[1])
        *(f[2]+(x1-x[2])*f[3]));
    fprintf(FP,"%f %f \n",x1, y);}
fclose(FP);
}
```

Now, look at the same newton's form but now, with equal intervals remember this form of the polynomial it is again newton's form but, for the data is evaluated or the data we will have tabulated at equal intervals of x . So "s" here is actually x_s , x_s is the point where you want to evaluate the polynomial x_s minus x_0 divided by "h" where h is our spacing between the data points, so the "s" written here is " x_s ".

(Refer Slide Time: 09:21)

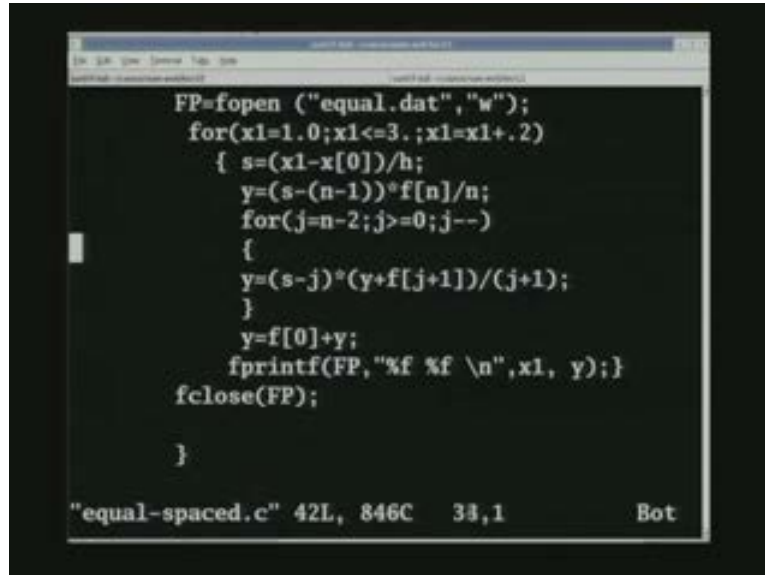
$$P_n(x_s) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_0 + \dots + \frac{s(s-1)\dots(s-n+1)}{n!}\Delta^n f_0$$

where $s = (x_s - x_0)/h$, with $h = \Delta x$, the uniform spacing in x -values.

x_s is the point at which we want to evaluate minus x_0 , x_0 is the first data point which we have tabulated divided by h , where h is Δx given that is the difference distance between the two tabulated values and that is same for all values it is unlike the previous case, where we had “2.5”, “3.75”, “5”, “6.25”. So this is, we could, so here also we actually have equal space but we did not use that. For example, this is a good example of that “3.75” minus “2.5”, “one point two five” “1.25” etcetera. So we could use the same data to actually get used and to use the same data and actually use this functional form, how the polynomial and obtain a polynomial and compare that, so we will see that in this here. So, that functional form so we call that as equal space dot c.

So this is the program which should implement that particular quantity again we have a set of an array which is x and the function f , we store them in this array. Here, what I have done is, I have taken the log x program, the log x function and I created this function. So I have, I am going back and doing this particular problem using equal space data, this particular problem, that is I take log x function log base 10 x and I evaluate that function between 1 and 8 in some set of discrete intervals and then I would get an interpolating polynomial through that. So, I just create this set of discrete points at equal intervals h , where h I choose them to be “.5”. So I evaluate between 1 and 3 in this case, between 1 and 3 I evaluated a certain number at every point x equal to “.5” and then I could and then, I have a set x of i 's x_i 's and f of i is x_i and f_i this will be the tabulated value then.

(Refer Slide Time: 11:04)



```
FP=fopen ("equal.dat", "w");
for(x1=1.0;x1<=3.;x1=x1+.2)
{ s=(x1-x[0])/h;
  y=(s-(n-1))*f[n]/n;
  for(j=n-2;j>=0;j--)
  {
    y=(s-j)*(y+f[j+1])/(j+1);
  }
  y=f[0]+y;
  fprintf(FP,"%f %f \n",x1, y);}
fclose(FP);
}
```

"equal-spaced.c" 42L, 846C 33,1 Bot

So then, I am writing this in this particular thing on to the screen the x_i and f_i values and then I would go and construct that divided difference. So here, it is the same as the newton's form as I said, so we will use the same divided difference method, no difference the only change being that, in the earlier case in the newton's method when we had a set of discrete points, we had them at arbitrary intervals and the arbitrary intervals the divided interval were made as f of i plus 1 minus f of i or f of i minus f of i minus 1 divided by x of i minus x of i minus 1. We have to divide it by the spacing of the data but since here, it is all equally spaced as we saw all that, the spacing the Δx that is x_i plus 1 minus x_i is absorbed into this function here and we write that s as x_s minus x_0 into h .

So Δf_0 is simply remember is f of i minus f of i plus 1. So that is, for the evenly spaced data, so we will have just only Δf of i minus f of i minus 1 and these are further differences of the divided difference that is the 2nd order differences and the 3rd order differences etcetera. So that is what exactly, what we are doing here. So again we have like in the newton's polynomial case starting with j equal to 0. So remember this all this points are exactly the same, once we have constructed the function here, the discrete set of points from this function $\log x$ we have constructed a discrete set of points x of i and f of i and then, this part of the program starting here is exactly the same as what is in the previous case of newton's polynomial for except that the divided difference we do not divide this f_i minus f_{i-1} by the interval x_i minus x_{i-1}

Instead of that we just simply construct, the divided differences as the difference between the function values. Again, so let us start with j equal to 0 that is the first divided differences and i will go from one to n ; i will go from 1 to n since j is 0 in steps of 1 and we store the first function value at j equal to 0 in the $d f_1$ and then we construct df that is df as the difference between 1 and 0 f of 1 and f of 0 and then we go back, since this is i greater than 1 this line is executed only at i greater than 1 so we go back here .

So we go to i equal to 2 and we come back here and the 1 which we have computed the value which we computed as f of 1 minus f of 0 is now stored into $d f 1$. So remember $d f 1$ is now f of 1 minus f of 0 and then we go to the next line where we compute now since, i is 2, f of 2 minus f of 1 and then we store this value which was f of 1 minus f of 0 into now f of 1, j is 0. So this is now one, i is 2 and j is 0. So i minus 1 is 1, so f of 1 now, stores the 1st divided difference that is f of 1 minus f of 0 into that and we continue the loop and the next f of 3 minus f of 2 will be stored in f of 2 and f of 4 minus, f of 5 minus f of 4 will be stored in f of 4 etcetera.

So we will use just like what we did earlier, we just store all the divided differences into the function f and then we change j value, now we go to j to 1 that is the 2nd divided difference and because, we have stored all the 1st divided differences into the function now, we do not have to worry we just have to take the difference between f again. So that is the advantage I would show you this graph this plot again so this is exactly the same as this so now all the distances are equal between x_0, x_1, x_2, x_3, x_4 . So what we did again was computing the difference between these functions and storing them, computing the difference between these functions which are f of $x_0, x_1, x_1, x_2, x_2, x_3, x_3, x_4$ and storing them in f_1, f_2, f_3, f_4 that is what we have done in the j equal to 0 loop just now. Now we are going to change j equal to 1.

When you change j equal to 1 again we would continue the same thing that is j equal to 1, i equal to 1 and i equal to 1. we will compute now f of now 2 j is 1, so f of 2 minus f of 1, that is what are going to compute, that is taking f of 2 minus f of one which is since, f of 1 had this and f of 2 had this it is actually taking f of x_1, x_2 minus f of x_0, x_1 which is nothing but f of x_0, x_1, x_2 . So that is again being stored in this value here in the 2nd time loop, 2nd loop, i equal to 2 we come back here and store, now i equal to 2 we will compute remember when i equal to 2 we store this f of 2 minus f of 1 into $d f 1$ and compute f of 3 minus f of 2 now j is 1, i is 2, so this is 2 j is 1, i is 2, so this is 3, so this is f of 3 minus f of 2 and then, we come to the next step and store the 1 which we computed as f of 2 minus f of 1 into f of 2 that is, i is 2, j is 1, so f of 2 is what it is so. We will store it in this value.

So that is saying that, this function now has gone into f of 2 and the next difference will go that is this will go into f of 3 etcetera. So we will compute exactly the same thing, so we get all the divided differences at the end again as $f_1, f_0, f_1, f_2, f_3, f_4$. So we get f_0, f_1, f_2, f_3, f_4 as the divided differences and then we can use the nested form again like in the newton's form. So only so far between the equally spaced data and the one, the unequal spaced data in the newton's form is the construction of the divided difference where we will not divide the function by x_i minus x_i plus 1 plus x_i that is the only difference and then we have the nested form again in this form.

So now, the nested form is slightly different, now coming to the nested form, the nested form here is slightly different because it is this form, you are going to use because this functions now involves the value at which we have to evaluate the polynomial. So in that sense it is slightly different, it is now the coefficients of the polynomial now has to be evaluated at every value we want to get in that is, we have to get this s value computed at

every-, so this s value has to be computed at every point where we want to evaluate the polynomial. So we have just completed the calculation of Δf_0 , $\Delta^2 f_0$, $\Delta^3 f_0$, $\Delta^4 f_0$ but these has to be computed at every point because it is x_s minus x_0 by h where x_s is the value at which we want to evaluate the polynomial.

So that is what is done here. We can see that, we can see that “ s ” now, we were going to evaluate the polynomial starting from 1 to 3 at intervals of “.2” we tabulated-, we have tabulated values at the intervals of “.5” but, we want the polynomial in between the interval, so we are going to evaluate that in intervals of .2. So s is as I said now, x in this thing is now x_1 here, so x_1 minus x_0 by h is your “ s ” as x_1 changes s also changes and then, we would say that the function value, we start with the first term, the first term is-, the first term is just f of 0 as you can see that the first term is just f of 0 and the 2nd term is s into Δf_0 that is f of 1. So that is f of 1 is my Δf_0 f of 2. Remember, I stored these things in to my function values so this is my f of 1 and this is my f of 2 and this is f of 3 etcetera. That is what is been done here?

you start from, start with f_0 plus this value that is s into, so we have chosen our y value, to be start with this that is s times f of, f of n by n that is what we are going to do, here. And then we are going to go backwards, so what we will do is we are going to start from here, and then we go backwards and complete that so our n in this particular case is equal to we will be having 4 points, we have 7 points. So n is equal to six let us say, so we would start from that value, so fist we start with the n th term or the last term here.

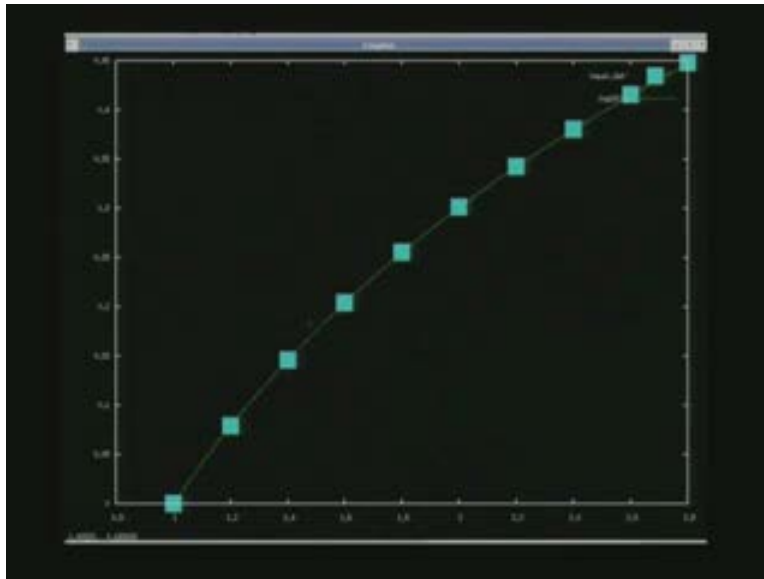
So we will take f of n remember this is, f of 1, f of 2, f of 3 and this is f of n , so we will start with the last value and that has to be divided by n factorial, so that is what we have to do. So now this is going to be remember, n factorial is 1 into 2 into 3 up to n . Now I am going to do this in a nested loop, so I am going to take out, I am going to make this n now this gets multiplied, I take only s minus n plus 1 divided by n and that gets multiplied by s minus n plus 2 divided by n minus 1 etcetera, as we go further that is what the loop here is This is a nested loop but, we start from the highest order term, so s into n minus 1 that is s minus n plus 1 into f of n by n that is the first value which we have taken which is exactly the same as this 1 which we take.

So s minus n plus 1 by n instead of n factorial into the f of n that is, what this is. So that is the start with and then I run a loop starting from n minus 2 to 0, j equal to n minus 2 to 0 and then go step by step, dividing it each time adding to that function the next f value that is f n minus 1 so j is n minus 2, so i add to this value here, f n minus 2 and divide it by n minus 1, i just divide by n minus 1 remember this was divided by n the next is n minus 1. So since, f is in the bracket this function also gets divided by n minus 1 it is already n into n minus 1, so by the time I complete this loop, I have n factorial for the 1st term n minus 1 factorial for the 2nd term n minus 2 factorial for the 3rd term etcetera, and the last I just sum up the function to f of 0 that is this term.

So that will give me the full value and I just print it out into a file, here called equal dot dat. So that is what this program is? So summarize this program I have taken the $\log x$ function, I have taken the $\log x$ function and then I have tabulated at from 1 to 3 in

intervals of “.5” and then I wrote that and I stored that into a file into an array of x of i and f of i and then, I use that f of i to compute my divided differences without dividing it by x of i minus x of i minus 1 because, it is equally spaced and then, I use a nested loop to compute once I have the divided differences, I use a nested loop to compute my polynomial at different values starting from 1 to 3 at intervals of “.2” and I wrote that into a file here called equal dot dat.

(Refer Slide Time: 26:39)



I opened a file here and I wrote that in to a file and I closed the file here that is the program for computing this thing, now we will go back, we will go and just run this program and then see what do we get. So we will run this program, we need a math library there so that is, and then we would run this program, so that is the values at which I have tabulated this thing so I have tabulated it at 1, values at “1”, “1.5”, “2”, “2.5”, “3” etcetera. So this is the value at which I tabulated the function.

So 1,2,3,4,5 data points starting from 0 to 4, so that is the function value at this point and this is the, these are the divided differences I got, the coefficients basically of the polynomial that is Δf_0 , $\Delta^2 f_0$, $\Delta^3 f_0$ etcetera, are also listed here as “.176”, “.0-”, “.0511” minus “.023” etcetera, these are the coefficients and then I would just run a program with this. I just plot those the data points. This is the data points plotted against the line which we can see here, the line which we can see here is the log x function that is the function which we had and this square symbols here, these square symbols here are the interpolated values and you can see that the interpolated value actually goes over the function quite well. so interpolation works pretty well.

(Refer Slide Time: 27:00)

```
Equal-spaced.c

#include<math.h>
#include<stdio.h>
main()
{
    int i, j, n;
    float x1, x[15], f[15], y, df, df1, h, s;
    FILE *FP;
    h=0.5;
    i=0;
    for(x1=1.0; x1<=3.0; x1=x1+h)
    {
        x[i]=x1;
        f[i]=log10(x1);
    }
}
```

(Refer Slide Time: 27:10)

```
printf("%d %f %f\n", i, x[i], f[i]);
i=i+1;
}
n=i-1;
for(j=0; j<n; j++)
{
    df=f[j];
    for(i=1; i<=(n-j); i++)
    {
        df1=df;
        df=f[i+j]-f[i+j-1];
        if(i>1) f[i+j-1]=df1;
    }
}
```

(Refer Slide Time: 27:19)

```
f[n]=df;
printf("%d %f \n",j, f[j+1]);
}
P=fopen ("equal.dat","w");
for(x1=1.0;x1<=3.;x1=x1+.2)
{
s=(x1-x[0])/h;
y=(s-(n-1))*f[n]/n;
for(j=n-2;j>=0;j--)
{
y=(s-j)*(y+f[j+1])/(j+1);
}
y=f[0]+y;
fprintf(FP,"%f %f \n",x1, y);
fclose(FP);
}
}
```

So we should see, so we saw now two different methods of doing this. So, one is the using the divided difference newton's formula and this is the newton gregory formula in which we have equally spaced data. The difference being the way which we would implement this, implement the nested loop is slightly different in this case you would start from the bottom and then we would go up, up to this value computing s for every value which we want to evaluate.

(Refer Slide Time: 28:03)

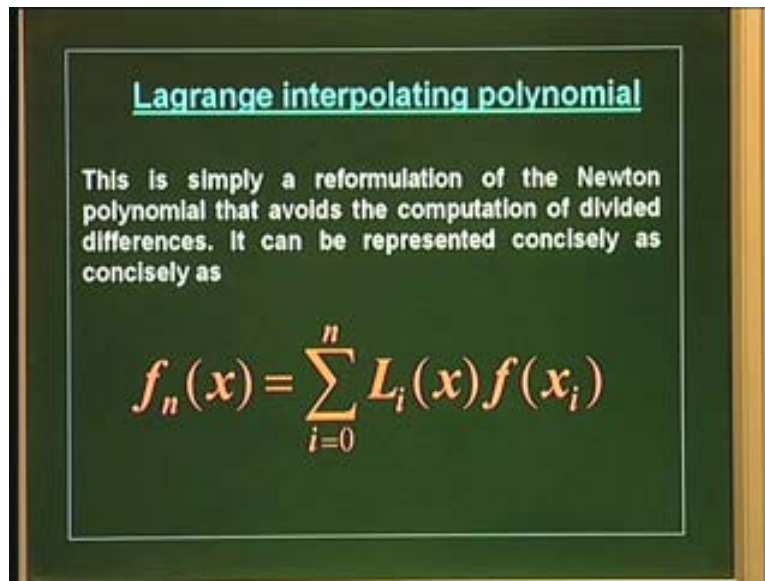
$$P_n(x_s) = f_0 + s\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_0 + \dots + \frac{s(s-1)\dots(s-n+1)}{n!}\Delta^n f_0$$

where $s = (x_s - x_0) / h$, with $h = \Delta x$, the uniform spacing in x -values.

In the case of the newton's form you do not need that this is actually have once you have the polynomial functions, we can simple evaluate the function for every value of x. So

this is a slightly different nested form otherwise, it is similar and again the divided differences are computed only by taking the difference between the function and the divided difference, the difference between the first divided differences etcetera, that is the form of this. so now, now will look at the next form that is, the lagrange interpolation formula.

(Refer Slide Time: 28:41)

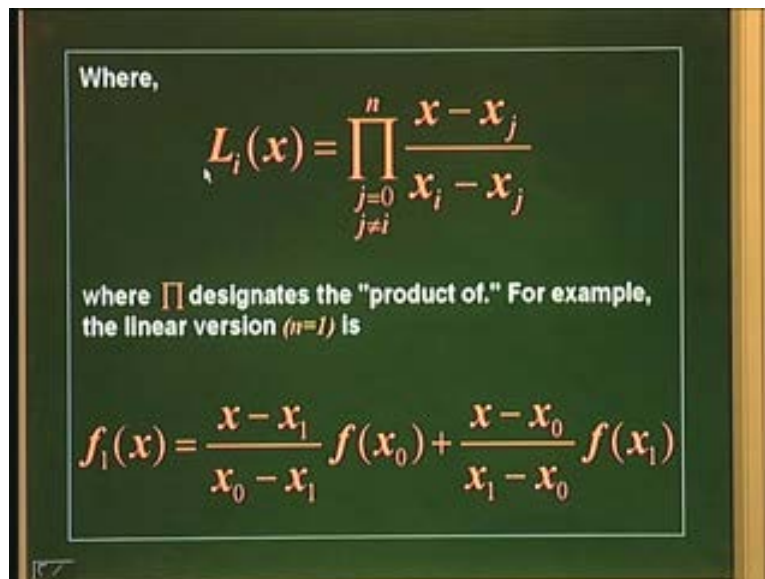


Lagrange interpolating polynomial

This is simply a reformulation of the Newton polynomial that avoids the computation of divided differences. It can be represented concisely as

$$f_n(x) = \sum_{i=0}^n L_i(x) f(x_i)$$

(Refer Slide Time: 29:18)



Where,

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}$$

where \prod designates the "product of." For example, the linear version ($n=1$) is

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

We again see an implementation of this lagrange interpolation formula now, and we said that in this case it has been represented by f_n of x here we said that this can be written as i going from 0 to n and L_i is some operator L_i of x here into f of x_i . Now this function, L_i of x

is of order n all the time, so that is what which we saw, if you write it for just the first order term, it will be simply written by this.

So l_i of x remember, r is the product going from j equal to 0 to n of x minus x_j divided by x_i minus x_j , we saw the derivation of that in the last class and so we can this product runs over all the values from j equal to 0 to n except for j equal to i so here n , this is n plus 1 quantities and then so n plus 1 minus 1 , so there are n products, n terms here. So for n equal to 1 , it is just 1 that is, let us say you have a polynomial n equal to one that is going through x_0 and x_1 and then you will have x minus x_1 by x_0 minus x_1 as the 1st term and the 2nd term would be x minus x_0 divided by x_1 minus x_0 as the 2nd term and you can see that at x_1 f of x , x equal to x_1 this will give me the f of x_1 and x equal to x_0 this will give me f of x_0 .

So, that is the basic idea of the lagrange interpolation thing, the formula is that every term is a functional value at that, at that i , is if i write it as f of x_i , so now this is the 2nd term the 2nd order term would be x minus x_1 into x minus x_2 by x_0 minus x_1 into x_0 minus x_2 and continues up to 2 , there will be 3 terms in this each of them is a 2nd order polynomial, in the case in n equal to 2 each of them is a 2nd order polynomial, in the case of n equal to 1 each of them, each of the terms is the 1st order polynomial and now if n equal to 3 there will be 4 terms, each of them would be 3rd order polynomial etcetera and the idea being that each term would give me the function value each $l_i f_x$ into f of, will give me a function value at that i , so that is the whole idea of this thing. So we have to again go back and then look at the implementation of this particular formula in this. That is what we will now see.

(Refer Slide Time: 30:43)

and the second-order version is

$$f_2(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} f(x_2)$$

(Refer Slide Time: 31:49)

```
#include<math.h>
#include<stdio.h>
main()
{
int i,j,n;
float x1,x[15],f[15],y,df,df1,x1;
FILE *FP,*fp1;
x[0]=2.5;x[1]=3.75;x[2]=5.0;x[3]=6.25;
f[0]=-28.62;f[1]=-159.26;f[2]=-513.97;f
[3]=-1265.45;
n=3;
FP=fopen ("lagrange.dat","w");
for(x1=x[0];x1<=x[n];x1=x1+.1)
{
"lagrang.c" 31L, 676C          1,2-9          Top
```

So here is the program which would implement this new formula, this formula that is the lagrange formula that is saying that the function value is equal to f of i the function is at any value, at any point x is equal to l_i of x into f of x_i , i going from 0 to n . so now you remember this, l_i goes from 0 to n l_i of x and f of x_i . So when, if you have a polynomial which goes through n , let us say 3 n equal to 3, we will say 3 points or 4 points that is the example, which we are going to see it goes through 4 points and then there will be 5 terms in this, it goes from 0 to n , 0 to 4 it will go that will be 5 terms in this formula and each of them l_i being given by this product x_i minus x , x minus x_j divided by x_i minus x_j for j going from 0 to n for j is not equal to i that is the implementation which we are going to see

so what we have to see is the polynomial has l_i of x into f of x_i , sum over i l_i of x into f of x_i , i going from 0 to n that is what we want to see the implementation of that. So we again have the program here with 2 arrays, x of 0 and x and f , where x is the tabulated values we are using the same, same data points as we had done before and then f of 0 so that is, f of 0, 1, 2, 3 that is the function value tabulated. So that is what we have, the function values tabulated that is n is 3 in this particular case and then, so now we are going to store the result from this into a data file called lagrange dot dat that is what we are going to do and again we are going to evaluate this polynomial at various, starting from x of 0 to x of n in steps of .1.

That is what we want to do. And again, so this is now a slightly different implementation, so it is not a nested form or something like that now the point here is that at every x value which we want to evaluate the polynomial, we have to compute this product, so every x value for, every i , we have to compute this product because this product is different from every x value. So we cannot evaluate a set of coefficients and then compute the polynomial at every x but we have to actually compute the polynomial at every x value.

So it looks difficult but it is very, as we see in the program it is much more easier to implement in this particular case that is what we see here. So we want to evaluate it at let us say x_1 equal to x of 0 or x we are going to get, some value x_1 starting from x of 0 to x of n in steps of ".1"

so we start with y of-, for every x value we start with the function value is finally going to be written into y . So we start with y equal to 0 and then, we say that we go from i equal to 0 to n , so that is we have to do that-, as you remember the function value is $\sum l_i f_i$, l_i of x into f_i of x or f of x_i . So $\sum l_i f_i$ is the value which you want to actually finally obtain, so that i loop goes from 0 to n as we saw it in steps of 1 and then we want to compute the product, so the product is actually would finally come out as \prod which i have initialized as 1 and then i do the j loop here.

So the j loop is this 1 so i want to evaluate this product, so i just 1st put this as 1 and i go from j equal to 0 to n except for j equal to i and i multiply x minus x_j divided by x_i minus x_j where x is the value which we want to evaluate. since is very simple to implement, so x_1 is the value at which you want to evaluate so x_1 minus x_j divided by x_i minus x_j is the product for all j values except j equal to i this is-, this notation is j not equal to i so some programming detail here j not equal to i , if j not equal to i , so this loop is executed only when j is not equal to i otherwise j goes from 0 to n and i initialize this function to one value and now i get the product here x_i minus x_0 divided by x_i minus x_0 x_1 minus x_0 divided by x_1 minus x_0 x_1 minus x_0 and then-, so if i start with i equal to 0 and j equal to 0 is not executed because j is equal to i so j starts from 1 so I will get x_1 minus x_0 divided by x_1 minus x_0 and then x_1 minus x_0 divided by x_2 minus x_0 , x_0 minus x_2 etcetera. So all j values, so you get all the product. So and then, I multiply that product by the functional value at i and then I run from i goes through 0 to 1 and then I have the final function y , so that is the sum, the sum is here the product is here. We 1st evaluate the product that gets me the l_i of x and then I take the sum multiplying the l_i of x with the f of i and i sum it up here and then I get the function value at that x_1 and i tabulate I write it into the file here. That is the file.

So I can do this, so I put it into Lagrange dot dat and we just plot that again and then see. So here is, this is slightly different we just evaluate we just compute this. So that was the one remember data dot dat was where we stored the function value the tabulated values and this is the function values. So again the green points here the green squares which you can see here are the tabulated values and this round points are the interpolated values, you can see that it goes through this very well and you can see that it is the same as, it will give the same accuracy or same result as what we got from the newton's polynomials there is no difference between these two.

(Refer Slide Time: 39:23)

```
Lagrang.c
#include<math.h>
#include<stdio.h>
main()
{
    int i,j,n;
    float x1,x[15],f[15],y,df,df1,xl;
    FILE *FP,*fp1;
    x[0]=2.5;x[1]=3.75;
    x[2]=5.0;x[3]=6.25;
    f[0]=-28.62;f[1]=-159.26;
    f[2]=-513.97;f[3]=-1265.45;
    n=3;
    FP=fopen ("lagrange.dat","w");
```

(Refer Slide Time: 39:33)

```
for(x1=x[0];x1<=x[n];x1=x1+.1)
{
    y=0.0;
    for(i=0;i<=n;i++)
    {
        df=1.0;
        for(j=0;j<=n;j++)
        {
            if(j!=i)
            {
                df=df*(x1-x[j])/(x[i]-x[j]);
            }
        }
    }
}
```

so we see that we can actually evaluate polynomials we can get the function values or the, we can interpolate given set of tabulated values using we saw 3 different methods for equally spaced data 1 and for unequally spaced data 2 methods that is, we saw the newton's method and the Lagrange's method.

(Refer Slide Time: 39:45)

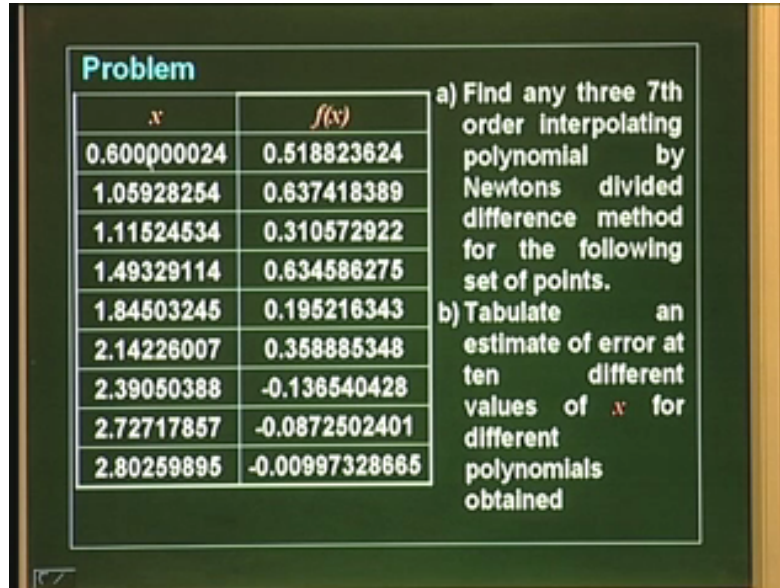
```
y=y+df*f[i];
}
fprintf(FP,"%f %f \n",x1, y);
}
fclose(FP);
}
```

So now, does it mean that if you give me any interpolated any set of discrete function that I can evaluate this function anywhere. So is that possible all the time, so that is the question which we have to ask and we will find that this is not always true there will be always an error in the polynomial and that is something which I can demonstrate here to you, so I will show you this in program that, so here is a program which is similar to again using the lagrange's method but I am using another set of data points here. So I choose another set of data points which is tabulated in this to inter dot data this data file which is actually something which we have it here. I will show you the data file.

(Refer Slide Time: 40:47)

```
#include<math.h>
#include<stdio.h>
main()
{
int i,j,n;
float x1,x[15],f[15],y,df,df1,x1;
FILE *FP,*fp1;
//fp1=fopen("data.dat","r");
fp1=fopen("tointer.data","r");
i=0;
while(!feof(fp1))
{fscanf(fp1,"%f" "%f",&x[i],&f[i]); i++;
}
fclose(fp1);
"lagrang1.c" 38L, 828C      3,2-9      Top
```

(Refer Slide Time: 41:16)



Problem

x	$f(x)$
0.60000024	0.518823624
1.05928254	0.637418389
1.11524534	0.310572922
1.49329114	0.634586275
1.84503245	0.195216343
2.14226007	0.358885348
2.39050388	-0.136540428
2.72717857	-0.0872502401
2.80259895	-0.00997328665

a) Find any three 7th order interpolating polynomial by Newtons divided difference method for the following set of points.

b) Tabulate an estimate of error at ten different values of x for different polynomials obtained

So I want to, I want to interpolate this particular set of data, so I have this data file here and I want to write an interpolation polynomial to get the values in between this. So that is something we should, we can try I have entered these set of data into a file, so that file is I called it to inter dot data, so this file contains all this this tabulated values of x and f of x and what does this program do this program first read that file so, again I wanted to show you here a way to read the files off so I want to read this file so i have used this fscanf function to read that file.

So first I opened the file called to inter dot data and then I go from and I read that file using the fscanf. I set the while it reads that line by line till it finds a end of file statement there. So every file will have an end of file character so till it finds an end of file character it will keep reading, so this while loop here closes at this point so, this is a while loop. While loop has only one function to be executed one statement to be executed that is to scan this file called fp 1, where fp 1 is the to inter dot data so that is how you open a file and you could just read that off and while reading it also counts the number of lines that is done by the i equal to 0, there is other ways of getting the number of lines in a data file but this is a very simple way of doing it i equal to 0 we have initiated it and I am just scanning that thing till it finds the end of file character and I increment i at every time and after I read this file I close the file .

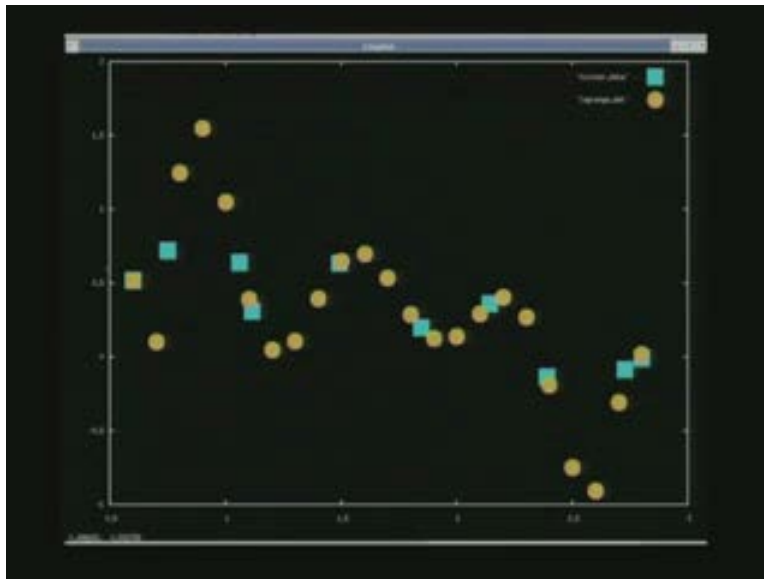
It over counts it by 2. So I just set my number of data points as i minus 2. So that is because it starts from 1 in counting, so I set it by 2 that is the number of, that is my number of data file the data points which I have and in this file, to inter dot data, so then I use lagrange's after that I have the data in x of i and f of i you remember when you use the scanf function you are given ampersand sign here when you are reading you are pointing it to that addresses and storing the data files in this addresses to which this is pointing to, x of i and f of i is the arrays which I have to store the f value and the x value

and once I have that, I use exactly the one as the same before the Lagrange's form and I use the Lagrange's form to evaluate the function, to evaluate the polynomial and I am running it from x of 0 to x of n at a subset ".1".

So I have a set of data points here and I construct a Lagrange's from polynomial with this and I evaluate the function values the polynomial values in between this starting from this to this in steps of ".1", that is what I am doing. So that is what we would see here so and then, this is the this part is exactly the same as we saw before that we had y value and we had the product here that is evaluating the l of i the l_i of x_1 , l_i of x_1 and multiplying it by f of x_i and doing the sum here and the values I got, I am storing it into this thing called Lagrange's dot dat again so we will see this we will run this program and see what do we get.

Now I have to run this program, so I run this program, so that is the data points which we have just tabulated here, I just put it here again now that is the value which we are getting here. So now you can see.

(Refer Slide Time: 45:33)



Now this again the squares, the squares here are my function values which is given to me the squares are the function the values of which are given to me and the dot, this circular points are the interpolated values and you can see that wherever the circular points were close to this square points I have a good agreement where it is but it is quite ambiguous when it is away from that for example, this line decided to go down and go up and here for example, it is going up quite a bit and then coming down.

So we are not very sure that whether this is actually for example here this has to go down and go up this point. So, it is not very clear to us in the first in the initial value, initial stage you can see that when the curve starts, look at the square disc as the square points and the circular disc, they agree very well at the first point because that is exactly the

same value, so function value and the polynomial agrees wherever they are close they are exactly the same, so they agree and the next point it goes it varies a lot so given this set of square points it is not clear to us that this line shown by this circular discs is a good representation. So that is what I wanted to show you, so it is not clear that the polynomial which we got is the correct polynomial or, what is the error in this?

(Refer Slide Time: 47:10)

```
Lagrang1.c
#include<math.h>
#include<stdio.h>
main()
{
    int i,j,n;
    float x1,x[15],f[15],y,df,df1,xl;
    FILE *FP,*fp1;
    fp1=fopen("tointer.data","r");
    i=0;
    while(!feof(fp1))
    {
        fscanf(fp1,"%f" "%f",&x[i],&f[i]);
        i++;
    }
}
```

(Refer Slide Time: 47:19)

```
fclose(fp1);
n=i-2;
for(i=0;i<=n;i++)
printf("%d %f %f\n",i,x[i],f[i]);
xl=x[n];
FP=fopen("lagrange.dat","w");
for(x1=x[0],x1<=x[n],x1=x1+.1)
{
    y=0.0;
    for(i=0;i<=n;i++)
    {
        df=1.0;
```

There is some error, what is the error in this? And that is, what we want to quantify and that is, what we would look at in the next part.

(Refer Slide Time: 47:29)

```
for(j=0;j<=n;j++)
{
    if(j!=i)
    {
        df=df*(x1-x[j])/(x[i]-x[j]);
    }
}
y=y+df*f[i];
}
fprintf(FP,"%f %f \n",x1, y);
}
fclose(FP);
}
```

(Refer Slide Time: 47:40)

Error in the Interpolating polynomial

Let $P_n(x)$ be a polynomial of degree n which interpolates a real valued function $f(x)$ at x_0, \dots, x_n . If $f(x)$ is the actual value of the function at x , the error in this interpolation is $e_n = f(x) - P_n(x)$.

We will now try to estimate this error. Consider any point different from x_0, \dots, x_n . If $P_{n+1}(x)$ is a polynomial of degree $n+1$ which interpolates $f(x)$ at x_0, \dots, x_n and at \bar{x} , then $P_{n+1}(\bar{x}) = f(\bar{x})$.

We are given a polynomial p_n of x of degree n and we have which interpolates between 0 and n the function value, tabulated function value etcetera. Now the actual value of the function should be of course f of x , so f of x is the actual value of the function at any point x , let us say this is the function and this function is tabulated at x_0 up to x_n and we have constructed a polynomial p_n of x and we have constructed in such a way as we just saw in this particular example, that we have constructed it such a way that this p_n of x or the n the order polynomial agrees completely with f of x at these points.

So now, if x is any point which is different from $x_0, x_1, x_2, \dots, x_n$. Then, what is the error? We can define the error as f of x the actual value of the function which we do not know of course minus p_n of x . So that will be the error. So f of x is something which we do not know because, x has been tabulated f of x has been tabulated only for discrete set of points so that is x_0 to x_n and then we have f of x here and f of x minus the polynomial value at that x this we know now this is my error so now question is can we do not know this exactly but the question is can we estimate this error and so we will use the lagrange form, the newton form for doing this analysis here.

So let us, try to estimate this error how different is the p_n of x from f of x , so that is what we would try to get. So let us say, if p_n of x is a polynomial of degree n plus 1 which interpolates from f of x at x_0 to x_n and at x bar, so that is the idea. We will try to do, as I said x is a point which is not, which does not match with any of this x_0 to x_n . So now, we say, we assume that we know this, let us take this x as x bar that is what we have done here and we say that we know the function value at this point f of x bar and the polynomial is actually not of the order n it is of order n plus one because it also goes through x bar we force it to go through x bar.

So x bar is some value between x_0 and x_n , so now since it goes through x_0 and x_n and also through x bar. We have n plus 1, so n plus 2 points, this is the n plus 1 points here and then x bar so n plus 2 points and the polynomial which goes through all of that would be an order of degree n plus 1 so that, then we have the exact equation that p_{n+1} of x bar is equal to f of x bar because that is way p_{n+1} is constructed.

(Refer Slide Time: 50:59)

But according to newtons formula this can be written as

$$P_{n+1}(x) = P_n(x) + f[x_0, \dots, x_n, \bar{x}] \prod_{j=0}^n (x - x_j)$$

It gives,

$$f(\bar{x}) = p_{n+1}(\bar{x}) = p_n(\bar{x}) + f[x_0, \dots, x_n, \bar{x}] \prod_{j=0}^n (\bar{x} - x_j)$$

So remember, that p_n of x was a polynomial which went through n points n plus 1 points that is, x_0 to x_n and then p_{n+1} is a polynomial which goes through x_0 to x_n and also through x bar which is some value of x between x_0 and x_n . So we have that polynomial so then we can write p_{n+1} of x as so p_n of x plus p_n of x is the n th order polynomial

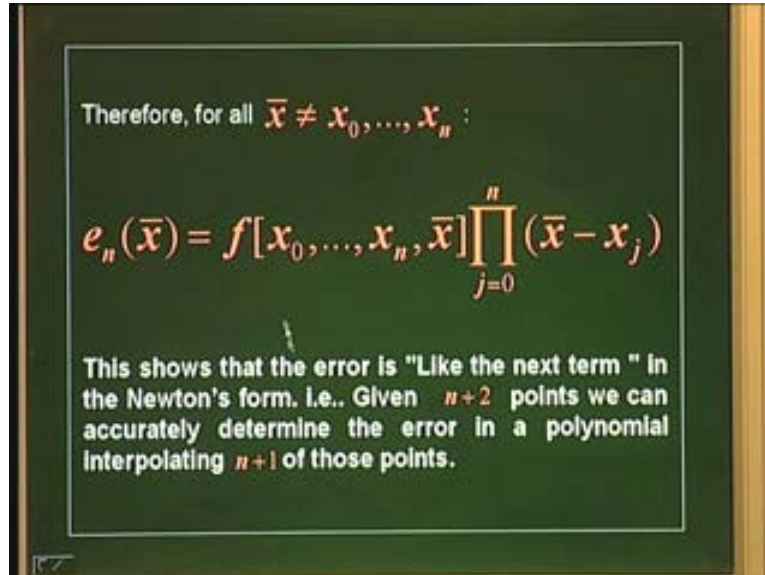
which went through x_0 to x_n and then we have an additional term, the last coefficient would be then f of $x_0 x_n x$ bar remember, the newton's form. So we have f of x_0 to x_n to x bar and then a product here that is x minus x_0 , x minus x_1 , x minus x_2 , x minus x_n into x minus x bar, x minus x_2 , x minus x_n up to that it will be there that is the last term. So remember, once again I want to emphasize this point that is p_n of x is a polynomial which went from which went through n plus one points x_0 to x_n and then we added an extra point x bar which is between x_0 and x_n and then we constructed a n plus 1 th order polynomial which also went through this .

Now I want to write this n plus one th order polynomial as this polynomial plus something, if I want to do that then what I would get would be this. So that is the last term, this is the last point this point is somewhere in between but this is an additional point which we have had. So, I have to construct this coefficient which is now f bracket $x_0 x_n$ and x bar and it will be multiplied by x minus this will be the n th order term n plus 1 th order term because this is a polynomial of order n and this is a polynomial of order n plus 1 and the n plus 1 th order term would then be x minus x_0 into x minus x_1 to x minus x_n , that will be n plus one th order. And then, if I evaluate this polynomial at x bar that will be exactly the same as f of x bar because, we are forced to go through that. So f of x bar is p_n plus 1 of x bar which is p_n of x bar plus this additional term into x bar minus x_j .

So now, what do we see, what do you wanted to see in the earlier thing was this difference, this difference at x bar at x equal to x bar what is the error of the polynomial n th order polynomial, if I evaluate this polynomial at any value x bar which is between this but not one of this. So that is exactly, what we are getting here, so we get that difference as this term. So what do we see is that the error in this polynomial is actually in this polynomial for any value x bar which is not any of these values is given by the next term in the polynomial so that is if you are given a polynomial of order n plus 1 it is like n plus 2 th order polynomial, n plus 2 th term. So n plus 2nd term in this.

So the next term in the polynomial is the error, the error now we can get this so the error in the polynomial would be like the next term. So can we make now again, we do not know this value set at every point so, can we make an estimate of this as I said we do not know x bar. So we assumed here, that we know f of x bar, when we write that I said f of x bar is p_n of x bar we said that we forced the polynomial to go through that point for that we need to know f of x bar so, otherwise we cannot construct this function here. So we do not know this f of x bar in reality but we can see that the error is of this form like the next term in the polynomial. So now, question is can we make an estimate of this, so that is what we should try to see whether we could make an estimate and then we could compute some examples of this actually.

(Refer Slide Time: 54:00)

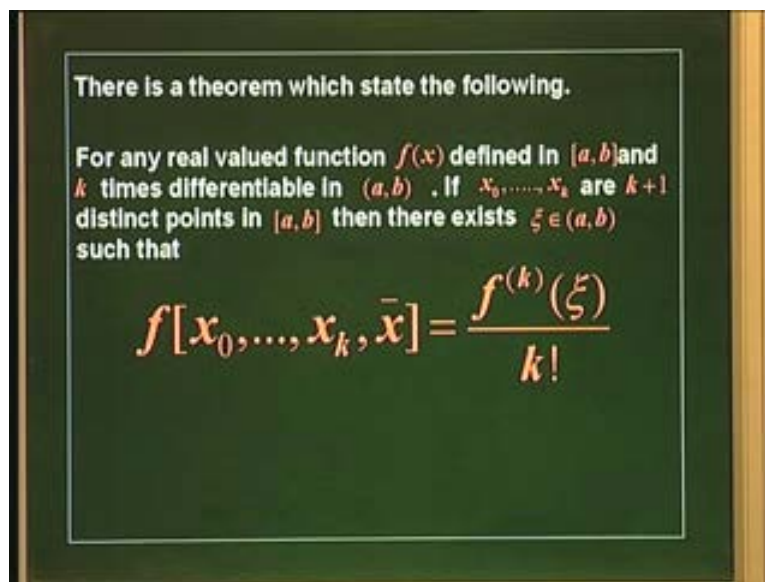


Therefore, for all $\bar{x} \neq x_0, \dots, x_n$:

$$e_n(\bar{x}) = f[x_0, \dots, x_n, \bar{x}] \prod_{j=0}^n (\bar{x} - x_j)$$

This shows that the error is "Like the next term " in the Newton's form. i.e.. Given $n+2$ points we can accurately determine the error in a polynomial interpolating $n+1$ of those points.

(Refer Slide Time: 55:00)



There is a theorem which state the following.

For any real valued function $f(x)$ defined in $[a, b]$ and k times differentiable in (a, b) . If x_0, \dots, x_k are $k+1$ distinct points in $[a, b]$ then there exists $\xi \in (a, b)$ such that

$$f[x_0, \dots, x_k, \bar{x}] = \frac{f^{(k)}(\xi)}{k!}$$

So what we **sue** is this theorem, this is called the mean value theorem, it says that if there is a function f of x which is defined between a and b , and it is k times differentiable in this interval a and b and so if x_0 to x_k are k plus 1 distinct points in a and b . And then, there exists a value between this interval a and b , such that this function is equal to the k th derivative of that function at that point divided by k factorial, this is the theorem. So I am not going to give a proof of this theorem, except to show you that in the first order k 's we can see this very easily. We call that derivative mean value theorem and we can see this very easily for the first order k . So the statement is again the following what we want to get is an estimate of this quantity.

Remember we did not know f of x bar, so unless we know the f of x bar we cannot compute this coefficient going from x_0 to x_k . We cannot compute that coefficient, so we need an estimate and what it turns out is that, this quantity is equal to the k th derivative divided by the k factorial. So we have an estimate of that of that already for some value of x , for some value of x between a and b this given by ζ here, so that is an estimate so we can actually maximize this derivative we can actually, since the function is differentiable in this interval k th time differentiable, so we could actually look at the k th order we do not know the function actually, what I am trying to say is that the estimate tells us that it is the k th derivative.

So if you assume the function to be smooth and differentiable then we already have an idea about how bad that function that polynomial would be or the error in that polynomial at any point would be, it will be of the order k th derivative divided by k factorial. So we see that as the degree of the polynomial increases, it increases pretty fast we will continue on this theme again in the next class and look at some details of what is the implication of such a term would be, that would be again in the next class.