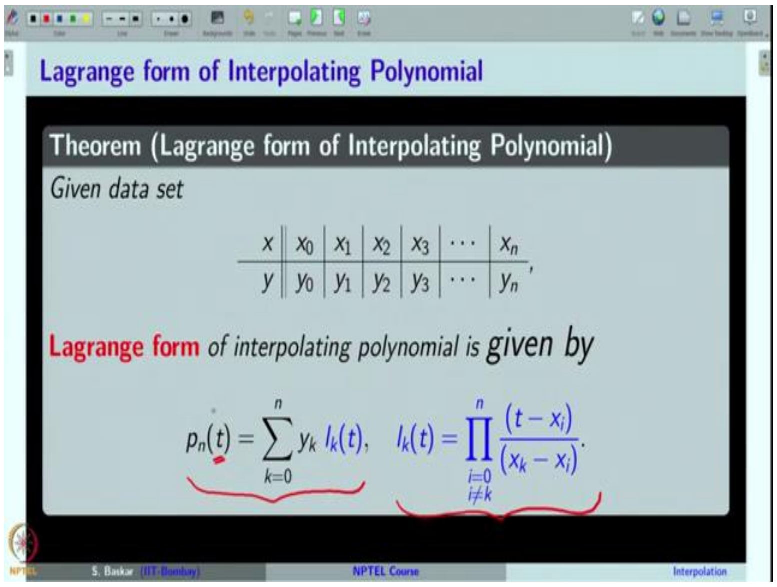
Numerical Analysis

Prof. S. Baskar
Department of Mathematics
Indian institute of Technology – Bombay

Lecture – 43
Polynomial Interpolation: Implementation of Lagrange Form as Python Code

Hi, we have learned the Construction of interpolating Polynomials for a given data set. We have learned two ways of constructing interpolating polynomials; one is, Newton's form of interpolating polynomials and another one is the Lagrange form of interpolating polynomials. In this class we will learn to implement the Lagrange Form of interpolating Polynomial as a Python Code.

(Refer Slide Time: 00:52)



Lagrange form of Interpolating Polynomial

Theorem (Lagrange form of Interpolating Polynomial)

Given data set

x	x_0	x_1	x_2	x_3	$\cdots$	x_n
y	y_0	y_1	y_2	y_3	$\cdots$	y_n

Lagrange form *of interpolating polynomial is given by*

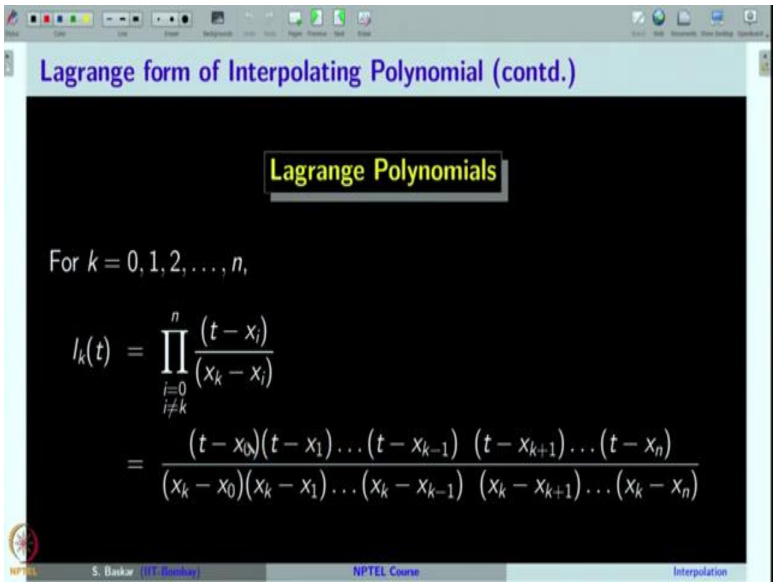
$$\underbrace{p_n(t) = \sum_{k=0}^{n} y_k\, l_k(t)}, \quad \underbrace{l_k(t) = \prod_{\substack{i=0\\ i\neq k}}^{n} \frac{(t-x_i)}{(x_k-x_i)}}.$$

S. Baskar (IIT-Bombay) NPTEL Course Interpolation

Let us quickly recall the Lagrange form of the interpolating polynomial. We are given a data set where we have *x* coordinate and *y* coordinate. The *x* coordinate includes $n + 1$ node points and the *y* coordinate includes the corresponding values of the given function at the node points $x_0, x_1, \cdots, x_n$. Now, given this data set, we can construct the Lagrange form of the interpolating polynomial which is given by $p_n(t) = \sum_{k=0}^{n} y_k l_k(t)$.

Where all this l_k's are the Lagrange polynomials given by this expression. Here, I would like to highlight the change in the notation when compared to the notation we have used in our lecture session. I have used the variable *t* here. Whereas in the lecture session we have used *x* instead of *t* here. This is just for the sake of convenience in the coding path.

(Refer Slide Time: 02:25)



Lagrange form of Interpolating Polynomial (contd.)

Lagrange Polynomials

For $k = 0, 1, 2, \ldots, n$,

$$l_k(t) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{(t - x_i)}{(x_k - x_i)}$$

$$= \frac{(t - x_0)(t - x_1)\ldots(t - x_{k-1})\ (t - x_{k+1})\ldots(t - x_n)}{(x_k - x_0)(x_k - x_1)\ldots(x_k - x_{k-1})\ (x_k - x_{k+1})\ldots(x_k - x_n)}$$

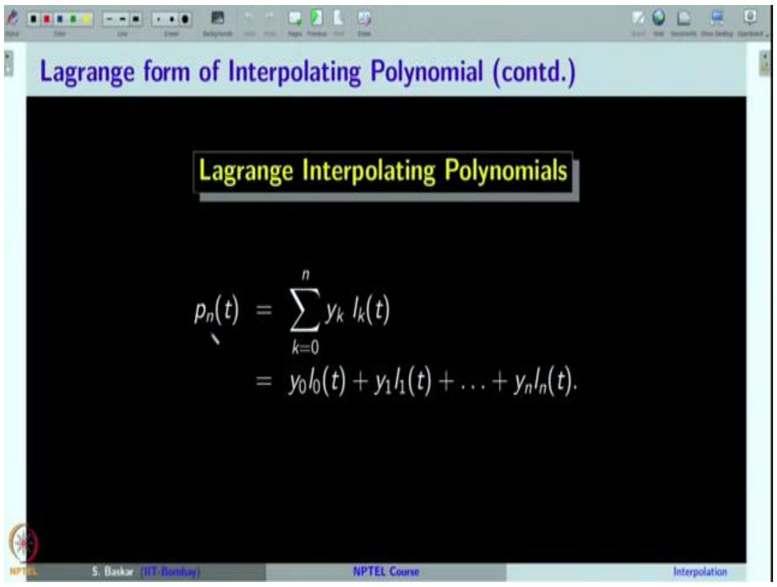
S. Baskar (IIT-Bombay) NPTEL Course Interpolation

In this lecture video, let us take the Lagrange polynomial and have a close look at it. First point to note is that we have to construct the Lagrange polynomial for every k = 0, 1, 2 up to n. Remember what is this n? This n is coming from the data set where we have given n + 1 number of node points and the corresponding values at the node points. So, for each node point we have to construct the Lagrange polynomial as the first step.

Here you can notice that for each k, $l_k(t)$ is written as the product of the terms $\frac{t-x_i}{x_k-x_i}$, where the product excludes the term $i = k$. That is, $l_k(t)$ is given by this expression. Here t is the variable, you can see t appears in the numerator where all this x_i's are the node points given in our data set. Note that the Lagrange polynomial depends on the node points and therefore depends on the number of node points given in the data set.

Whereas they are independent of the y values given in the data set. Also note that the denominator does not depend on t, it only depends on the node points and the numerator includes t. These are something which we have to remember when we are coding. Since the denominator does not depend on t, we can compute the denominator once for each k and store in an array of dimension k whereas the numerator has to be computed for every t.

(Refer Slide Time: 04:59)



Lagrange form of Interpolating Polynomial (contd.)

Lagrange Interpolating Polynomials

$$p_n(t) = \sum_{k=0}^{n} y_k l_k(t)$$
$$= y_0 l_0(t) + y_1 l_1(t) + \ldots + y_n l_n(t).$$

Let us keep this observation in mind and go ahead. Once the Lagrange polynomial is computed for each t. That is, once this quantity is computed for each t and for all k, then we can go to obtain the interpolating polynomial $p_n(t)$ using this formula. This is given by $y_0 l_0(t) + y_1 l_1(t) + \cdots + y_n l_n(t)$. Remember, this summation has to be done for every t at which we want to obtain the value of the interpolating polynomial $p_n(t)$.

Let us now go to the coding part. **(Video Starts: 05:49)** Let us see, what are all the inputs that we are taking in our code. First is the function f that needs to be approximated and then we will take the number of nodes as the input. We will also take the node points as the n dimensional array. And finally, we have to take the t values at which we want to finally obtain the interpolating polynomial values which may be taken as an m dimensional array.

Remember, n has to be given as an input. Then we have to generate the nodes and separately we should also generate the variable t at which we want the polynomial values. Let us get into the code and see how to take these inputs. Of course, the first part of the code is, as usual containing the import commands where we are importing two modules, numpy and mathplotlib.

I hope, I do not need to explain you these commands now we are familiar with these modules, as we have used them in our previous lectures of the python code. Let us go to see the inputs. As we have seen in our previous course, if you want to define a function through its expression, you have to use lambda and then the variable. Here I am using x for the variable of the function f.

Therefore, we have to type lambda x and then a colon. After this, we have to give the expression of the function. Here, I am interested to develop the interpolating polynomial to approximate the function $sin\,2\,x$. Therefore, my expression is sin(2*x). Remember sin is not an inbuilt command in python but it is included in the numpy module. Therefore, we have to give np.sin. The next input is the number of nodes in the data set.

It means, I am interested in constructing the Lagrange’s interplating polynomial of degree 3 here because I am giving the input as 4. Suppose I want to develop the Lagrange polynomial of degree 2 then remember, I have to give n = 3 that will generate a quadratic polynomial. Because from our theory class we have seen that if your data set has n + 1 nodes then the interpolating polynomial should be of degree less than or equal to n.

Therefore, if you have 3 node points then you should have the degree as something less than or equal to 2. Here note that in the theory part we have used the n + 1 but the same here we are using only n. This is just for the programming convenience, so, you have to keep this discrepancy in mind between the notation used theoretically and the notation used in the code.

Let us go to the next input, here we are taking the node points, as our next input observe that we are using the command called arrange again. This command is borrowed from the numpy module and hence we have to prefix np to the command arrange. Well, what does the command arrange do? Let us see the arrange command generates an array with value starting from – 1.5.

That is what we have given here and then ends with the number approximately equal to 1. I have given it 1.001 just to take care that 1 is surely included in our node point. And it generates in between numbers with step size given like this that is 2.5 divided by (n – 1) where n is given here. Observe that 2.5 is nothing but the length of the interval – 1.5 to 1. So, this arrange command precisely gives us the partition of the interval – 1.5 to 1 with n number of partition points.

This is precisely what we want as nodes. So, we are generating a set of equally spaced nodes in the interval – 1.5 to 1 using the arrange command and stored in the variable x. This is what we are doing at this line. Our final input is the variable t at which we want to finally, get the approximate values of our function f through the interpolating polynomials. Again, we are generating points between the first value of the array x which is in our case – 1.5.

That is stored in the variable x[0] and it goes till the last point of the x array which is denoted by x of length of x – 1. Remember length of x is nothing but n here. That is the way we have created the array x. Therefore, we are again creating a partition of the interval – 1.5 to approximately 1. But now, with the step size much less, it is fixed here as 0.001. This is because I want to get the approximation of my function f through the interpolating polynomial at many points in the interval – 1.5 to 1.

That is why I am generating the t variable with step size much smaller. Well, I hope you have understood the input commands. Let us now go to the main program well, before going into the main construction of the interpolating polynomial, let us see how we want to get the output. Finally, from our code, the output is obviously the interpolating polynomial values at the points specified in the variable t.

Note that we are not going to construct a polynomial explicitly but only the point wise values of the polynomial are computed in our code. We would also like to display the graph of the polynomial, along with the graph of the function f. With this note, let us now go into the main construction of the interpolating polynomial. As I have mentioned earlier, the denominator of the Lagrange polynomial that is this expression does not depend on t.

Therefore, this part of the Lagrange polynomial can be constructed once for all and then stored in a variable. The denominator is therefore, computed separately and let us store it in a variable called ld Lagrange denominator and use this expression in the denominator to compute the Lagrange polynomial for every t here. You see, you do not need to compute this value for every t that will reduce the computational cost drastically.

Let us see how to compute this denominator terms. Let us first see how many loops are needed to compute this. Well, we need two loops to compute ld’s, one is the loop with index k and it varies from 0 to n. Remember, n is the number of nodes that we have taken in our data set. That is, we have taken n + 1 nodes in our data set. Then we should also have another for loop to compute this product and this for loop will run from 0 to n.

And let us use the index i for this loop. Remember in this loop we have to exclude one term which is precisely i = k. Let us now see how to compute ld[k]. For each k remember ld includes the product. Therefore, we will initialize ld with the value 1 in it and the length of the array ld is n. That is, it has to start from 0 and it should go up to n – 1. And ld is computed in this loop.

Let us go into this loop and see how we are computing the product involved in the expression of ld. Remember we have initialized ld with value 1. That means ld will hold the value 1 when it enters into the loop for the first time. Remember, the first loop will run for k = 0 to n – 1. That will correspond to ld of k and in each ld of k we have a product. That product will run from again 0 to n – 1 and we have to exclude one term that is i = k.

Again I emphasize, theoretically we are taking totally n + 1 node points, in the input we have given 3 as n + 1. But for the notational convenience we have stored it in the variable n only. Therefore, wherever you see, n in the code you have to theoretically remember that it is n + 1 there is a slight confusion here. Generally, theoretically, we take the index from 0 ,1, $\cdots, n$ and take the data set as $x_0, x_1, \cdots, x_n$.

Therefore, we have n + 1 node points that gives us the polynomial of degree less than or equal to n. Since we do not want to define our variable as n + 1 we are just giving the name n for it. But keeping in mind that wherever we have n it is actually n + 1. So, this is slightly confusing but we have to remember this and then see the code accordingly. Therefore, all our loops should go from 0 to n – 1. Theoretically, it will go from 0 to n.

Let us now understand how this inner loop works, as we have initialized ld[k] as 1 initially, when you come for i = 0, you have ld[k] = 1. Therefore, ld[k] will be 1 * x[k] – x[0] and that will be computed and stored in the variable ld[k]. When you come for i = 1, that is, once this line is computed then again, the control of the python code will go to the first line of the for loop.

Again, it will come check whether i is not equal to k, if this is satisfied again, it will go into computing this product. Suppose this happens at i = 1 then what will be now stored in ld[k]? Well, it will be ld[k] which now has this value into it will take this value that is x[k] – x[i] now, i =1. Therefore, it takes 1 and then it puts this product value into ld[k]. Again, the control will go to the first line of the for loop.

Then it will check whether i is not equal to k. If i = 2 and i is not equal to k then again it will come to find ld[k]. And that will be now this entire product will come here into x[k] – x[2]. So, like that this product will go on for each i ranging from 0 to n – 1 and every time it will check this condition, if it is not satisfied then it will find the product, otherwise it will simply skip to the next value of i.

At the end of this loop, we will have precisely ld[k] which is given mathematically as this expression. So, we have computed ld[k] for each k = 0, 1 upto n – 1 in the python code, whereas mathematically it should be 0 to n. Now, we will go to compute the Lagrange polynomial note that the Lagrange polynomial have to be computed for each *t*. Also, the polynomial *p* has to be computed for each *t*.

Therefore, it may be computationally efficient to construct both l and p together. Let us first understand how the loops are to be made to compute l and p. Well, you can clearly see that the outer loop is a for loop for t variable. This is because both l and p have to be computed for each t. Let us use the index j for this for loop and the outermost loop is running for the t variable with index as j.

Therefore, it has to run from 0 up to 1 less than the length of the t array. For each t, we have to compute l[k] where k running from 0 to n – 1 and for each k we have to compute the numerator product which needs another loop, whose index is denoted by i. You can see from this expression that for each t we have to find all the l[k]'s. What are they? They are the Lagrange polynomials and k ranges from 0 to n – 1 in the code and for each l[k] for each t.

The l[k] computation includes a product. Remember this product is very much similar to how we computed ld. So, therefore, once you understand how we computed this product, this can be similarly coded once you find this product. Then, finally, you have to divide it by ld[k] and that is precisely the expression for l[k] that we have seen in our slides. Let us see how to do this product, so, we are doing this product in this line.

Remember for each k, we have to find l[k] and for each l[k] we have to do the product and in this product we have to take care that i should not be equal to k. This is something similar to what we did in computing ld[k] also. The same idea will go on here and we can compute l[k] for each k = 0 to n – 1 and the product will run from 0 to again n – 1, excluding i = k. Once you finish this product then you have to come out of this loop and then find l[k] divided by the denominator term**.**

That is computed here you will divide the numerator product by this denominator product and that is precisely defined as the Lagrange polynomial. Remember this has to be done for each t and observe that this denominator value will not change by changing t. That is why once for all computed it here and stored in the array ld and we are using it directly here. Otherwise, this entire product has to be once again calculated here for each t that will drastically increase the computational cost but here, we are not doing it.

Therefore, we are saving all this computation. So, we have computed all the Lagrange polynomials l[k]’s. Remember this l[k]’s are computed for every t. So that is very important this needs to be done for every t. Once we are done with all the l[k]’s for a given t then we have to go for finding the value of the interpolating polynomial at t and store it in the variable p[j]. So, this is what we are doing here.

Also, you find all the Lagrange polynomials for a particular t. And then come to find p of t with this expression where all this Lagrange polynomials are plugged in here and then you have to multiply them with the corresponding function values. Let us see how to do it in the code. So, as the last part of this main loop, we are now running a loop with index i varying from 0 to n – 1 and here we are summing the terms in the interpolating polynomials.

For this we are first initializing all the p arrays as 0. Remember p should have the length as the length of the variable t because we want to find the interpolating polynomials value at each t. Therefore, p should have the same length as t variable and when we are initializing this array, we are initializing it with value 0. Because we are going to run the loop for summation, whereas when we were initializing ld here, we were initializing it with value 1.

Similarly, the Lagrange polynomial l was initialized with again value 1 and that is done for each value of t. Because once you find the value of p at a time then when you go to find the value at the next t variable you have to re-initialize this l. That is why we are putting this initialization within this loop. All other variables are initialized previously, so, once you get the polynomial value at one particular t again, you are going back to this for loop.

You are forgetting all the Lagrange polynomials constructed for the previous t. Now, you are initializing again the Lagrange polynomials as 1 with the array length as n and then again going and computing the Lagrange polynomials for now the new value of t and then once Lagrange polynomials are constructed. Then you are coming and using the interpolating polynomial formula with the function value fv which we have initialized at the beginning itself, fv is taken as the function values at the nodes.

Remember we have taken the function values at the nodes and saved that values in fv. And this variable is finally, used when we are constructing the interpolating polynomial here because if you recall $p(t) = \sum_{i=0}^{n} f(x_i) l_i(t)$. That is the formula, $f(x_i)$ is computed and stored in the variable fv[i] and l[i]'s are computed here and then this summation is done in this loop.

Remember the only thing is, theoretically, we are running from 0 to n that we are making 0 to n – 1 in the python code by appropriately giving the value of n. So, this completes the construction of the Lagrange polynomial and now let us see how to take the output. Just for the information, we are printing the number of nodes used and the degree of the corresponding polynomial which is 1 less obviously.

And then we are showing the output in the form of a graph I will leave it to you to understand how this graph is plotted. Let us go to see the output of this program. Well, the code is executed and the output is shown here. You can see that our print command is executed and the output is shown here. You can also see that our plotting commands are executed and the graph is generated and shown here.

The black dots denote the position of the data given in our data set. Red solid line represents the graph of the function f and the blue dashed line represents the graph of the computed interpolating polynomial $p_n(x)$ using the Lagrange form of the interpolating polynomial for the given data set. We can change the value of n to see the corresponding data positions. Let us go to do that. Let us change the value of n as 4.

With this, we are supposed to get a cubic polynomial. Let us see, the position of the data sets are now shown here we have given 4 node points and the corresponding function values are obtained in the code. And then they are plotted here and the interpolating polynomial is shown again in the blue line. Now, it is a polynomial of degree 3 that is cubic polynomial and you can see that the interpolating condition is satisfied. How are you seeing this?

At the node points the function value and the polynomial values are coinciding. That is precisely the interpolation condition. In the previous case, we have plotted the quadratic polynomial. Let me go back to that and show you when you put n = 3. It means you have generated the quadratic interpolating polynomial. You have 3 node points and therefore, the degree of the polynomial is 2.

And the quadratic interpolating polynomial is shown in the blue line and the polynomial value and the exact value are coinciding nicely at the node points confirming the interpolating condition. You can also increase the number of nodes, as you want and also you can change the function here and you can play around with this. Let me put the value of n as, say 5 and you can see now you will have the interpolating polynomial as the fourth degree polynomial which is shown here.

And the corresponding graph of the fourth degree interpolating polynomial is shown in the blue line. **(Video Ends: 34:52)** I hope you understood the python implementation of the Lagrange form of interpolating polynomials. Thank you for your attention.